



포팅 매뉴얼

[버전](#)

[사용한 포트](#)

[도커\(Docker\)](#)

[프론트 도커파일](#)

[백엔드 도커파일](#)

[플라스크 도커파일](#)

[젠킨스\(Jenkins\)](#)

[프론트 아이템 파이프라인 및 리눅스 명령어들 설명](#)

[백엔드 아이템 파이프라인](#)

[NginX설정](#)

[Nginx 설치법](#)

[SSL 설정\(CertBot\)](#)

[Nginx sites-available 코드](#)

[우분투 기본설정](#)

[젠킨스 설치방법](#)

[젠킨스란?](#)

[젠킨스 설치 \(Docker\)](#)

[Jenkins 환경설정 \(플러그인 미러서버 변경\)](#)

[Jenkins 접속](#)

[젠킨스 기본 설정](#)

[Jenkins 내부에 Docker 패키지,Docker-compose 설치](#)

[젠킨스 파이프라인 설정](#)

[플러그인 설치](#)

[GitLab Credential 등록 \(Username with password\)](#)

[Jenkins Webhook Integration 설정](#)

[도커허브](#)

[Ubuntu Credential 추가](#)

버전

도커 **25.0.4**

젠킨스 **2.451**

사용한 포트

- 젠킨스(CICD) : 8080
- 리액트(프론트) : 3000
- 스프링부트(백엔드) : 8082
- 플라스크(빅데이터) : 5000

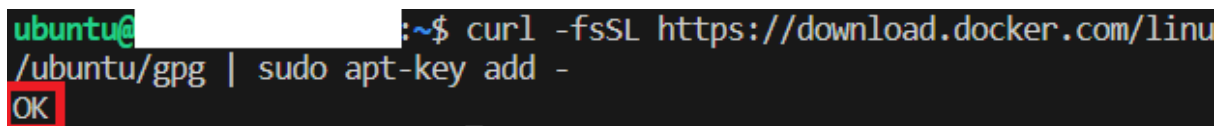
도커(Docker)

- Docker 설치 전 필요한 패키지 설치
 - gnupg-agent : OpenPGP 표준 규격의 데이터 통신을 암호화하고 서명할 수 있는 패키지 (추가)

```
sudo apt-get -y install apt-transport-https ca-certificates  
curl gnupg-agent software-properties-common
```

- Docker에 대한 GPG key 인증 진행

```
curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> |  
sudo apt-key add -
```



```
ubuntu@:~$ curl -fsSL https://download.docker.com/linux/  
/ubuntu/gpg | sudo apt-key add -  
OK
```

- Docker 레포지토리 등록

```
sudo add-apt-repository "deb [arch=amd64] <https://downloa  
d.docker.com/linux/ubuntu> $(lsb_release -cs) stable"
```

- 패키지 리스트 갱신

```
sudo apt-get -y update
```

- Docker 패키지 설치

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

- Docker 일반 유저에게 권한 부여(**linux docker sudo 없이 사용하기**)
 - 도커는 항상 root로 실행되기 때문에, sudo를 사용하여 명령어를 입력해야 했습니다.
 - 사용자를 docker 그룹에 추가하여 sudo를 사용하지 않아도 docker 명령이 사용될 수 있도록 진행하고자 합니다.

```
sudo usermod -aG docker ubuntu // ubuntu가 사용자, 도커 그룹 유  
저 추가  
sudo service docker restart // 도커 서비스 재시작  
exit  
sudo su - ubuntu // 현재 사용자 재 로그인
```

```
ubuntu@:~$ sudo su - ubuntu
ubuntu@:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

프론트 도커파일

```
FROM node:20.11 AS build

RUN mkdir /app

WORKDIR /app

COPY package.json /app/

RUN npm install

COPY . /app/

RUN npm run build
```

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

백엔드 도커파일

```
FROM docker
COPY --from=docker/buildx-bin:latest /buildx /usr/libexec/d
ocker/cli-plugins/docker-buildx

FROM openjdk:17-jdk
EXPOSE 8082
ARG JAR_FILE=./build/libs/*.jar

COPY ${JAR_FILE} app.jar

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

플라스크 도커파일

```
FROM python:3.9

# 작업 디렉토리를 설정합니다.
WORKDIR /flask

COPY . /flask

# 필요한 파일을 복사하고 패키지를 설치합니다.
RUN apt-get update
RUN apt-get -y install libgl1-mesa-glx
RUN pip install flask_cors
RUN pip install --no-cache-dir Flask Flask-Migrate Flask-SQ
LAlchemy mysql-connector-python scipy
RUN pip install gunicorn

# Flask 애플리케이션을 실행합니다.
CMD ["gunicorn", "flask_recommender:app", "-w", "4", "--bin
d", "0.0.0.0:5000"]
```

젠킨스(Jenkins)

프론트 아이템 파이프라인 및 리눅스 명령어들 설명

```
pipeline {
  agent any
  tools {nodejs "nodejs"}

  environment {
    imageName = "이미지이름 넣기"
    registryCredential = 'dockerhub' //젠킨스에 등록해놓은
자격증명서
    dockerImage = ''

    releaseServerAccount = 'ubuntu'
    releaseServerUri = '서비스 URL EC2서버'
    releasePort = '리액트가 사용하는 포트'
  }

  stages {
    stage('Git Clone') {
      steps {
        git branch: 'FE',
            credentialsId: 'gitlab', //젠킨스에 등록해
놓은 자격증명서
            url: 'URL.git' //git clone할 때 사용하는
url
      }
    }
    stage('Image Build & DockerHub Push') {
      steps {
        dir('frontend') {
          script {
            docker.withRegistry('', registryCre
dential) {
              sh "docker buildx create --use
```

```

--name mybuilder"
        sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:latest --push ."
    }
}
}
}
stage('Before Service Stop') {
    steps {
        sshagent(credentials: ['ubuntu-c106']) {
            sh '''
                if test "`ssh -o StrictHostKeyChecking=
no $releaseServerAccount@$releaseServerUri "docker ps -aq -
-filter ancestor=mooncheolhwan/frontend:latest"`"; then
                    ssh -o StrictHostKeyChecking=no $releas
eServerAccount@$releaseServerUri "docker stop $(docker ps -
aq --filter ancestor=mooncheolhwan/frontend:latest)"
                    ssh -o StrictHostKeyChecking=no $releas
eServerAccount@$releaseServerUri "docker rm -f $(docker ps
-aq --filter ancestor=mooncheolhwan/frontend:latest)"
                    ssh -o StrictHostKeyChecking=no $releas
eServerAccount@$releaseServerUri "docker rmi mooncheolhwan/
frontend:latest"
                fi
            '''
        }
    }
}
stage('DockerHub Pull') {
    steps {
        sshagent(credentials: ['ubuntu-c106']) {
            sh "ssh -o StrictHostKeyChecking=no $re
leaseServerAccount@$releaseServerUri 'sudo docker pull $ima
geName:latest'"
        }
    }
}
}

```

```

stage('Service Start') {
    steps {
        sshagent(credentials: ['ubuntu-c106']) {
            sh '''
                ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri "sudo docker rm -f fe"
                ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri "sudo docker run -i -e TZ=Asia/Seoul --name fe -p $releasePort:$releasePort -d $imageName:latest"
            '''
        }
    }
}

stage('Service Check') {
    steps {
        sshagent(credentials: ['ubuntu-c106']) {
            sh '''
                #!/bin/bash

                for retry_count in \$(seq 20)
                do
                    if curl -s "URL" > /dev/null
                    then
                        curl -d '{"text":"Release Complete"}' -H "Content-Type: application/json" -X POST 메타모스트 웹훅
                        break
                    fi

                    if [ $retry_count -eq 20 ]
                    then
                        curl -d '{"text":"Release Failed"}' -H "Content-Type: application/json" -X POST 메타모스트 웹훅
                        exit 1
                    fi
                done
            '''
        }
    }
}

```

```

    echo "The server is not alive yet. Retry health check in 5 seconds..."
    sleep 5
done
'''
}
}
}
}
}
```

위 코드로 진행되었습니다.

단계는 6단계로,

1. Git Clone : Git 저장소에서 소스 코드를 가져옵니다.
2. Image Build & DockerHub Push : Docker 이미지를 빌드하고 DockerHub에 푸시합니다.
3. Before Service Stop : 이전에 실행된 Docker 컨테이너를 중지하고 제거합니다.
4. DockerHub Pull : 새로 업데이트된 Docker 이미지를 서버로 가져옵니다.
5. Service Start : 새로운 Docker 컨테이너를 시작합니다.
6. Service Check : 서비스가 올바르게 시작되었는지 확인합니다.

agent any : 어떤 에이전트에서든 실행될 수 있다.

tools : 파이프라인에서 사용하는 도구

environment : 환경 변수 정의함.

stages : 파이프라인의 주요 단계 정의

steps : 각 단계에서 실행할 명령, shell 스크립트로 실행됨.

- `docker buildx create --use --name mybuilder`
 - Docker Buildx 빌더를 생성하고 사용할 이름을 지정하는 명령
 - `-use` : 새로 생성된 빌더를 사용하도록 설정합니다.
 - `-name mybuilder` : 새로 생성된 빌더의 이름을 "mybuilder"로 지정합니다.
 - mybuilder 빌더를 사용하여 다중 아키텍처 이미지를 빌드할 수 있게 됩니다.

- `docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:latest --push .`
 - Docker Buildx를 사용하여 다중 아키텍처 이미지를 빌드하고 푸시하는 명령
 - `-platform linux/amd64,linux/arm64` : 빌드할 이미지의 플랫폼을 지정합니다. 여기서는 Linux의 amd64 아키텍처와 arm64 아키텍처를 지정했습니다. 즉, 이 명령은 amd64와 arm64 아키텍처용 이미지를 동시에 빌드합니다.
 - `t $imageName:latest` : 빌드한 이미지에 태그를 지정합니다. \$imageName은 환경 변수로 설정된 이미지 이름을 나타냅니다. latest는 태그로서, 이 이미지가 최신 버전임을 나타냅니다.
 - `-push` : 빌드된 이미지를 레지스트리에 푸시합니다.
 - `.` : 현재 디렉토리에서 Dockerfile을 사용하여 이미지를 빌드합니다.

- `if test "\ssh -o StrictHostKeyChecking=no \\$releaseServerAccount@\\$releaseServerUri "docker ps -aq --filter ancestor=mooncheolhwan/frontend:latest"\""; then`
 - 원격 서버에서 실행 중인 모든 컨테이너의 ID를 가져와서 filter 옵션을 사용하여 특정 이미지를 뽑아냅니다.
 - 필터된 ID가 mooncheolhwan/frontend:latest이면 아래코드가 실행됩니다.
- `"ssh -o StrictHostKeyChecking=no \\$releaseServerAccount@\\$releaseServerUri "docker stop $(docker ps -aq --filter ancestor=mooncheolhwan/frontend:latest)"`
 - `docker stop` 명령을 사용하여 원격 서버에서 실행 중인 특정 이미지를 기반으로한 컨테이너를 중지합니다.
- `"ssh -o StrictHostKeyChecking=no \\$releaseServerAccount@\\$releaseServerUri "docker rm -f $(docker ps -aq --filter ancestor=mooncheolhwan/frontend:latest)"`
 - `docker rm -f` 명령을 사용하여 원격 서버에서 실행 중인 특정 이미지를 기반으로한 컨테이너를 강제로 제거합니다.
- `"ssh -o StrictHostKeyChecking=no \\$releaseServerAccount@\\$releaseServerUri "docker rmi mooncheolhwan/frontend:latest"`
 - `docker rmi` 명령을 사용하여 해당 이미지를 원격 서버에서 제거합니다.

- `ssh -o StrictHostKeyChecking=no \`
`\$releaseServerAccount@\$releaseServerUri "sudo docker rm -f fe"`
 - 원격 서버에 SSH를 통해 접속한 후, `sudo docker rm -f fe` 명령을 실행합니다.
 - 이 명령은 fe라는 이름의 Docker 컨테이너를 강제로 제거합니다.
 - `f` 옵션은 강제로 컨테이너를 제거
- `ssh -o StrictHostKeyChecking=no \`
`\$releaseServerAccount@\$releaseServerUri "sudo docker run -i -e`
`TZ=Asia/Seoul --name fe -p \$releasePort:\$releasePort -d`
`\$imageName:latest"`
 - 원격 서버에 SSH를 통해 접속한 후, 새로운 Docker 컨테이너를 실행합니다.
 - `i` : 인터랙티브 모드로 실행합니다.
 - `e TZ=Asia/Seoul` : 컨테이너 내부의 시간대를 Asia/Seoul로 설정합니다.
 - `-name fe` : 컨테이너의 이름을 fe로 지정합니다.
 - `p \$releasePort:\$releasePort` : 호스트의 `\$releasePort` 포트를 컨테이너의 동일한 포트로 매핑합니다.
 - `d` : 백그라운드 모드로 실행합니다.
 - `\$imageName:latest` : 실행할 이미지의 이름과 태그를 지정합니다.
 - `\$imageName` 은 환경 변수로 설정된 이미지 이름을 나타냅니다.

마지막 step인 Service Check 코드에 대해 설명하자면,

- `for retry_count in \$(seq 20)` : 20번의 반복을 수행하는 for 루프를 시작합니다.
- `if curl -s "URL" > /dev/null` : curl을 사용하여 특정 URL로 HTTP GET 요청을 보냅니다. -s 옵션은 silent 모드로 출력을 표시하지 않습니다. 응답은 /dev/null로 보냅니다.
- `curl -d '{"text":"Release Complete"}' -H "Content-Type: application/json" -X POST 메타`
`모스트 웹훅` : Release Complete 메시지를 포함한 JSON 데이터를 지정된 웹훅 URL로 POST 요청을 보냅니다.
- `break`: 반복문을 종료합니다.
- `if [$retry_count -eq 20]` : 20번의 시도가 모두 실패한 경우 실행합니다.
- `curl -d '{"text":"Release Fail"}' -H "Content-Type: application/json" -X POST 메타`
`모스트 웹훅` : Release Fail 메시지를 포함한 JSON 데이터를 지정된 웹훅 URL로 POST 요청

을 보냅니다.

- exit 1: 스크립트를 종료하고 오류 코드 1을 반환합니다.
- `echo "The server is not alive yet. Retry health check in 5 seconds..."`: 서버가 아직 준비되지 않았음을 나타내는 메시지를 출력합니다.

백엔드 아이템 파이프라인

```
pipeline {
    agent any

    environment {
        imageName = "muncheolhwan/c106-back"
        registryCredential = 'dockerhub-c106'
        dockerImage = ''

        releaseServerAccount = 'ubuntu'
        releaseServerUri = 'j10c106.p.ssafy.io'
        releasePort = '8082'
    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: 'sh',
                    credentialsId: 'gitlab-muncheolhwan',
                    url: 'https://lab.ssafy.com/s10-bigdata-recom-sub2/S10P22C106.git'
            }
            post {
                success {
                    sh 'echo "Successfully Cloned Repository"'
                }
                failure {
                    sh 'echo "Fail Cloned Repository"'
                }
            }
        }
    }
}
```

```

stage('Jar Build') {
    steps {
        sh 'chmod +x ./gradlew'
        sh './gradlew clean bootJar'
        // sh './gradlew build'
    }
    post {
        success {
            echo 'gradle build success'
        }

        failure {
            echo 'gradle build failed'
        }
    }
}

stage('Image Build & DockerHub Push') {
    steps {
        script {
            docker.withRegistry('', registryCredent
ial) {
                sh "docker buildx create --use --na
me mybuilder"

                sh "docker buildx build --platform
linux/amd64,linux/arm64 -t $imageName:latest --push ."
            }
        }
    }
}

stage('Before Service Stop') {
    steps {
        sshagent(credentials: ['ubuntu-c106']) {
            sh '''
            if test "`ssh -o StrictHostKeyChecking=
no $releaseServerAccount@$releaseServerUri "docker ps -aq -
-filter ancestor=$imageName:latest"`"; then
                ssh -o StrictHostKeyChecking=no $releas
eServerAccount@$releaseServerUri "docker stop $(docker ps -

```

```

    aq --filter ancestor=$imageName:latest)"
        ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri "docker rm -f $(docker ps
-aq --filter ancestor=$imageName:latest)"
        ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri "docker rmi $imageName:latest"
    fi
    ''
}
}
}
stage('DockerHub Pull') {
    steps {
        sshagent(credentials: ['ubuntu-c106']) {
            sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker pull $imageName:latest'"
        }
    }
}
stage('Service Start') {
    steps {
        sshagent(credentials: ['ubuntu-c106']) {
            sh '''
                ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri "sudo docker run -i -e
TZ=Asia/Seoul --name muncheolhwan -p $releasePort:$releasePort -d $imageName:latest"
            '''
        }
    }
}
}
}

```

NginX설정

Nginx 설치법

- NginX설치

```
sudo apt install nginx -y
```

- NginX 설치 확인

```
sudo systemctl status nginx
```

- 방화벽 확인
sudo ufw status
- 방화벽 허용
sudo ufw allow 80

SSL 설정(CertBot)

- Let's Encrypt는 사용자가 웹사이트를 위한 무료, 자동화된, 공개 인증 기관(CA)을 제공하는 비영리 서비스입니다. 이를 통해 HTTPS를 통한 보안 연결을 쉽게 설정할 수 있습니다.

```
sudo apt-get install letsencrypt
```

- CertBot 설치
방법1)

```
sudo apt-get -y install python3-certbot-nginx
```

- SSL 인증서 발급
 - Certbot NginX연결

```
sudo certbot --nginx
```

- 도메인 입력
- 기존 인증서 재설치 시도 : 1번 선택

- redirect 2번 선택
- /etc/cron.d에서 자동으로 갱신되는 스크립트가 설치중 기록된다.

Nginx sites-available 코드

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;
    index index.html index.htm index.nginx-debian.html;
    server_name _;
    include /etc/nginx/conf.d/service-url.inc;
    location / {
        proxy_pass $service_url;
    }
}

server {
    listen 443 ssl http2;
    server_name back;

    # ssl 인증서 적용하기
    ssl_certificate /etc/letsencrypt/live/j10c106.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j10c106.p.ssafy.io/private.pem;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header Host $http_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;
    }
    location /users {
        proxy_pass http://127.0.0.1:8082;
```

```

    proxy_set_header Host $http_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;
}
location /recommend {
    proxy_pass http://127.0.0.1:5000;
    proxy_set_header Host $http_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;
}
}

server {
    if ($host = j10c106.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80 ;
    listen [::]:80 ;
    server_name j10c106.p.ssafy.io;
    return 404; # managed by Certbot
}

```

우분투 기본설정

해당 배포 실전은 개념에 대한 설명보단, 실행한 코드 위주로 기록용으로 작성됨을 알려드립니다.^^

- 서버시간 변경 (ec2 접속후 진행)

```

sudo timedatectl set-timezone Asia/Seoul
date

```

- 미리 서버 변경

sudo vi /etc/apt/sources.list :%s/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kakao.com/

```
# deb-src http://archive.canonical.com/ubuntu focal partner
deb http://security.ubuntu.com/ubuntu focal-security main restricted
# deb-src http://security.ubuntu.com/ubuntu focal-security main restricted
deb http://security.ubuntu.com/ubuntu focal-security universe
# deb-src http://security.ubuntu.com/ubuntu focal-security universe
deb http://security.ubuntu.com/ubuntu focal-security multiverse
# deb-src http://security.ubuntu.com/ubuntu focal-security multiverse
~
```

- 패키지 업데이트, 업그레이드

sudo apt update

해당 명령어 입력시 아래처럼 오류가 나온다.

```
Reading package lists... Done
E: The repository 'http://ppa.launchpad.net/certbot/certbot/ubuntu focal Release' does not have a Release file.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
```

- 가상 메모리 할당

free -h 로 현재 메모리 용량 확인

- 스왑 영역 할당

sudo dd if=/dev/zero of=/swapfile bs=128M count=64 혹은
sudo fallocaate -l 8G /swapfile

- swapfile 읽기 쓰기 권한을 업데이트

sudo chmod 600 /swapfile

- swapfile 생성

```
sudo mkswap /swapfile
```

```
Setting up swspace version 1, size = 8 GiB (8589930496 bytes)
```

- 스왑 공간에 스왑 파일을 추가하여 스왑 파일을 즉시 사용

```
sudo swapon /swapfile
```

- 프로시저가 성공적인지 확인

```
sudo swapon -s
```

Filename	Type	Size	Used	P
priority				
/swapfile	file	8388604	0	-
2				

- 시스템이 재부팅 되어도 swap 유지할 수 있도록 설정

```
sudo echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

```
/swapfile none swap sw 0 0
```

- swap영역이 할당 되었는지 확인

```
free -h
```

	total	used	free	shared	buff/cache
Mem:	15Gi	938Mi	13Gi	1.0Mi	1.1Gi
Swap:	8.0Gi	0B	8.0Gi		

젠킨스 설치방법

젠킨스란?

- 지속적인 통합 및 지속적인 배포(CI/CD)를 관리하기 위한 오픈 소스 자동화 도구입니다.
- 젠킨스를 사용하면 소프트웨어 개발 프로세스를 자동화하여 개발자가 코드를 통합하고 테스트하며, 빌드하고 배포하는 등의 작업을 자동으로 수행할 수 있습니다.
- Job(작업)
 - Jenkins에서 실행하는 각각의 작업을 나타냅니다.
 - Job은 소프트웨어 빌드, 테스트, 배포 등 다양한 작업을 수행할 수 있습니다.
- Build(빌드)
 - 소스 코드를 실행 가능한 소프트웨어로 변환하는 과정을 의미합니다.
 - Jenkins를 사용하여 빌드 작업을 자동화할 수 있습니다.
- Pipeline(파이프라인)
 - 여러 단계의 작업을 연결하여 자동화 프로세스를 구성하는 도구입니다.
 - 소프트웨어 개발 및 배포 과정을 단계별로 정의하고 실행할 수 있습니다.
- Plugin(플러그인)
 - Jenkins의 기능을 확장하기 위해 사용되는 확장 모듈입니다.
 - 다양한 프로젝트 유형 및 환경에 맞게 Jenkins를 사용할 수 있습니다.

젠킨스 설치 (Docker)

- Jenkins 이미지 받기
 - Java 17 버전 이용

```
docker pull jenkins/jenkins:jdk17
```

- Jenkins 컨테이너 실행

```
docker run -d --restart always --env JENKINS_OPTS=--httpPort=8080 -v /etc/localtime:/etc/localtime:ro -e TZ=Asia/Seoul -p 8080:8080 -v /jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -v /usr/local/bin/docker-compose
```

```
se:/usr/local/bin/docker-compose --name jenkins -u root jenkins/jenkins:jdk17
```

- 위 코드 설명
 - `d` : 컨테이너를 데몬으로 띄운다.
 - `-restart always` : 컨테이너가 어떤 이유로 종료되더라도, 자동으로 재시작 하는 옵션
 - `e TZ=Asia/Seoul` : 환경변수 설정(내부 시간을 Asia/Seoul 설정)
 - `p 8080:8080` : 컨테이너 외부와 내부 포트에 대해 포워딩
 - 왼쪽 : Host Port
 - 오른쪽 : Container Port
 - `v /tec/localtime:/etc/localtime:ro` : Host OS의 localtime을 컨테이너의 localtime과 동기화
 - `v /jenkins:/var/jenkins_home` : 도커 컨테이너의 데이터는 컨테이너가 종료되면 사라지기 때문에, 볼륨 마운트 옵션을 이용하여 Jenkins 컨테이너의 `/var/jenkins_home` 디렉토리를 Host OS의 `/jenkins`와 연결하여 데이터를 유지한다.
 - `-name jenkins` : 도커 컨테이너의 이름을 설정하는 옵션
 - `u root` : 컨테이너가 실행될 리눅스 사용자 계정 지정(root)

- Jenkins 종료

```
docker stop jenkins
```

- 포트가 개방되어있는지, 확인하기

```
netstat -nltp
```

Jenkins 환경설정 (플러그인 미러서버 변경)

- Jenkins 데이터가 있는 디렉토리에 update-center-rootCAs 하위 디렉토리 생성

```
sudo mkdir /jenkins/update-center-rootCAs
```

- CA파일 다운로드

```
sudo wget <https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt> -O /jenkins/update-center-rootCAs/update-center.crt
```

- Jenkins 플러그인 다운로드시 미러사이트로 대체될 수 있도록 설정

```
sudo sed -i 's#<https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com/lework/jenkins-update-center/master/updates/tencent/update-center.json#>' /jenkins/hudson.model.UpdateCenter.xml
```

- Jenkins 컨테이너 재시작

```
docker restart jenkins
```

Jenkins 접속

- 해당 url:8080으로 접속하면 젠킨스 화면이 뜬다.

Getting Started

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

- 웹 페이지 비밀번호를 확인하기
 - Jenkins 컨테이너의 bash 셸에 접속

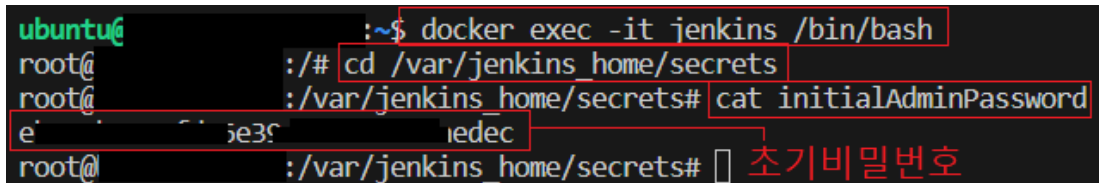
```
docker exec -it jenkins /bin/bash
```

- 해당하는 디렉토리로 이동한다

```
cd /var/jenkins_home/secrets
```

- 초기 비밀번호를 확인한다. 이후 exit으로 bash셸을 나간다.

```
cat initialAdminPassword
exit
```



```
ubuntu@~$ docker exec -it jenkins /bin/bash
root@:/# cd /var/jenkins_home/secrets
root@:/var/jenkins_home/secrets# cat initialAdminPassword
e'...'ie3C'iedec
root@:/var/jenkins_home/secrets#
```

초기비밀번호

젠킨스 기본 설정

- Jenkins 기본 설정은 아래 url에 정리되어있습니다^^

https://velog.io/@m_moon_c/Jenkins-이용법12

Jenkins 내부에 Docker 패키지, Docker-compose 설치

- 컨테이너 재시작 마다 하는 작업
- 젠킨스 컨테이너 접속

```
docker exec -it jenkins /bin/bash
```

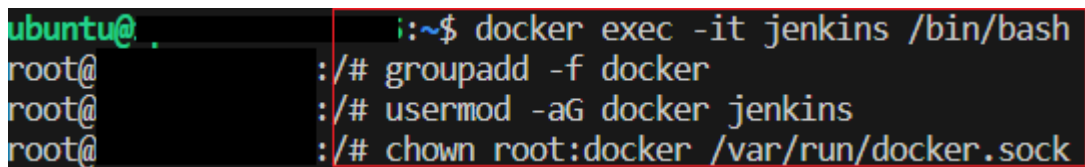
- Docker Repository 등록 및 docker-ce 패키지 설치(AMD64)

```
apt-get update && apt-get -y install apt-transport-https ca
-certificates curl gnupg2 software-properties-common && cur
l -fsSL <https://download.docker.com/linux/$>(. /etc/os-rel
ease; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey &&
```

```
add-apt-repository "deb [arch=amd64] <https://download.docker.com/linux/$>(. /etc/os-release; echo "$ID") $(lsb_release -cs) stable" && apt-get update && apt-get -y install docker-ce
```

- Docker Jenkins에서 Host Docker 접근권한 부여

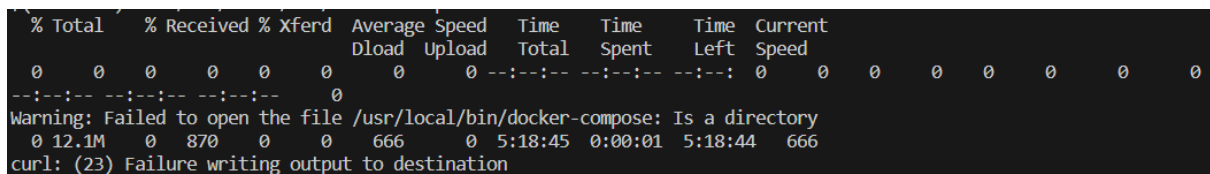
```
groupadd -f docker
usermod -aG docker jenkins
chown root:docker /var/run/docker.sock
```



```
ubuntu@. :~$ docker exec -it jenkins /bin/bash
root@:/# groupadd -f docker
root@:/# usermod -aG docker jenkins
root@:/# chown root:docker /var/run/docker.sock
```

- Docker Compose 다운로드

```
curl -L "<https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```



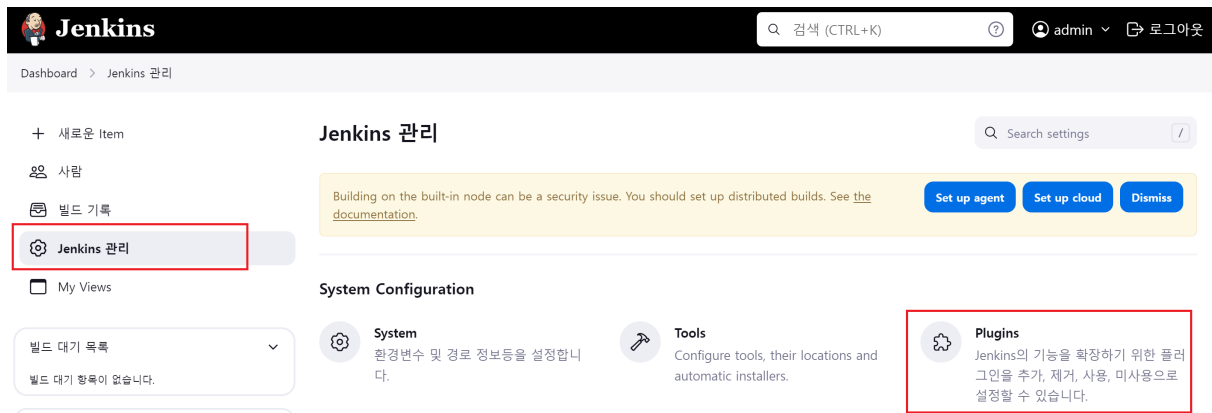
```
% Total % Received % Xferd Average Speed Time Time Time Current
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Warning: Failed to open the file /usr/local/bin/docker-compose: Is a directory
0 12.1M 0 870 0 666 0 5:18:45 0:00:01 5:18:44 666
curl: (23) Failure writing output to destination
```

- /usr/local/bin/docker-composer 권한 변경

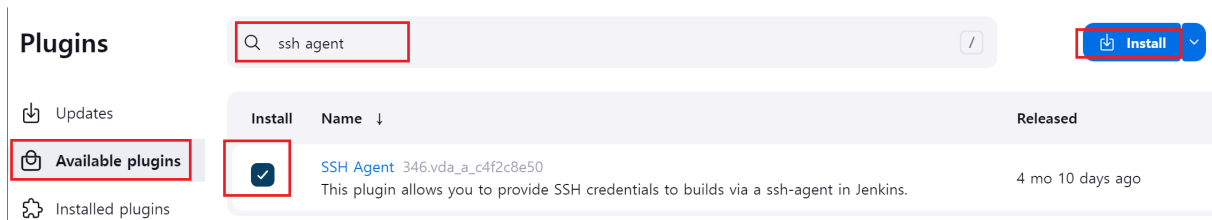
```
chmod +x /usr/local/bin/docker-compose
```

젠킨스 파이프라인 설정

플러그인 설치



The screenshot shows the Jenkins Dashboard. The top navigation bar includes the Jenkins logo, a search bar, and user information (admin). The left sidebar contains links to '새로운 Item', '사람', '빌드 기록', 'Jenkins 관리' (highlighted with a red box), and 'My Views'. The main content area is titled 'Jenkins 관리' and features a yellow warning banner about distributed builds. Below this is the 'System Configuration' section with links to 'System' and 'Tools'. A 'Plugins' section is also visible, with a red box highlighting the 'Plugins' link and its description.

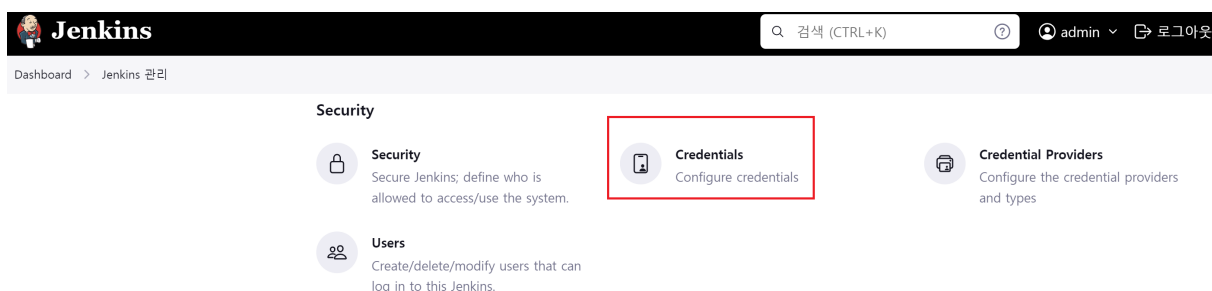


The screenshot shows the 'Plugins' page in Jenkins. The left sidebar has 'Available plugins' highlighted with a red box. The main content area has a search bar with 'ssh agent' entered. Below the search bar is a table of available plugins. The 'SSH Agent' plugin is highlighted with a red box, showing its name, version (346.vda_a_c4f2c8e50), and release date (4 mo 10 days ago). The 'Install' button for this plugin is also highlighted with a red box.

- 플러그인 설치 목록
 - SSH Agent : ssh 커멘드 입력에 사용
 - Docker / Docker Commons / Docker Pipeline / Docker API : docker 이미지 생성에 사용
 - Generic Webhook Trigger : 웹훅을 통해 브랜치 merge request 이벤트 발생 시 Jenkins 자동 빌드에 사용
 - GitLab / GitLab API / GitLab Authentication : GitLab 레포지토리 이용시 사용
 - NodeJS : Node.js 빌드시 사용

GitLab Credential 등록 (Username with password)

- Jenkins관리 - Credentials 클릭



The screenshot shows the 'Security' page in Jenkins. The left sidebar contains links to 'Security', 'Users', and 'Credentials' (highlighted with a red box). The main content area is titled 'Security' and contains three sections: 'Security' (Secure Jenkins; define who is allowed to access/use the system.), 'Users' (Create/delete/modify users that can log in to this Jenkins.), and 'Credentials' (Configure credentials) (highlighted with a red box). The 'Credential Providers' section is also visible, with the text 'Configure the credential providers and types'.


- Add credentials 클릭

Jenkins 관리 > Credentials

Credentials

T	P	Store ↓	Domain	ID
---	---	---------	--------	----

Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global) ▼ Add credentials

- 여기서 필요한 GitLab Token 발급받는다.

https://velog.io/@m_moon_c/GitLab-Token-받기

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

 Gitlab 계정 아이디 입력

☐ Treat username as secret ?

Password ?



Concealed 발급 받은 토큰 넣어주기

ID ?

gitlab-muncheolhwan Credential 대한 별칭

Description ?

gitlab connection with token | Credential 설명

- GitLab Credential 등록 (**API Token**)

New credentials

Kind

GitLab API token

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

API token

.....

ID ?

api- [REDACTED]

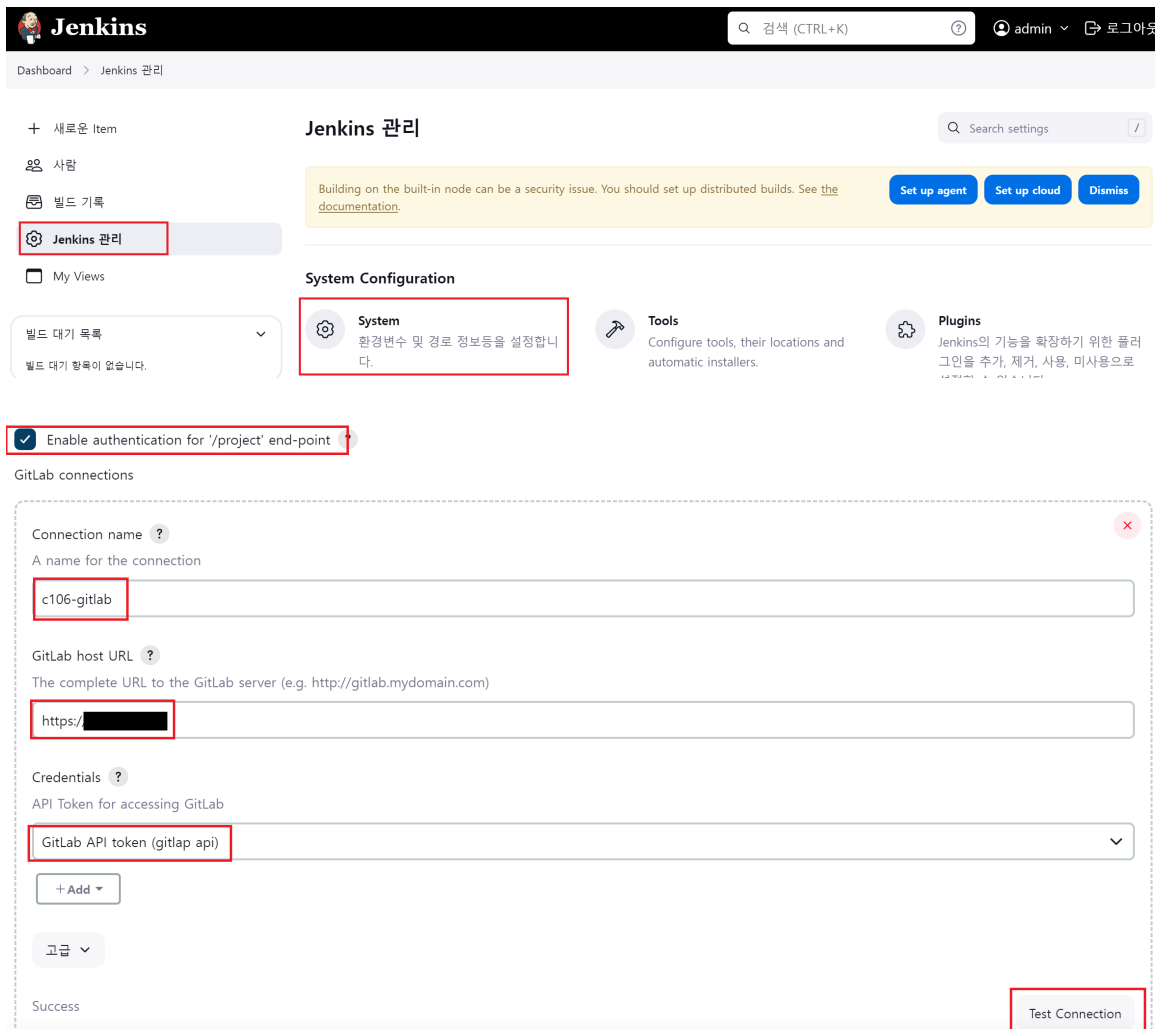
Description ?

gitlap api



- GitLab 커넥션 추가

Test 진행 후 저장!



Jenkins Webhook Integration 설정

- 젠킨스 파이프라인 아이템 만들기

아래 이미지 버튼 체크체크 (밀줄 그은, url은 깃랩 웹훅에서 쓰인다)

고급 > Secret token (Generate 클릭 후 토큰 메모장에 복사)

저장하기.

Enter an item name

c106

» Required field



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

다양한 환경에서의 테스트, 플랫폼 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Branch Pipeline

a set of Pipeline projects according to detected branches in one SCM repository.

OK

Configure

General

Advanced Project Options

Pipeline



Build when a change is pushed to GitLab. GitLab webhook URL: <http://c106>

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

고급 ^

☒ Enable [ci-skip] ?

☒ Ignore WIP Merge Requests ?

Labels that launch a build if they are added (comma-separated) ?

☒ Set build description to build cause (eg. Merge request or Git Push) ?

☐ Build on successful pipeline events

Pending build name for pipeline ?

☐ Cancel pending merge request builds on update ?

Allowed branches

☒ Allow all branches to trigger this job ?

☐ Filter branches by name ?

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

Generate

• Gitlab Webhook 지정

Secure

Deploy

Operate

Monitor

Analyze

Settings

General

Integrations

Webhooks

Access Tokens

Repository

Merge requests

CI/CD

Packages and regist...

Monitor

Search page

Webhook

Webhooks enable you to send notifications to web applications in response to events in a webhook.

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.

Secret token

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

Trigger

☒ Push events

☐ All branches

☐ Wildcard pattern

☒ Regular expression

Regular expressions such as `^(feature|hotfix)/` are supported.

도커허브

- 도커허브 토큰생성

<https://hub.docker.com/>
도커 허브 사이트 로그인

New Access Token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access Token Description *

jenkins-token

Access permissions

Read, Write, Delete

Read, Write, Delete tokens allow you to manage your repositories.

[Cancel](#) [Generate](#)

Access Tokens

Tokens marked [\(AUTO-GENERATED\)](#) are created on your behalf by Docker Desktop for the CLI to use for authentication. You can have a maximum of 5 auto-generated tokens associated with your account. [Learn more](#)

<input type="checkbox"/>	Description	Source	Scope	Last Used	Created
New Access Token					

Copy Access Token

When logging in from your Docker CLI client, use this token as a password. [Learn more](#)

ACCESS TOKEN DESCRIPTION

jenkins-token

ACCESS PERMISSIONS

Read, Write, Delete

발급받은 토큰은 1번만 볼 수 있으니,
저장해놓도록 하자

To use the access token from your Docker CLI client:

1. Run `docker login -u muncheolhwan`
2. At the password prompt, enter the personal access token.

[Copy](#)

WARNING: This access token will only be displayed once. It will not be stored and cannot be retrieved. Please be sure to save it now.

[Copy and Close](#)

- 도커허브 레포지토리 생성

dockerhub Explore **Repositories** Organizations Search Docker Hub ctrl+K ? M

Search by repository name Q All Content ▼ Create repository

/ **c106-bigdata**
Contains: No content | Created: less than a minute ago Security unknown 0 0 Public

Create repository

Namespace ▼ Repository Name *

Short description

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility
Using 1 of 1 private repositories. [Get more](#)

☒ **Public**
Appears in Docker Hub search results

☐ **Private**
Only visible to you

Cancel Create

Pushing images
You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to replace tagname with your desired image repository tag.

- 도커허브 젠킨스 Credential 추가

New credentials

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

도커허브에서 사용하는 계정아이디 입력

☐

Treat username as secret ?

Password ?

발급받은 토큰 입력

ID ?

젠킨스 내부에서 사용되는 별칭

Description ?

Create

Ubuntu Credential 추가

Kind

SSH Username with private key

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

ID ?

ubuntu-c106 젠킨스에서 사용할 별칭

Description ?

Username

ubuntu SSH원격서버호스트 계정명

☐ Treat username as secret ?

Private Key

☒ Enter directly

Key

aws발급해준 pem키의 내용을 메모장으로 열고, 해당 내용을 복사하여 붙여넣기

Enter New Secret Below

```

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAzad5W7/gkRn6L9kt0LpZ6p715YTE0gRfT0JA8coyzz6W90L9
un900d2q1hY11swv13PvhwF5Gwrt0d4cTvoS3ZdyGPJhk1ch4MkyY2IouWUEdb
dfj1ngEvYBIMqfkyaa2nrm/0GorBjWkR03e3H/wSe9Wk13G1Y/vvA=
-----END RSA PRIVATE KEY-----

```

Create

New credentials

Kind

GitLab API token

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

API token

.....

ID ?

api-

Description ?

gitlap api

Create