

# BE/FE Jenkins 스크립트 및 Docker 설정 파일

BE/FE Jenkins 스크립트 및 Docker 설정 파일

## Jenkins pipeline 정리

**dev-be** : 백엔드 기능 통합 개발용 브랜치 (구축 완료)

**dev-fe** : 프론트엔드 기능 통합 개발용 브랜치 (구축 완료)

**devlop** : dev-be와 dev-fe를 병합하여 최종 점검하는 브랜치

**master** : 실제 운영 브랜치

\*백엔드 개발 브랜치로 변경 및 merge request 명령을 받으면 자동빌드 되도록 해야됨. ( 현재는 테스트 ) 'feature/infra' → 'dev-be'

## 9. BackEnd 배포 ( dev-fe )

아래 Jenkins plugin 설치하기

Generic Webhook Trigger Plugin

GitLab API Plugin

GitLab Plugin

Stage View → 배포 과정 쉽게 볼 수 있음

### 9.1 infra/docker-compose.yml 작성

```
services:
  backend:
    build:
      context: ../backend
      dockerfile: Dockerfile
    image: ptsd-app
    container_name: ptsd-app
    ports:
```

```

- "8081:8000"
volumes:
- jenkins-data:/var/jenkins_home
- /var/run/docker.sock:/var/run/docker.sock
environment:
- ENV=production
restart: always
networks:
- app-network

mosquitto:
image: eclipse-mosquitto:2
container_name: mosquitto
ports:
- "1883:1883"
- "9001:9001"
volumes:
- /home/ubuntu/S12P31D101/backend/PTSD/core/mosquitto/mosquitto.conf:/mosquitto/mosquitto.conf
- mosquitto-data:/mosquitto/data
- mosquitto-log:/mosquitto/log
command: mosquitto -c /mosquitto/mosquitto.conf
restart: always
networks:
- app-network

networks:
app-network:
external: true

volumes:
jenkins-data:
mosquitto-data:
mosquitto-log:

```

## 9.2. DockerFile 작성

```
# backend/Dockerfile
```

```
# 1. Python 이미지 기반
FROM python:3.10-slim
```

```
# 2. 작업 디렉토리
WORKDIR /app
```

```
# 3. 의존성 파일 복사 및 설치
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
# 4. 전체 소스 복사
COPY . .
```

```
# 5. 환경 변수 로딩을 위한 uvicorn 실행 (FastAPI 진입점: PTSD.main:app)
CMD ["uvicorn", "PTSD.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

### 9.3. Jenkins pipeline을 이용한 자동배포 스크립트

```
pipeline {
    agent any
    environment {
        DEPLOY_ENV = credentials('DEPLOY_ENV') // .env.prod 환경변수 파일 (FastAPI)
    }

    stages {
        stage('Check Branch') {
            steps {
                script {
                    def targetBranch = env.gitlabTargetBranch ?: env.BRANCH_NAME
                    if (targetBranch != null && targetBranch != 'dev-be') {
                        currentBuild.result = 'ABORTED'
                        error("This pipeline only runs for merge requests to dev-be branch")
                    }
                }
            }
        }
    }
}
```

```

stage('Clean Workspace') {
    steps {
        echo '🧹 Cleaning workspace before checkout'
        deleteDir()
    }
}

stage('CheckOut') {
    steps {
        echo '📦 Cloning PTSD project...'
        git branch: 'dev-be',
            credentialsId: 'account',
            url: 'https://lab.ssafy.com/s12-final/S12P31D101.git'
        sh 'ls -R'
        echo '✅ CheckOut finished!'
    }
}

stage('Build') {
    steps {
        echo '🏗️ Start building PTSD project...'
        dir('backend') {
            withCredentials([file(credentialsId: 'DEPLOY_ENV', variable: 'ENV_
                sh '''
                    cp "$ENV_FILE" .env.prod
                    cat .env.prod
                ''')
        }
        echo '✅ Build stage finished!'
    }
}

stage('Deploy') {
    steps {
        script {
            dir('infra') {

```

```

        withCredentials([file(credentialsId: 'DEPLOY_ENV', variable: 'ENV_FILE')]) {
            sh '''
                cp "$ENV_FILE" "$WORKSPACE/backend/.env"
                chmod 600 "$WORKSPACE/backend/.env"
            '''
        }
        sh 'docker-compose down || true'
        sh 'docker-compose build --no-cache'
        sh 'docker-compose up -d'
        sh 'sleep 20'
        echo '🚀 Deploy finished!'
    }
}
}
}
}

post {
    success {
        echo '✅ Pipeline succeeded!'
    }

    failure {
        echo '❌ Pipeline failed! Logs below:'
        dir('infra') {
            withCredentials([file(credentialsId: 'DEPLOY_ENV', variable: 'ENV_FILE')]) {
                sh '''
                    cp "$ENV_FILE" "$WORKSPACE/backend/.env"
                    chmod 600 "$WORKSPACE/backend/.env"
                '''
            }
            sh 'docker-compose down || true'
            sh 'docker-compose build --no-cache'
            sh 'docker-compose up -d'
        }
    }
}

always {

```

```

        echo '🧹 Cleaning up workspace after pipeline'
        deleteDir()
    }
}
}

```

## 10. FrontEnd 배포 ( dev-fe )

아래 Jenkins plugin 설치하기

NodeJs Plugin

### 10.1. frontend/docker-compose.yml 작성

```

services:
  react:
    image: s12p31d101-react
    build:
      context: .
      dockerfile: Dockerfile
    container_name: react
    ports:
      - "3000:80"
    networks:
      - app-network
    restart: always

networks:
  app-network:
    external: true

```

### 10.2. DockerFile 작성

```

# 빌드 단계
FROM node:22.13.0-alpine as builder

# 작업 디렉토리 설정

```

```
WORKDIR /app
```

```
# 의존성 설치
```

```
COPY package*.json ./
```

```
RUN npm install
```

```
# 앱 소스 복사 및 빌드
```

```
COPY . .
```

```
RUN npm run build
```

```
# 프로덕션 단계
```

```
FROM nginx:alpine
```

```
# 빌드 결과물 복사
```

```
# COPY --from=builder /app/build /usr/share/nginx/html
```

```
COPY --from=builder /app/dist /usr/share/nginx/html
```

```
# 포트 설정
```

```
EXPOSE 80
```

```
# 실행 명령어
```

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

```
# 왜 안바뀌니?
```

### 10.3. Nginx.conf 작성

```
# frontend/mafia/nginx.conf
```

```
server {
```

```
    listen 80;
```

```
    location / {
```

```
        root /usr/share/nginx/html;
```

```
        try_files $uri $uri/ /index.html;
```

```
    }
```

```
}
```

### 10.4. Jenkins pipeline를 이용하여배포

```

pipeline {
  agent any

  stages {
    stage('FE-Check Branch') {
      steps {
        script {
          // def targetBranch = 'dev-fe'
          def targetBranch = env.gitlabTargetBranch ?: env.BRANCH_NAME
          if (targetBranch != null && targetBranch != 'dev-fe') {
            currentBuild.result = 'ABORTED'
            error("This pipeline only runs for merge requests to dev-fe bran
          }
        }
      }
    }
  }

  stage('FE-CheckOut') {
    steps {
      echo 'Start CheckOut PTSD project...'
      git branch: 'dev-fe',
        credentialsId: 'account',
        url: 'https://lab.ssafy.com/s12-final/S12P31D101.git'
      echo 'CheckOut finished!'
    }
  }

  stage('FE-Build') {
    steps {
      echo 'Start building PTSD project...'
      nodejs(nodeJSInstallationName: 'NodeJS 22.13.0') {
        dir('frontend') {
          sh '''
            npm install
            CI=false npm run build
          '''
        }
      }
    }
  }
}

```



```

    }
    echo 'Build finished!'
  }
}

stage('FE-Deploy') {
  steps {
    script {
      dir('frontend') {
        sh '''
          docker-compose down || true
          docker-compose build --no-cache
          docker-compose up -d
        '''

        sh """
          echo "Reloading Nginx configuration..."
          docker exec nginx nginx -s reload
          echo "Nginx reloaded successfully"
        """

        // 컨테이너 상태 확인
        sh '''
          echo "Waiting for containers to start..."
          sleep 30
          docker ps
        '''
      }
    }
  }
}

post {
  success {
    echo 'Frontend pipeline successful !!'
  }
  failure {

```

```

    echo 'Frontend pipeline failed !!'
    dir('frontend') {
        sh 'docker compose logs'
    }
}
}
}
}

```

## ▼ ⚠ React 배포 에러

### ✅ 문제 요약

The screenshot shows a Jenkins pipeline execution. The pipeline consists of several steps: Start, FE-Check Branch, FE-CheckOut, FE-Build, FE-Deploy, Post Actions, and End. The FE-Build step is marked with a red 'X', indicating a failure. Below the pipeline view, the console output for the FE-Build step is shown, detailing the error.

```

0 Found unhandled java.io.IOException exception:
1 No installation NodeJS 22.13.0 found, Please define one in manager Jenkins.
2 PluginClassLoader for nodejs//jenkins.plugins.nodejs.NodeJSBuildWrapper.setup(NodeJSBuildWrapper.java:158)
3 PluginClassLoader for workflow-basic-steps//org.jenkinsci.plugins.workflow.steps.CoreWrapperStepExecution2.doStart(CoreWrapperStep.java:121)
4 PluginClassLoader for workflow-step-api//org.jenkinsci.plugins.workflow.steps.GeneralNonBlockingStepExecution.Lambda$run$0(GeneralNonBlockingStepExecution.java:77)
5 java.base/java.util.concurrent.Executors$RunnableAdapter.call(Unknown Source)
6 java.base/java.util.concurrent.FutureTask.run(Unknown Source)
7 java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)
8 java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
9 java.base/java.lang.Thread.run(Unknown Source)

```

## 1. Jenkins 빌드 실패

nodejs(nodeJSInstallationName: 'NodeJS 22.13.0')

- Jenkins에 등록되지 않은 NodeJS 이름 사용
- "NodeJS 22.13.0" 이라는 이름의 Node.js 설치가 없음

### 🔧 해결 방법

#### 🔧 Jenkins에 Node.js 버전 등록

## 1. Jenkins > Jenkins 관리

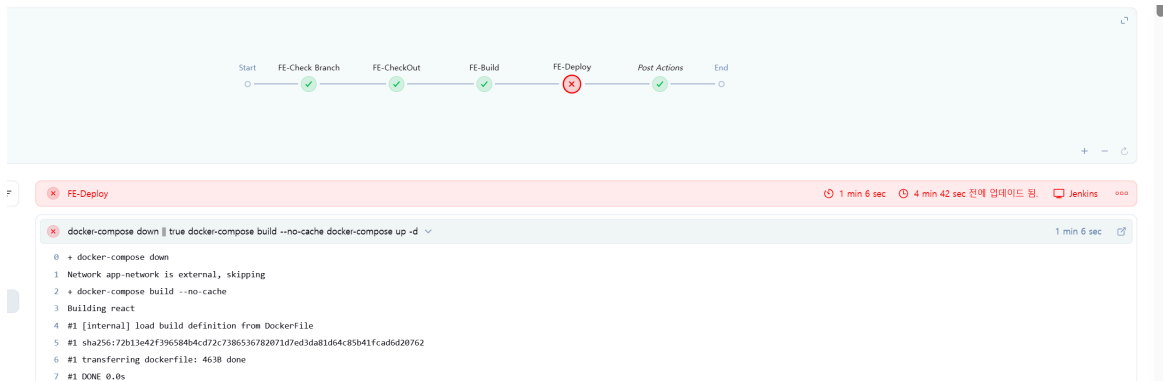
## 2. Global Tool Configuration

### 3. NodeJS 섹션에서

- [NodeJS 추가] 클릭
- 이름: NodeJS 22.13.0
- ☒ 자동 설치 체크
- 버전: 22.x 중 가능한 최신 선택

### 4. 저장 후 다시 빌드

 Pipeline과 동일한 이름 필수 → NodeJS 22.13.0




The image shows a Jenkins Pipeline Run View. The pipeline consists of several stages: Start, FE-Check Branch, FE-CheckOut, FE-Build, FE-Deploy, Post Actions, and End. The FE-Deploy stage is highlighted in red, indicating a failure. Below the pipeline view, the console output for the FE-Deploy stage is shown, detailing the execution of docker-compose commands and the resulting error.

```
docker-compose down || true; docker-compose build --no-cache; docker-compose up -d
```

```
0 + docker-compose down
1 Network app-network is external, skipping
2 + docker-compose build --no-cache
3 Building react
4 #1 [internal] load build definition from Dockerfile
5 #1 sha256:72b13e42f396584b4cd72c7386536782071d7ed3da81d64c85b41fca6d20762
6 #1 transferring dockerfile: 463B done
7 #1 DONE 0.0s
```

## 두 번째 에러

 **docker compose** 명령 인식 실패

```
docker compose ...
exit code 125
```

## 원인

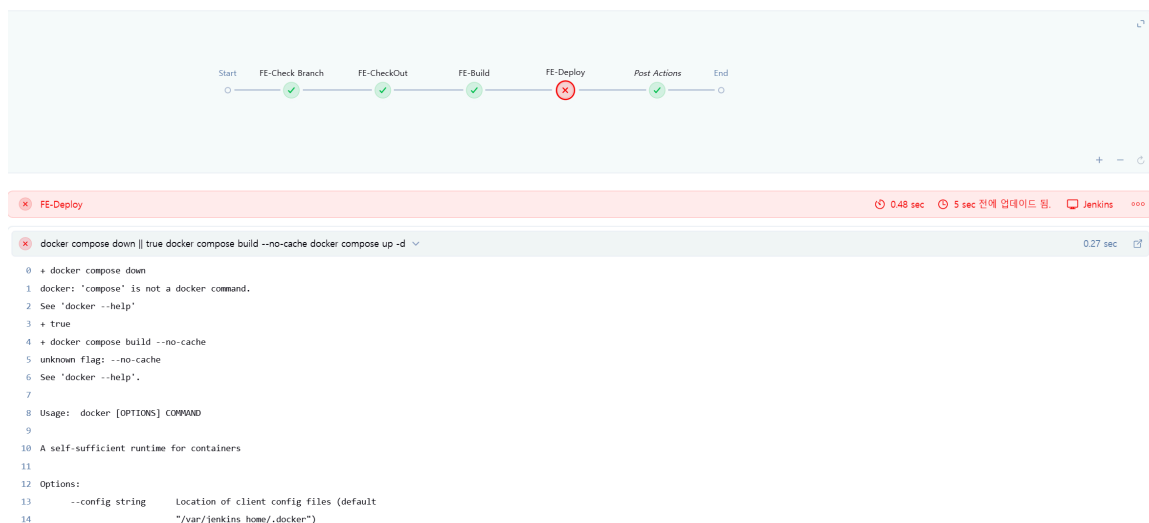
- Jenkins 환경의 Docker 버전이 낮아 **docker compose** 명령어 인식 불가

## 해결 방법

**docker-compose** (하이픈 포함된 명령어)로 변경

```
docker-compose down || true
docker-compose build --no-cache
docker-compose up -d
```

## ✓ 세 번째 에러



The image shows a Jenkins pipeline console output. The pipeline consists of several stages: Start, FE-Check Branch, FE-CheckOut, FE-Build, FE-Deploy, Post Actions, and End. The FE-Deploy stage is marked with a red 'X' and an error message. The error message is as follows:

```
0 + docker: compose down
1 docker: 'compose' is not a docker command.
2 See 'docker --help'
3 + true
4 + docker compose build --no-cache
5 unknown flag: --no-cache
6 See 'docker --help'.
7
8 Usage: docker [OPTIONS] COMMAND
9
10 A self-sufficient runtime for containers
11
12 Options:
13   --config string      Location of client config files (default
14                        "/var/jenkins_home/.docker")
```

## 📦 빌드 파일 경로 에러

```
COPY --from=builder /app/build /usr/share/nginx/html
ERROR: ... "/app/build": not found
```

## 🎯 원인

- 최신 Vite 기반 프로젝트는 `build` 폴더 대신 `dist` 폴더 생성

## ✓ 해결 방법

`Dockerfile` 수정:

```
# 수정 전
```

```
COPY --from=builder /app/build /usr/share/nginx/html
```

# 수정 후

```
COPY --from=builder /app/dist /usr/share/nginx/html
```

## ✅ nginx.conf 설정

현재 nginx는 다음과 같이 구성됨:

🔄 FastAPI는 **/service/** 로 프록시 전달

```
location /service/ {  
    proxy_pass http://ptsd-app;  
}
```

🔄 React는 **/** 로 프록시 전달

```
location / {  
    proxy_pass http://react;  
}
```

→ 따라서 프론트엔드에서 API 요청을 보낼 때 반드시 **/service** prefix를 포함해야 함

## ❌ Nginx 빌드 실패 정리

### 💣 문제 요약

#### 📌 증상

- Jenkins에서 Nginx가 빌드 실패
- `nginx: [emerg] host not found in upstream "jenkins"` 에러 발생

## 에러 로그

```
bash
복사편집
nginx: [emerg] host not found in upstream "jenkins" in /etc/nginx/nginx.conf
```

## 원인

### ! 컨테이너 이름 불일치

nginx.conf 의 설정:

```
proxy_pass http://jenkins:8080/jenkins;
```

▼ 그런데 실제 Docker 컨테이너 이름은:

```
jenkins-v2501
```

즉, Nginx 설정 파일이 존재하지 않는 호스트(jenkins)로 요청을 보내고 있었음 → 빌드 실패

## 해결 방법

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS	PORTS
file3a42dbac0	s12p31d101-react	"/docker-entrypoint..."	react	19 minutes ago	Up 19 minutes	0.0.0.0:3000->80/tcp, [::]:3000->80/tcp
30e7707f1a55	ptsd-app	"uvicorn PTSD.main:a..."	ptsd-app	22 hours ago	Up 22 hours	0.0.0.0:8081->8080/tcp, [::]:8081->8080/tcp
8bac4a3fd569	eclipse-mosquitto:2	"/docker-entrypoint..."	mosquitto	22 hours ago	Up 22 hours	0.0.0.0:1883->1883/tcp, [::]:1883->1883/tcp
0.0.0.0:9001->9001/tcp, [::]:9001->9001/tcp	nginx:latest	"/docker-entrypoint..."	nginx	22 hours ago	Up About an hour	0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp
fe11f9a76c30	jenkins/jenkins:2.501	"/usr/bin/tini -- /u..."	jenkins-v2501	5 days ago	Up 5 days	0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp
b3504632205f	postgres:15	"docker-entrypoint.s..."	ptsd	12 days ago	Up 12 days (healthy)	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp
50000/tcp	redis:7	"docker-entrypoint.s..."	redis	12 days ago	Up 12 days	0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp
ded084ae3684						
2166e91afb5c						

## nginx.conf 수정

컨테이너 이름을 정확히 일치시켜야 함:

## ✓ 수정 전

```
proxy_pass http://jenkins:8080/jenkins;
```

## ✓ 수정 후

```
proxy_pass http://jenkins-v2501:8080/jenkins;
```

| 💡 docker ps 명령어로 정확한 컨테이너 이름을 확인한 뒤 설정

## 💡 팁: 도커 컨테이너 이름 확인

```
docker ps
```

출력 예시:

```
CONTAINER ID  IMAGE                ... NAMES
...          jenkins/jenkins    ... jenkins-v2501 ✓
```