

# CI/CD 구축

## 빌드 배포 메뉴얼

### 1. 사전 준비 사항

#### 1.1. EC2 서버 발급 및 초기 설정

```
sudo apt update # 최신 패키지 정보 업데이트: Ubuntu 서버의 패키지 목록을 최신화  
sudo apt upgrade # 보안 패치 적용: 기존에 설치된 패키지들을 최신 버전으로 업그레이드
```

#### 1.2. pem키 설정 및 서버 접속 → 외부 로컬 PC 설정할 때 활용 (필수 아님)

##### 1. .pem 파일 로컬에 저장

```
#PowerShell  
wsl --install -d Ubuntu
```

##### 2. ubuntu 계정에 대한 비밀번호 설정

```
sudo passwd  
su root
```

##### 3. 서버 접속

```
#폴더 생성  
mkdir .ssh  
  
# 자신 경로에 pem키 있는지 확인  
cd /mnt/c/Users/Bae/Desktop/ssafy/pem
```

```
#해당 경로로 복사
cp /mnt/c/Users/Bae/Desktop/ssafy/pem/K12D101T.pem ~/.ssh/

#권한 설정(읽기)
chmod 400 ~/.ssh/K12D101T.pem

#서버 접속
ssh -i ~/.ssh/K12D101T.pem ubuntu@k12d101.p.ssafy.io
```

#### 4. EC2 접속 화면

```
rokmc3038@DESKTOP-PHKCE21:~$ ssh -i ~/.ssh/K12D101T.pem ubuntu@k12d101.p.ssafy.io
The authenticity of host 'k12d101.p.ssafy.io (3.35.216.68)' can't be established.
ED25519 key fingerprint is SHA256:8+LH7uNru0JSaIpyhFGiR9GU8Y9+pXZ9SbEwVVdE2n4.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'k12d101.p.ssafy.io' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.8.0-1024-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Mon Apr 28 04:13:09 UTC 2025

System load:  0.08               Processes:    131
Usage of /:   1.6% of 309.95GB   Users logged in: 0
Memory usage: 4%                IPv4 address for eth0: 172.26.2.57
Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.
```

## 2. 리눅스 방화벽 설정

### 2.1. 예시

```
# 필수 포트 허용
sudo ufw allow 22    # SSH
sudo ufw allow 80    # HTTP
sudo ufw allow 443   # HTTPS
sudo ufw allow 3000  # WEB (Frontend)
sudo ufw allow 8081  # FastAPI was (Backend)
sudo ufw allow 8080  # Jenkins
```

### 2.2. 필수 포트 허용하기

## ▼ ufw 적용 순서 방법

제공되는 EC2의 ufw(우분투 방화벽)는 기본적으로 활성화(Enable) 되어 있고, ssh 22번 포트만 접속 가능하게 되어 있습니다.

포트를 추가할 경우 6번부터 참고하시고,  
처음부터 새로 세팅해 보실 경우에는 1번부터 참고하시기 바랍니다.

1. 처음 ufw 설정 시 실수로 ssh접속이 안되는 경우를 방지하기 위해  
ssh 터미널을 여유있게 2~3개 연결해 놓는다.

2. ufw 상태 확인

```
$ sudo ufw status
```

```
Status : inactive
```

3. 사용할 포트 허용하기 (ufw inactive 상태)

```
$ sudo ufw allow 22
```

3-1 등록된 포트 조회하기 (ufw inactive 상태)

```
$ sudo ufw show added
```

```
Added user rules (see 'ufw status' for running firewall):
```

```
ufw allow 22
```

4. ufw 활성화 하기

```
$ sudo ufw enable
```

```
Command may disrupt existing ssh connections. Proceed with operation (Y) or
```

4.1 ufw 상태 및 등록된 rule 확인하기

```
$ sudo ufw status numbered
```

```
Status: active
```

To	Action	From
--	-----	----
[ 1] 22	ALLOW IN	Anywhere
[ 2] 22 (v6)	ALLOW IN	Anywhere (v6)

5. 새로운 터미널을 띄워 ssh 접속해 본다.

```
C:\> ssh -i 팀.pem ubuntu@팀.p.ssafy.io
```

6. ufw 구동된 상태에서 80 포트 추가하기

```
$ sudo ufw allow 80
```

6-1. 80 포트 정상 등록되었는지 확인하기

```
$ sudo ufw status numbered
```

Status: active

To	Action	From
--	-----	----
[ 1] 22	ALLOW IN	Anywhere
[ 2] 80	ALLOW IN	Anywhere
[ 3] 22 (v6)	ALLOW IN	Anywhere (v6)
[ 4] 80 (v6)	ALLOW IN	Anywhere (v6)

6-2. allow 명령을 수행하면 자동으로 ufw에 반영되어 접속이 가능하다.

7. 등록한 80 포트 삭제 하기

```
$ sudo ufw status numbered
```

Status: active

To	Action	From
--	-----	----
[ 1] 22	ALLOW IN	Anywhere
[ 2] 80	ALLOW IN	Anywhere
[ 3] 22 (v6)	ALLOW IN	Anywhere (v6)
[ 4] 80 (v6)	ALLOW IN	Anywhere (v6)

7-1. 삭제할 80 포트의 [번호]를 지정하여 삭제하기

번호 하나씩 지정하여 삭제한다.

```
$ sudo ufw delete 4
```

```
$ sudo ufw delete 2
```

```
$ sudo ufw status numbered (제대로 삭제했는지 조회해보기)
```

Status: active

To	Action	From
--	-----	----

```
[ 1] 22                ALLOW IN  Anywhere
[ 2] 22 (v6)           ALLOW IN  Anywhere (v6)
```

7-2 (중요) 삭제한 정책은 반드시 enable를 수행해야 적용된다.

```
$ sudo ufw enable
```

Command may disrupt existing ssh connections. Proceed with operation (Y) or: (n) abort: y

기타

- ufw 끄기

```
$ sudo ufw disable
```

## ufw 적용 후 ufw 상태

```
ubuntu@ip-172-26-2-57:~$ sudo ufw status numbered
Status: active

    To Action      From
    --
[ 1] 22      ALLOW IN  Anywhere
[ 2] 80      ALLOW IN  Anywhere
[ 3] 443     ALLOW IN  Anywhere
[ 4] 3000    ALLOW IN  Anywhere
[ 5] 8081    ALLOW IN  Anywhere
[ 6] 8080    ALLOW IN  Anywhere
[ 7] 22 (v6)  ALLOW IN  Anywhere (v6)
[ 8] 80 (v6)  ALLOW IN  Anywhere (v6)
[ 9] 443 (v6) ALLOW IN  Anywhere (v6)
[10] 3000 (v6) ALLOW IN  Anywhere (v6)
[11] 8081 (v6) ALLOW IN  Anywhere (v6)
[12] 8080 (v6) ALLOW IN  Anywhere (v6)
```

## 3. 도커 설치 및 설정

도커 공식 문서 참조 → <https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository>

### 3.1. Docker 레포지토리 설치

```
# 다음 명령을 실행하여 충돌하는 모든 패키지를 제거
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman
```

```
# 시스템의 패키지 목록을 최신화
```

```
sudo apt-get update
```

```
# SSL 인증서와 curl 도구 설치 (보안 통신과 파일 다운로드에 필요)
```

```
sudo apt-get install ca-certificates curl
```

```
# Docker의 GPG 키를 저장할 디렉토리 생성 (권한: 0755)
```

```
sudo install -m 0755 -d /etc/apt/keyrings
```

```
# Docker의 공식 GPG 키를 다운로드 (패키지 인증에 사용)
```

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/ke
```

```
# 다운로드한 GPG 키를 모든 사용자가 읽을 수 있도록 권한 설정
```

```
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

### 3.2. 레포지토리 추가

```
# Docker 공식 레포지토리를 시스템의 소프트웨어 소스에 추가
```

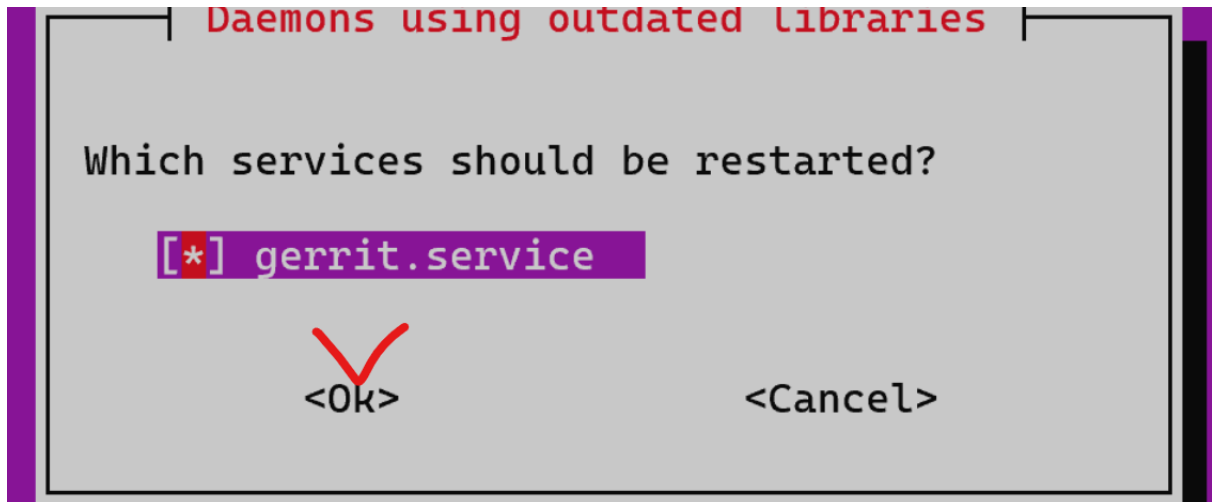
```
# - arch=$(dpkg --print-architecture): 시스템 아키텍처 확인 (예: amd64)
```

```
# - VERSION_CODENAME: Ubuntu 버전 코드네임 (예: focal)
```

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/do
```

### 3.3. Docker 패키지 설치

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plug
```



### 3.4. Docker 버전 확인

```
docker --version
Docker version 28.1.1, build 4eba377
```

### 3.5. Docker 설치 확인 및 권한 설정

```
# 현재 사용자를 docker 그룹에 추가
sudo usermod -aG docker $USER
#변경사항 적용
newgrp docker
#권한 확인
groups
`ubuntu adm dialout cdrom floppy sudo audio dip video plugdev netdev lxd dc
```

## 4. Jenkins 설치

### 4.1. 호스트 특정 디렉토리에 마운트

```
cd /home/ubuntu && mkdir jenkins-data
```

## 4.2. Jenkins Docker 컨테이너 기동

### 4.2.1 jenkins docker 할 때 --network app-network \ 네트워크 생성

```
docker run -d \  
-v /home/ubuntu/jenkins-data:/var/jenkins_home \  
-v /var/run/docker.sock:/var/run/docker.sock \  
-v /home/ubuntu/docker/proxy:/proxy \  
-p 8080:8080 \  
-e JENKINS_OPTS="--prefix=/jenkins" \  
--group-add $(getent group docker | cut -d: -f3) \  
-e TZ=Asia/Seoul \  
--restart=on-failure \  
--name jenkins \  
--network app-network \  
jenkins/jenkins:lts-jdk17
```

# jenkins/jenkins:2.501 → 젠킨스 공식 사이트에서 최신버전 명시

# -v 호스트의 /home/ubuntu/jenkins-data 디렉토리를 컨테이너의 /var/jenkins\_home로 마운트  
# -v 호스트의 Docker 소켓을 컨테이너에 마운트  
# -v 호스트의 해당 폴더로 마운트  
# -e Jenkins의 URL 접두사를 '/jenkins'로 설정  
# -e Docker 명령어를 젠킨스 내에서 실행할 권한을 부여  
# -- 실패했을 경우 재시작  
# -- jenkins로 이름 지정  
# --network app-network 컨테이너를 app-network에 연결하여 nginx 등 다른 서비스와 연결  
# -- JDK17버전을 명시적으로 지정

### 4.2.1. network app-network \ 네트워크 생성

```
docker network create app-network → 생성  
21bac02d0da13a711c43412be843c2c0b5622ef2cd44bc260701831785cd65b5
```



```
docker restart jenkins 확인
```

### 4.3. Jenkins 환경설정

```
cd /home/ubuntu/jenkins-data

mkdir update-center-rootCAs
#Jenkins가 업데이트 센터에 접속할 때 사용할 SSL 인증서를 제공
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-
#Jenkins가 기본 업데이트 센터 대신 Tencent 미러를 사용하도록 설정
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githu
#그 후 재시작
sudo docker restart jenkins
```

### 4.4. config 보안 설정 확인

```
vi config.xml
#true가 되어 있어야함
<useSecurity>true</useSecurity>
<securityRealm class="hudson.security.HudsonPrivateSecurityRealm">
<disableSignup>true</disableSignup>
```

### 4.5. Jenkins 초기 설정

앞서 4.2에서 Jenkins를 도커 컨테이너로 띄웠음.

1. <http://k12d101.p.ssafy.io:8080/jenkins/> 접속

# Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

## 비번 조회

```
docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

# Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

## Install suggested plugins

Install plugins the Jenkins community finds most useful.

## Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.492.3

## 2. Install suggested plugins : 초기 플러그인 모두 설치

## Getting Started

### Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ Timestampers	✓ Workspace Cleanup	✓ Ant	✓ Gradle
✓ Pipeline	✓ GitHub Branch Source	✓ Pipeline: GitHub Groovy Libraries	✓ Pipeline Graph View
✓ Git	✓ SSH Build Agents	✓ Matrix Authorization Strategy	✓ PAM Authentication
✓ LDAP	🔄 Email Extension	✓ Mailer	🔄 Dark Theme

```

** Gson API
** Git client
Git
** GitHub
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
** Pipeline Graph Analysis
** Metrics
Pipeline Graph View
Git
** EDDSA API
** Trilead API
SSH Build Agents
Matrix Authorization Strategy
PAM Authentication
LDAP
```

\*\* - required dependency

Jenkins 2.501

### 3. Getting Started - 계정 생성

계정명 : admin

암호 : S12P31D101PTSD

이름 : ptsd

이메일주소 :

user@example.com (실제 본인 이메일)

## Create First Admin User

계정명

admin

암호

....

암호 확인

....

이름

d101

이메일 주소

bjy556@gmail.com

#### 4. 초기 Jenkins URL은 주소 고정

- a. 현재는 http 이지만 추후 nginx로 https 리다이렉션할 예정.

## Instance Configuration

Jenkins URL:

http://j12d109.p.ssafy.io:8080/jenkins/

#### 5. Jenkins plugin 설치하기

Generic Webhook Trigger Plugin

GitLab API Plugin

GitLab Plugin

Stage View → 배포 과정 쉽게 볼 수 있음

#### 4.6. Jenkins 내 docker 명령어 실행

DooD ( Docker outside of Docker ) 방식 적용

- jenkins 컨테이너 내부에서 EC2 Docker 데몬에 접근하는 방식
- CI/CD 파이프라인 내에서 Docker 빌드 작업할 때 주로 사용됨.

### Jenkins 안에 Docker를 설치하기 위해서 Jenkins 컨테이너에 접속

```
docker exec -it -u root jenkins bash
```

### Jenkins 안에 Docker를 설치

```
# 필요한 패키지 설치
apt-get update
apt-get install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Docker의 공식 GPG 키 추가
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

# Docker repository 설정
echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg \
    $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

# 패키지 목록 업데이트
apt-get update

# Docker CLI만 설치
apt-get install -y docker-ce-cli
```

## 5. Jenkins, Gitlab 연동하기

## 5.1. Jenkins plugin 설치

Jenkins관리 → Plugins 클릭 ( stage view 추가 )

Install	Name ↓	Released
<input checked="" type="checkbox"/>	<b>GitLab</b> 1.9.7 <b>Build Triggers</b> This plugin allows <b>GitLab</b> to trigger Jenkins builds and display their results in the GitLab UI.	12 days ago
<input checked="" type="checkbox"/>	<b>Generic Webhook Trigger</b> 2.2.5 <b>notification</b> <b>github</b> <b>webhook</b> <b>Build Parameters</b> <b>gitlab</b> <b>Build Triggers</b> <b>bitbucket</b> <b>bitbucket-server</b> <b>jira</b> Can receive any HTTP request, extract any values from JSON or XML and trigger a job with those values available as variables. Works with GitHub, GitLab, Bitbucket, Jira and many more.	3 mo 14 days ago
<input checked="" type="checkbox"/>	<b>GitLab API</b> 5.6.0-97.v6603a_83f8690 <b>Library plugins (for use by other plugins)</b> This plugin provides <b>GitLab4J API</b> for other plugins.	5 mo 26 days ago

## 5.2. Gitlab 연결 Personal Access Token 발급하기

저장소 접근과 코드 체크아웃을 위해 사용됨.

GitLab과 Jenkins를 연결할 때는 GitLab 계정의 액세스 토큰을 사용해야 합니다. ( 프로젝트 토큰 X )

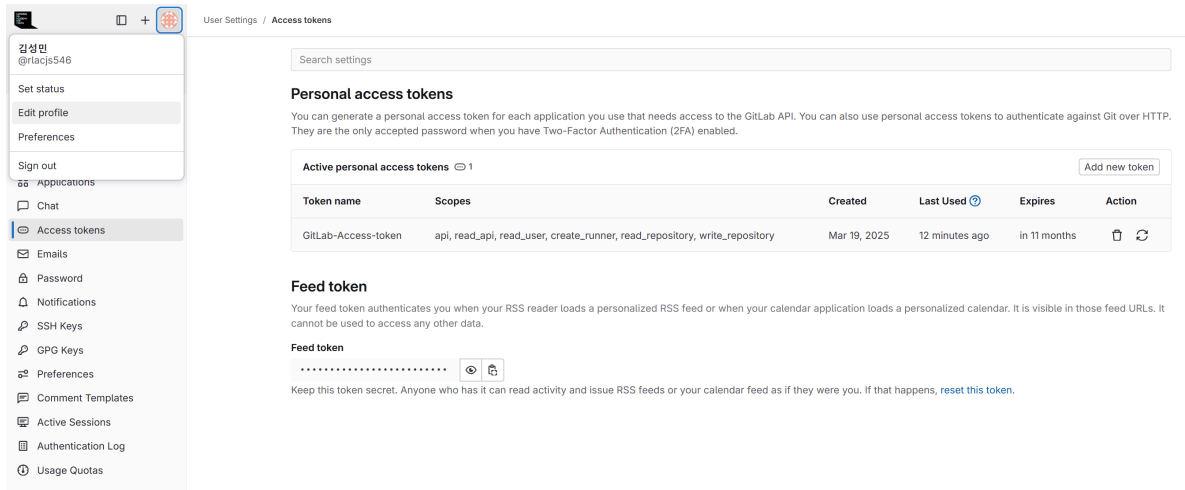
본인 계정 페이지 → Edit profile → Access tokens

- **Scopes (권한) 선택 → 전체 선택해도 무방함**
  - **api** : GitLab API를 사용하여 프로젝트 및 리포지토리에 접근 가능 (필수)
  - **read\_repository** : 저장소 읽기 권한 (Clone 가능)
  - **write\_repository** : Jenkins가 리포지토리에 Push 권한이 필요할 경우 선택

깃랩 Access Token 발급한 화면

▼ Access Token ( Gitlab )

txo\_ohx5VGDhCRmE-nJw



### 5.3. Gitlab 연결 Gitlab API Token 발급하기

GitLab API와의 통합 기능( webhook, mr 등 )을 위해 사용됨.

대상 프로젝트 페이지 → Settings → Access Tokens

토큰 생성 시 권한 설정 후 발급한다.

### 5.4. Jenkins Credential 등록하기

Jenkins 관리 → Credentials → System → Global credentials → Add Credentials

발급한 2개의 토큰을 각각 등록한다.

- **Personal Access Token (개인 액세스 토큰)**
  - Jenkins에서 등록 시 타입: "Username with password"
  - Username: GitLab 사용자 이름
  - Password: 발급받은 Personal Access Token
  - ID: 'account' (또는 원하는 식별자)

→ 주로 저장소 접근 및 코드 체크아웃에 사용

- **GitLab API Token (GitLab API 토큰)**
  - Jenkins에서 등록 시 타입: "GitLab API token"



- API token: 발급받은 Project Access Token
- ID: 'GitLab-API-Token' (또는 원하는 식별자)

→ 주로 웹훅, Merge Request 처리 등 GitLab 플러그인의 기능에 사용

s12-final **s12P31D101** Access tokens

Your new project access token has been created.

Search page

### Project access tokens

Generate project access tokens scoped to this project for your applications that need access to the GitLab API. You can also use project access tokens with Git to authenticate over HTTP(S). [Learn more.](#)

Your new project access token

L7xEP547VnZ5-4sxy4Sp

Make sure you save it - you won't be able to access it again.

Active project access tokens 1 Add new token

Token name	Scopes	Created	Last Used	Expires	Role	Action
GitLab API Token	api, read_api, create_runner, read_repository, write_repository	Apr 28, 2025	Never	in 4 weeks	Guest	

Jenkins

Dashboard > Jenkins 관리 > Credentials

### Credentials

T	P	Store	Domain	ID	Name
		System	(global)	account	rlacjs546/***** (personal access token)
		System	(global)	GitLab-API-Token	GitLab API token (GitLab-API-Token (personal access token))
		System	(global)	application-secret	application-secret.yml (Spring Boot application-secret config)
		System	(global)	env-credentials	.env (환경 변수 관리)

### Stores scoped to Jenkins

P	Store	Domains
	System	(global)









아이콘: S M L

로컬 관리할 자격증명은 Secret file로 만들어 따로 등록함.

## Global credentials (unrestricted)

[+ Add Credentials](#)










Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 <a href="#">account</a>	rlacjs546/***** (personal access token)	Username with password	personal access token 
 <a href="#">GitLab-API-Token</a>	GitLab API token (GitLab-API-Token (personal access token))	GitLab API token	GitLab-API-Token (personal access token) 
 <a href="#">application-secret</a>	application-secret.yml (Spring Boot application-secret config)	Secret file	Spring Boot application-secret config 
 <a href="#">env-credentials</a>	.env (환경 변수 관리)	Secret file	환경 변수 관리 

## 5.5. Pipeline 생성

jenkins : 새로운 item → pipeline

### Build Triggers

- ☐ Build after other projects are built 
- ☐ Build periodically 
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://jenkins:8080/github-webhook/> 
  - Enabled GitLab triggers
    - ☐ Push Events 
    - ☐ Push Events in case of branch delete 
    - ☐ Opened Merge Request Events 
    - ☐ Build only if new commits were pushed to Merge Request 
    - ☒ Accepted Merge Request Events 
    - ☐ Closed Merge Request Events 

Rebuild open Merge Requests ?

Never

☒ Approved Merge Requests (EE-only) ?

☒ Comments ?

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급 ▾

☐ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

- 고급 → 시크릿 토큰 생성 → Generate 버튼을 클릭 후
  - jenkins secret token 생성

GitLab : 해당 프로젝트 → setting → webhook

- URL과 jenkins Secret token 입력

## Webhook

[Webhooks](#) enable you to send notifications to web applications in response to events in a group or project. We recommend

### URL

URL must be percent-encoded if it contains one or more special characters.

☒ Show full URL

☐ Mask portions of URL

Do not show sensitive data such as tokens in the UI.

### Custom headers </> 0

No custom headers configured.

### Name (optional)

### Description (optional)

### Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

### Trigger

## 5.6. Jenkins-Gitlab 연결 확인

### 1. Jenkins에서 Gitlab연결 테스트

Jenkins관리 → System → GitLab 섹션

Dashboard > Jenkins 관리 > System >

GitLab

☒ Enable authentication for '/project' end-point ?

GitLab connections

Connection name ?  
A name for the connection

ptsd

GitLab host URL ?  
The complete URL to the GitLab server (e.g. http://gitlab.mydomain.com)

https://lab.ssafy.com/

Credentials ?  
API Token for accessing GitLab

GitLab API token

+ Add

고급 ▾

Test Connection

Save Apply

## 2. GitLab에서 웹훅 테스트

GitLab 웹훅 페이지 하단 "Test" 클릭 후 이벤트 유형 선택

성공 화면

Hook executed successfully: HTTP 200

Q Search page

### Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

Webhooks 1

Add new webhook

dev-be  
http://j12d109.p.ssafy.io:8080/jenkins/project/dev-be

dev-be Test Edit Delete

Merge request events Push events SSL Verification: enabled

## EC2 내부 DB 관련 파일 경로

```
ubuntu@ip-172-26-2-54:~/docker$ ls
db proxy
ubuntu@ip-172-26-2-54:~/docker$ cd db
ubuntu@ip-172-26-2-54:~/docker/db$ ls
docker-compose.yml mysql
```

## EC2 내부 Nginx 관련 파일 경로

```
ubuntu@ip-172-26-2-54:~/docker$ ls
db proxy
ubuntu@ip-172-26-2-54:~/docker$ cd proxy
ubuntu@ip-172-26-2-54:~/docker/proxy$ ls
conf.d data docker-compose.yml nginx.conf
ubuntu@ip-172-26-2-54:~/docker/proxy$
```

```
sudo mkdir -p ~/docker/db
sudo chown -R ubuntu:ubuntu ~/docker
cd ~/docker
ls
cd db
nano docker-compose.yml → 파일 생성
```

## 6. PostgreSQL, Redis 컨테이너 기동

### 6.1. docker-compose.yml

```
# EC2 내부 경로 : /home/ubuntu/docker/db/docker-compose.yml

services:
  postgres:
    image: postgres:15
    container_name: postgres
    restart: always
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER}"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - app-network

  redis:
    image: redis:7
```

```

container_name: redis
ports:
  - "${REDIS_PORT}:6379"
volumes:
  - redis_data:/data
command: redis-server --requirepass '${REDIS_PASSWORD}' --appendonly
networks:
  - app-network

volumes:
  postgres_data:
  redis_data:

networks:
  app-network:
    external: true

```

## 6.2 docker 명령어 실행

```

#해당 폴더 위치로 이동
cd home/ubuntu/docker/db/docker-compose.yml

# 현재 디렉토리의 docker-compose.yml 파일 기반으로 서비스 시작
docker compose up -d

# 특정 docker-compose.yml 파일 지정하여 서비스 시작
docker compose -f docker-compose.yml up -d

# 현재 디렉토리의 docker-compose.yml 파일 기반으로 서비스 중지 및 삭제
docker compose down

```

## 6.3 DB init 설정하기

```

# PostgreSQL 컨테이너 접속
docker exec -it ptsd bash

```

```

# PostgreSQL에 root 유저로 접속 (Postgres는 보통 'postgres' 사용자로 접속해)
psql -U ptsd

# 데이터베이스 사용
\c ptsd

# 권한 부여
GRANT ALL PRIVILEGES ON DATABASE ptsd TO ptsd;

# 확인하기
\du
\l

# 나가기
\q

컨테이너 다시띄우기

docker compose down
docker compose up -d

```

## 7. NGINX 컨테이너 기동

### docker-compose.yml

```

# EC2 내부 경로 : home/ubuntu/docker/proxy/docker-compose.yml
services:
  nginx:
    image: nginx:latest
    container_name: nginx
    ports:      # 포트 매핑 (호스트:컨테이너)
      - "80:80"  # HTTP 트래픽용 포트
      - "443:443" # HTTPS 트래픽용 포트
    volumes:    # 볼륨 마운트 (호스트 경로:컨테이너 경로)
      - ./nginx.conf:/etc/nginx/nginx.conf

```



```

- ./conf.d/etc/nginx/conf.d
- ./data/certbot/conf:/etc/letsencrypt # SSL 인증서 저장 위치
- ./data/certbot/www:/var/www/certbot # Let's Encrypt 인증 파일 위치
  - /home/ubuntu/inquiry/images:/var/www/inquiry/images # 호스트 디렉토리
  - ./frontend/dist:/usr/share/nginx/html # 프론트 빌드 결과물 연결
command: "/bin/sh -c 'while ;; do sleep 6h & wait $$(!); nginx -s reload; c
networks:
  - app-network
restart: always

certbot:
  image: certbot/certbot
  container_name: certbot
  volumes:
  - ./data/certbot/conf:/etc/letsencrypt
  - ./data/certbot/www:/var/www/certbot
  networks:
    - app-network
  entrypoint: "/bin/sh -c 'trap exit TERM; while ;; do certbot renew; sleep 12
networks:
  app-network: # app-network에 연결 (백엔드와 통신하기 위함)
  external: true # 컨테이너가 종료되면 항상 재시작

```

## 8. CertBot Https 인증서 발급/적용

CertBot 전용 발급 nginx.conf 작성 → 발급 후 완전한 nginx.conf 변경

### 8.1. CertBot 전용 발급 nginx.conf

```

# EC2 내부 경로 : home/ubuntu/docker/proxy/nginx.conf

events {}

http {
  server {

```

```

listen 80;
server_name k12d101.p.ssafy.io;


location /.well-known/acme-challenge/ {
    root /var/www/certbot;
    allow all;
}


location / {
    return 404;
}
}
}

```

```

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;


events {
    worker_connections 1024;
}


http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;


    upstream backend {
        server ptsd-app:8000;
    }


    server {
        listen 80;
        listen [::]:80;
        server_name k12d101.p.ssafy.io;


        location /.well-known/acme-challenge/ {

```

```

    root /var/www/certbot;
}

location / {
    return 301 https://$server_name$request_uri;
}
}

server {
    listen 443 ssl;                # SSL 포트 리스닝
    listen [::]:443 ssl;          # IPv6 지원
    server_name k12d101.p.ssafy.io; # 도메인 이름 설정
    server_tokens off;            # 서버 버전 정보 숨김

    # SSL 인증서 설정
    ssl_certificate /etc/letsencrypt/live/k12d101.p.ssafy.io/fullchain.pem; #
    ssl_certificate_key /etc/letsencrypt/live/k12d101.p.ssafy.io/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf; # SSL 옵션
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # FastAPI
    location / {
        proxy_pass http://backend;
        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;
    }

    # 🔥 Jenkins 추가
    location /jenkins {
        proxy_pass http://jenkins:8080/jenkins;
        proxy_http_version 1.1;
        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;
    }
}

```

```
}
}
```

docker compose restart nginx → 컨테이너 재시작

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS	PORT
5fc6412341d6	<u>nginx:latest</u>	"/docker-entrypoint..."	nginx	7 minutes ago	Restarting (1) 4 seconds ago	
eb08a185b255	<u>certbot/certbot</u>	"/bin/sh -c 'trap ex..."	certbot	9 minutes ago	Up 9 minutes	80/tcp
dbf9f7376c69	redis:7	"docker-entrypoint.s..."	redis	19 minutes ago	Up 19 minutes	0.0.0.0:6379->6379/tcp
0.0:6379->6379/tcp, [::]:6379->6379/tcp	postgres:15	"docker-entrypoint.s..."	postgres	19 minutes ago	Up 19 minutes (healthy)	0.0.0.0:5432->5432/tcp
42f8f0a591a4	ptsd	"/usr/bin/tini -- /u..."	jenkins	3 hours ago	Up 2 hours	0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp, 50000/tcp

## 8.2. 폴더 생성 및 권한 설정

# Certbot 관련 디렉토리 생성

mkdir -p data/certbot/conf # 인증서가 저장될 경로

mkdir -p data/certbot/conf # 인증 챌린지 파일이 저장될 경로

# 디렉토리 소유권 및 권한 설정

sudo chown -R ubuntu:ubuntu data/certbot # ubuntu 사용자로 소유권 변경

sudo chmod -R 755 data/certbot # 읽기/쓰기 권한 설정

# 인증서 발급 명령어 실행

# --webroot: 웹 서버 루트 경로 방식으로 인증

# -w: 웹 루트 디렉토리 지정

# -d: 인증서를 발급받을 도메인

# --force-renewal: 기존 인증서가 있어도 강제로 갱신

ubuntu@ip-172-26-2-57:~/docker/proxy\$ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
30e7707f1a55	ptsd-app	"uvicorn PTSD.main:a..."	38 minutes ago	Up 38 minutes
8bac4a3fd569	eclipse-mosquitto:2	"/docker-entrypoint...."	38 minutes ago	Up 38 minutes
fe11f9a76c30	nginx:latest	"/docker-entrypoint...."	About an hour ago	Up About an hour
b3594632205f	jenkins/jenkins:2.501	"/usr/bin/tini -- /u..."	5 days ago	Up 5 days

```
ded084ae3684 postgres:15 "docker-entrypoint.s..." 12 days ago
2166e91afb5c redis:7 "docker-entrypoint.s..." 12 days ago Up
```

### 8.3. Certbot 인증서 발급 완료

```
ubuntu@ip-172-26-2-54:~/docker/proxy$ docker compose exec certbot certbot
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Requesting a certificate for j12d109.p.ssafy.io
```

```
Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/j12d109.p.ssafy.io/fullchain.pem
Key is saved at: /etc/letsencrypt/live/j12d109.p.ssafy.io/privkey.pem
This certificate expires on 2025-06-18.
These files will be updated when the certificate renews.
```

NEXT STEPS:

- The certificate will need to be renewed before it expires. Certbot can automa

```
-----
If you like Certbot, please consider supporting our work by:
* Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
* Donating to EFF: https://eff.org/donate-le
-----
```

### 8.4. 인증서 발급이 성공되면 SSL pem 파일 작성

```
# Diffie-Hellman 파라미터 생성 (SSL 보안 강화)
# 2048비트 키를 사용하여 생성
ubuntu@ip-172-26-2-57:~/docker/proxy$
명령어 -> openssl dhparam -out data/certbot/conf/ssl-dhparams.pem 2048
```

### 8.5. Nginx 작성 ( 최종본 )

```

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    # ✅ FastAPI 백엔드 (ptsd 서비스)
    upstream ptsd-app {
        server ptsd-app:8000;
    }

    # ✅ React 프론트엔드 (react 서비스)
    upstream react {
        server react:80;
    }

    # ✅ HTTP (Let's Encrypt 인증서 발급용)
    server {
        listen 80;
        listen [::]:80;
        server_name k12d101.p.ssafy.io;

        # certbot 인증 파일 경로
        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
            allow all;
        }

        # 나머지 HTTP 요청은 HTTPS로 리디렉션
        location / {
            return 301 https://$server_name$request_uri;
        }
    }
}

```

```

    }
}

# ✅ HTTPS 서비스 블록 (실제 서비스용)
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name k12d101.p.ssafy.io;
    server_tokens off;

    # SSL 인증서 설정
    ssl_certificate /etc/letsencrypt/live/k12d101.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k12d101.p.ssafy.io/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # 보안 헤더 추가
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header Referrer-Policy "no-referrer" always;

    # ✅ React 프론트엔드
    location / {
        proxy_pass http://react;
        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 $scheme;
    }

    # ✅ FastAPI 백엔드
    location /api/ {
        proxy_pass http://ptsd-app;
        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 $scheme;
}
# ✅ WebSocket 경로 추가

location /ws/ {
    proxy_pass http://ptsd-app/ws/;
    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 $scheme;
}

# ✅ Jenkins
location /jenkins {
    proxy_pass http://jenkins-v2501:8080/jenkins;
    proxy_http_version 1.1;
    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;
}

# ✅ 인증서 갱신용 (자동 renew)
location /.well-known/acme-challenge/ {
    root /var/www/certbot;
    allow all;
}
}
} //

```



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 이용하여 배포

```

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 branch")
          }
        }
      }
    }

    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 image shows a Jenkins Pipeline Run Summary and a detailed error log. The pipeline consists of the following steps: Start, FE-Check Branch, FE-CheckOut, FE-Build, FE-Deploy, Post Actions, and End. The FE-Build step is marked as failed with a red 'X' icon. The error log for the FE-Build step shows a 'Pipeline error' with a stack trace indicating a 'java.io.IOException' due to a missing 'NodeJS 22.13.0' installation in the Jenkins manager.

```

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.java:158)
3   PluginClassLoader for workflow-basic-steps//org.jenkinsci.plugins.workflow.steps.CoreWrapperStep$Execution2.doStart(CoreWrapperStep.java:121)
4   PluginClassLoader for workflow-step-api//org.jenkinsci.plugins.workflow.steps.GeneralNonBlockingStepExecution.lambda$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 the following 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 a red circle, indicating a failure. Below the pipeline view, the console output for the FE-Deploy stage is displayed, showing the execution of Docker Compose commands. The output indicates that the Docker Compose build failed due to a network 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

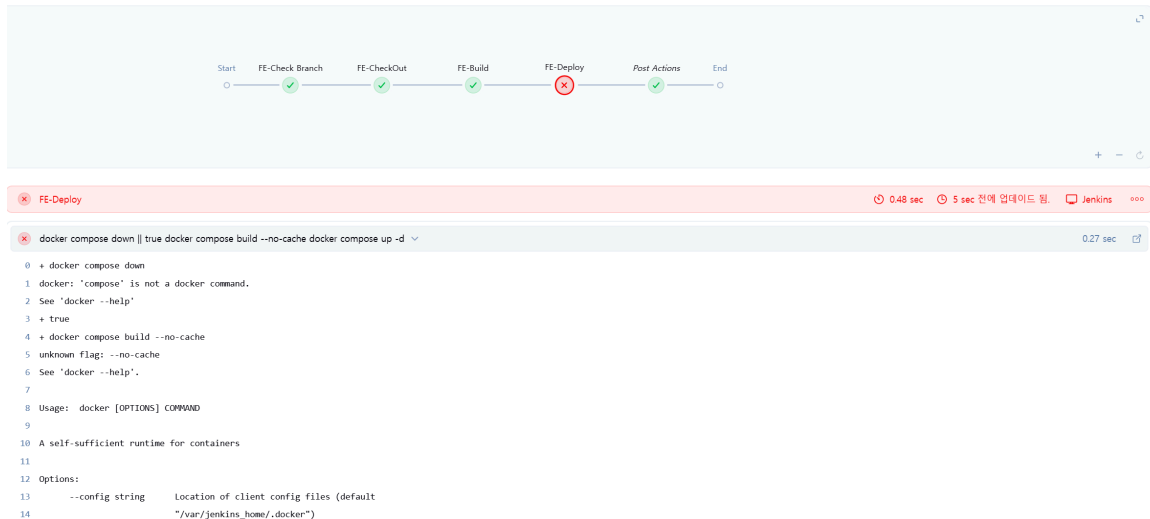
- Jenkins 환경의 Docker 버전이 낮아 `docker compose` 명령어 인식 불가

## 💡 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 screenshot shows a Jenkins pipeline with 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 a failure icon. The console output for FE-Deploy shows the command `docker compose down || true docker compose build --no-cache docker compose up -d` and the error message: `docker: 'compose' is not a docker command. See 'docker --help'`. The error occurs at line 0 of the script.

## 📦 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
f1e3a42dbac0	sl2p31d101-react	"/docker-entrypoint..."	react	19 minutes ago	Up 19 minutes	0.0.0.0:3000->80/tcp, [::]:3000->80/tcp
39e7707f1a55	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
fe11f9a76c30	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
b3594632205f	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
ded084ae3684	postgres:15	"docker-entrypoint.s..."	ptsd	12 days ago	Up 12 days (healthy)	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp
2166e91afb5c	redis:7	"docker-entrypoint.s..."	redis	12 days ago	Up 12 days	0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp

## 🖋️ 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 ✅
```

