



포팅 메뉴얼

▼ aws ec2

▼ Bash에서 PEM으로 ec2 접속

```
ssh -i I7A503T.pem ubuntu@i7a503.p.ssafy.io
```

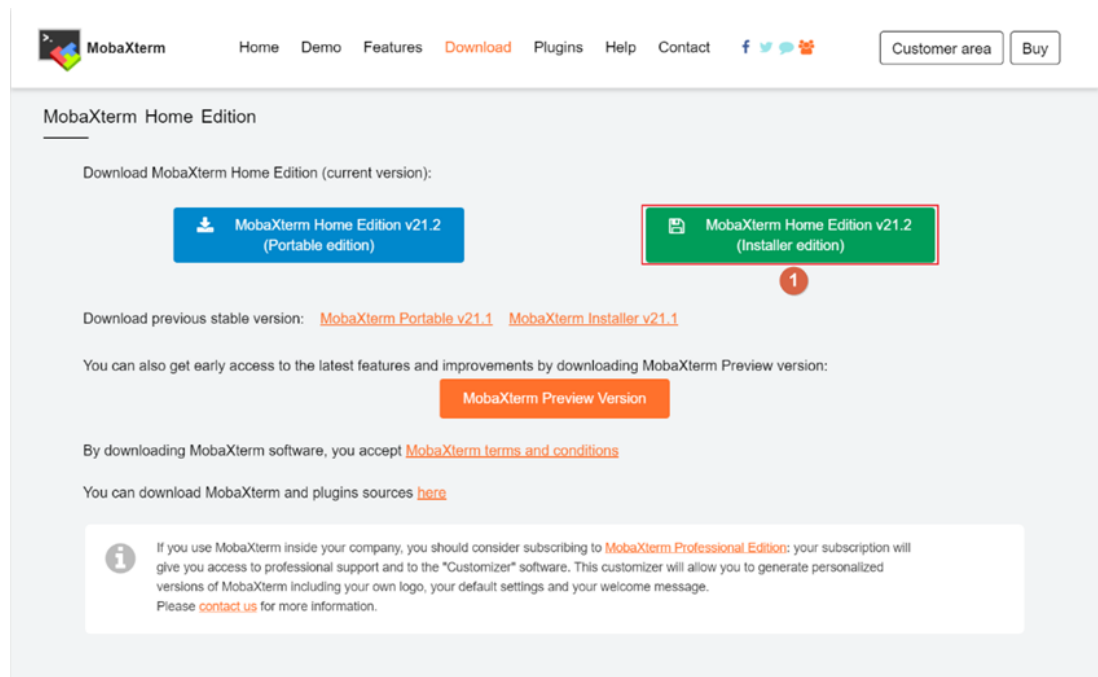
▼ 공통 ip 확인

```
curl ifconfig.me
```

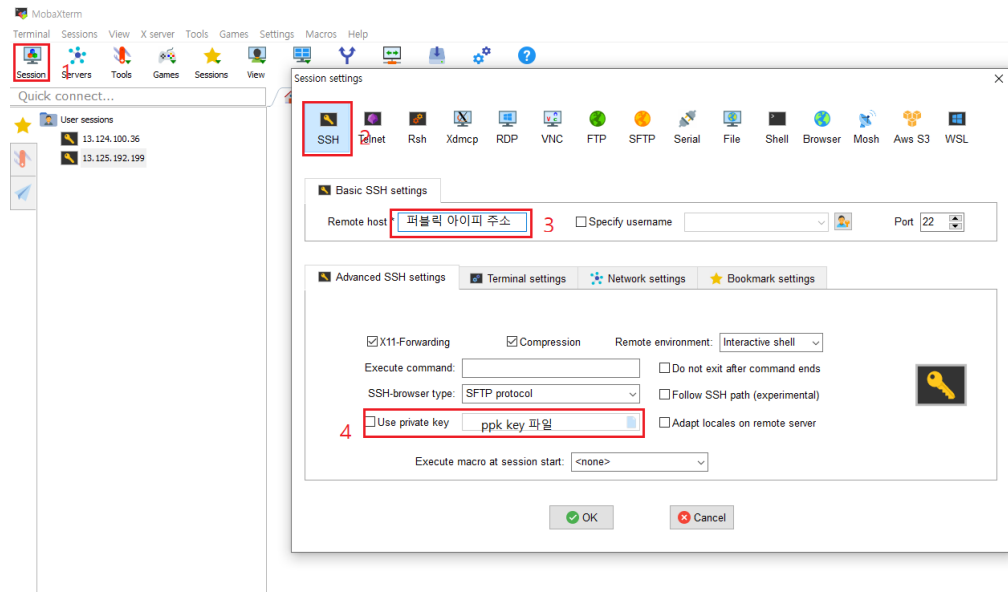
▼ 모바엑스텀

▼ 설치

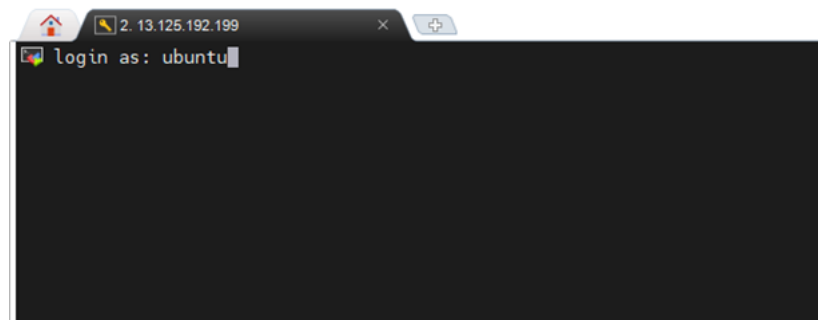
[설치페이지](#)



▼ 설정



▼ 접속



▼ ec2 시간 설정

```
$ sudo rm /etc/localtime
$ sudo ln -s /usr/share/zoneinfo/Asia/Seoul /etc/localtime
$ date
Fri Aug 8 15:48:27 KST 2014
```

▼ Java

▼ 설치

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install openjdk-11-jdk

#설치 확인
$ java -version
openjdk version "11.0.11" 2021-04-20
$ javac -version
javac 11.0.11
```

▼ 설정

```
$ sudo vim /etc/profile

# 맨 아래에 추가
...
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64 // 본인의 자바 설치 경로
export PATH=$JAVA_HOME/bin:$PATH
```

```
export CLASSPATH=$CLASSPATH:$JAVA_HOME/jre/lib/ext:$JAVA_HOME/lib/tools.jar
...

# 확인
$ source /etc/profile
$ echo $JAVA_HOME
/usr/lib/jvm/java-11-openjdk-amd64
```

▼ Docker

▼ 설치

```
# 패키지 업데이트
sudo apt-get update -y

# 기존에 있던 도커 삭제
sudo apt-get remove docker docker-engine docker.io -y

# 도커 설치
sudo apt-get install docker.io -y

# docker 서비스 실행
sudo service docker start

# 파일의 권한을 666으로 변경하여 그룹 내 다른 사용자도 접근 가능하게 변경
sudo chmod 666 /var/run/docker.sock

# ubuntu 유저를 docker 그룹에 추가
sudo usermod -aG docker $USER
sudo service docker restart

# 버전확인
docker --version

docker ps
```

▼ 설정 별 설명

```
docker run --name jenkins-docker -d -p 8000:8080 -p 8888:50000 -v /home/jenkins:/var/jenkins_home -u root jenkins/jenkins:lts
```

- **d**: detached mode, 백그라운드에서 컨테이너가 실행되게 한다.
- **p**: 서버의 9090포트와 컨테이너 내부 8080포트를 연결한다.
- **v**: 서버의 `/home/jenkins` 경로와 컨테이너 내부 `/var/jenkins_home` 경로를 마운트한다. 이것을 하는 이유는, Jenkins 설치 시 ssh 키값 생성, 저장소 참조 등을 용이하게 하기 위함입니다.
- **-name**: 실행될 컨테이너의 이름을 jenkins-docker으로 설정한다.
- **u**: 실행할 사용자를 root으로 설정한다.
- 포트는 ec2 인스턴스의 8000, 8888번 포트를 도커 컨테이너의 8080, 50000번 포트에 대응시킨다.

▼ Docker-Compose

▼ 설치

```
#설치
sudo curl -L https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

#권한
sudo chmod +x /usr/local/bin/docker-compose

#버전확인
docker-compose --version
```

▼ Nginx

▼ 설치

```
sudo apt update
sudo apt upgrade
```

```

sudo apt autoremove

# 설치
sudo apt install nginx

# 실행
sudo systemctl start nginx

# 상태 보기
sudo service status nginx
sudo apt install net-tools
netstat -lntp

#제거
sudo apt remove nginx
sudo apt purge nginx

```

▼ Docker로 설치

```

#image로 다운
docker pull nginx

#docker 실행
docker run --name front-nginx -v /home/ubuntu/compose/jenkins/workspace/spring-boot-ci-cd/frontend/build:/usr/share/nginx/html

```

▼ https 설정

```

sudo apt-get install letsencrypt -y

# nginx 중단
sudo service nginx stop

# certbot 인증서 발급 동의, 이메일 수신은 미동의
sudo certbot certonly --standalone -d j7a302.p.ssafy.io

sudo vim /etc/nginx/sites-available/default

...
# 80포트 접근 시 443 포트로 리다이렉트
server {
    if ($host = beanzido.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80 ;
    listen [::]:80 ;
    server_name beanzido.com;
    return 404; # managed by Certbot
}

# domain을 두개 연결해서 사용하고 싶다면 똑같은걸 만들기만 하면 된다.
server {
    if ($host = k7a206.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80 ;
    listen [::]:80 ;
    server_name k7a206.p.ssafy.io;
    return 404; # managed by Certbot
}

server {
    index index.html index.htm index.nginx-debian.html;
    server_name beanzido.com; # managed by Certbot
    root /home/ubuntu/compose/jenkins/workspace/release/frontend/build/;
    location / {
        root /home/ubuntu/compose/jenkins/workspace/release/frontend/build/;
        try_files $uri $uri/ @router;
    }
    location /chat-server{
        proxy_pass http://13.125.39.100:8091;
        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-Forwarded-For $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    location /keyword-server{

```

```

    proxy_pass http://13.125.39.100:8092;
}
location @router{
    rewrite ^(.+)$ /index.html last;
}

ssl_certificate /etc/letsencrypt/live/beanzido.com/fullchain.pem; # managed by Certbot
ssl_certificate_key /etc/letsencrypt/live/beanzido.com/privkey.pem; # managed by Certbot
listen 443 ssl; # managed by Certbot

}

server {
    index index.html index.htm index.nginx-debian.html;
    server_name k7a206.p.ssafy.io; # managed by Certbot
    root /home/ubuntu/compose/jenkins/workspace/front/frontend/build/;
    location / {
        root /home/ubuntu/compose/jenkins/workspace/front/frontend/build/;
        try_files $uri $uri/ @router;
    }
    location /chat-server{
        proxy_pass http://13.125.39.100:8061;
        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-Forwarded-For $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    location /keyword-server{
        proxy_pass http://13.125.39.100:8062;
    }
    location @router{
        rewrite ^(.+)$ /index.html last;
    }

    ssl_certificate /etc/letsencrypt/live/k7a206.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/k7a206.p.ssafy.io/privkey.pem; # managed by Certbot
    listen 443 ssl; # managed by Certbot

}
...

# nginx 재시작
sudo service nginx restart

```

▼ Redis

▼ 설치

```

sudo apt-get install redis-server
sudo vim /etc/redis/redis.conf

...
bind 0.0.0.0
requiredpass
...

sudo systemctl restart redis-server

```

▼ 도커 설치

```

# 이미지 다운
docker pull redis

# redis-cli 네트워크 구성
docker network create redis-net
docker network ls

# conf 작성 경로는 아무곳이나 가능 도커 -v에만 경로 잘 넣어주면 됨
...
# yes로 변경시 구동되지 않음
daemonize no
bind 0.0.0.0
protected-mode no

#변경하고자 하는 포트
port 6379

#logfile "redis.log" #이 옵션 사용시 파일로 로그가 저장되고 프롬프트는 노출되지 않음

```

```
#workingdir을 지정
#dir /data

# SECURITY
requirepass "a302ef1234"

# CLIENTS
maxclients 10000
...

# redis 서버 도커로 설치
docker run --name redis-server -p 6379:6379 -v /home/ubuntu/redis/redis.conf:/usr/local/etc/redis/redis.conf --network redis-net
docker run --rm --name dev-redis-chat -p 8070:6379 -v /home/ubuntu/compose/env/redis1.conf:/usr/local/etc/redis/redis.conf -d

# redis-cli로 redis 서버 접속
docker run -it --network redis-net --rm redis redis-cli -h redis-server

# 비밀번호 입력
auth a302ef1234
```

▼ MySQL

▼ 설치

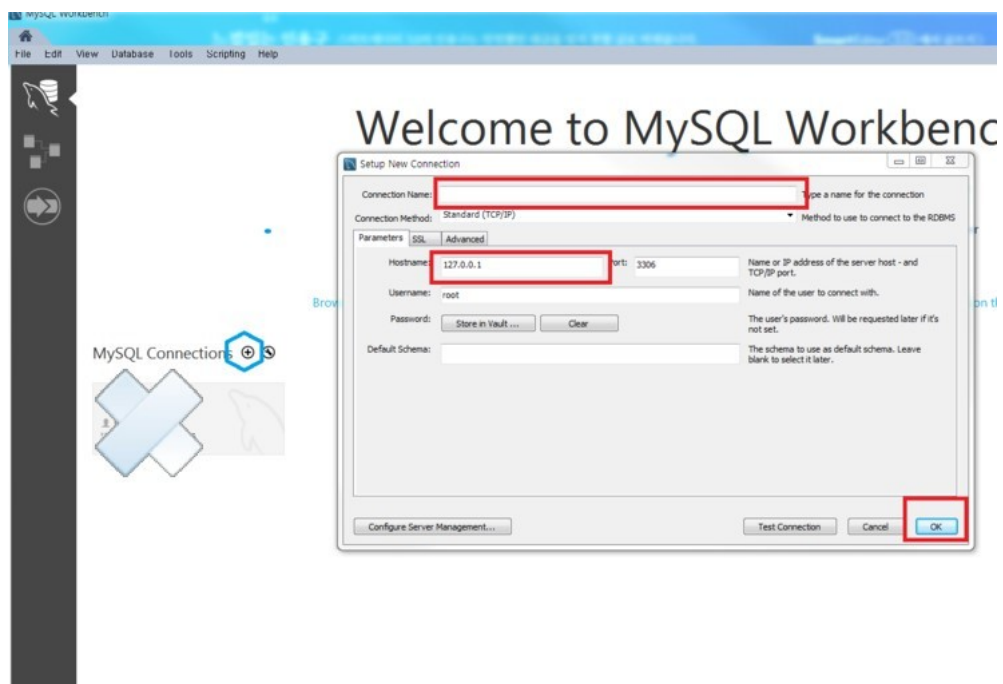
```
sudo apt-get update
sudo apt-get upgrade
sudo apt install mysql-server
dpkg -l | grep mysql-server
```

▼ 설정

```
cd /etc/mysql/mysql.conf.d
sudo vi mysqld.cnf

# ip를 0.0.0.0 으로 변경
...
bind-address 0.0.0.0
...
```

▼ Workbench 접속



▼ Jenkins

▼ compose로 설치

```
$ mkdir compose && cd compose
$ mkdir jenkins-dockerfile && cd jenkins-dockerfile
$ vim Dockerfile

...#Dockerfile
FROM jenkins/jenkins:ls

USER root
RUN apt-get update &&\
    apt-get upgrade -y &&\
    apt-get install -y openssh-client
...

$ cd ..
$ vim docker-compose.yml

...#docker-compose.yml
version: "3"
services:
  jenkins:
    container_name: jenkins-compose
    build:
      context: jenkins-dockerfile
      dockerfile: Dockerfile
    user: root
    ports:
      - 8000:8000
      - 8888:50000
    volumes:
      - /home/ubuntu/compose/jenkins:/var/jenkins_home
      - /home/ubuntu/compose/.ssh:/root/.ssh
    ...

$ mkdir jenkins
$ mkdir .ssh

#파일 빌드
docker-compose up --build -d

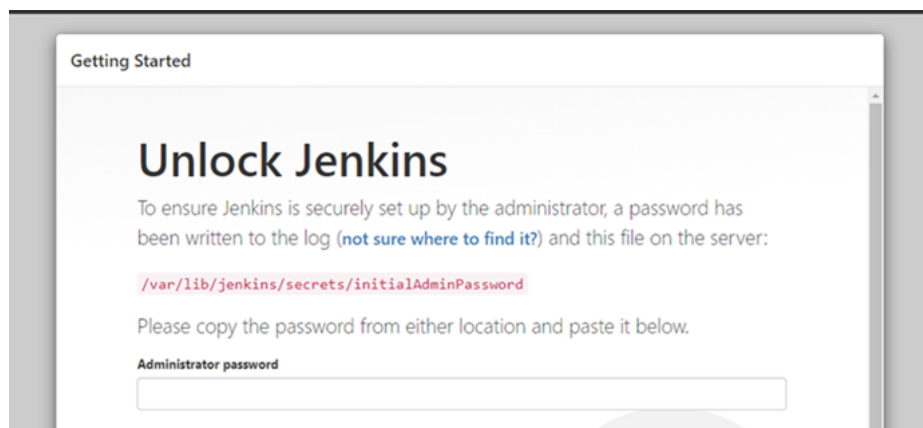
#이미지 확인
docker image ls

#실행중인 도커 확인
docker ps
```

▼ 설정

▼ 비밀번호 입력

http://[ec2-주소-입력]:8000 으로 접속 후 비밀번호 입력



```
#비밀번호 확인하기
docker logs jenkins-compose

*****
*****
*****
```

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

비밀번호임
b3a3499b0e9d4c718143f649ee7f59d9

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

▼ 기본 설정



Getting Started

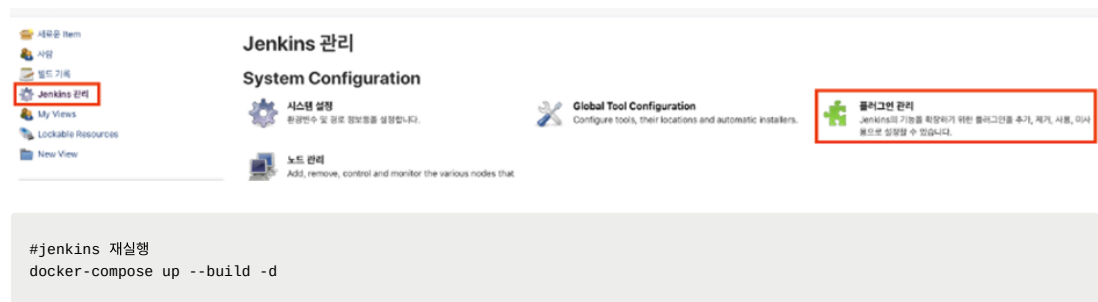
Create First Admin User

계정명: master
암호: *****
암호 확인: *****

계정 설정

URL은 설정 스킵

▼ 플러그인 추가 설치



GIT, DOCKER, NodeJS 등 플러그인 설치 후 재시작

Jenkins

Jenkins 관리

My Views

Lockable Resources

Credentials

New View

빌드 대기 목록

자동으로 새로고침

Configure Global Security

Secure Jenkins; define who is allowed to access/use the system.

Configure Credentials

Configure the credential providers and types

Global Tool Configuration

Configure tools, their locations and automatic installers.

Dashboard > Global Tool Configuration

Add NodeJS

NodeJS

Name

NodeJS 16.1.0

Required

Install automatically

Install from nodejs.org

Version

NodeJS 16.1.0

Force 32bit architecture

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail

Global npm packages to install

Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax 'packageName@version'

Global npm packages refresh hours

72

Duration, in hours, before 2 npm cache update. Note that 0 will always update npm cache

Add Installer

Delete Installer

Save

Apply

빌드 환경

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s) ?
- ☐ Provide Configuration files ?
- ☐ Add timestamps to the Console Output
- ☐ Inspect build log for published Gradle build scans

- ☒ Provide Node & npm bin/ folder to PATH

NodeJS Installation

Specify needed nodejs installation where npm installed packages will be provided to the PATH

NodeJS 16.16.0

npmrc file

- use system default -

Cache location







Default (~/.npm or %APP_DATA%\npm-cache)

- ☐ Terminate a build if it's stuck

jenkins 설정 변경해야 shell에서 npm 쓸 수 있다

▼ Item 등록


새로운 Item

 사람
 빌드 기록
 Jenkins 관리
 My Views
 Lockable Resources
 New View


Enter an item name

spring-boot-ci-cd


» Required field




Freestyle project
 이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.




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, 플레폼 특성 빌드, 기타 동등 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.



GitHub Organization
 Scans a GitHub organization (or user account) for all repositories matching some defined markers.



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

OK

▼ GIT 연동

General 소스 코드 관리 빌드 유발 빌드 환경 Build 빌드 후 조치

☐ 이 빌드는 매개변수가 있습니다
☐ 빌드 안함
☐ 필요한 경우 concurrent 빌드 실행

고급...

소스 코드 관리

☐ None
☒ **Git**

Repositories

Repository URL
 Credentials **Add**
 고급...
 Add Repository

Branches to build

Branch Specifier (blank for 'any')
 Add Branch

Repository browser (자동)

Additional Behaviours **Add**

☐ Subversion

빌드 유발

여기서 빨간 오류 나는건 아직 사용자 설정을 안해줘서 그런거니 계속 진행

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain
 Kind
 Scope
 Username
 Password
 ID
 Description

Add Cancel

추가 후 **Credentials**를 추가한 계정으로 변경

p.s. 만약 특정 브랜치에서 가져오고 싶다면 branches to build에서 master를 변경

▼ WebHook 등록

빌드 유발

- ☐ 빌드를 원격으로 유발 (예: 스크립트 사용)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://10.216.253.160:7009/project/webhook_test

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

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

고급...

General 스스 코드 관리 빌드 유발 빌드 환경 Build 빌드 후 조치

Comment (regex) for triggering a build

Jenkins please retry a build

- ☒ Enable [ci-skip]
- ☒ Ignore WIP Merge Requests

Labels that forces builds 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

Include

master

⚠ Following patterns don't match any branch in source repository: master

Exclude

- ☐ Filter branches by regex
- ☐ Filter merge request by label

Secret token

053cfd717013fa60a73d108284aff13a

Generate

맨 아래 우측 Generate 누르면 토큰 생성됨. 토큰 복사

Search settings

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.

URL

`http://[redacted]/project/webhook_test`

URL must be percent-encoded if necessary.

Secret token

`[redacted]da63f9b0ad`

Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.

Trigger

☒ **Push events**

`master`

URL is triggered by a push to the repository

깃 랩의 세팅에 있는 웹훅으로 이동 후 URL은 젠킨스 주소로 하고 복사한 토큰 넣은 다음 PushEvents는 master로 설정

Add webhook

Project Hooks (1)

`http://3.38.115.242:8000/project/spring-boot-ci-cd` Test ▾ Edit Delete

Push Events SSL Verification: enabled

Hook executed successfully: HTTP 200

추가하면 밑에 hooks에 생기는데 여기서 test설정 후 edit을 누르고 push를 했을 때 성공해야한다.

p.s. 만약 특정 브랜치에서 푸시 발생시에만 자동으로 배포하고 싶다면 2번째 사진에서 Filter branches By Name의 include에서 master를 변경

▼ Execute shell 설정

Build Steps

Execute shell ?

Command

See [the list of available environment variables](#)

```
cd BE/Server/business
chmod +x ./gradlew
./gradlew clean build
```

고급...

Add build step ▾

```

# 현재 위치는 jenkins docker에서 연결한 것을 다운 받은 폴더임
# jar로 만들기 위해 해당 spring 프로젝트가 있는 폴더로 이동해야함
cd BE/Server/eureka

# 해당 폴더에 권한이 없으므로 권한 부여
chmod +x ./gradlew

# gradlew를 빌드하여 jar 파일로 만들기
./gradlew clean build

cd ..
cd auth
chmod +x ./gradlew
./gradlew clean build

cd ..
cd business
chmod +x ./gradlew
./gradlew clean build

cd ..
cd apigateway
chmod +x ./gradlew
./gradlew clean build

```

▼ compose로 서비스 서버 설치

```

# compose 폴더로 이동
cd /home/ubuntu/compose

# 만드려면 서버의 폴더 생성 및 dockerfile 생성
# 해당 서버는 spring boot 설정임
mkdir chat && cd chat
vim Dockerfile
...
# docker의 기본 설정을 openjdk:8-jdk 이미지로 하겠다 선언
FROM openjdk:8-jdk

# jar을 실행할때 기본 설정을 다음 위치에 있는 설정 파일로 하겠다 선언
ENTRYPOINT java -Dspring.config.location=/appconfig/application-env.properties,classpath:/application.properties -jar /deploy,

# 해당 도커의 포트에서 8080을 쓰겠다
EXPOSE 8080
...

#만드려면 서버의 폴더 생성 및 dockerfile 생성
# 해당 서버는 fast api 설정임
mkdir keyword && cd keyword
vim Dockerfile
...
FROM ubuntu:latest

ENV LANG=C.UTF-8
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && \
    apt-get install -y --no-install-recommends tzdata g++ curl

# install java
RUN apt-get install -y openjdk-8-jdk
ENV JAVA_HOME="/usr/lib/jvm/java-1.8-openjdk"

# install python
RUN apt-get install -y python3-pip python3-dev
RUN cd /usr/local/bin && \
    ln -s /usr/bin/python3 python && \
    ln -s /usr/bin/pip3 pip && \
    pip install --upgrade pip

COPY ./requirements.txt /app/requirements.txt
WORKDIR /app
RUN pip3 install --upgrade -r requirements.txt
...

#docker-compose.yml 수정
cd ..
vim docker-compose.yml

...#docker-compose.yml
version: "3"

```

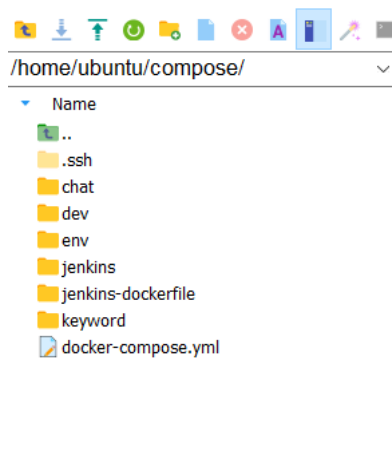
```

services:
  jenkins:
    container_name: jenkins-compose
    build:
      context: jenkins-dockerfile
      dockerfile: Dockerfile
    user: root
    ports:
      - 8000:8080
      - 8888:50000
    volumes:
      - /home/ubuntu/compose/jenkins:/var/jenkins_home
      - /home/ubuntu/compose/.ssh:/root/.ssh
  chat:
    container_name: chat-compose
    build:
      context: chat
      dockerfile: Dockerfile
    ports:
      - 8091:8080
    volumes:
      - /home/ubuntu/compose/jenkins/workspace/release/BackEnd/Spring/business/build/libs:/deploy
      - /home/ubuntu/compose/env:/appconfig
  deploy:
    resources:
      reservations:
        memory : 6g
  keyword:
    container_name: keyword-compose
    build:
      context: keyword
      dockerfile: Dockerfile
    ports:
      - 8092:8080
    volumes:
      - /home/ubuntu/compose/jenkins/workspace/release/BackEnd/Python/keyword:/app
      - /home/ubuntu/compose/keyword/.env:/app/.env
    command: uvicorn main:app --host 0.0.0.0 --port 8080 --reload
...

# 이미지를 저장하고 불러오고 싶은 경우 다음 처럼 폴더를 공유 시킨다.
volumes:
  - /tmp/a606/profile:/tmp/a606/profile

```

만약 지금까지 잘 따라 왔다면 구조는 다음 사진과 같음



▼ ssh 등록

```

# 젠킨스 도커에 접속
docker exec -it jenkins-compose bash

# 나오는 질문 모두 엔터
ssh-keygen -t rsa

# 해당 명령어 후 나온 값(키) 복사
cat /root/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCMAYEXx7IrDKHCM2xVvHgXQfN1qovkpCtmNi76cHD1rPwmPnMzHdImw8LuXDjbT7wB0xJdqpRDxtzWM66Ie0y

```

```
# 도커 나오기
exit

# 공개키 등록(맨 아래에 등록)
vim ~/.ssh/authorized_keys

# 젠킨스 도커에 접속
docker exec -it jenkins-compose bash

# 젠킨스 도커에서 aws로 접속 되는지 확인 yes 누르면된다.
#$(/sbin/ip route | awk '/default/ { print $3 }')는 공개 아이피임
ssh ubuntu@$(/sbin/ip route | awk '/default/ { print $3 }')
```

ssh 등록하기(jenkins에서 aws에 접근하여 서버를 실행 시키기 위해 필요하다)

도커 파일은 실행 할 명령어를 넣어 둔 파일이라 생각하면 된다.

젠킨스에서 먼저 명령어를 실행하고 docker-compose up 하면 도커 파일이 실행 되는 것.

```
ssh -t -t ubuntu@ip-172-26-13-177 <<EOF
cd /home/ubuntu/compose
docker-compose build --no-cache eureka
docker-compose build --no-cache auth-api
docker-compose build --no-cache business-api
docker-compose build --no-cache apigateway
exit
EOF
```

jenkins item 설정에서 shell에 다음을 추가하면 자동으로 서버를 실행한다.

만약 해당 서버에 접속이 안된다면

spring project의 properties에서

server.port = 8080

server.address = localhost

를 삭제해야한다.

▼ 환경변수

▼ 배포

▼ chat

application-env.properties

```
SERVER_IP=13.125.39.100
DB_PORT=3306
DB_NAME=chat
DB_OPTION=useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Seoul&zeroDateTimeBehavior=convertToNull&rewriteBatchedStatements=true
DB_USER=a206
DB_PASSWORD=a206ef123

#redis
REDIS_PORT_CHAT=8030
REDIS_PORT_CHAT2=8035
REDIS_HOST=13.125.39.100
REDIS_PASSWORD=a206ef123

#aws
AWS_ID=AKIA5G5U22HHXEIPADS0
AWS_PASSWORD=kLGaFVb0VRN5xEtIzG3iCdFo0tBdmnZRDC09CTS

#filter
BADWORD_PATH=/appconfig/badword.json

#rabbitmq
RABBIT_PORT=8033
RABBIT_USER=a206
RABBIT_PASSWORD=a206ef1234
EXCHANGE_NAME=beanzido.exchange
```



```
ROUTING_KEY=beanzido.oing.#
QUEUE_NAME=beanzido.queue
```

▼ keyword

.env

```
SERVER_IP=13.125.39.100
REDIS_PORT_KEYWORD=8031
REDIS_PORT_MESSAGE=8030
JSON_PATH=/app/
```

▼ front

.env

```
REACT_APP_KAKAO_JAVASCRIPT_KEY=4b9547fbda5feef9e59d1e1a8b615859
REACT_APP_SOCKET_URL=wss://beanzido.com/chat-server/map
REACT_APP_SEND_URL=beanzido.com
```

▼ 개발

▼ chat

application-env.properties

```
SERVER_IP=13.125.39.100
DB_PORT=3306
DB_NAME=dev_chat
DB_OPTION=useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Seoul&zeroDateTimeBehavior=convertToNull&rewriteBatchedStatements
DB_USER=a206
DB_PASSWORD=a206ef123

#redis
REDIS_PORT_CHAT=8070
REDIS_PORT_CHAT2=8075
REDIS_HOST=13.125.39.100
REDIS_PASSWORD=a206ef123

#aws
AWS_ID=AKIA5G5U2HHXEIPADSO
AWS_PASSWORD=kLGaFVb0vRN5xEtIzG3iCdFo00tBdmnZRDC09CTs

#filter
BADWORD_PATH=/appconfig/badword.json

#rabbitmq
RABBIT_PORT=8073
RABBIT_USER=a206
RABBIT_PASSWORD=a206ef1234
EXCHANGE_NAME=dev-beanzido.exchange
ROUTING_KEY=dev-beanzido.oing.#
QUEUE_NAME=dev-beanzido.queue
```

▼ keyword

.env

```
SERVER_IP=13.125.39.100
REDIS_PORT_KEYWORD=8071
REDIS_PORT_MESSAGE=8070
JSON_PATH=/app/
```

▼ front

.env

```
REACT_APP_KAKAO_JAVASCRIPT_KEY=115b30fd6167503b318dc5a5252f7283
REACT_APP_SOCKET_URL=wss://k7a206.p.ssafy.io/chat-server/map
REACT_APP_SEND_URL=k7a206.p.ssafy.io
```