

# 빌드 및 배포 환경

## DockerFile

- FrontEnd

```
FROM node:20-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
# 타입 검사 없이 빌드 실행
RUN NEXT_IGNORE_TS_ERRORS=1 npm run build

# 프로덕션 환경에서는 dev dependencies 제외
FROM node:20-alpine AS production
WORKDIR /app
COPY --from=build /app/package*.json ./
RUN npm ci --only=production
COPY --from=build /app/.next ./next
COPY --from=build /app/public ./public
# next.config.js가 없을 경우에도 실패하지 않음
COPY --from=build /app/next.config.mjs ./

EXPOSE 3000
CMD ["npm", "run", "start"]
```

- BackEnd

```
# 1. OpenJDK 이미지 사용
FROM openjdk:17-jdk-alpine AS build

# 2. 작업 디렉토리 설정
WORKDIR /app
```

# 3. 필요한 파일 복사 (전체가 아닌 필요한 부분만)

COPY ./src ./src

COPY ./build.gradle ./build.gradle

COPY ./settings.gradle ./settings.gradle

COPY ./gradlew ./gradlew

COPY ./gradle ./gradle

# 4. Gradle 파일 실행 권한 부여 (Linux 환경에서 필요)

RUN chmod +x ./gradlew

# 5. Gradle 빌드 실행

# 테스트용 RUN 코드

# RUN ./gradlew bootJar -x test -PincludePackage=com.b110.jjeonchongmu.domain.test

RUN ./gradlew bootJar -x test --no-daemon

# 6. 런타임 이미지 생성

FROM openjdk:17-jdk-alpine

WORKDIR /app

COPY --from=build /app/build/libs/\*.jar app.jar

# 7. 실행 명령어 설정

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]

## Docker Compose

- blue-green의 각 파일은 컨테이너 이름만 상이하며 내용은 동일함.

services:

client-blue:

build:

context: ./client

dockerfile: Dockerfile

container\_name: client-blue

ports:

```

- "3001:3000"
depends_on:
- server-blue
environment:
- NODE_ENV=production
- NEXT_PUBLIC_API_URL=/api
# - NEXT_PUBLIC_API_URL=https://server-blue:8080
restart: always
networks:
- app-network

server-blue:
build:
context: ./server
dockerfile: Dockerfile
container_name: server-blue
ports:
- "8081:8080"
environment:
- SPRING_PROFILES_ACTIVE=${SPRING_PROFILES}
- MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
- MYSQL_DATABASE=${MYSQL_DATABASE}
- MYSQL_USER=${MYSQL_USER}
- MYSQL_PASSWORD=${MYSQL_PASSWORD}
- SPRING_DATASOURCE_URL=jdbc:mysql://my-db:3306/${MYSQL_DA
DATABASE}?useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=
UTF-8&allowPublicKeyRetrieval=true
- SPRING_DATASOURCE_USERNAME=${MYSQL_USER}
- SPRING_DATASOURCE_PASSWORD=${MYSQL_PASSWORD}
- JWT_SECRET=${JWT_SECRET}
- JWT_ACCESS_TOKEN_VALIDITY=${JWT_ACCESS_TOKEN_VALIDITY}
- JWT_REFRESH_TOKEN_VALIDITY=${JWT_REFRESH_TOKEN_VALIDIT
Y}
- MY_DB=${MY_DB}
- EXTERNAL_BANK_API_URL=${EXTERNAL_BANK_API_URL}
- EXTERNAL_BANK_API_KEY=${EXTERNAL_BANK_API_KEY}
- EXTERNAL_BANK_API_ACCOUNTTYPE=${EXTERNAL_BANK_API_AC
COUNTTYPE}

```

```
- EXTERNAL_USER_API_URL=${EXTERNAL_USER_API_URL}
- SPRING_DATA_REDIS_HOST=${SPRING_DATA_REDIS_HOST}
- SPRING_DATA_REDIS_PORT=${SPRING_DATA_REDIS_PORT}
- SPRING_DATA_REDIS_PASSWORD=${SPRING_DATA_REDIS_PASSWORD}
RD}
```

depends\_on:

my-db:

condition: service\_healthy

my-cache-server:

condition: service\_healthy

networks:

- app-network

restart: always

my-db:

image: mysql:8.0

environment:

- MYSQL\_ROOT\_PASSWORD=\${MYSQL\_ROOT\_PASSWORD}

- MYSQL\_DATABASE=\${MYSQL\_DATABASE}

- MYSQL\_USER=\${MYSQL\_USER}

- MYSQL\_PASSWORD=\${MYSQL\_PASSWORD}

# - MYSQL\_ROOT\_HOST=%

command: --default-authentication-plugin=mysql\_native\_password

volumes:

# - ./mysql\_data:/var/lib/mysql

- mysql\_data\_volume:/var/lib/mysql

- ./mysql-init:/docker-entrypoint-initdb.d

ports:

- 3307:3306

networks:

- app-network

healthcheck:

test: [ "CMD", "mysqladmin", "ping" ]

interval: 5s

retries: 10

my-cache-server:

image: redis

```

ports:
  - 6379:6379
# command: redis-server --port 6379 --protected-mode yes
command: redis-server --port 6379 --requirepass ${SPRING_DATA_REDIS_PASSWORD}
volumes:
  # - ./redis_data:/data
  - redis_data_volume:/data
networks:
  - app-network
healthcheck:
  test: [ "CMD", "redis-cli", "-a", "${SPRING_DATA_REDIS_PASSWORD}",
"ping" ]
  interval: 5s
  retries: 10

# nginx:
# image: nginx:latest
# ports:
#   - "3001:80"
#   - "443:443"
# volumes:
#   - ./nginx/conf.d:/etc/nginx/conf.d      # 로컬 설정 파일을 컨테이너 내부로
마운트
#   - /etc/letsencrypt:/etc/letsencrypt:ro  # SSL 인증서 마운트
# depends_on:
#   - client
# restart: always
# networks:
#   - app-network

volumes:
  mysql_data_volume:
  redis_data_volume:

networks:
  app-network:
    driver: bridge

```

## Jenkins 파이프라인

```
// 한국 시간으로 날짜 포맷팅하는 함수
def getKoreanTime() {
    def now = new Date()
    def koreanTimeZone = TimeZone.getTimeZone("Asia/Seoul")
    def sdf = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
    sdf.setTimeZone(koreanTimeZone)
    return sdf.format(now)
}

// 현재 활성화된 서버(Blue/Green) 확인 함수
def getCurrentActiveServer() {
    // nginx 설정 파일에서 현재 활성화된 서버 확인
    def nginxConf = sh(script: "cat ${PROJECTS_DIR}/${PROJECT_NAME}/n
    ginx/conf.d/bluegreen.conf", returnStdout: true).trim()

    // 백엔드 서버 확인
    if (nginxConf.contains("server localhost:8081") && !nginxConf.contains
    ("# server localhost:8081")) {
        return "blue"
    } else if (nginxConf.contains("server localhost:8082") && !nginxConf.con
    tains("# server localhost:8082")) {
        return "green"
    }

    // 기본값은 blue
    return "blue"
}

// 서버 상태 체크 함수 최적화
def checkServerHealth(port) {
    def maxRetries = 2
    def retryInterval = 3

    // 내부 포트 확인 - 출력 없이 수행
```

```

sh(script: "docker exec server-${env.TARGET_SERVER} netstat -tulpn | g
rep 8080", returnStatus: true)

// 헬스체크 시도
for (int i = 0; i < maxRetries; i++) {
    try {
        def response = sh(script: "curl -s -m 5 http://j12b110.p.ssafy.io:${po
rt}/api/test/health", returnStdout: true).trim()

        // 응답 확인 - 성공 조건 단순화
        if (response && (response.contains("UP") || response.contains("ok")
|| response.contains("OK"))) {
            return true // 성공하면 즉시 반환
        }

        // 마지막 시도가 아닐 때만 대기 메시지 출력
        if (i < maxRetries - 1) {
            echo "재시도 중... (${i+1}/${maxRetries})"
            sh "sleep ${retryInterval}"
        }
    } catch (Exception e) {
        // 마지막 시도가 아닐 때만 대기
        if (i < maxRetries - 1) {
            echo "재시도 중... (${i+1}/${maxRetries})"
            sh "sleep ${retryInterval}"
        }
    }
}

// 로그로 서버 상태 확인 - 간소화
echo "API 확인 실패, 로그 확인 중..."
return sh(script: "docker logs server-${env.TARGET_SERVER} | grep 'Sta
rted JjeonchongmuApplication'", returnStatus: true) == 0
}

pipeline {
    agent any

```

```

options {
    // 빌드 타임아웃 설정
    timeout(time: 30, unit: 'MINUTES')
    // 병렬 빌드 처리 최적화
    parallelsAlwaysFailFast()
    // 깃 체크아웃 깊이 최적화
    skipDefaultCheckout(true)
}

environment {
    // GitLab 저장소 정보
    GIT_REPO = 'lab.ssafy.com/s12-fintech-finance-sub1/S12P21B110.git'
    BRANCH_NAME = 'develop'
    PROJECT_NAME = 'S12P21B110'
    PROJECTS_DIR = '/var/jenkins_home/projects'
    // Docker Compose 설정
    DOCKER_COMPOSE_BLUE = 'docker-compose.blue.yml'
    DOCKER_COMPOSE_GREEN = 'docker-compose.green.yml'
    // Nginx 설정 파일 경로
    NGINX_BLUEGREEN_CONF = '${PROJECTS_DIR}/${PROJECT_NAME}/
nginx/conf.d/bluegreen.conf'
    // 빌드 캐시 설정
    GRADLE_OPTS = "-Dorg.gradle.daemon=false -Dorg.gradle.parallel=true -Dorg.gradle.workers.max=4"
    DOCKER_BUILDKIT = "1"
}

triggers {
    // gitlab 웹훅
    gitlab(triggerOnPush: true, triggerOnMergeRequest: true, branchFilter
Type: 'NameBasedFilter', includeBranchesSpec: 'develop')
}

stages {
    stage('Checkout') {
        steps {
            script {
                // projects 디렉토리로 이동

```



```

sh "mkdir -p ${PROJECTS_DIR}"

// 프로젝트 디렉토리가 이미 존재하는지 확인
def projectExists = fileExists "${PROJECTS_DIR}/${PROJECT_
NAME}"

if (projectExists) {
    // 빠른 업데이트를 위해 fetch와 reset만 수행
    dir("${PROJECTS_DIR}/${PROJECT_NAME}") {
        withCredentials([usernamePassword(credentialsId: 'gitlab-
credentials', usernameVariable: 'GIT_USER', passwordVariable: 'GIT_PAS
S')) {
            sh "git remote set-url origin https://${GIT_USER}:${GIT_
PASS}@${GIT_REPO}"
            sh "git fetch --depth=1 origin ${BRANCH_NAME}"
            sh "git reset --hard origin/${BRANCH_NAME}"
        }
    }
} else {
    // 없으면 새로 shallow clone (빠른 클론을 위해)
    dir("${PROJECTS_DIR}") {
        withCredentials([usernamePassword(credentialsId: 'gitlab-
credentials', usernameVariable: 'GIT_USER', passwordVariable: 'GIT_PAS
S')) {
            sh "git clone --depth=1 -b ${BRANCH_NAME} https://
${GIT_USER}:${GIT_PASS}@${GIT_REPO}"
        }
    }
}
}
}

stage('Initialize Environment') {
    parallel {
        stage('Check Nginx Config') {
            steps {
                script {

```

```

        // bluegreen.conf 파일이 있는지 확인
        def nginxConfExists = fileExists "${PROJECTS_DIR}/${PROJECT_NAME}/nginx/conf.d/bluegreen.conf"

        if (!nginxConfExists) {
            echo "Blue/Green Nginx 설정 파일이 없습니다. 새로 생성합니다."

            sh "mkdir -p ${PROJECTS_DIR}/${PROJECT_NAME}/nginx/conf.d"

            writeFile file: "${PROJECTS_DIR}/${PROJECT_NAME}/nginx/conf.d/bluegreen.conf", text: "'# 블루/그린 설정...'"
        }

        echo "Blue/Green Nginx 설정 파일이 준비되었습니다."
    }
}

stage('Check Docker Compose Files') {
    steps {
        script {
            dir("${PROJECTS_DIR}/${PROJECT_NAME}") {
                // docker-compose 파일 존재 확인 및 생성
                def blueComposeExists = fileExists "docker-compose.blue.yml"

                def greenComposeExists = fileExists "docker-compose.green.yml"

                if (!blueComposeExists || !greenComposeExists) {
                    echo "Blue/Green Docker Compose 파일 생성..."
                    sh "cp -f docker-compose.yml docker-compose.blue.yml || true"

                    sh "cp -f docker-compose.yml docker-compose.green.yml || true"

                    // 병렬 처리를 위해 sed 명령 분리
                    sh '''
                        # Blue 서버 설정
                        sed -i 's/server:/server-blue:/g' docker-compose.blue.yml
                    '''
                }
            }
        }
    }
}

```

```

ue.yml &
    sed -i 's/client:/client-blue:/g' docker-compose.bl
e.yml &
    sed -i 's/- "8080:8080"/- "8081:8080"/g' docker-c
ompose.blue.yml &
    sed -i 's/- "3000:3000"/- "3001:3000"/g' docker-c
ompose.blue.yml &

    # Green 서버 설정
    sed -i 's/server:/server-green:/g' docker-compose.
green.yml &
    sed -i 's/client:/client-green:/g' docker-compose.gr
een.yml &
    sed -i 's/- "8080:8080"/- "8082:8080"/g' docker-c
ompose.green.yml &
    sed -i 's/- "3000:3000"/- "3002:3000"/g' docker-c
ompose.green.yml &

    wait # 모든 백그라운드 작업 완료 대기
    ...
}
}
}
}
}
}
}
}

stage('Determine Target & Prepare Environment') {
    parallel {
        stage('Determine Target Server') {
            steps {
                script {
                    try {
                        // 현재 활성 서버 확인
                        env.ACTIVE_SERVER = getCurrentActiveServer()
                    } catch (Exception e) {
                        echo "현재 활성 서버를 확인할 수 없습니다. 기본값으로 blue

```

를 사용합니다."

```
        env.ACTIVE_SERVER = "blue"
    }

    // 대상 서버 결정
    env.TARGET_SERVER = (env.ACTIVE_SERVER == "blue") ?
"green" : "blue"

    // 포트 설정
    env.TARGET_BACKEND_PORT = (env.TARGET_SERVER ==
"blue") ? "8081" : "8082"
    env.TARGET_FRONTEND_PORT = (env.TARGET_SERVER =
= "blue") ? "3001" : "3002"

    echo "현재 활성 서버: ${env.ACTIVE_SERVER}"
    echo "배포 대상 서버: ${env.TARGET_SERVER}"

    // 사용할 Docker Compose 파일 결정
    env.DEPLOY_COMPOSE_FILE = (env.TARGET_SERVER ==
"blue") ? "${DOCKER_COMPOSE_BLUE}" : "${DOCKER_COMPOSE_GREE
N}"

    echo "사용할 Docker Compose 파일: ${env.DEPLOY_COMP
OSE_FILE}"
    }
    }
}

stage('Fast Tests') {
    steps {
        dir("${PROJECTS_DIR}/${PROJECT_NAME}/server") {
            // 빠른 컴파일 테스트만 실행
            sh '''
                chmod +x ./gradlew
                ./gradlew classes -x test --parallel --build-cache || true
            '''
            echo "기본 컴파일 검증 완료"
        }
    }
}
```

```

    }
  }
}

stage('Prepare Environment Variables') {
  steps {
    dir("${PROJECTS_DIR}/${PROJECT_NAME}") {
      withCredentials([
        string(credentialsId: 'mysql-root-password', variable: 'MYSQL_ROOT_PASSWORD'),
        string(credentialsId: 'mysql-database', variable: 'MYSQL_DATABASE'),
        string(credentialsId: 'mysql-app-user', variable: 'MYSQL_USER'),
        string(credentialsId: 'mysql-app-password', variable: 'MYSQL_PASSWORD'),
        string(credentialsId: 'my-db-host', variable: 'MY_DB'),
        string(credentialsId: 'jwt-secret', variable: 'JWT_SECRET'),
        string(credentialsId: 'external-bank-api-url', variable: 'EXTERNAL_BANK_API_URL'),
        string(credentialsId: 'external-bank-api-key', variable: 'EXTERNAL_BANK_API_KEY'),
        string(credentialsId: 'external-bank-api-accounttype', variable: 'EXTERNAL_BANK_API_ACCOUNTTYPE'),
        string(credentialsId: 'external-user-api-url', variable: 'EXTERNAL_USER_API_URL'),
        string(credentialsId: 'jwt-access-token-validity', variable: 'JWT_ACCESS_TOKEN_VALIDITY'),
        string(credentialsId: 'jwt-refresh-token-validity', variable: 'JWT_REFRESH_TOKEN_VALIDITY'),
        string(credentialsId: 'spring-data-redis-host', variable: 'SPRING_DATA_REDIS_HOST'),
        string(credentialsId: 'spring-data-redis-port', variable: 'SPRING_DATA_REDIS_PORT'),
        string(credentialsId: 'spring-data-redis-password', variable: 'SPRING_DATA_REDIS_PASSWORD'),
        string(credentialsId: 'next-public-api-url', variable: 'NEXT_PUBLIC_API_URL')
      ])
    }
  }
}

```

```

    }) {
        // Docker Compose용 .env 파일 생성
        sh """
            echo "MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASS
WORD}" > .env
            echo "MYSQL_DATABASE=${MYSQL_DATABASE}" >> .en
v
            echo "MYSQL_USER=${MYSQL_USER}" >> .env
            echo "MYSQL_PASSWORD=${MYSQL_PASSWORD}" >> .
env
            echo "MY_DB=${MY_DB}" >> .env
            echo "JWT_SECRET=${JWT_SECRET}" >> .env
            echo "EXTERNAL_BANK_API_URL=${EXTERNAL_BANK_A
PI_URL}" >> .env
            echo "EXTERNAL_BANK_API_KEY=${EXTERNAL_BANK_AP
I_KEY}" >> .env
            echo "EXTERNAL_BANK_API_ACCOUNTTYPE=${EXTERN
AL_BANK_API_ACCOUNTTYPE}" >> .env
            echo "EXTERNAL_USER_API_URL=${EXTERNAL_USER_API
_URL}" >> .env
            echo "JWT_ACCESS_TOKEN_VALIDITY=${JWT_ACCESS_T
OKEN_VALIDITY}" >> .env
            echo "JWT_REFRESH_TOKEN_VALIDITY=${JWT_REFRESH
_TOKEN_VALIDITY}" >> .env
            echo "SPRING_DATA_REDIS_HOST=${SPRING_DATA_REDI
S_HOST}" >> .env
            echo "SPRING_DATA_REDIS_PORT=${SPRING_DATA_REDI
S_PORT}" >> .env
            echo "SPRING_DATA_REDIS_PASSWORD=${SPRING_DATA
_REDIS_PASSWORD}" >> .env

            # 추가 환경 변수 설정
            echo "SPRING_PROFILES=development" >> .env
            echo "SPRING_DATASOURCE_URL=jdbc:mysql://${MY_D
B}:3306/${MYSQL_DATABASE}" >> .env
            echo "SPRING_DATASOURCE_USERNAME=${MYSQL_USE
R}" >> .env
            echo "SPRING_DATASOURCE_PASSWORD=${MYSQL_PAS

```

```

SWORD}" >> .env
    ""

    // 프론트엔드용 .env 파일 생성
    sh "mkdir -p client"
    sh "echo \"NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}\" > client/.env"

    // Docker 빌드 캐시 최적화를 위한 병렬 빌드
    sh "DOCKER_BUILDKIT=1 COMPOSE_DOCKER_CLI_BUILD=1
docker compose -f ${env.DEPLOY_COMPOSE_FILE} build --parallel"
    sh "docker compose -f ${env.DEPLOY_COMPOSE_FILE} down || true"
    sh "docker compose -f ${env.DEPLOY_COMPOSE_FILE} up -d"
}
}
}
}

stage('Health Check') {
    steps {
        script {
            echo "대상 서버(${env.TARGET_SERVER}) 헬스체크 시작... 포트:
${env.TARGET_BACKEND_PORT}"

            // 폴링 방식으로 서버 준비 상태 확인
            def maxAttempts = 10
            def waitInterval = 10
            def isHealthy = false

            for (int i = 0; i < maxAttempts && !isHealthy; i++) {
                if (i > 0) sh "sleep ${waitInterval}"

                // 로그 확인은 첫 시도와 마지막 시도에만 수행
                if (i == 0 || i == maxAttempts - 1) {
                    sh "docker logs server-${env.TARGET_SERVER} --tail 50 ||
echo '로그를 확인할 수 없습니다'"

```

```

    }

    echo "헬스체크 시도 중... (${i+1}/${maxAttempts})"
    isHealthy = checkServerHealth(env.TARGET_BACKEND_PORT)

T)    if (isHealthy) break
    }

    if (!isHealthy) {
        error "서버 시작 실패: ${env.TARGET_SERVER}"
    }

    echo "대상 서버(${env.TARGET_SERVER}) 정상 작동 확인!"
}
}
}

stage('Switch Traffic') {
    steps {
        script {
            // nginx 설정 파일 경로 명시적 정의
            def nginxConfPath = "${PROJECTS_DIR}/${PROJECT_NAME}/
nginx/conf.d/bluegreen.conf"
            def nginxTempFile = "${PROJECTS_DIR}/${PROJECT_NAME}/n
ginx/conf.d/bluegreen_temp_${BUILD_NUMBER}.conf"

            echo "원본 설정 파일 경로: ${nginxConfPath}"
            echo "임시 파일 경로: ${nginxTempFile}"

            // 현재 설정 파일 복사
            sh "cp ${nginxConfPath} ${nginxTempFile}"
            sh "ls -la ${nginxTempFile}"

            // 설정 파일 수정 (최적화: 한 번의 sed 명령으로 변경)
            if (env.TARGET_SERVER == "blue") {
                // Blue 서버로 트래픽 전환
                sh ""
                sed -i -e 's/# server localhost:3001;/server localhost:3001;/g'
            }
        }
    }
}

```



```

\\
    -e 's/server localhost:3002;/# server localhost:3002;/g'
\\
    -e 's/# server localhost:8081;/server localhost:8081;/g' \\
    -e 's/server localhost:8082;/# server localhost:8082;/g'
${nginxTempFile}
    ""
} else {
    // Green 서버로 트래픽 전환
    sh ""
    sed -i -e 's/server localhost:3001;/# server localhost:3001;/g'
\\
    -e 's/# server localhost:3002;/server localhost:3002;/g'
\\
    -e 's/server localhost:8081;/# server localhost:8081;/g' \\
    -e 's/# server localhost:8082;/server localhost:8082;/g'
${nginxTempFile}
    ""
}

// 설정 파일 적용
sh "cp ${nginxTempFile} ${nginxConfPath}"

// SSH를 사용하여 호스트에 명령 실행 (병렬 처리)
sshagent(['ssh-ec2-ubuntu']) {
    // 임시 파일 복사 및 적용을 한 번에 처리
    def sshResult = sh(script: ""
        scp -o StrictHostKeyChecking=no ${nginxTempFile} ubuntu@j12b110.p.ssafy.io:/tmp/bluegreen.conf &&
        ssh -o StrictHostKeyChecking=no ubuntu@j12b110.p.ssafy.io '
            sudo cp /tmp/bluegreen.conf /etc/nginx/conf.d/bluegreen.conf &&
            sudo nginx -t &&
            sudo systemctl reload nginx &&
            rm -f /tmp/bluegreen.conf
        '
    "", returnStatus: true)

```

```

        if (sshResult == 0) {
            echo "트래픽이 ${env.TARGET_SERVER} 서버로 성공적으로 전
환되었습니다!"
            sh "rm -f ${nginxTempFile}"
        } else {
            echo "트래픽 전환 중 문제가 발생했습니다."
            error "Nginx 설정 적용 실패"
        }
    }
}
}
}

stage('Verify and Clean') {
    parallel {
        stage('Verify Deployment') {
            steps {
                script {
                    // 새로운 활성 서버 확인
                    def newActiveServer = getCurrentActiveServer()
                    echo "새 활성 서버 확인 결과: ${newActiveServer}, 기대값:
${env.TARGET_SERVER}"

                    if (newActiveServer != env.TARGET_SERVER) {
                        error "트래픽 전환이 제대로 이루어지지 않았습니다."
                    }

                    echo "무중단 배포 검증 완료! 현재 활성 서버: ${newActiveServ
er}"

                    sh "curl -s http://j12b110.p.ssafy.io:${env.TARGET_BACKEN
D_PORT}/api/test/health || echo '헬스 체크 접근 불가'"
                }
            }
        }
    }

    stage('Clean Old Deployment') {
        steps {



```

```

        script {
            echo "이전 서버(${env.ACTIVE_SERVER}) 정리 중..."
            // 프리닝 전 잠시 대기 (안정화 시간)
            sh "sleep 5"
            sh "docker ps | grep ${env.ACTIVE_SERVER} || echo '이전
서버 컨테이너를 찾을 수 없습니다'"

            // 불필요한 이미지와 볼륨만 정리 (컨테이너는 유지)
            sh "docker system prune -f --volumes"

            echo "무중단 배포 완료! 이전 서버는 유지 상태입니다."
        }
    }
}

post {
    success {
        script {
            // 성공 알림 및 정리 작업
            def kTime = getKoreanTime()
            echo ""
             =====
=====
            🎉 무중단 배포 성공! 🎉
            📋 작업명: ${env.JOB_NAME}
            📋 빌드 번호: #${env.BUILD_NUMBER}
            🕒 완료 시간: ${kTime}
            💻 활성 서버: ${env.TARGET_SERVER}
            🌐 접속 URL: https://j12b110.p.ssafy.io
            =====
            == 
            ""
            // 성공 시에도 Mattermost에 알림 추가
            dir("${PROJECTS_DIR}/${PROJECT_NAME}") {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnSt

```

```

dout: true).trim()
        def Author_Email = sh(script: "git show -s --pretty=%ae", return
nStdout: true).trim()
        def Commit_Hash = sh(script: "git show -s --pretty=%h", return
Stdout: true).trim()
        def Commit_Msg = sh(script: "git show -s --pretty=%s", return
Stdout: true).trim()

        mattermostSend(
            color: 'good',
            message: """" *무중단 배포 성공*


- 작업명*: ${env.JOB_NAME}
- 빌드*: #${env.BUILD_NUMBER}
- 브랜치*: ${BRANCH_NAME}
- 커밋*: ${Commit_Hash} - ${Commit_Msg}
- 작성자*: ${Author_ID} <${Author_Email}>
- 완료 시간*: ${kTime}
- 활성 서버*: ${env.TARGET_SERVER}""",
            endpoint: 'https://meeting.ssafy.com/hooks/1m96q98oybfi9k
67nkjipxjddy',
            channel: 'B110-Jenkins'
        )
    }
}
}

failure {
    script {
        // 실패 알림
        def kTime = getKoreanTime()
        echo """"
         =====
        =====
         무중단 배포 실패! 
         작업명: ${env.JOB_NAME}
         빌드 번호: #${env.BUILD_NUMBER}
         실패 시간: ${kTime}
        =====
    }
}

```

== ❌

"""

// 실패 시 Mattermost에 알림

dir("\${PROJECTS\_DIR}/\${PROJECT\_NAME}") {

def Author\_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()

def Author\_Email = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()

def Commit\_Hash = sh(script: "git show -s --pretty=%h", returnStdout: true).trim()

def Commit\_Msg = sh(script: "git show -s --pretty=%s", returnStdout: true).trim()

mattermostSend(

color: 'danger',

message: """❌ \*무중단 배포 실패\*

• \*작업명\*: \${env.JOB\_NAME}

• \*빌드\*: #\${env.BUILD\_NUMBER}

• \*브랜치\*: \${env.BRANCH\_NAME}

• \*커밋\*: \${Commit\_Hash} - \${Commit\_Msg}

• \*작성자\*: \${Author\_ID} <\${Author\_Email}>

• \*실패 시간\*: \${kTime}

• \*로그 확인\*: \${env.BUILD\_URL}console""",

endpoint: 'https://meeting.ssafy.com/hooks/1m96q98oybfi9k67nkjipxjddy',

channel: 'B110-Jenkins'

)

}

}

}

always {

echo """



=====

==

작업 공간 정리 중...



임시 파일 및 아티팩트 제거

= 

=====

"""

cleanWs()

}

}

}