



포팅 매뉴얼

개발 환경

[Frontend](#)

[Backend](#)

EC2

[도커 설치](#)

[SSL 인증서 발급](#)

공통

[MySQL](#)

CC24

[Front](#)

[Back](#)

[Docker-compose](#)

OMM

[Front](#)

[Back](#)

[Fastapi](#)

[Docker-compose](#)

Jenkins 설정

[Jenkins 설치 후 시작](#)

[Jenkins 안에 Docker, Docker Compose 설치](#)

[Jenkins 접속](#)

[파이프라인 생성](#)

[pipeline script 작성](#)

[빌드 트리거 등록](#)

[GitLab 프로젝트 WebHook 설정](#)

[CC24 pipeline](#)

[OMM pipeline](#)

개발 환경

Frontend

React	18.0.28
Node.js	16.18.0
VSCode	1.74.2
tailwind	3.2.7
npm	8.19.2
eslint	8.36.0
react-redux	8.0.5
vite	4.2.0

Backend

Spring Boot	2.7.9
Java	11
IntelliJ	2022.3.1
Node.js	18.15.0
FastAPI	0.95.0
Python	3.10.1
MySQL	latest
Redis	latest

EC2

도커에 컨테이너로 프론트 서버, 백 서버, 데이터베이스, 젠킨스를 띄워서 진행했습니다.

도커 설치

```
$ sudo apt-get update
$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg \
```

```
lsb-release
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
$ echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

EC2 서버에 도커를 설치합니다. 추후 docker 명령어를 편리하게 사용하기 위해 docker 그룹도 설정합니다.

```
$ usermod -aG docker $USER

$ groups $USER

$ service docker restart
```

SSL 인증서 발급

https를 적용하기 위해 ssl 인증서를 발급받아야 합니다. ssl 인증서는 letsencrypt를 사용하여 발급했습니다.

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

설치를 실패했다면 먼저 `apt update` 명령어를 통해서 운영체제에서 사용 가능한 패키지들과 그 버전에 대한 정보를 업데이트해야 합니다.

letsencrypt를 설치한 다음에는 해당 도메인에 ssl 인증서를 발급받아야 합니다.

```
$ letsencrypt certonly --standalone -d {도메인}
```

해당 명령어 입력 후, 이메일(선택), 서비스 이용 동의(필수), 정보 수집(선택) 등을 하고 나면 .pem 키 발급이 완료됩니다. 발급된 키는 `/etc/letsencrypt/live/{도메인}/` 에서 확인할 수 있습니다.

공통

MySQL

각 EC2에 mysql을 설치합니다. mysql은 컨테이너로 띄웠습니다.

```
$ docker run --name mysql -e MYSQL_ROOT_PASSWORD={password} -v mysql-volume:/var/lib/mysql-p 3306:3306 -d mysql:latest
```

mysql로 접속하여 계정과 데이터베이스를 설정합니다.

```
$ docker exec -it mysql bash

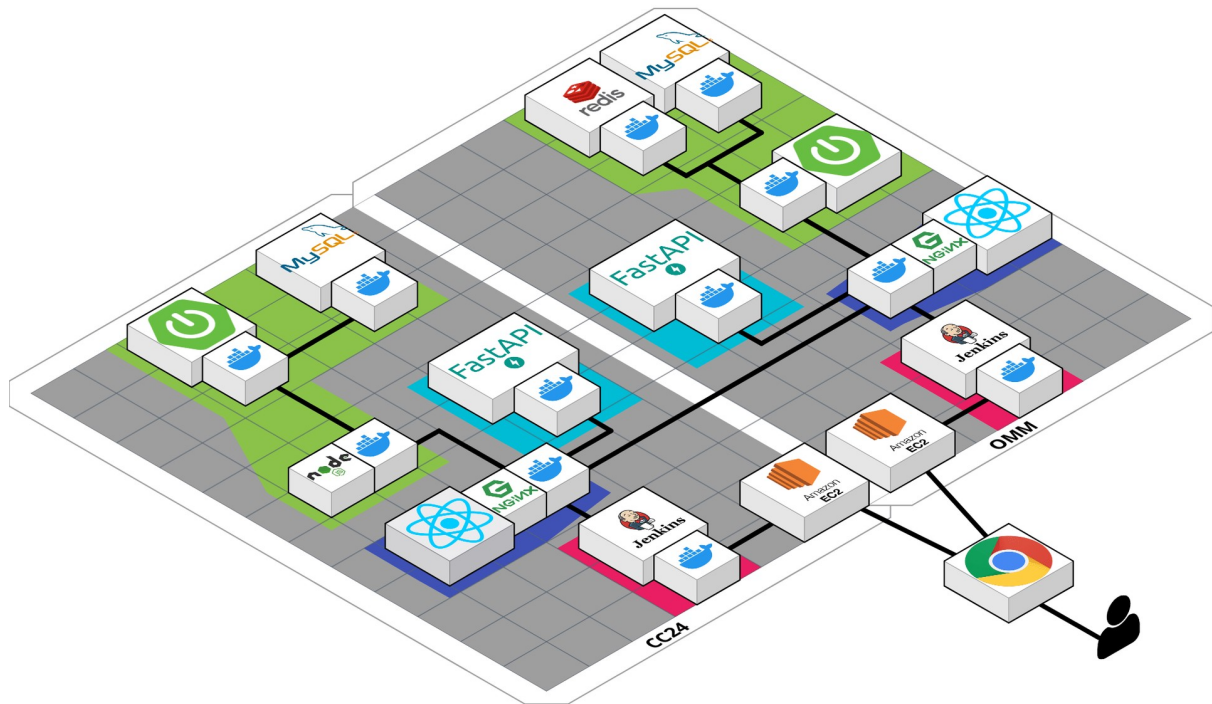
$ mysql -uroot -p
$ mysql> create user {계정아이디}@"%" identified by {비밀번호};
$ mysql> create database {DB명};
$ mysql> GRANT ALL privileges ON {DB명}.* TO {계정아이디}@"%";
```

백엔드 서버에서 mysql을 사용할 수 있도록 네트워크를 설정합니다.

```
$ docker network create {네트워크이름}
$ docker network connect {네트워크이름} mysql
```

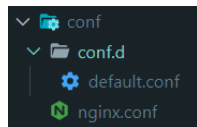
CC24

먼저 CC24 사이트입니다. CC24의 아키텍처 구상도입니다.



Front

프론트는 React를 이용하여 개발했습니다. React는 Nginx를 이용하여 배포했습니다.



```
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    client_max_body_size 50M;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile on;
    #tcp_nopush on;

    keepalive_timeout 65;

    #gzip on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*.conf;
```

```
server_names_hash_bucket_size 64;
}
```

nginx.conf 파일입니다. nginx 관련 설정을 할 수 있습니다. `default.conf`를 사용할 수 있도록 설정 파일 경로를 include 했습니다.

```
upstream node {
    server node:4424;
}

upstream spring {
    server spring:3324;
}

upstream fast {
    server fast:8000;
}

server {
    listen      80;
    server_name {도메인};
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name {도메인};

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /api/spring {
        proxy_pass http://spring/api/spring;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /api/node {
        proxy_pass http://node/api/node;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /api/fast {
        proxy_pass http://fast/api/fast;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    ssl_certificate /etc/letsencrypt/live/{도메인}/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/{도메인}/privkey.pem;
}
```

default.conf 파일입니다. `upstream` 변수를 통해 `server` 설정에서 nginx가 받아들인 요청을 어떤 서버로 흘려보내 줄 것인지 결정할 수 있습니다. 컨테이너의 이름과 포트를 지정합니다. 그리고 `server` 블록에서 특정 location 요청에 대한 `proxy_pass`를 설정합니다.

```
# https://hub.docker.com/_/node
# alpine, mininum => 경량화된 이미지
# BUILDER
# node 이미지 가져오기
FROM node:16-alpine AS builder

# 현재 디렉토리 설정
WORKDIR /app

COPY package*.json ./

# npm install로 node_modules 설치
RUN npm install

# 현재 경로의 파일 모두 복사
COPY . ./

# 빌드
```

```

RUN npm run build

# https://hub.docker.com/_/nginx
# RUNNING
FROM nginx:1.23.4-alpine

# nginx에서 필요한 설정파일 옮기
COPY ./conf /etc/nginx/

# builder에서 빌드한 바이너리를 실행할 이미지로 전달해주기 위해 copy --from 옵션을 사용하여 실행 이미지로 전달한다
COPY --from=builder /app/build /usr/share/nginx/html

```

도커 파일입니다.

해당 도커 파일은 `docker build -t front .` 명령어를 통해 이미지로 빌드하고, `docker run --name front -p 80:80 -p 443:443 -d --network cc24 front` 명령어를 통해서 이메일을 컨테이너로 실행할 수 있습니다.

Back

백은 spring boot, nodejs, fastapi를 이용하여 개발했습니다.

Spring boot

```

# https://hub.docker.com/_/gradle
# BUILDER
FROM gradle:8.0.2-jdk11-alpine as builder
WORKDIR /build

# 그래들 파일이 변경되었을 때만 새롭게 의존패키지 다운로드 받게함
COPY build.gradle settings.gradle /build/
RUN gradle build -x test --parallel --continue > /dev/null 2>&1 || true

# 애플리케이션 빌드
COPY . /build
RUN gradle clean build -x test --parallel

# RUNNING
# https://hub.docker.com/_/openjdk
FROM openjdk:11-slim

# 빌더 이미지에서 jar 파일 복사
COPY --from=builder /build/build/libs/{projectname}-0.0.1-SNAPSHOT.jar .

EXPOSE 3324

# 컨테이너 시작할 때 실행되는 명령
ENTRYPOINT [
    "java",
    "-jar",
    "-Djava.security.egd=file:/dev/./urandom",
    "-Dsun.net.inetaddr.ttl=0",
    "{projectname}-0.0.1-SNAPSHOT.jar"
]

```

도커파일입니다.

해당 도커 파일은 `docker build -t spring .` 명령어를 통해 이미지로 빌드하고, `docker run --name spring -p 3324:3324 -d --network cc24 spring` 명령어를 통해서 이메일을 컨테이너로 실행할 수 있습니다.

```

server:
  servlet:
    context-path: /api/spring
    encoding:
      charset: UTF-8
      enabled: true
      force: true
    port: 3324

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://mysql:3306/cc24?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC&characterEncoding=UTF-8
    username: {mysql 계정아이디}

```

```
password: {mysql 비밀번호}

jpa:
  database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
  hibernate:
    ddl-auto: update
  properties:
    hibernate:
      format_sql: true
```

application.yml 파일입니다.

Nodejs

```
INFURA_API_KEY={인플라 아이디}
ISSUER_ID=0x0307f4d8571dd76c61c0b5c8eefb8057e849cb70e201c1b0d331ee976d9537b73b
ISSUER_PRIVATE_KEY=0x8a734dc20fa2e661d118bd7b60b2fb9356d961b42067a475572bb60e0af941d7
STORAGE_BUCKET={s3 버킷 주소}
```

firebase admin 키를 발급받아야 합니다.

[Firebase] firebase admin sdk 세팅하기

Firebase는 간단한 database를 포함한 백엔드를 제공하는 서비스이다. 여러가지 기능이 있지만, 그 중 firebase Authentication를 활용하면 로그인 구현을 굉장히 편하게 할 수 있다. 로그인 세션, 토큰 등을 관리해주며 소셜 로그인

의 U까지
<https://velog.io/@zero-black/Firebase-firebase-admin-sdk-세팅하기>



```
#어떤 이미지로부터 새로운 이미지를 생성할지를 지정
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . ./

#환경변수 NODE_ENV 의 값을 development 로 설정
ENV NODE_ENV development

EXPOSE 4424

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

도커 파일입니다.

해당 도커 파일은 `docker build -t node .` 명령어를 통해 이미지로 빌드하고, `docker run --name node-p 4424:4424 -d --network cc24 node` 명령어를 통해서 이메일을 컨테이너로 실행할 수 있습니다.

Fastapi

```
FROM python:latest

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

도커 파일입니다.

해당 도커 파일은 `docker build -t fast .` 명령어를 통해 이미지로 빌드하고, `docker run --name fast -p 8000:8000 -d --network cc24 fast` 명령어를 통해서 이메일을 컨테이너로 실행할 수 있습니다.

Docker-compose

```
version: "3.7"

services:
  spring:
    container_name: spring
    build:
      context: ./spring
      dockerfile: Dockerfile
    ports:
      - 3324:3324
    networks:
      - cc24
  node:
    container_name: node
    env_file:
      - ./nodejs/.env
    build:
      context: ./nodejs
      dockerfile: Dockerfile
    ports:
      - 4424:4424
    networks:
      - cc24
  fast:
    container_name: fast
    build:
      context: ./fastapi
      dockerfile: Dockerfile
    ports:
      - 8000:8000
    networks:
      - cc24
  front:
    container_name: front
    build:
      context: ./front
      dockerfile: Dockerfile
    volumes:
      - /etc/letsencrypt/:/etc/letsencrypt/
    ports:
      - 80:80
      - 443:443
    networks:
      - cc24
networks:
  cc24:
    external:
      name: cc24
```

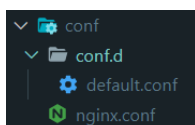
도커 컴포즈 파일을 사용하여 쉽게 빌드할 수 있습니다.

`docker-compose up -d` 명령어를 통해 컨테이너를 생성할 수 있습니다.

OMM

OMM 포팅 매뉴얼은 CC24 포팅 매뉴얼과 동일하기 때문에 설명을 생략하겠습니다.

Front



```

user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    client_max_body_size 50M;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile on;
    #tcp_nopush on;

    keepalive_timeout 65;

    #gzip on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*.conf;
    server_names_hash_bucket_size 64;
}

```

```

upstream back {
    server back:5000;
}

upstream fast {
    server fast:8000;
}

server {
    listen 80;
    server_name {도메인};
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name {도메인};

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://back/api;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    # 소켓 관련 설정
    location /api/chat {
        proxy_pass http://back/api/chat;
        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;
    }

    # 소켓 관련 설정
    location /api/matching {
        proxy_pass http://back/api/matching;
        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;
    }
}

```



```

    }

    location /api/fast {
        proxy_pass http://fast/api/fast;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    ssl_certificate /etc/letsencrypt/live/j8c2081.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j8c2081.p.ssafy.io/privkey.pem;
}

```

```

# BUILDER
FROM node:16-alpine AS builder

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . ./

ENV NODE_OPTIONS=--max_old_space_size=2048

RUN npm run build

FROM nginx:1.23.4-alpine

COPY ./conf /etc/nginx/

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


```

환경 변수에 카카오 맵 API 키를 발급받아 작성합니다.

1. 카카오 API 발급

Kakao Developers

카카오 API를 활용하여 다양한 어플리케이션을 개발해보세요. 카카오 로그인, 메시지 보내기, 친구 API, 인공지능 API 등을 제공합니다.

 <https://developers.kakao.com/>

kakao developers

2. 개발자 등록 및 앱 생성

3. 웹 플랫폼 추가

- 앱 선택 – [플랫폼] – [Web 플랫폼 등록] – 사이트 도메인 등록

2. 카카오맵 API를 환경변수 파일(.env)에 넣기

Back

```

# BUILDER
FROM gradle:8.0.2-jdk11-alpine as builder
WORKDIR /build

COPY build.gradle settings.gradle /build/
RUN gradle build -x test --parallel --continue > /dev/null 2>&1 || true

COPY . /build
RUN gradle clean build -x test --parallel

# RUNNING
FROM openjdk:11-slim

COPY --from=builder /build/build/libs/back-0.0.1-SNAPSHOT.jar .

```

EXPOSE 3324

```
ENTRYPOINT [
    "java",
    "-jar",
    "-Djava.security.egd=file:/dev/./urandom",
    "-Dsun.net.inetaddr.ttl=0",
    "back-0.0.1-SNAPSHOT.jar"
]
```

```
server:
  servlet:
    context-path: /api
    encoding:
      charset: UTF-8
      enabled: true
      force: true
    port: 5000

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://mysql:3306/omm?serverTimezone=UTC&characterEncoding=UTF-8
    username: {mysql 계정아이디}
    password: {mysql 비밀번호}

  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true

  redis:
    host: redis
    port: 6379
    password: {redis 비밀번호}

  jwt:
    header: Authorization
    secret: {서버 비밀키}
    token-validity-in-seconds: 86400

  url:
    fastapi: http://172.17.0.1:8000/api/fast
    ommfront: https://{도메인}
    cc24front: https://{도메인}
    nodeapi: https://{도메인}
```

Fastapi

```
FROM python:latest

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Docker-compose

```
version: '3.7'

services:
  redis:
    image: redis
    container_name: redis
```

```

ports:
  - 6379:6379
command: redis-server --requirepass rladbsal123@@ --port 6379
networks:
  - omm
back:
  container_name: back
  build:
    context: ./back
    dockerfile: Dockerfile
  ports:
    - 5000:5000
  depends_on:
    - redis
  networks:
    - omm
fast:
  container_name: fast
  build:
    context: ./fastapi
    dockerfile: Dockerfile
  ports:
    - 8000:8000
  networks:
    - omm
front:
  container_name: front
  build:
    context: ./front
    dockerfile: Dockerfile
  volumes:
    - /etc/letsencrypt:/etc/letsencrypt/
  ports:
    - 80:80
    - 443:443
  networks:
    - omm
networks:
  omm:
    external:
      name: omm

```

Jenkins 설정

Jenkins 설치 후 시작

```

docker run --name jenkins -d -p 5555:8080 -v /home/ubuntu/volumes/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock
docker start jenkins

```

Jenkins 안에 Docker, Docker Compose 설치

```

//Jenkins 접속
docker exec -it jenkins bash

//Jenkins 안에 Docker 설치
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

//Docker Compose 설치
sudo curl -L "https://github.com/docker/compose/releases/download/{VERSION}/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/
sudo chmod +x /usr/local/bin/docker-compose

```

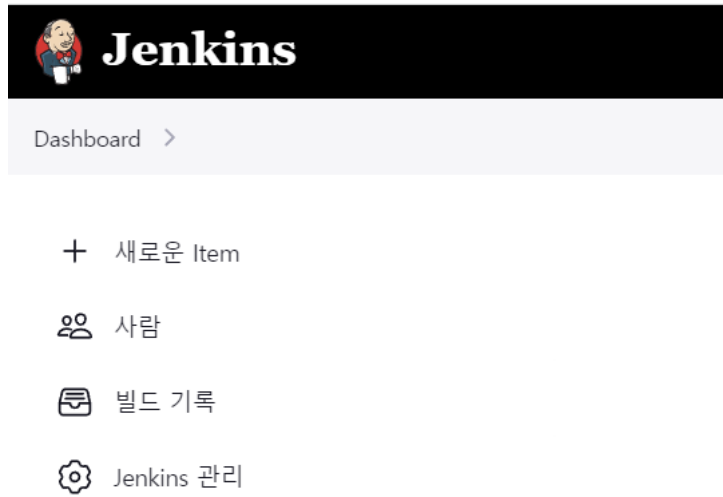
Jenkins 접속

- CC24 : j8c208.p.ssafy.io:5555

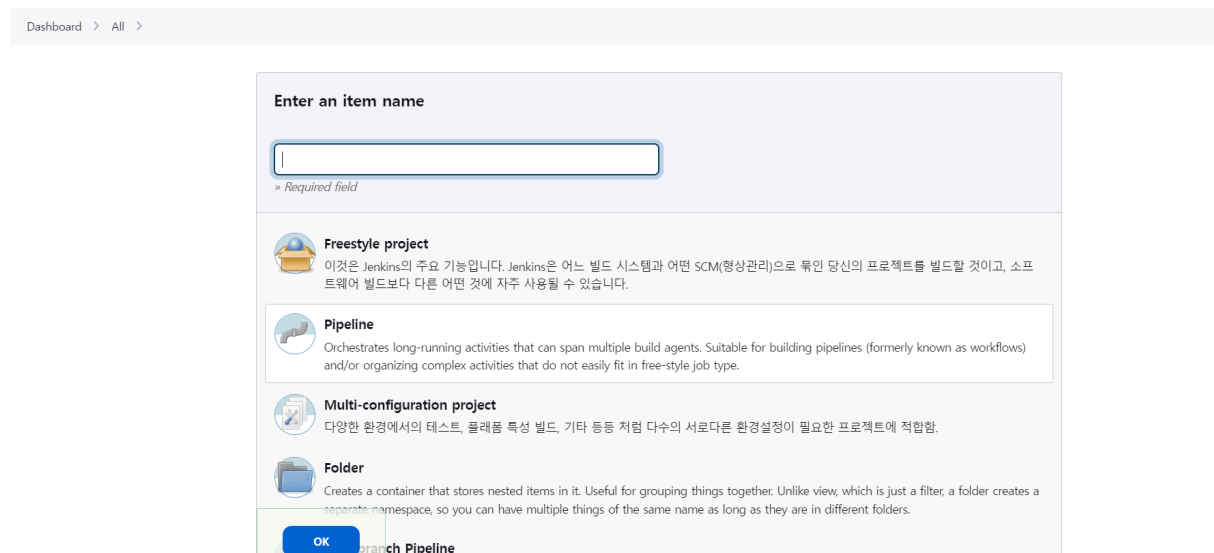
- `OMM : j8c2081.p.ssafy.io:5555`

파이프라인 생성

- 새로운 Item 클릭



- item 이름 입력 후 Pipeline 선택 후 OK



pipeline script 작성

Configure

- General
- Advanced Project Options
- Pipeline**

Definition

Pipeline script

Script ?

1

try sample Pipeline...

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

저장 **Apply**

Dashboard > item이름 > 구성

빌드 트리거 등록

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://j8c208.p.ssafy.io:5555/project/cc24> ?

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

On push to source or target branch

- ☒ Approved Merge Requests (EE-only)
- ☒ Comments

Merge 이벤트에 트리거 이벤트 등록

- 고급 > Secret Token > Generate

그룹 ^

- ☒ Enable [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 ?
- ☐ Filter branches by regex ?
- ☐ Filter merge request by label

Secret token ?

Generate

GitLab 프로젝트 WebHook 설정

s08-blockchain-contract-sub2 > S08P22C208 > Webhook Settings > Webhook

Q Search page

Webhook

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

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

Secret token

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

Trigger

- ☒ **Push events**

Push to the repository.
- ☐ **Tag push events**
A new tag is pushed to the repository.

- 위에서 생성한 토큰 입력

Project Hooks (2)

http://j8c208.p.ssafy.io:5555/project/cc24 Push Events Merge Requests Events SSL Verification: enabled	Test ▾ Edit Delete
http://j8c2081.p.ssafy.io:5555/project/omm Push Events Merge Requests Events SSL Verification: enabled	Test ▾ Edit Delete

CC24 pipeline

```
pipeline {
    agent any

    environment {
        BUILD_DIR = "CC24"
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'develop', credentialsId: 'cc24_pipeline', url: 'https://lab.ssafy.com/s08-blockchain-contract-sub2/S08P22C
            }
        }

        stage('Build') {
            steps {
                sh 'pwd'
                dir('${BUILD_DIR}/spring/src/main/resources') {
                    sh 'cp /var/jenkins_home/initfile/cc24/application.yml ./application.yml'
                }
                sh 'cp /var/jenkins_home/initfile/cc24/.env ./${BUILD_DIR}/nodejs/.env'
                sh 'cp /var/jenkins_home/initfile/cc24/cc24-3d5b1-firebase-adminsdk-flw87-a3579a4c47.json ./${BUILD_DIR}/nodejs/cc24-3
            }
        }

        stage('Deploy End') {
            steps {
                dir('CC24') {
                    sh 'docker-compose up --build -d'
                }
                sh 'docker image prune --all --force'
                mattermostSend color: '#fc1100', message: "CC24 Deploy End! (${env.JOB_NAME}) #(${env.BUILD_NUMBER}) (<${env.BUILD_URL
            }
        }
    }
}
```

OMM pipeline

```
pipeline {
    agent any

    environment {
        BUILD_DIR = "OMM"
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'develop', credentialsId: 'omm_pipeline', url: 'https://lab.ssafy.com/s08-blockchain-contract-sub2/S08P22C
            }
        }

        stage('Build') {
            steps {
                sh 'pwd'
                dir("${BUILD_DIR}/back/src/main/resources") {
                    sh 'cp /var/jenkins_home/initfile/omm/application.yml ./application.yml'
                }
                sh 'cp /var/jenkins_home/initfile/omm/.env ${BUILD_DIR}/front/.env'
                dir("${BUILD_DIR}/front") {
                    sh '''
                        sed -i 's/win32-x64/linux-x64/g' package.json
                    '''
                }
            }
        }

        stage('Deploy End') {
            steps {
                dir('OMM') {
                    sh 'docker-compose up --build -d'
                }
                sh 'docker image prune --all --force'
                mattermostSend color: '#32a852', message: "OMM Deploy End! (${env.JOB_NAME}) #(${env.BUILD_NUMBER}) (<${env.BUILD_URL
            }
        }
    }
}
```

} }