



Porting Manual : Chat Composer

자율 A504 Team Chat...주희PT

0. 목차

0. 목차

I. 개발 환경

1. 프로젝트 기술 스택

2. ERD

3. Server Directory 구조

API Server

GPU Server

4. 설정 파일

>> API Server <<

Nginx, Jenkins, MySQL, Redis, RabbitMQ: [docker-compose.yml](#)

Nginx : [default.conf](#)

Jenkins-Backend : [Jenkinsfile](#)

Jenkins-Frontend : [Jenkinsfile](#)

Backend : [Dockerfile](#)

Frontend : [Dockerfile](#)

>> GPU Server <<

Nginx, Flask, Fast API: [docker-compose.yml](#)

Nginx : [nginx.conf](#)

Gateway(Flask) : [Dockerfile](#)

Riffusion(Flask): [Dockerfile](#)

Diffusion(Fast API): [Dockerfile](#)

5. 환경 변수 파일

Backend : [application.yml](#)

Frontend : [.env](#)

II. AI 모델 테스트

1. Cuda 설치

2. 가상환경

MiniConda 설치

3. Riffusion

가상환경 시작

4. Diffusion

가상환경 시작

III. 빌드 및 배포

1. docker 설치

2. Docker Compose 설치

3. SSL 인증

certbot docker container

4. DB 및 Infra 배포

[Nginx, Jenkins, MySQL, Redis, RabbitMQ](#)

[Dockerfile 저장](#)

[방법 1 \) 수동 배포](#)

[방법 2 \) Jenkins 사용](#)

[IV. 외부 서비스](#)

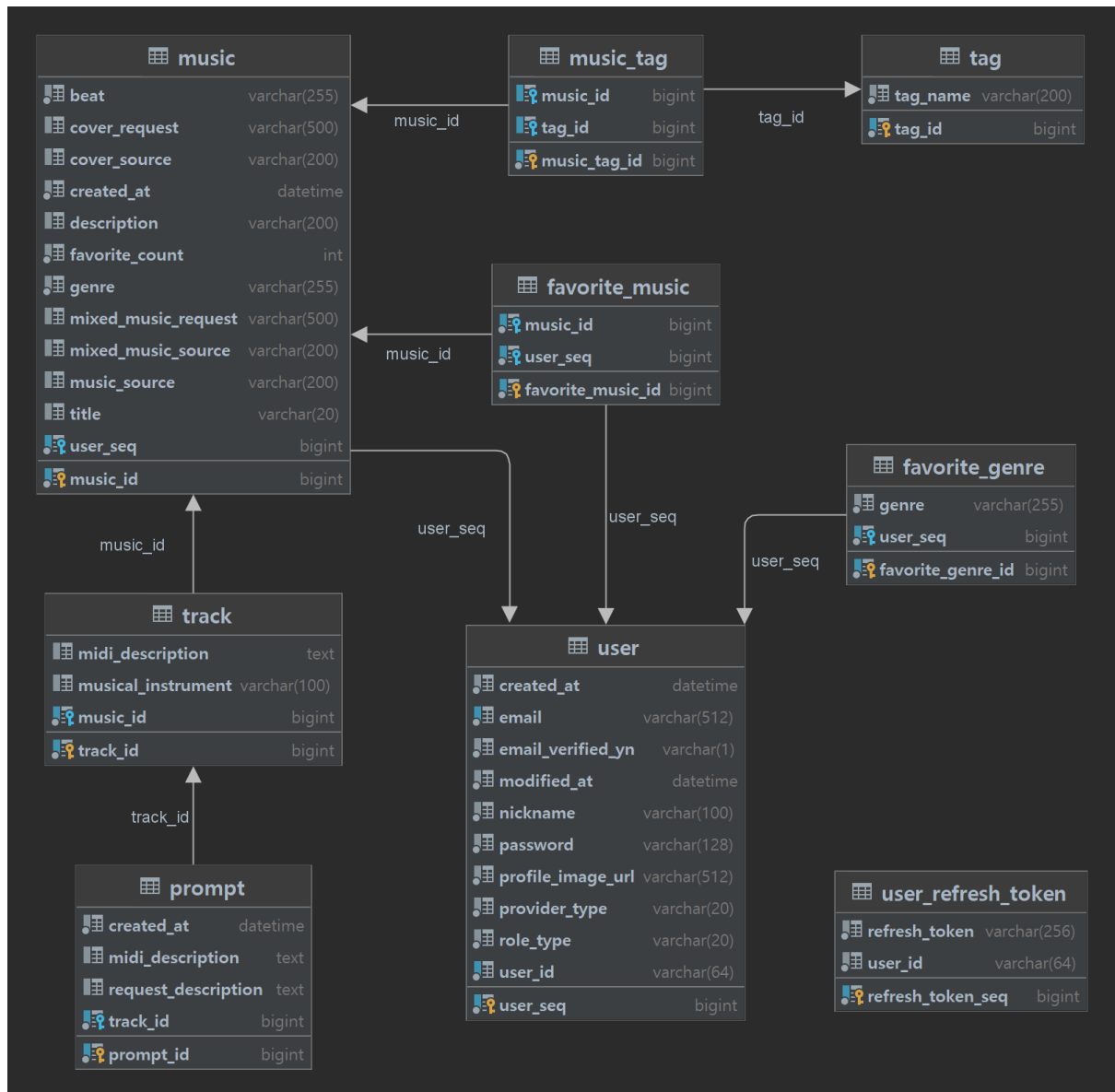
[Google Login](#)

I. 개발 환경

1. 프로젝트 기술 스택

- CI/CD
 - AWS EC2
 - Ubuntu Ubuntu 20.04 LTS (GNU/Linux 5.4.0-1018-aws x86_64)
 - Docker 23.0.4
 - Jenkins 2.402
 - Nginx 1.23.4
- Backend
 - IntelliJ IDEA 2022.3.1
 - JVM zulu-11
 - SpringBoot Gradle 2.7.11
 - Spring Security 5.7.8
 - JWT 0.11.5
 - Swagger v3
 - QueryDSL 5.0.0
 - RabbitMQ 3.11.14
 - Flask
 - Fast API
- Frontend
 - Visual Studio Code 1.77.1
 - Node.js 18.12.1
 - TypeScript 4.7.4
 - React 18.2.0
 - Next.js 13.4.2
 - Recoil 0.7.7
 - React-Query
 - tone.js 14.7.77
 - tailwind css 3.3.1
- Database
 - MySQL 8.0.33
 - Redis 7.0.11

2. ERD



3. Server Directory 구조

API Server

```

ubuntu
├── jenkins
│   └── workspace
│       ├── chatcomposer_backend
│       └── chatcomposer_frontend
├── mysql
│   ├── data
│   └── nginx
│       ├── conf.d
│       └── default.conf
├── rabbitmq
│   ├── data
│   ├── log
│   └── redis
└── docker-compose.yml
  
```

GPU Server

```

ubuntu
├── gateway
  
```

```

| L Dockerfile
| miniconda3
| models
| L diffusion
| L riffusion
| nginx
| L conf.d
| L nginx.conf
| riffusion
| L Dockerfile
| stable_diffusion
| L Dockerfile
| L docker-compose.yml

```

4. 설정 파일

>> API Server <<

Nginx, Jenkins, MySQL, Redis, RabbitMQ: docker-compose.yml

~/docker-compose.yml

```

version: '3'

services:
  nginx:
    image: nginx
    container_name: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - ~/nginx/conf.d:/etc/nginx/conf.d
      - /etc/letsencrypt:/etc/letsencrypt
    command:
      ['nginx', '-g', 'daemon off;']

  mysql:
    image: mysql
    container_name: mysql
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: [mysql root 비밀번호]
    volumes:
      - ~/mysql/data:/var/lib/mysql
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    restart : always
    #restart: unless-stopped

  redis:
    image: redis
    container_name: redis
    ports:
      - 6379:6379
    command: /bin/sh -c "redis-server --requirepass [redis 비밀번호]"
    volumes:
      - ~/redis/data:/data
      - ~/redis/conf/redis.conf:/usr/local/etc/redis/redis.conf
    environment:
      - TZ=Asia/Seoul
    restart: always
    network_mode: host

  rabbitmq:
    image: rabbitmq:3-management-alpine
    container_name: rabbitmq
    volumes:
      - ~/rabbitmq/data:/var/lib/rabbitmq/
      - ~/rabbitmq/log:/var/log/rabbitmq
    ports:
      - "4002:5672"
      - "4003:15672"
    environment:
      RABBITMQ_ERLANG_COOKIE: "RabbitMQ-My-Cookies"
      RABBITMQ_DEFAULT_USER: "admin"
      RABBITMQ_DEFAULT_PASS: "[RabbitMQ 비밀번호]"
    user: "1000:1000"

```

```
jenkins:
  image: jenkins/jenkins
  container_name: jenkins
  volumes:
    - /usr/bin/docker:/usr/bin/docker
    - /var/run/docker.sock:/var/run/docker.sock
    - ~/jenkins:/var/jenkins_home
  ports:
    - 8095:8080
  privileged: true
  user: root
  restart: unless-stopped
```

Nginx : default.conf

볼륨 설정이 ~/nginx/conf.d/etc/nginx/conf.d 와 같이 되어있고

/etc/nginx/nginx.conf 파일에 include /etc/nginx/conf.d/*.conf 설정됨

~/nginx/conf.d/default.conf

```
server {
    listen 80;
    listen [::]:80;
    server_name [서비스 도메인];

    # Redirect to https
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name [서비스 도메인];
    client_max_body_size 10M;

    ssl_certificate /etc/letsencrypt/live/[서비스 도메인]/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/[서비스 도메인]/privkey.pem;
    ssl_prefer_server_ciphers on;

    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;
    proxy_set_header X-Forwarded-Proto https;
    #proxy_set_header Upgrade $http_upgrade;
    #proxy_set_header Connection "Upgrade";
    proxy_headers_hash_bucket_size 512;
    proxy_redirect off;

    location / {
        proxy_pass http://[서비스 도메인]:3000/;
    }
    location /api/ {
        proxy_connect_timeout 3000;
        proxy_send_timeout 3000;
        proxy_read_timeout 3000;
        send_timeout 3000;
        proxy_pass http://[서비스 도메인]:8080/;
        proxy_set_header X-Forwarded-Prefix /api;
    }
    location /api/chatgpt {
        proxy_connect_timeout 3000;
        proxy_send_timeout 3000;
        proxy_read_timeout 3000;
        send_timeout 3000;
        proxy_pass http://[서비스 도메인]:3000/api/chatgpt;
    }
    location /api/papago {
        proxy_connect_timeout 3000;
        proxy_send_timeout 3000;
        proxy_read_timeout 3000;
        send_timeout 3000;
        proxy_pass http://[서비스 도메인]:3000/api/papago;
    }
}
```

Jenkins-Backend : Jenkinsfile

Gitlab backend/ [Jenkinsfile](#)

```
pipeline {
  agent any

  stages {
    stage('application.yml load'){
      steps {
        dir("backend") {
          withCredentials([file(credentialsId: 'back-credential', variable: 'configFile')]){
            script {
              sh 'cp $configFile ./src/main/resources/application.yml'
            }
          }
        }
      }
    }

    stage('Gradle Build') {
      steps {
        dir("backend") {
          sh 'chmod +x gradlew'
          sh './gradlew clean build -x test'
        }
      }
    }

    stage('Backend Docker Build') {
      steps {
        dir("backend") {
          sh 'docker build -t chco-backend:latest .'
        }
      }
    }

    stage('Backend Deploy') {
      steps {
        sh 'docker rm -f backend'
        sh 'docker run -d --name backend -p 8080:8080 -u root chco-backend:latest'
      }
    }

    stage('Finish') {
      steps {
        sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
      }
    }
  }
}
```

Jenkins-Frontend : [Jenkinsfile](#)

Gitlab frontend/ [Jenkinsfile](#)

```
pipeline {
  agent any

  tools {
    nodejs 'node18'
  }

  stages {
    stage('.env load'){
      steps {
        dir("frontend") {
          withCredentials([file(credentialsId: 'front-credential', variable: 'configFile')]){
            script {
              sh 'cp $configFile ./env'
            }
          }
        }
      }
    }

    stage('Frontend Docker Build') {
      steps {
        dir("frontend") {
          sh 'docker build -t chco-frontend:latest .'
        }
      }
    }
  }
}
```

```

    stage('Frontend Deploy') {
        steps {
            sh 'docker rm -f frontend'
            sh 'docker run -d --name frontend -p 3000:3000 -u root chco-frontend:latest'
        }
    }

    stage('Finish') {
        steps {
            sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
        }
    }
}
}

```

Backend : Dockerfile

Gitlab backend/ [dockerfile](#)

```

FROM azul/zulu-openjdk:11

ARG JAR_FILE=build/libs/ChatComposer-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar

EXPOSE 8080
ENV TERM=xterm

CMD ["java", "-jar", "app.jar"]

```

Frontend : Dockerfile

Gitlab frontend/ [Dockerfile](#)

```

FROM node:18.12.1

WORKDIR /app

COPY package*.json yarn.lock ./

ENV NODE_ENV=production

RUN yarn install --frozen-lockfile
COPY . .
RUN yarn build

EXPOSE 3000

CMD ["yarn", "start"]

```

>> GPU Server <<

Nginx, Flask, Fast API: [docker-compose.yml](#)

```

version: '3'

services:
  nginx:
    image: nginx
    container_name: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - ~/nginx/conf.d:/etc/nginx/conf.d
      - /etc/letsencrypt:/etc/letsencrypt
    command:
      ['nginx', '-g', 'daemon off;']

  gateway:
    build: ./gateway
    image: gateway
    container_name: gateway
    volumes:
      - ./gateway:/app/
    ports:
      - 3000:3000

```

```
diffusion:
  build: ./stable_diffusion
  image: diffusion
  container_name: diffusion
  runtime: nvidia
  devices:
    - /dev/nvidia-modeset:/dev/nvidia-modeset
    - /dev/nvidia-uvmm:/dev/nvidia-uvmm
    - /dev/nvidia-uvmm-tools:/dev/nvidia-uvmm-tools
    - /dev/nvidia0:/dev/nvidia0
    - /dev/nvidiaactl:/dev/nvidiaactl
  volumes:
    - ./stable_diffusion:/app/
    - ~/models/diffusion:/app/model
  ports:
    - 8885:8885
riffusion:
  build: ./riffusion
  image: riffusion
  container_name: riffusion
  runtime: nvidia
  devices:
    - /dev/nvidia-modeset:/dev/nvidia-modeset
    - /dev/nvidia-uvmm:/dev/nvidia-uvmm
    - /dev/nvidia-uvmm-tools:/dev/nvidia-uvmm-tools
    - /dev/nvidia0:/dev/nvidia0
    - /dev/nvidiaactl:/dev/nvidiaactl
  volumes:
    - ./riffusion:/app/
    - ~/models/riffusion:/app/model
  ports:
    - 3313:3313
```

Nginx : nginx.conf

- 볼륨 설정이 ~/nginx/conf.d/etc/nginx/conf.d 와 같이 되어있고
/etc/nginx/nginx.conf 파일에 include /etc/nginx/conf.d/*.conf 설정됨

~/nginx/conf.d/ `nginx.conf`

```
server {
    listen 80;
    listen [::]:80;
    server_name [GPU 서버 도메인];

    # Redirect to https
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name [GPU 서버 도메인];

    ssl_certificate /etc/letsencrypt/live/[GPU 서버 도메인]/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/[GPU 서버 도메인]/privkey.pem;
    ssl_prefer_server_ciphers on;

    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;
    proxy_set_header X-Forwarded-Proto https;
    #proxy_set_header Upgrade $http_upgrade;
    #proxy_set_header Connection "Upgrade";
    proxy_headers_hash_bucket_size 512;
    proxy_redirect off;

    location / {
        proxy_connect_timeout 300;
        proxy_send_timeout 300;
        proxy_read_timeout 300;
        send_timeout 300;
        proxy_pass http://[GPU 서버 도메인]:3000/;
    }

    location /music/ {
        proxy_connect_timeout 300;
        proxy_send_timeout 300;
        proxy_read_timeout 300;
    }
}
```



```

    send_timeout 300;
    proxy_pass http://[GPU 서버 도메인]:3313/;
}

location /album/ {
    proxy_pass http://[GPU 서버 도메인]:8885/;
}

}

```

Gateway(Flask) : Dockerfile

```

FROM python:3.9-slim-buster
WORKDIR /app

RUN python -m pip install --upgrade pip
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 3000
COPY . .
CMD [ "python", "gateway.py" ]

```

Riffusion(Flask): Dockerfile

```

FROM nvidia/cuda:12.1.0-base-ubuntu20.04

WORKDIR /app
ADD . /app

# Since wget is missing
RUN apt-get update && apt-get install -y wget

#Install MINICONDA
RUN wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh -O Miniconda.sh && \
    /bin/bash Miniconda.sh -b -p /opt/conda && \
    rm Miniconda.sh

ENV PATH /opt/conda/bin:$PATH

# Install gcc as it is missing in our base layer
RUN apt-get update && apt-get -y install gcc

RUN conda install python=3.9.16
RUN conda env create -f environment.yaml

# Copy your source code
COPY . /app

# Expose the necessary ports
EXPOSE 3313

# Make RUN commands use the new environment:
SHELL ["conda", "run", "-n", "riffusion", "/bin/bash", "-c"]

CMD ["conda", "run", "-n", "riffusion", "python", "-m", "riffusion.server", "--host", "0.0.0.0", "--port", "3313", "--checkpoint", "/a

```

Diffusion(Fast API): Dockerfile

```

FROM nvidia/cuda:12.1.0-base-ubuntu20.04

WORKDIR /app
ADD . /app

# Since wget is missing
RUN apt-get update && apt-get install -y wget

#Install MINICONDA
RUN wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh -O Miniconda.sh && \
    /bin/bash Miniconda.sh -b -p /opt/conda && \
    rm Miniconda.sh

ENV PATH /opt/conda/bin:$PATH

# Install gcc as it is missing in our base layer
RUN apt-get update && apt-get -y install gcc

RUN conda env create -f environment.yml

```

```
# Make RUN commands use the new environment:
SHELL ["conda", "run", "-n", "diffusion", "/bin/bash", "-c"]

# Copy your source code
COPY . /app

# Expose the necessary ports
EXPOSE 8885

# Run the application
ENTRYPOINT ["/opt/conda/envs/diffusion/bin/uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8885"]
```

5. 환경 변수 파일

Backend : application.yml

- Jenkins Credential로 관리

~/jenkins/workspace/chatcomposer_backend/backend/src/main/resources/application.yml

```
serverHost: [서비스 도메인]
baseUrl: https://[서비스 도메인]/api

# Server setting
server:
  port: 8080
  servlet:
    encoding:
      charset: UTF-8
      enabled: true
      force: true
  forward-headers-strategy: FRAMEWORK

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://${serverHost}:3306/chatcomposer?serverTimezone=Asia/Seoul
    username: a504
    password: [MySQL 비밀번호]

  redis:
    host: ${serverHost}
    port: 6379
    database: 0
    password: [Redis 비밀번호]

  servlet:
    multipart:
      max-request-size: 100MB
      max-file-size: 100MB

  rabbitmq:
    host: ${serverHost} # rabbitMQ host (docker로 띄웠음)
    port: 4002 # default port
    username: admin # default username
    password: [RabbitMQ 비밀번호] # default password

  jpa:
    show-sql: true
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
        show-sql: true
        format_sql: true
        use-sql-comments: true
    open-in-view: false
    generate-ddl: true

  security:
    oauth2:
      client:
        registration:
          google:
            clientId: [google oauth clientId]
            clientSecret: [google oauth clientSecret]
            redirectUri: "${baseUrl}/login/oauth2/code/google"
            scope:
              - email
              - profile
```

```
# chatgpt key 설정
chatgpt:
  api-key: [ChatGpt api key]

logging:
  level:
    org:
      hibernate:
        type:
          descriptor:
            sql: TRACE
    com:
      amazonaws:
        util:
          EC2MetadataUtils: ERROR

# 토큰 관련 secret Key 및 RedirectUri 설정
app:
  auth:
    tokenSecret: [secret key]
    tokenExpiry: 84400000
    refreshTokenExpiry: 604800000
  oauth2:
    authorizedRedirectUris:
      - https://[서비스 도메인]/oauth/redirect

# jwt secret key 설정
jwt:
  secret: '[JWT 비밀 키]'
  master: '[JWT 마스터 키]'

# cors 설정
cors:
  allowed-origins: 'http://localhost:3000,https://[서비스 도메인],https://accounts.google.com'
  allowed-methods: GET,POST,PUT,DELETE,OPTIONS,PATCH
  allowed-headers: '*'
  max-age: 3600

# AWS
cloud:
  aws:
    region:
      static: ap-northeast-2
    stack:
      auto: false
    s3:
      bucket: chatcomposer
  # AWS IAM
  credentials:
    access-key: [aws access key]
    secret-key: [aws secret key]

# Swagger springdoc-ui Configuration
springdoc:
  packages-to-scan: com.a504.chatcomposer
  default-consumes-media-type: application/json;charset=UTF-8
  default-produces-media-type: application/json;charset=UTF-8
  swagger-ui:
    path: chatcomposer-ui.html # Swagger UI 경로 => localhost:8080/chatcomposer-ui.html
    tags-sorter: alpha # alpha: 알파벳 순 태그 정렬, method: HTTP Method 순 정렬
    operations-sorter: alpha # alpha: 알파벳 순 태그 정렬, method: HTTP Method 순 정렬
  api-docs:
    path: /api-docs/json
    groups:
      enabled: true
  cache:
    disabled: true
```

Frontend : .env

- Jenkins Credential로 관리

~/jenkins/workspace/chatcomposer_frontend/frontend/ `.env`

```
OPENAI_API_BASE = "[ChatGPT key]"
X_NAVER_CLIENT_ID = "[네이버 API 클라이언트 ID]"
X_NAVER_SECRET_ID = "[네이버 API 비밀 ID]"

GOOGLE_CLIENT_ID="[구글 API 클라이언트 ID]"
GOOGLE_SECRET="[구글 API 비밀 키]"

SECRET=[next.js auth 비밀 키]
```

II. AI 모델 테스트

1. Cuda 설치

- 적합한 드라이버 찾기

```
sudo apt-get install -y ubuntu-drivers-common  
  
ubuntu-drivers devices
```

- 다운로드

```
sudo apt-get install -y {위에서 찾은 드라이버}  
# ex. sudo apt-get install -y nvidia-driver-530  
  
# 재부팅  
sudo reboot
```

- 확인

```
nvidia-smi
```

2. 가상환경

MiniConda 설치

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh \  
&& sh Miniconda3-latest-Linux-x86_64.sh  
  
# 재부팅 후  
conda config --set auto_activate_base false
```

- 가상환경 생성

```
conda create -n riffusion python=3.9  
conda create -n diffusion python=3.9
```

3. Riffusion

가상환경 시작

```
conda activate riffusion  
  
# 모델 디렉토리로 이동  
cd riffusion
```

- 필요한 패키지 설치

```
conda install -c conda-forge ffmpeg  
python -m pip install -r requirements.txt
```

- API Server 실행

서비스에 사용되는 모델 인퍼런스 서버

```
python -m riffusion.server --host {호스트} --port {포트}
```

- Streamlit 실행

다양한 기능을 테스트 해볼 수 있는 Playground

```
python -m riffusion.streamlit.playground
```

- CLI 실행

커맨드라인으로 테스트할 수 있는 인터페이스

```
# 가능한 커맨드 확인 후 실행  
python -m riffusion.cli -h
```

4. Diffusion

가상환경 시작

```
conda activate diffusion
```

- 필요한 패키지 설치

```
python -m pip install -r requirements.txt
```

- API 서버 실행

```
uvicorn main:app --port 8885
```

III. 빌드 및 배포

1. docker 설치

- 필요한 패키지 설치

Apt 패키지 관리자를 업데이트 Docker 설치에 필요한 패키지들을 설치합니다.

```
sudo apt-get update && sudo apt-get install \  
  apt-transport-https \  
  ca-certificates \  
  curl \  
  software-properties-common
```

- Docker 공식 gpg key와 저장소 추가

아래의 명령어를 사용하여 Docker 공식 gpg key와 저장소를 추가합니다.

```
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-key fingerprint 0EBFCD88
```

- Docker 패키지 검색 확인

현재의 우분투 버전에서 Docker 패키지가 검색이 되는지 확인합니다.

```
sudo apt-get update && sudo apt-cache search docker-ce

# 정상적으로 패키지가 검색된다면 아래와 같은 내용이 출력됩니다.
# docker-ce - Docker: the open-source application container engine
```

• Docker CE Install

아래의 명령어로 Docker CE 버전을 설치합니다.

```
sudo apt-get update && sudo apt-get install docker-ce
```

• 도커 재실행

그리고 일반 사용자 계정으로 docker 명령어를 사용하기 위해서 아래의 명령어로 그룹을 추가 후 도커를 재실행합니다.

```
sudo usermod -aG docker $USER

sudo systemctl restart docker
```

- 일반 사용자를 docker 그룹에 추가하지 않을 경우, sudo 권한이 아닌 일반 사용자로 docker 명령 실행 시 아래와 같은 오류가 발생할 수 있습니다.

```
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.39/containers/json?all=1: dial unix /var/run/docker.sock: connect: permission denied
```

• 설치 확인

설치가 제대로 되었는지 확인합니다.

```
docker run hello-world
```

2. Docker Compose 설치

Install the Compose standalone

How to install Docker Compose - Other Scenarios

 <https://docs.docker.com/compose/install/other/>



1. To download and install Compose standalone, run :

```
sudo curl -SL \
https://github.com/docker/compose/releases/download/v2.16.0/docker-compose-linux-x86_64 \
-o /usr/local/bin/docker-compose
```

2. Apply executable permissions to the standalone binary in the target path for the installation.

실행 권한 설정

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

- If the command `docker-compose` fails after installation, check your path. You can also create a symbolic link to `/usr/bin` or any other directory in your path. For example:

심볼릭 링크 설정

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

- 정상 설치 확인

```
docker-compose -v
```

3. SSL 인증

certbot docker container

```
docker run -it --rm --name certbot \
-p 80:80 \
-v '/etc/letsencrypt:/etc/letsencrypt' \
-v '/var/lib/letsencrypt:/var/lib/letsencrypt' \
certbot/certbot certonly -d '[서비스 도메인]' --standalone \
--server https://acme-v02.api.letsencrypt.org/directory
```

4. DB 및 Infra 배포

Nginx, Jenkins, MySQL, Redis, RabbitMQ

```
# docker-compose.yml 파일의 위치에서 실행 (현재 home directory)
cd /home/ubuntu 또는 cd ~
docker compose up -d
```

Dockerfile 저장

- Backend
 - 프로젝트 back 폴더 최상위에 Backend Dockerfile을 위치시키고 Gitlab Repository에 push
- Frontend
 - 프로젝트 front 폴더 최상위에 Frontend Dockerfile을 위치시키고 Gitlab Repository에 push

방법 1) 수동 배포

1. ec2 서버에서 git pull 받기
2. `application.yml` 저장

```
# 백엔드 환경파일 위치로 이동
cd ~/[GitLab Repository 명]/backend/src/main/resources/

# application.yml 생성 후 열기
sudo vim application.yml

##### vim 진입 #####
# Insert 모드로 변경
i

# => application.yml 작성 후 esc를 눌러 command 모드로 변경

# 저장하고 나가기
:wq
```

3. `.env` 저장

```
# 프론트엔드 환경파일 위치로 이동
cd ~/[GitLab Repository 명]/frontend/

# .env 생성 후 열기
sudo vim .env
```

```
##### vim 진입 #####
# Insert 모드로 변경
i

# => .env 작성 후 esc를 눌러 command 모드로 변경

# 저장하고 나가기
:wq
```

4. 다음 명령어 실행 (각 Dockerfile의 위치)

- back

```
# ~/[GitLab Repository 명]/backend/

sudo ./gradlew clean build -x test \
    && docker build -t chco-backend:latest . \
    && docker rm -f backend \
    && docker run -d --name backend -p 8080:8080 -u root chco-backend:latest \
    && docker images -qf dangling=true | xargs -I{} docker rmi {}
```

- front

```
# ~/[GitLab Repository 명]/frontend/

docker build -t chco-frontend:latest .
docker rm -f frontend
docker run -d --name frontend -p 3000:3000 -u root chco-frontend:latest
docker images -qf dangling=true | xargs -I{} docker rmi {}
```

▼ 방법 2) Jenkins 사용

[접속 후 환경설정]

Jenkins (1)

1. http://[서비스 도메인]:[jenkins_port] 접속

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

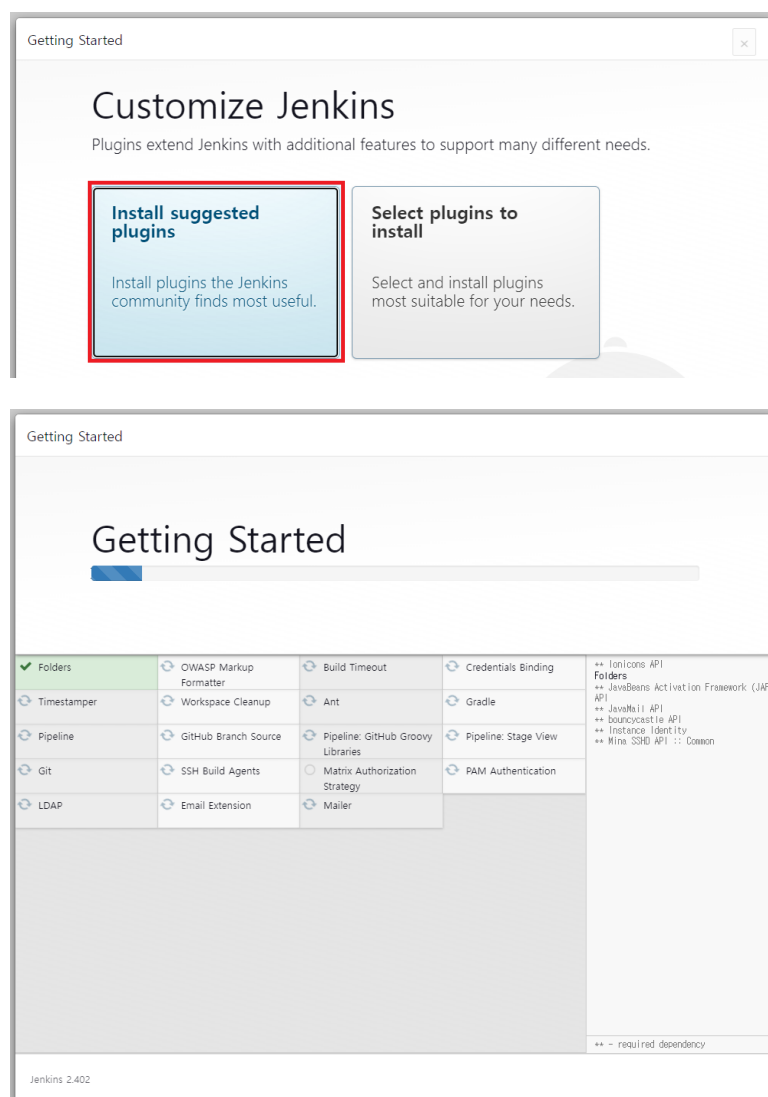


Continue

- **Administrator password** : ec2서버에서 **docker logs [jenkins docker container name]** 입력 후 나오는 password를 입력

```
*****  
*****  
*****  
  
jenkins initial setup is required. An admin user has been created and a password generated.  
Please use the following password to proceed to installation:  
  
##### Administrator password #####  
  
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword  
  
*****  
*****  
*****
```

2. 플러그인 설치



- Install suggested plugins 선택

3. Create First Admin User 계정 등록

Getting Started

Create First Admin User

계정명
a504

암호

암호 확인

이름
chatcomposer

이메일 주소

Jenkins 2.402 Skip and continue as admin Save and Continue

- 계정명 : 임의 작성
- 암호 : 임의 작성
- 암호 확인 : 임의 작성
- 이름 : 임의 작성
- 이메일 주소 : 임의 작성

⇒ Save and Continue

4. Instance Configuration

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

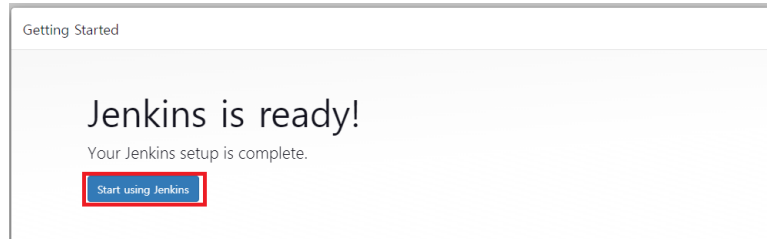
The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.402 Not now Save and Finish

- Jenkins URL : **현재 접속한 URL** (http://[서비스 도메인]:[jenkins_port]) 입력

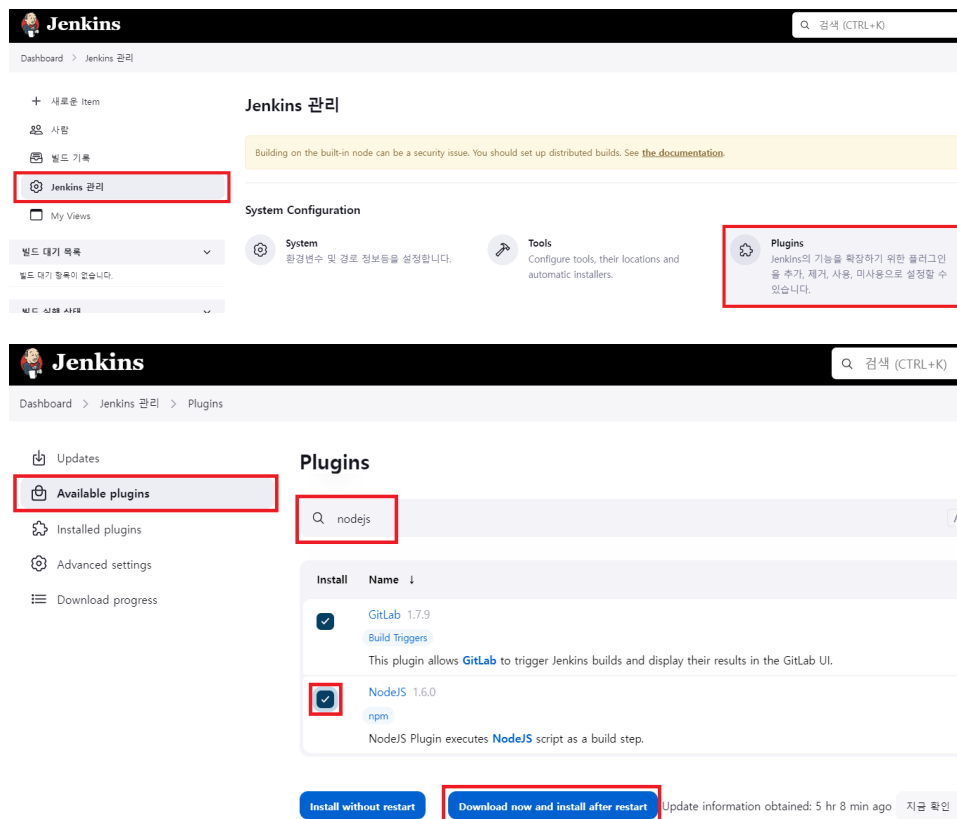
⇒ Save and Finish

5. Jenkins is ready!



⇒ Start using jenkins

6. Dashboard > Jenkins 관리 > Plugins > Available plugins

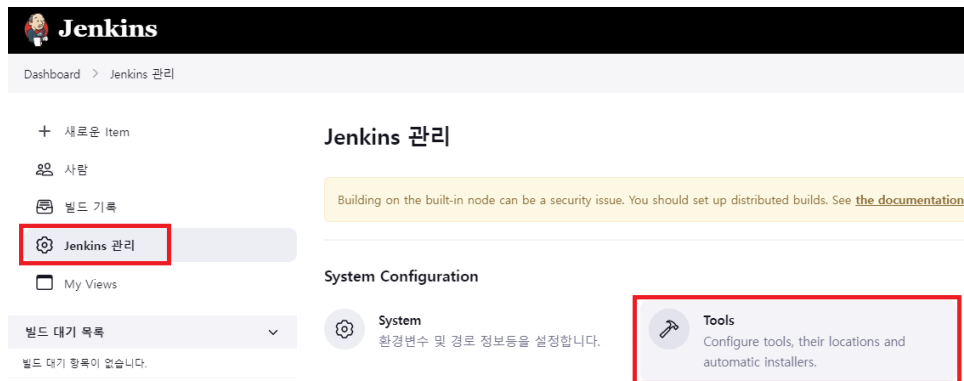


→ [메인 페이지로 돌아가기](#)
(설치된 플러그인을 바로 사용할 수 있습니다.)

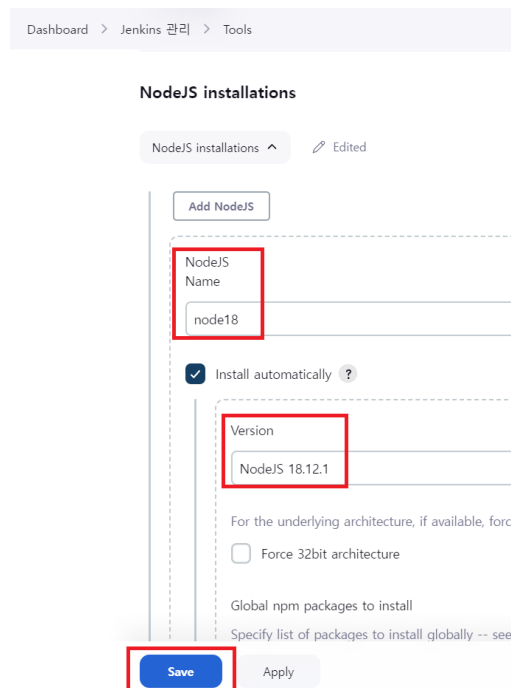
→ ☒ 설치가 끝나고 실행중인 작업이 없으면 Jenkins 재시작.

⇒ **Gibb Lab**, **NodeJS** 검색하여 추가

7. Dashboard > Jenkins 관리 > Tools



NodeJS

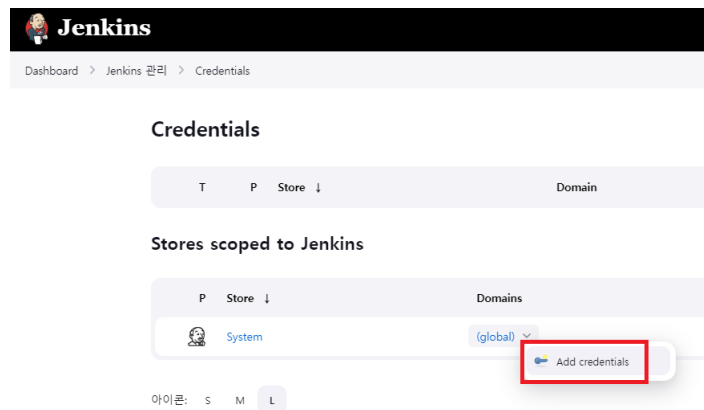
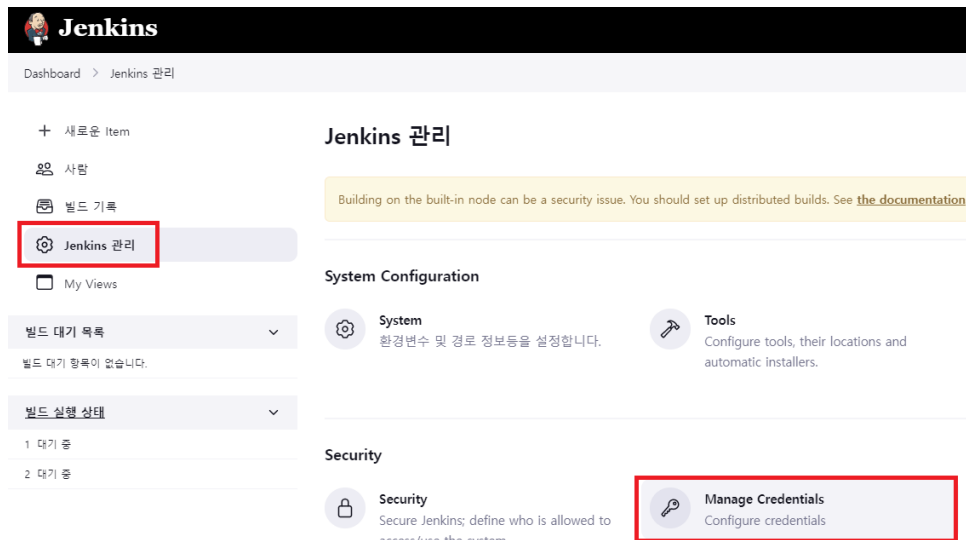


- Name : 임의 설정
- Version : **NodeJS 18.12.1** (사용할 버전 임의 선택)

⇒ Save

[Gitlab Repository 연결]

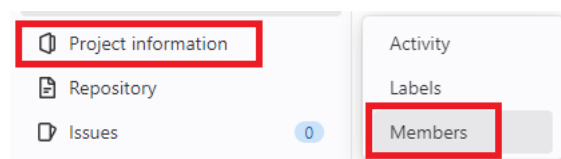
8. Dashboard > Jenkins 관리 > Manage Credentials



- (global)에 커서를 대면 나오는 우측 화살표 버튼을 눌러 **Add Credentials** 클릭

Gitlab (1)

9. Project infomagion > Members



Max role

Developer ▾

Maintainer ▾

Developer ▾

Maintainer ▾

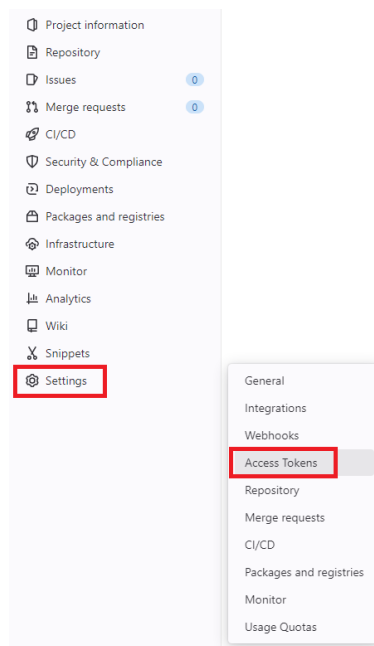
Developer ▾

Developer ▾

- Maintainer 권한 확인

⇒ 권한이 Maintainer가 아닐 경우 팀장(Maintainer권한자)에게 권한 변경 요청
또는 Maintainer가 직접 진행

10. Settings > Access Tokens



Q 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.](#)

Add a project access token

Enter the name of your application, and we'll return a unique project access token.

Token name
chatcomposer-access-token

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

Expiration date
2023-05-26

Select a role
Maintainer

Select scopes
Scopes set the permission levels granted to the token. [Learn more.](#)

☐ api
Grants complete read and write access to the scoped project API, including the Package Registry.

☒ read_api
Grants read access to the scoped project API, including the Package Registry.

☒ read_repository
Grants read access (pull) to the repository.

☐ write_repository
Grants read and write access (pull and push) to the repository.

Create project access token

- Token name : 임의 설정
- Expiration date : 임의 설정 (프로젝트 기간보다 조금 여유를 두고 설정하였음)
- Select a role : Maintainer
- Select scopes : 임의 설정 (✓ read_api, ✓ read_repository 체크)

⇒ Create project access token

① Your new project access token has been created.

Q Search page

Project Access Tokens

Generate project access tokens scoped to this project for your applications that need access to the GitLab API.

Your new project access token

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

Copy project access token

- access token을 발급받은 후, 최상단에서 Copy project access token

Jenkins (2)

11. 8 에서 Add Credentials 클릭 후 화면

Kind

- GitLab API token
- Username with password
- GitHub App
- GitLab API token**
- SSH Username with private key
- Secret file
- Secret text
- Certificate

New credentials

Kind
GitLab API token

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

API token
.....

ID ?
chatcomposer

Description ?

Create

- Kind : GitLab API token 선택
- Scope : Global (Jenkins, nodes, items, all child items, etc) 선택
- API token : 10 에서 발급받아 Copy했던 Gitlab의 project access token 입력

⇒ Create

12. Dashboard > Jenkins 관리 > System > Gitlab

Jenkins

Dashboard > Jenkins 관리

+ 새로운 Item

사람

빌드 기록

Jenkins 관리

My Views

빌드 대기 목록

빌드 대기 항목이 없습니다.

Jenkins 관리

Building on the built-in node can be a security issue. Yo

System Configuration

System
환경변수 및 경로 정보등을 설정합니다.

Dashboard > Jenkins 관리 > System >

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name

A name for the connection

chatcomposer-repository

Gitlab host URL

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

https://lab.com/

Credentials

API Token for accessing Gitlab

GitLab API token

Add ▾

고급 ▾

추가

저장 Apply

- Connection name : 임의 설정
- Gitlab host URL : 깃랩 호스트 입력
- Credentials : GitLab API token 선택 (11 에서 생성한 Credentials)

⇒ 저장

Gitlab (2)

13. 우측 상단 Profile > Edit profile > Access Tokens

📄 🔗 ✉️ ⚙️ 👤 ▾

👤

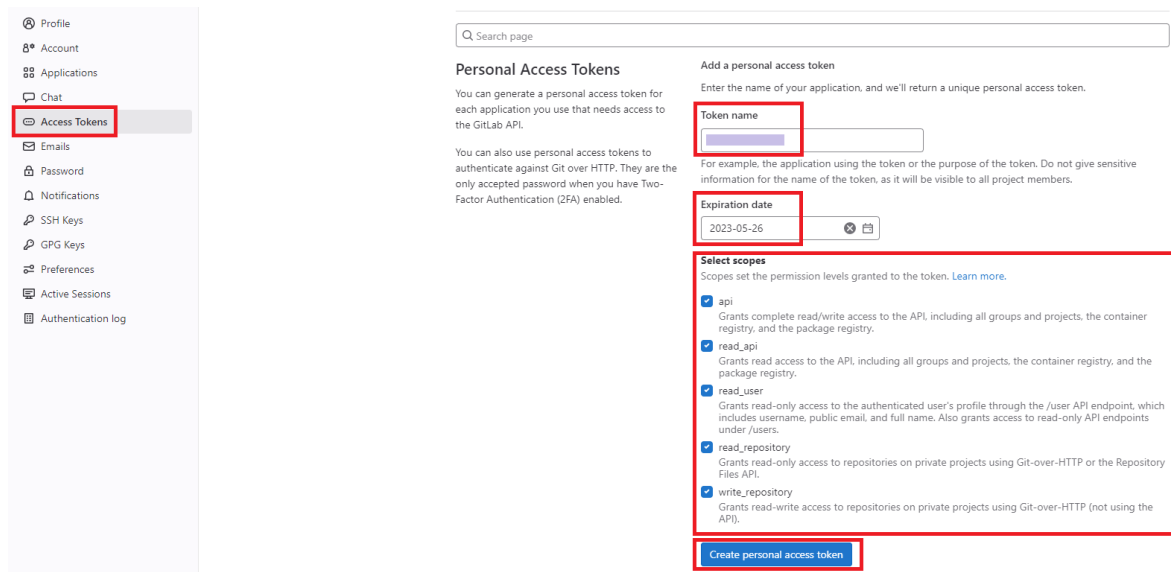
@

Set status

Edit profile

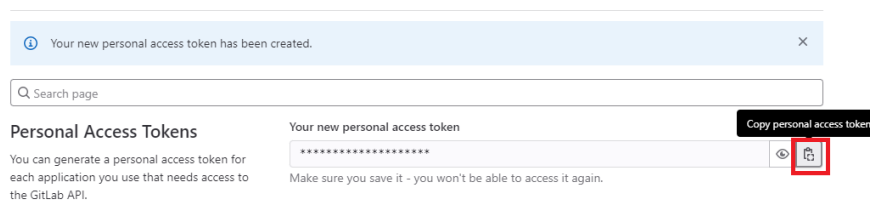
Preferences

Sign out



- Token name : 임의 설정
- Expiration date : 임의 설정
- Select scopes : 임의 설정 (✓ all 체크)

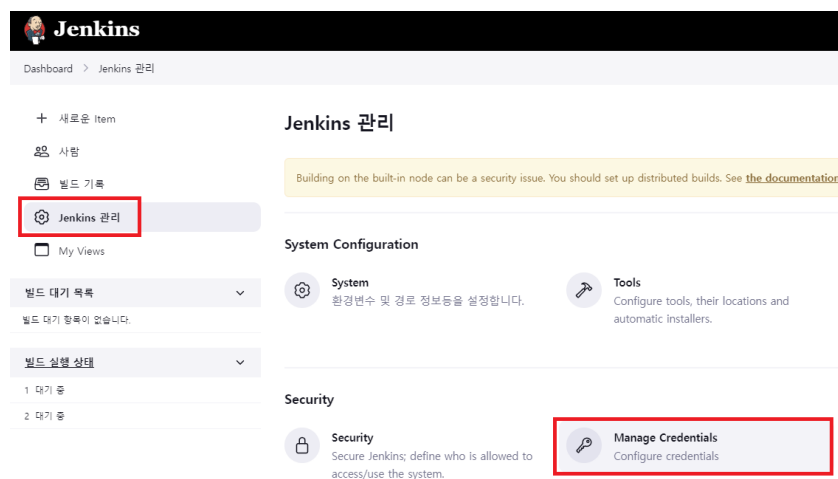
⇒ Create personal access token

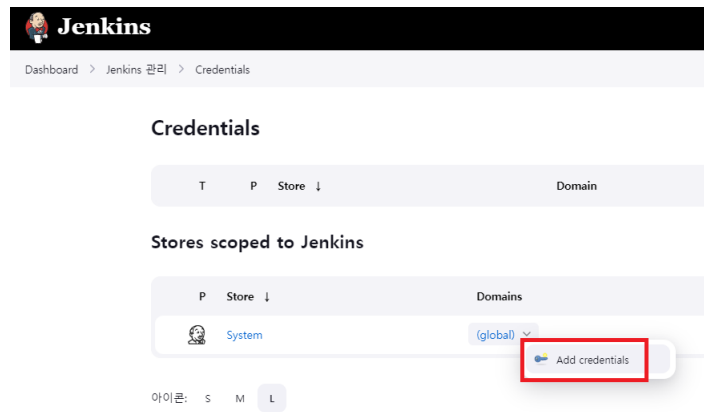


- access token을 발급받은 후, 최상단에서 Copy personal access token

Jenkins (3)

14. Dashboard > Jenkins 관리 > Manage Credentials





- (global)에 커서를 대면 나오는 우측 화살표 버튼을 눌러 **Add Credentials** 클릭

- Kind : Username with password 선택
- Scope : Global (Jenkins, nodes, items, all child items, etc) 선택
- Username : Gitlab ID (personal token을 발급받은 사용자)
- Password : Gitlab Password (13 에서 발급받아 Copy했던 Gitlab의 personal access token 입력)
- ID : 임의 설정

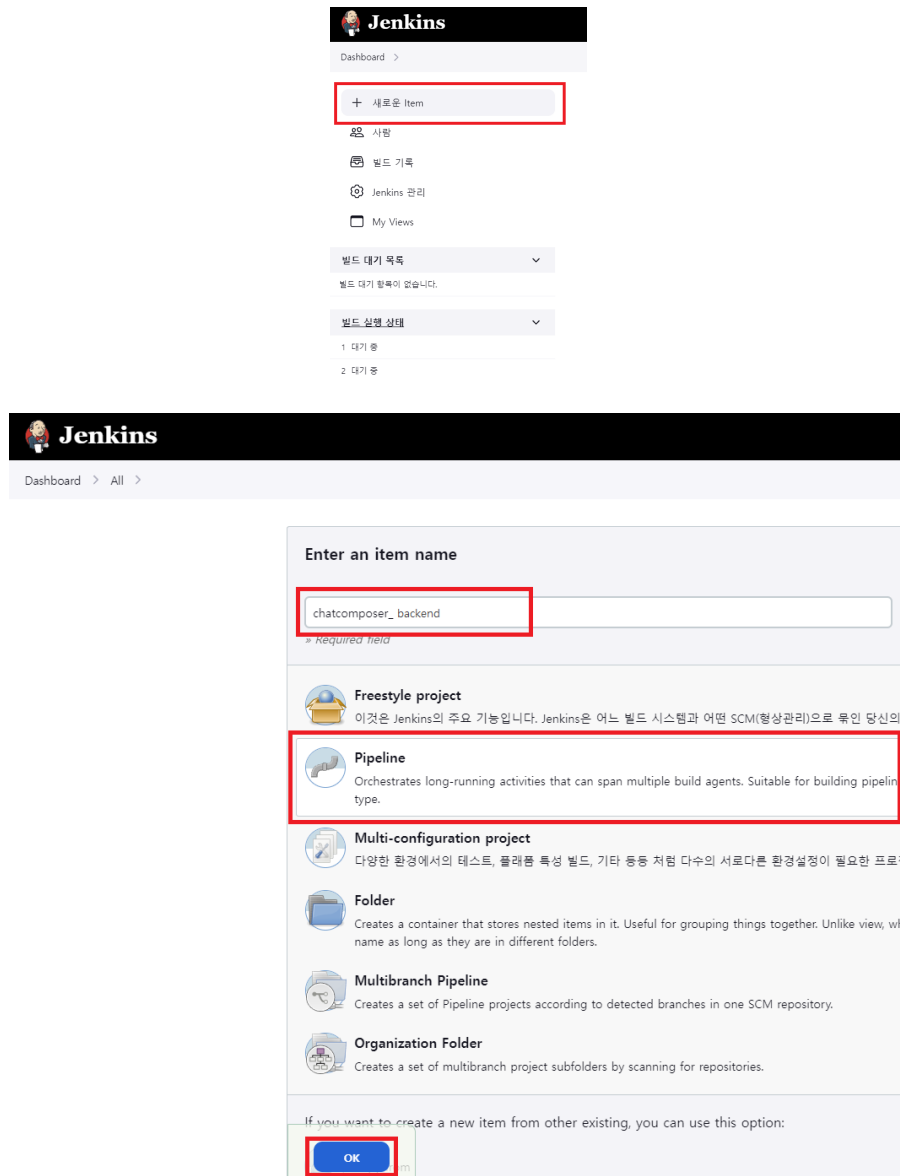
⇒ Create

[파이프라인 생성]

15. Dashboard > 새로운 Item



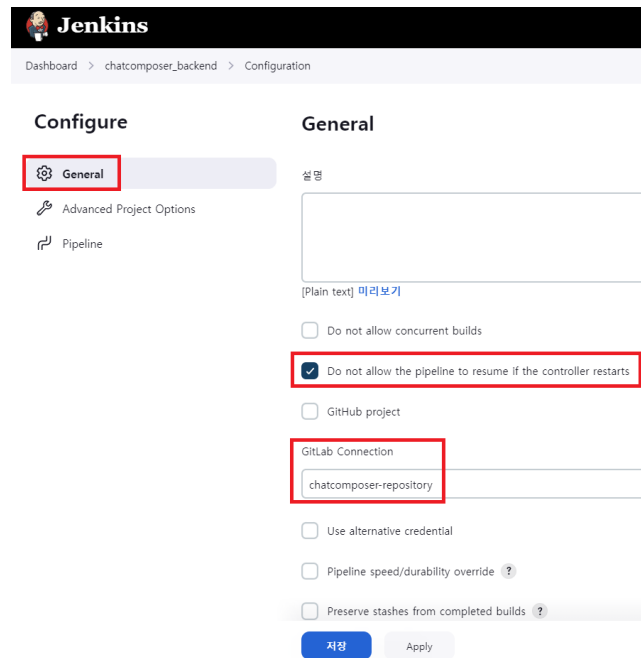
(15 ~ 는 chatcomposer_backend과 같이 chatcomposer_frontend도 동일 과정으로 진행)



- Enter an item name : 임의 설정 (프로젝트 명 입력)
- Pipeline 선택

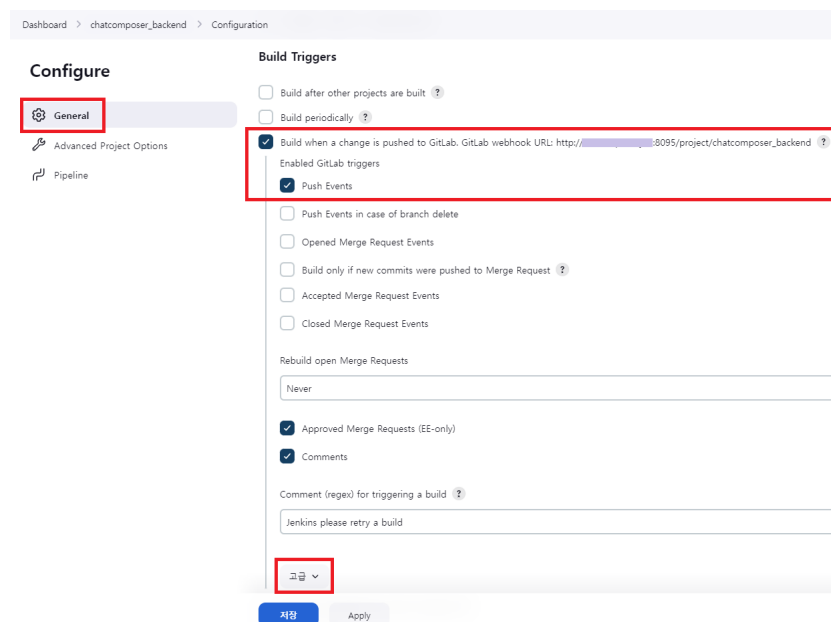
⇒ OK

16. OK 클릭 시 나오는 화면 (Dashboard > chatcomposer_backend > Configuration) > General



The image shows the Jenkins Configuration page for the 'chatcomposer_backend' project, specifically the 'General' tab. The 'General' tab is highlighted with a red box. Below it, the 'Advanced Project Options' and 'Pipeline' tabs are visible. The 'General' section includes a description field, a checkbox for 'Do not allow concurrent builds', a checked checkbox for 'Do not allow the pipeline to resume if the controller restarts' (highlighted with a red box), a checkbox for 'GitHub project', a 'GitLab Connection' dropdown menu with 'chatcomposer-repository' selected (highlighted with a red box), and checkboxes for 'Use alternative credential', 'Pipeline speed/durability override', and 'Preserve stashes from completed builds'. At the bottom, there are '저장' (Save) and 'Apply' buttons.

- ☒ Do not allow the pipeline to resume if the controller restarts 체크
- GitLab Connection : 12 에서 설정했던 Gitlab Connection name 선택



The image shows the Jenkins Configuration page for the 'chatcomposer_backend' project, specifically the 'Build Triggers' tab. The 'General' tab is highlighted with a red box. Below it, the 'Advanced Project Options' and 'Pipeline' tabs are visible. The 'Build Triggers' section includes checkboxes for 'Build after other projects are built', 'Build periodically', and a checked checkbox for 'Build when a change is pushed to GitLab, GitLab webhook URL: http://[url]:8095/project/chatcomposer_backend' (highlighted with a red box). Below this, there are checkboxes for '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', and 'Closed Merge Request Events'. There is a section for 'Rebuild open Merge Requests' with a dropdown menu set to 'Never'. Below that, there are checked checkboxes for 'Approved Merge Requests (EE-only)' and 'Comments'. A text field for 'Comment (regex) for triggering a build' contains 'Jenkins please retry a build'. At the bottom, there is a '고급' (Advanced) dropdown menu (highlighted with a red box) and '저장' (Save) and 'Apply' buttons.

Build Triggers

- ☒ Build when a change is pushed to GitLab, GitLab webhook URL: http://[서비스도메인]:[Jenkins port]/project/[pipeline item명] 체크
- Enabled GitLab triggers
 - ☒ Push Events 체크

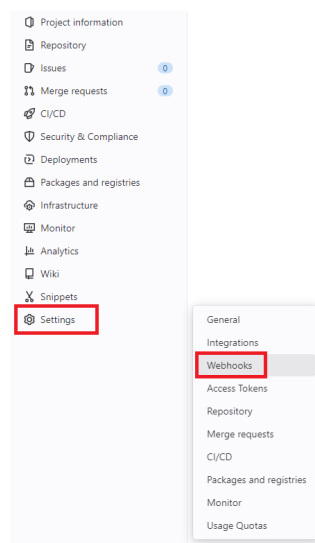
고급 클릭

- Secret token > Generate 클릭 후 Copy 해두기

[웹훅 설정]

Gitlab (3)

17. Settings > Webhooks



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.

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.

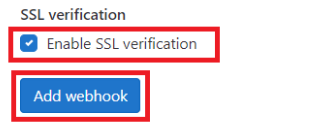
Secret token

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

Trigger
☒ Push events

 Push to the repository.

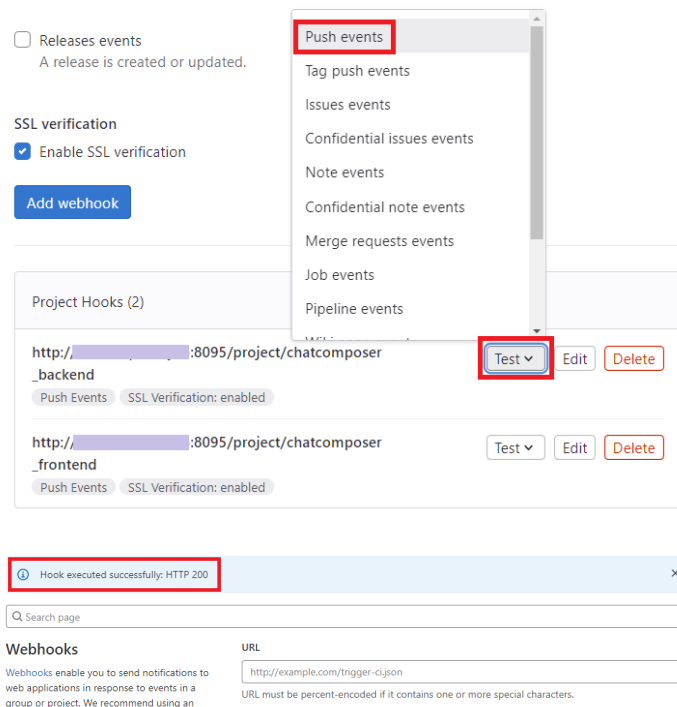
- URL : 16 에서 Build Triggers 설정했던 GitLab webhook URL: http://[서비스도메인]:[Jenkins port]/project/[pipeline item명] 입력
- Secret token : 16 에서 Generate해서 Copy 해둔 Secret token 입력
- Trigger : ☒ Push events 체크
 - 각각 backend, frontend 입력 (events를 감지하여 webhook이 발생할 대상 branch 설정)



SSL verification

- ☒ Enable SSL verification 체크

⇒ Add webhook



- 하단 Project Hooks 목록에서 Test > Push events 클릭 후 최상단에 HTTP 200이 뜨면 성공

[파이프라인 설정]

Jenkins (4)

18. 16 화면 (Dashboard > chatcomposer_backend > Configuration) > Pipeline

Dashboard > chatcomposer_backend > Configuration

Configure

General

Advanced Project Options

Pipeline

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://

Credentials ?

/*****

Add

고급

Add Repository

Branches to build ?

저장 Apply

- Definition : Pipeline script from SCM 선택
- SCM : Git 선택

Repositories

- Repository URL : GitLab Repository URL 입력
- Credentials : 14 에서 설정했던 User Credential 선택

Dashboard > chatcomposer_backend > Configuration

Configure

General

Advanced Project Options

Pipeline

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/backend

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Add

Script Path ?

backend/enkinsfile

☒ Lightweight checkout ?

Pipeline Syntax

저장 Apply

Dashboard > chatcomposer_frontend > Configuration

Configure

General

Advanced Project Options

Pipeline

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/frontend

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Add

Script Path ?

frontend/enkinsfile

☒ Lightweight checkout ?

Pipeline Syntax

저장 Apply

Branches to build

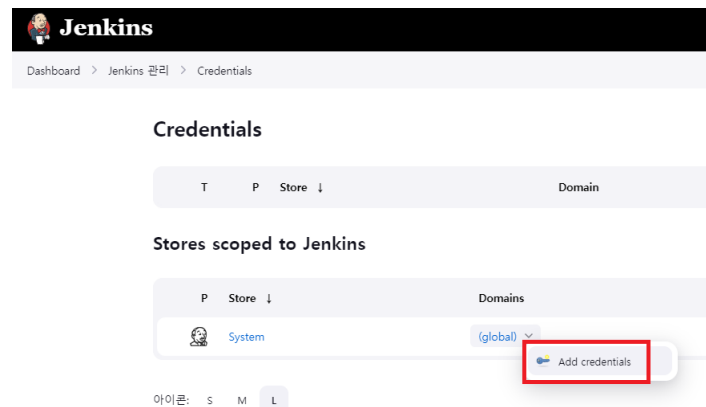
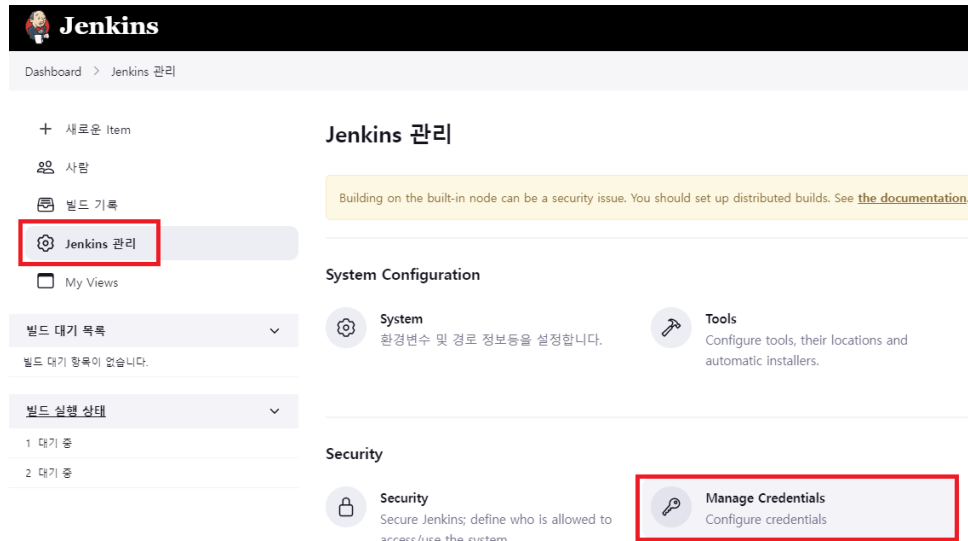
- Branch Specifier (blank for 'any') : 각각 ***/backend, */frontend** 입력 (trigger 발생 시 build 대상이 되는 branch 설정)

- Script Path : Gitlab Repository 최상위 디렉토리 기준으로 Jenkinsfile의 경로 입력

⇒ 저장

[환경 파일 관리]

19. Dashboard > Jenkins 관리 > Manage Credentials



- (global)에 커서를 대면 나오는 우측 화살표 버튼을 눌러 Add Credentials 클릭

20. New credentials

New credentials

Kind: Secret file

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

File: application.yml

ID: back-credential

Description:

Create

New credentials

Kind: Secret file

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

File: .env

ID: front-credential

Description:

Create

- Kind : Secret file 선택
- Scope : Global (Jenkins, nodes, items, all child items, etc) 선택
- File : 각각 application.yml, .env 파일 선택
- ID : 임의 설정

⇒ Create

21. 각각의 파이프라인(Jenkinsfile)의 docker image 생성 전 step에서 알맞은 위치에 copy

```
stage('application.yml load'){
  steps {
    dir("backend") {
      withCredentials([file(credentialsId: 'back-credential', variable: 'configFile')]){
        script {
          sh 'cp $configFile ./src/main/resources/application.yml'
        }
      }
    }
  }
}
```

```
stage('.env load'){
  steps {
    dir("frontend") {
      withCredentials([file(credentialsId: 'front-credential', variable: 'configFile')]){
        script {
          sh 'cp $configFile ./env'
        }
      }
    }
  }
}
```

- withCredentials([file(credentialsId: '[29]에서 설정한 ID]', variable: '[변수명]')]{ } : 이와 같은 방식으로 사용

Jenkins, Gitlab 설정 후 Jenkins에서 지금 빌드

→ EC2 server console에서 ~/jenkins/workspace/[git repository명]/ 위치에 브랜치가 생성되었는지 확인

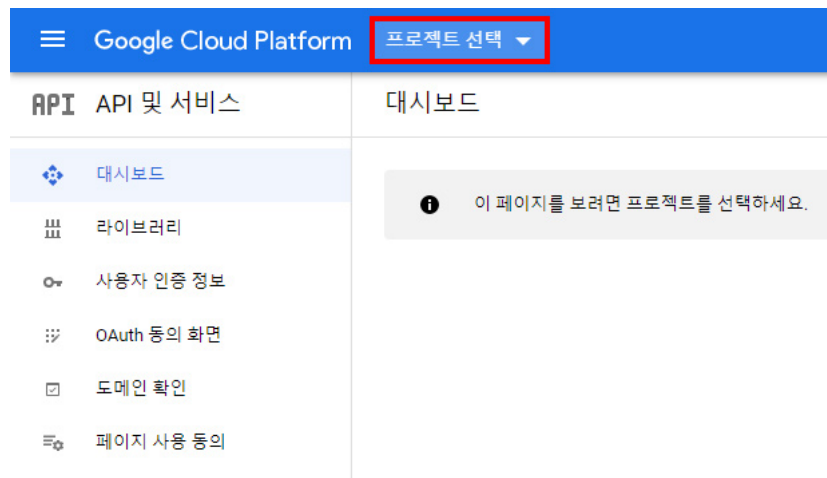
이후, 대상 브랜치에 push로 trigger가 실행되거나 Jenkins에서 지금 빌드를 누르면 배포 완료

IV. 외부 서비스

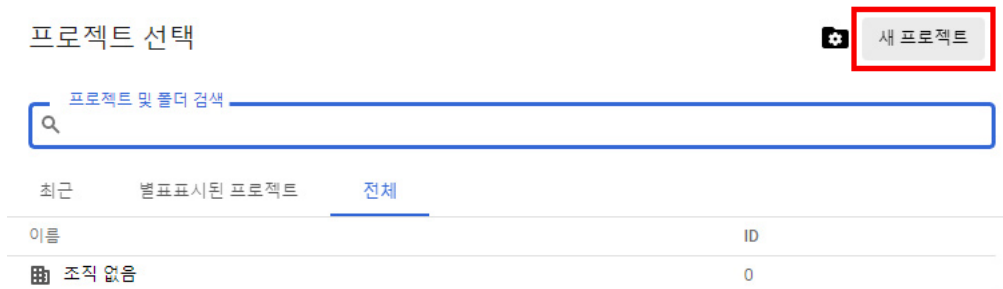
▼ Google Login

새 프로젝트 만들기

1. 화면 왼쪽 상단에 있는 **프로젝트 선택**을 클릭합니다.



2. 프로젝트 선택 창이 나타나면 **새 프로젝트**를 선택합니다.



3. (선택 사항) **프로젝트 이름**을 수정하고 **위치**를 선택한 후 **만들기** 버튼을 클릭합니다.

Google Cloud Platform

새 프로젝트

projects 할당량이 12개 남았습니다. 할당량 증가를 요청하거나 프로젝트를 삭제하세요. [자세히 알아보기](#)

[MANAGE QUOTAS](#)

프로젝트 이름 *

My Project 95254

프로젝트 ID: aerobic-layout-333901입니다. 나중에 변경할 수 없습니다. [수정](#)

위치 *

조직 없음 [찾아보기](#)

상위 조직 또는 폴더

[만들기](#) 취소

4. 새 프로젝트가 생성됩니다. 생성이 완료되기까지는 다소 시간이 소요될 수 있습니다.
5. 기존 프로젝트가 있는 경우, 다시 프로젝트 이름을 클릭한 후 생성된 프로젝트를 클릭해 선택합니다.

라이브러리 추가하기

1. 왼쪽 메뉴에서 라이브러리를 클릭합니다.

Google Cloud Platform

My Project 95254

API 및 서비스

API 및 서비스 + API 및 서비스

대시보드

라이브러리

사용자 인증 정보

OAuth 동의 화면

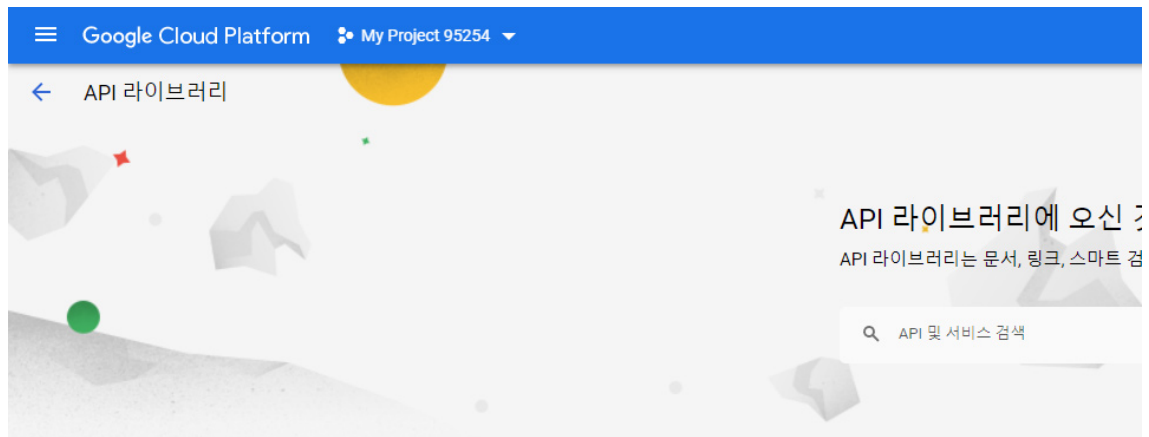
도메인 확인

페이지 사용 동의

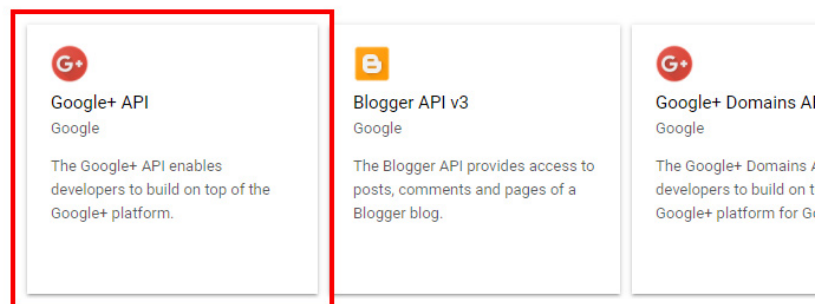
트래픽

No data is available for the selected API set

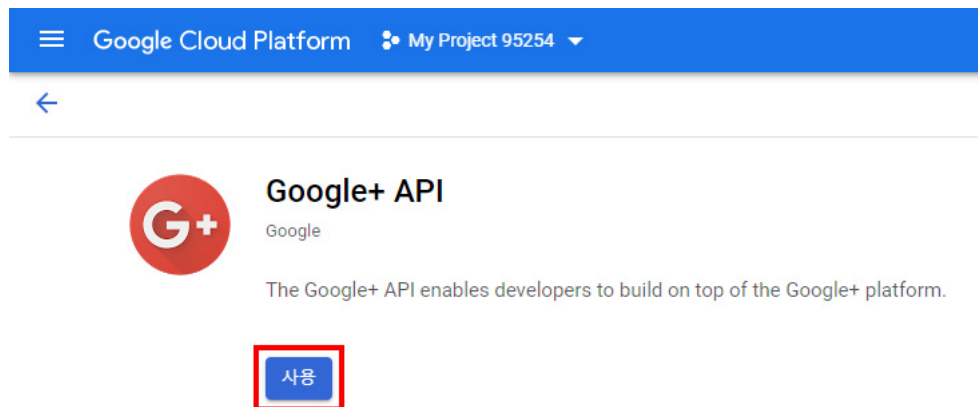
2. 페이지를 스크롤 해서 내려가면 소셜 항목에서 Google+ API를 클릭해 선택합니다.

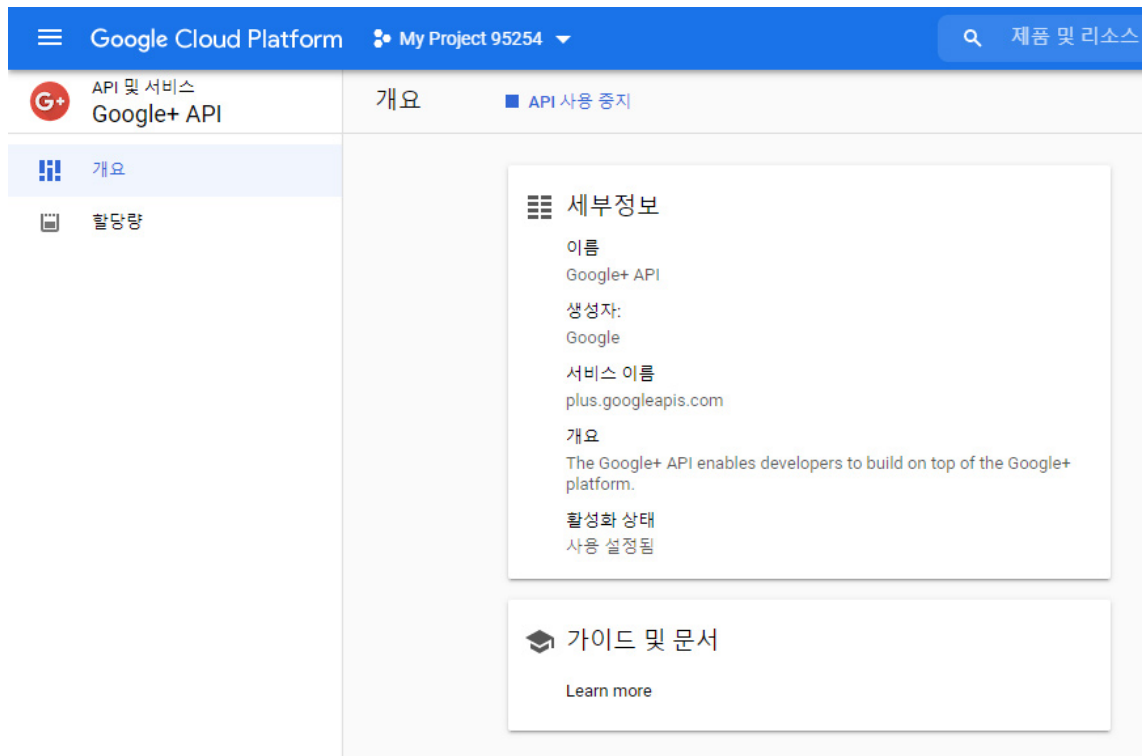


소셜



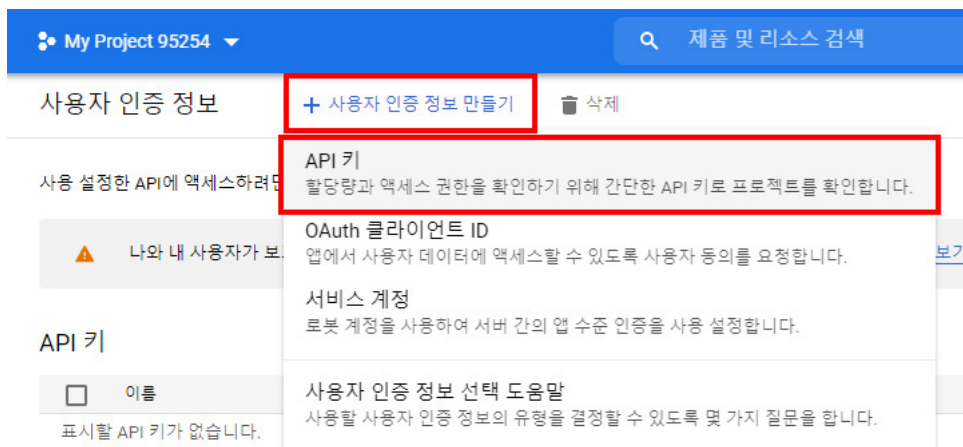
3. Google+ API 페이지로 이동한 후 **사용**을 클릭합니다.라이브러리가 추가된 후





API키 발급받기

1. 왼쪽 메뉴에서 **사용자 인증 정보**를 클릭합니다.
2. 상단의 **사용자 인증 정보 만들기**를 클릭한 후 **API 키**를 클릭합니다.



3. **API 키**가 생성되면 **복사 아이콘** () 을 클릭해 복사한 후 기록해 둡니다.

API 키 생성됨

애플리케이션에서 이 키를 사용하려면 키를 `key=API_KEY` 매개변수로 전달하세요.



닫기 키 제한

4. Additional Information을 확인하여 clientId, clientSecret을 확인한다.

Additional information

| | |
|----------|----------------------------------|
| 클라이언트 ID | |
| 생성일 | 2023년 4월 28일 PM 2시 45분 37초 GMT+9 |

클라이언트 보안 비밀번호

클라이언트 보안 비밀번호를 변경하는 중인 경우 다운로드 없이 수동으로 순환할 수 있습니다. [자세히 알아보기](#)

| | | |
|---------------|----------------------------------|-----|
| 클라이언트 보안 비밀번호 | | 🔍 ⬇ |
| 생성일 | 2023년 4월 28일 PM 2시 45분 37초 GMT+9 | |
| 상태 | ✔ 사용 설정됨 | |

[+ ADD SECRET](#)

5. 확인한 정보를 application.yaml에 추가합니다.

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            clientId: [클라이언트 ID를 입력하세요]
            clientSecret: [클라이언트 보안 비밀번호를 입력하세요]
            redirectUri: "${baseUrl}/login/oauth2/code/google"
            scope:
              - email
              - profile
```

앱 게시하기

- 왼쪽 메뉴에서 **OAuth 동의 화면**을 클릭합니다.
- 게시 상태** 항목에서 테스트의 **앱 게시** 버튼을 클릭합니다.

☰

Google Cloud Platform

My Project 95254 ▼

API

API 및 서비스

🔍

대시보드

📖

라이브러리

🔑

사용자 인증 정보

🔗

OAuth 동의 화면

☑

도메인 확인

⚙

페이지 사용 동의

OAuth 동의 화면

French Mood [앱 수정](#)

게시 상태 ?

테스트

앱 게시

사용자 유형

외부 ?

내부로 설정

3. 아래 이미지와 같은 확인 메시지가 나타나면 **확인** 버튼을 클릭합니다.

프로덕션으로 푸시하시겠어요?

Google 계정이 있는 모든 사용자가 앱을 사용할 수 있습니다.

[확인](#)이 필요한 방식으로 앱을 구성했습니다. 확인을 완료하려면 다음을 제공해야 합니다.

1. 앱 개인정보처리방침의 공식 링크
2. 범위 내에서 가져오는 Google 사용자 데이터를 어떻게 사용할 계획인지 보여주는 YouTube 동영상
3. 민감하거나 제한된 사용자 데이터에 액세스해야 하는 이유를 Google에 알리는 서면 설명서
4. Google Search Console에서 확인된 모든 도메인

[취소](#)

확인

4. 게시 상태가 **프로덕션** 단계로 전환되었습니다.

게시 상태



프로덕션 단계

[테스트로 돌아가기](#)

프로덕션 단계:

앱 상태를 '프로덕션 단계'로 설정하면 Google 계정이 있는 모든 사용자가 앱을 사용할 수 있게 됩니다. OAuth 화면 구성 방식에 따라서는 [확인을 위해 앱을 제출](#)해야 할 수도 있습니다.

테스트:

앱을 아직 테스트 및 빌드하는 중이라면 상태를 '테스트'로 설정할 수 있습니다. 이 상태에서는 제한된 수의 사용자를 대상으로 앱을 테스트할 수 있습니다.

[게시 상태 자세히 알아보기](#)