포팅 메뉴얼

▼ 1️⃣ 프로젝트 기술 스택

🖥️ Frontend

- lang:
  HTML5, CSS3, TypeScript `5.3.3`, Node.js `20.11.1`
- framework:
  React `18.2.0`, Next.js `14.2.2`
- library :
  style : styled-components `6.1.8`
  HTTP 통신: fetch
  formatter : eslint + prettier
  router : react-router-dom
- state management tool :
  Zustand `4.5.2`

🗄️ Database & Message Queue

🖥️ Backend

- Java open-JDK zulu `17.0.9`
- SpringBoot `3.2.4`
- Spring WebFlux `3.2.4`
- Gradle `8.5`
- Lombok `1.18.16`
- Hibernate `3.2.1`
- ThymeLeaf `3.1.2`
- JUnit `5.10.2`
- Mockito `5.7.0`
- Spring Security `6.2.2`
- Spring Cloud `4.1.2`
- MySQL connector `8.3.0`
- MongoDB Driver `4.11.1`
- Spring Kafka `3.1.3`
- Reactor Kafka `1.3.23`

- MySQL `8.4.0`
- MongoDB `7.0.9`
- Redis `7.2.4`
- Kafka `7.6.1`

👩‍❤️‍👩 협업 툴

- Gitlab
- Jira
- Notion
- Mattermost

- Embedded Mongo(flapdoodle) `4.11.0`
- java-jwt `4.4.0`
- jjwt `0.11.5`

🖱️ CI/CD

- docker `25.0.4`
- docker-compose `2.21.0`
- jenkins : `2.440.1`

🎨 UI/UX

- figma

⚒️ IDE

- IntelliJ 2023.3.4
- VSCode 1.85.2
- MySQL WorkBench 8.0.36

▼ 2️⃣ EC2 서버 환경 설정

(1) 우분투 서버 한국 표준시로 변경 (UTC+9)

```
sudo timedatectl set-timezone Asia/Seoul
```

(2) 카카오 미러 서버 활용

- 기본 서버가 *.ubuntu.com 이라는 해외 서버이기 때문에, 패키지 갱신 속도가 비교적 빠른 국내 미러 서버를 활용하는 것이 효율적임. 가장 많이 이용하는 성능 좋은 미러 서버는 카카오 서버

```
sudo sed -i 's/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kakao.com/g' /etc/apt/sources.list
```

- 미러 서버 업데이트 후

```
sudo apt-get -y update && sudo apt-get -y upgrade
```

(3) SWAP 영역 할당

- 스왑 영역 할당 (ex : 4GB)

```
sudo fallocate -l 4G /swapfile
```

- swapfile 권한 수정

```
sudo chmod 600 /swapfile
```

- sawpfile 생성

```
sudo mkswap /swapfile
```

- swapfile 활성화

```
sudo swapon /swapfile
```

- 시스템이 재부팅해도 swap 유지 설정

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

- swap 영역이 할당 확인

```
free -h
```

(4) docker 설치

```
# 0 Docker 설치 전 필요한 패키지 설치
sudo apt-get -y install apt-transport-https ca-certificates cur
l gnupg-agent software-properties-common

# 1 Docker에 대한 GPC Key 인증 진행
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -

# 2 Docker 레포지토리 등록 (AMD64)
sudo add-apt-repository "deb [arch=amd64] https://download.dock
er.com/linux/ubuntu $(lsb_release -cs) stable"
```

```
# 3 패키지 리스트 갱신
sudo apt-get -y update

# 4 docker 설치
sudo apt-get -y install docker-ce docker-ce-cli containerd.io

# 5 ubuntu유저에 권한 부여
sudo usermod -aG docker ubuntu

# 6 docker 재시작
sudo service docker restart

# 7 로그아웃 후 재접속 하면 도커 사용 가능
exit

# 8 docker 설치 버전 확인
docker -v
```

(5) docker compose설치

```
# 0 docker-compose 설치
sudo curl -L "https://github.com/docker/compose/releases/downlo
ad/v2.21.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/loca
l/bin/docker-compose

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

# 2 docker-compose 설치 버전 확인
docker-compose -v
```

▼ 3️⃣ Nginx 리버스 프록시 설정

1. docker 가상 네트워크 생성

📂 /home/ubuntu (위치 무관)

📍네트워크 이름 example : my-network

```
docker network create my-network
```

```
root@ip-172-26-0-97:/home/ubuntu# docker network ls
NETWORK ID     NAME                 DRIVER    SCOPE
8e63e3cd625c   bridge               bridge    local
6f14c0bd59aa   host                 host      local
e6b8ca46cd2c   my-network           bridge    local
56219ea7885f   none                 null      local
25182a5f6224   play-mongo-network   bridge    local
1bfeda5c807c   play-spark_default   bridge    local
root@ip-172-26-0-97:/home/ubuntu#
```

---

2. Jenkins / Nginx 컨테이너 설치

(nginx 리버스 프록시를 통해 jenkins로 접속하기 위하여 jenkins 함께 빌드)

📂 /home/ubuntu/

Jenkins 도커 파일

- Jenkins 컨테이너 안에 docker와 docker-compose 설치

📄 Dockerfile

```
FROM jenkins/jenkins:lts
USER root

RUN apt-get update && \
    apt-get -y install apt-transport-https \
      ca-certificates \
      curl \
      gnupg2 \
      software-properties-common && \
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-re
lease; echo "$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

RUN groupadd -f docker
```

```
RUN usermod -aG docker jenkins

# 도커 컴포즈 설치
RUN curl -L "https://github.com/docker/compose/releases/latest/
download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/
bin/docker-compose && \
    chmod +x /usr/local/bin/docker-compose
```

📄 docker-compose.yml

```
version: '3'
services:
  jenkins:
    build:
      context: .
      dockerfile: Dockerfile
    # image: jenkins/jenkins:lts
    container_name: jenkins
    volumes:
      - /home/ubuntu/jenkins:/var/jenkins_home #host의 jenkins_h
ome을 가져와서 ubuntu의 jenkins로 가져와서 추가
      - /home/ubuntu/.ssh:/var/jenkins_home/.ssh #젠킨스의 ssh의
명령어를 걸 때 호스트의 .ssh 인증서를 공용해서 씀
      - /var/run/docker.sock:/var/run/docker.sock #host의 docker
engine 사용을 위해 추가
    networks:
      - my-network

  nginx:
    image: nginx
    container_name: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - /home/ubuntu/pickitup/:/etc/nginx/pickitup/
      - /home/ubuntu/nginx/conf.d:/etc/nginx/conf.d # conf.d 를
만듦 (nginx를 통해서 jenkins)
      - /home/ubuntu/nginx/cert:/etc/cert # 인증서 파일을 공유시키기
위해서
      - /etc/letsencrypt:/etc/cert2
    restart: always # 꺼져도 다시 실행
    depends_on:
```

```
      - jenkins # jenkins가 실행되고 나서 nginx를 실행하겠다는 의미
    networks:
      - my-network # 네트워크는 my-network(가상네트워크 그룹을 만들어서
nginx랑 jenkins가 my-network 네트워크에서 사용한다.)

networks:
  my-network:
    external: true
```

---

3. SSL 와일드 카드 인증서 발급

(1) Let’s encrypt 설치

```
sudo apt update
sudo apt-get install letsencrypt -y
```

(2) 설치 확인

```
sudo certbot --help
```

```
root@ip-172-26-0-97:/home/ubuntu# sudo certbot --help

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

  certbot [SUBCOMMAND] [options] [-d DOMAIN] [-d DOMAIN] ...

Certbot can obtain and install HTTPS/TLS/SSL certificates.  By default,
it will attempt to use a webserver both for obtaining and installing the
certificate. The most common SUBCOMMANDS and flags are:

obtain, install, and renew certificates:
    (default) run   Obtain & install a certificate in your current webserver
    certonly        Obtain or renew a certificate, but do not install it
    renew           Renew all previously obtained certificates that are near
expiry
    enhance         Add security enhancements to your existing configuration
   -d DOMAINS       Comma-separated list of domains to obtain a certificate for

  (the certbot apache plugin is not installed)
  --standalone      Run a standalone webserver for authentication
  (the certbot nginx plugin is not installed)
  --webroot         Place files in a server's webroot folder for authentication
  --manual          Obtain certificates interactively, or using shell script
hooks

   -n               Run non-interactively
  --test-cert       Obtain a test certificate from a staging server
  --dry-run         Test "renew" or "certonly" without saving any certificates
to disk
```

(3) SSL 인증서 발급

DNS의 TXT 레코드를 이용하여 인증서를 발급 받을 수 있다. 서브도메인의 _acme-challenge에 해당하는 도메인을 cerbot이 생성한 난수로 등록해주면 된다.

여기서 도메인은 각자 구입한 도메인을 적어주면된다.

`-d "*.pickitup.online" -d "pickitup.online"` 이렇게 인증서를 발급받으면 구입한 도메인 앞에 모든 host 이름에 대해서 인증서를 공유할 수 있다.

```
sudo certbot certonly --manual --preferred-challenges dns -d
"*.yourdomain.com" -d "yourdomain.com"

sudo certbot certonly --manual --preferred-challenges dns -d
"*.pickitup.online" -d "pickitup.online"
```

위의 명령어를 치고 Enter를 누르면 _acme-challenge 하위 도메인에 등록해야할 난수를 던져준다. 해당 난수를 DNS 레코드에 등록해주면된다.

```
root@ip-172-26-0-97:/home/ubuntu# sudo certbot certonly --manual --preferred-challenges dns -d "*.yourdomain.com" -d "yourdomain.com"
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator manual, Installer None
Obtaining a new certificate
Performing the following challenges:
dns-01 challenge for yourdomain.com
dns-01 challenge for yourdomain.com

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
NOTE: The IP of this machine will be publicly logged as having requested this
certificate. If you're running certbot in manual mode on a machine that is not
your server, please ensure you're okay with that.

Are you OK with your IP being logged?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: Y

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please deploy a DNS TXT record under the name
_acme-challenge.yourdomain.com with the following value:

yOfI5qMLsp6hKp7iKY86wrdV9uKkwz6lrthFPx79G3E

Before continuing, verify the record is deployed.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Press Enter to Continue
```

📍 가비아 DNS 관리

TXT	_acme-challenge	yOfI5qMLsp6hKp7iKY86wrdV9uKkwz6lrthFPx79G3E	600		DNS 설정	수정 삭제

DNS 설정 정보를 아직 저장하지말고 터미널에서 Enter를 한번 더 누르면 난수를 하나 더 던져준다. 해당 난수도 추가로 DNS 레코드에 등록해줘야 한다.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please deploy a DNS TXT record under the name
_acme-challenge.yourdomain.com with the following value:

AImL_Edk4ZNNZdtx5L_xZw8eCcZqBSLPs8NtECiN-eA

Before continuing, verify the record is deployed.
(This must be set up in addition to the previous challenges; do not remove,
replace, or undo the previous challenge tasks yet. Note that you might be
asked to create multiple distinct TXT records with the same name. This is
permitted by DNS standards.)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Press Enter to Continue
```

📍 가비아 DNS 관리

TXT	_acme-challenge	yOfI5qMLsp6hKp7iKY86wrdV9uKkwz6lrthFPx79G3E	600		DNS 설정	수정 삭제
TXT	_acme-challenge	AImL_Edk4ZNNZdtx5L_xZw8eCcZqBSLPs8NtECiN-eA	600		DNS 설정	수정 삭제

이렇게 DNS 레코드를 저장해준 다음에 EC2 서버 터미널에서 Enter를 누르면 SSL 인증서가 성공적으로 발급된다.

(4) 인증서 발급 확인

아래 명령어를 통해 아까 등록한 도메인의 폴더가 생성되어있는 지를 확인하면 된다.

```
sudo ls /etc/letencrypt/live
```

4. Nginx conf 파일 설정

위에서 nginx 컨테이너를 실행시킬 때 아래와 같은 옵션을 통해 호스트의 conf.d 파일을nginx 컨테이너의 nginx/conf.d에 마운트 시켰었기 때문에 호스트의 `/home/ubuntu/ngnix/conf.d` 파일에 nginx 설정 파일을 작성해주면 된다.

(기본적으로 제공받은 도메인을 젠킨스 도메인으로 이용하도록 설정해주었다)

📂 /home/ubuntu/nginx/

📄 conf.d

```
## 젠킨스 서버
server {
    listen 80;
    server_name j10a406.p.ssafy.io;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name j10a406.p.ssafy.io;

    ssl_certificate /etc/cert/cert.pem;  # SSL 인증서 파일
    ssl_certificate_key /etc/cert/privkey.pem;  # SSL 키 파일
    ssl_trusted_certificate /etc/cert/chain.pem;

    location / {

            proxy_pass http://jenkins:8080;
            proxy_set_header Host $host:$server_port;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwa
rded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
```

```
            # Required for new HTTP-based CLI
            proxy_http_version 1.1;
            proxy_request_buffering off;
            proxy_buffering off; # Required for HTTP-based CLI
to work over SSL
            add_header 'X-SSH-Endpoint' 'j10a406.p.ssafy.io/' a
lways;
    }

}
```

▼ 4️⃣ Spring Cloud

discovery-service

- spring cloud service에서 관리하는 모든 서버들을 등록해서 서로 찾을 수 있도록 함

▼ application.yml

```
# server
server:
  port: 8761

spring:
  application:
    name: discoveryservice
  config:
    import:
      - optional:env.yml

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
  instance:
    hostname: ${discovery.host.ip}
```

▼ env.yml

```
discovery:
  host:
#    ip: localhost #로컬 개발 용
    ip: {server IP}
```

gateway-service

- 클라이언트 요청을 micro-service로 전달시켜주는 서버

▼ application.yml

```
server:
  port: 8000

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: ${eureka.host.url}

spring:
  application:
    name: gateway-service
  config:
    import:
      - optional:env.yml
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]':
            allowed-origins:
              - "http://localhost:8080"
              - "http://localhost:3000"
              - "https://edgescheduler.co.kr"
            allowed-methods:
              - GET
              - POST
              - PUT
              - PATCH
              - DELETE
              - OPTIONS
```

```
            allowed-headers:
              - "*"
            allow-credentials: true
      default-filters:
        - name: GlobalFilter
          args:
            baseMessage: Spring Cloud Gateway Global Filter
            preLogger: true
            postLogger: true
      routes:
        - id: notification-service
          uri: lb://NOTIFICATION-SERVICE
          predicates:
            - Path=/notification-service/**
          filters:
            - RewritePath=/notification-service/(?<segment>.
*), /$\{segment}
            - JwtAuthenticationFilter

        - id: schedule-service
          uri: lb://SCHEDULE-SERVICE
          predicates:
            - Path=/schedule-service/**
          filters:
            - RewritePath=/schedule-service/(?<segment>.*),
/$\{segment}
            - JwtAuthenticationFilter

        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user-service/uncheck
            - Method=GET
          filters:
            - RemoveRequestHeader=Cookie
            - RewritePath=/user-service/(?<segment>.*), /$\
{segment}
            - CustomFilter
            - LoggingFilter
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user-service/check
```

```
            - Method=GET
          filters:
            - RemoveRequestHeader=Cookie
            - RewritePath=/user-service/(?<segment>.*), /$\
{segment}
            - CustomFilter
            - LoggingFilter
            - JwtAuthenticationFilter
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user-service/auth/**
            - Method=GET
          filters:
            - RemoveRequestHeader=Cookie
            - RewritePath=/user-service/(?<segment>.*), /$\
{segment}
            - CustomFilter
            - LoggingFilter
            - JwtAuthenticationFilter
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user-service/members
            - Method=GET
          filters:
            - RemoveRequestHeader=Cookie
            - RewritePath=/user-service/(?<segment>.*), /$\
{segment}
            - CustomFilter
            - LoggingFilter
            - JwtAuthenticationFilter
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user-service/members/**
            - Method=GET
          filters:
            - RemoveRequestHeader=Cookie
            - RewritePath=/user-service/(?<segment>.*), /$\
{segment}
            - CustomFilter
            - LoggingFilter
```

```
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user-service/members/my/timezone
            - Method=PUT
          filters:
            - RemoveRequestHeader=Cookie
            - RewritePath=/user-service/(?<segment>.*), /$\
{segment}
            - CustomFilter
            - LoggingFilter
            - JwtAuthenticationFilter
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user-service/auth/token/refresh
            - Method=POST
          filters:
            - RemoveRequestHeader=Cookie
            - RewritePath=/user-service/(?<segment>.*), /$\
{segment}
            - CustomFilter
            - LoggingFilter
            - JwtAuthenticationFilterToken
```

▼ env.yml

```
token:
  secret: {JWT secret key}

eureka:
  host:
    url: http://{server IP}/eureka
```

▼ 4️⃣ DB & MQ

▼ (1) User-Service

MySQL

📄docker-compose.yaml

```
services:
  mysql:
    image: mysql:latest
```

```
    container_name: mysql-user-service
    restart: on-failure
    networks:
      - my-network
    volumes:
      - ./db:/var/lib/mysql
    ports:
      - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: {root password}
      MYSQL_DATABASE: {database name}
networks:
  my-network:
    external: true
```

Redis

📄docker-compose.yaml

```
services:
  redis: # container name
    image: redis:latest
    ports: # 바인딩할 포트:내부 포트
      - 6379:6379
    command: redis-server /usr/local/etc/redis/redis.conf
    volumes: # 마운트할 볼륨 설정
      - ${REDIS_DATA_PATH}:/data
      - ${REDIS_DEFAULT_CONFIG_FILE}:/usr/local/etc/redis/redis
    restart: on-failure
```

▼ (2) Schedule-Service

MySQL

📄docker-compose.yml

```
services:
  mysql:
    image: mysql:latest
    container_name: mysql
    restart: on-failure
    networks:
      - my-network
    volumes:
      - ./db:/var/lib/mysql
```

```
    ports:
      - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: {root password}
      MYSQL_DATABASE: {database name}
networks:
  my-network:
    external: true
```

Kafka Cluster

▼ 📄docker-compose.yml

```
services:
  zookeeper:
    image: confluentinc/cp-zookeeper
    container_name: zookeeper
    networks:
      - my-network
    ports:
      - "22181:2181"
    environment:
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_CLIENT_PORT: 2181
  kafka1:
    image: confluentinc/cp-kafka
    container_name: kafka1
    networks:
      - my-network
    depends_on:
      - zookeeper
    ports:
      - "19092:19092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAIN
TEXT,EXTERNAL:PLAINTEXT,LOCALHOST:PLAINTEXT
      KAFKA_LISTENERS: INTERNAL://0.0.0.0:9092,EXTERNAL://
0.0.0.0:19092,LOCALHOST://127.0.0.1:29092
      KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka1:9092,E
XTERNAL://kafka.edgescheduler.co.kr:19092,LOCALHOST://loca
lhost:29092
```

```
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
  kafka2:
    image: confluentinc/cp-kafka
    container_name: kafka2
    networks:
      - my-network
    depends_on:
      - zookeeper
    ports:
      - "19093:19093"
    environment:
      KAFKA_BROKER_ID: 2
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAIN
TEXT,EXTERNAL:PLAINTEXT,LOCALHOST:PLAINTEXT
      KAFKA_LISTENERS: INTERNAL://0.0.0.0:9093,EXTERNAL://
0.0.0.0:19093,LOCALHOST://127.0.0.1:29093
      KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka2:9093,E
XTERNAL://kafka.edgescheduler.co.kr:19093,LOCALHOST://loca
lhost:29093
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
  kafka3:
    image: confluentinc/cp-kafka
    container_name: kafka3
    networks:
      - my-network
    depends_on:
      - zookeeper
    ports:
      - "19094:19094"
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAIN
TEXT,EXTERNAL:PLAINTEXT,LOCALHOST:PLAINTEXT
      KAFKA_LISTENERS: INTERNAL://0.0.0.0:9094,EXTERNAL://
0.0.0.0:19094,LOCALHOST://127.0.0.1:29094
      KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka3:9094,E
XTERNAL://kafka.edgescheduler.co.kr:19094,LOCALHOST://loca
lhost:29094
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
  kafka-ui:
    image: provectuslabs/kafka-ui
```

```
    container_name: kafka-ui
    networks:
      - my-network
    depends_on:
      - kafka1
      - kafka2
      - kafka3
    ports:
      - "9090:8080"
    restart: always
    environment:
      KAFKA_CLUSTERS_0_NAME: "operation"
      KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: "kafka1:9092,kafk
a2:9093,kafka3:9094"
      KAFKA_CLUSTERS_0_ZOOKEEPER: "zookeeper:2181"
networks:
  my-network:
    external: true
```

▼ (3) Notification-Service

MongoDB

📄docker-compose.yml

```
services:
  mongo:
    image: mongo:latest
    container_name: mongo
    networks:
      - my-network
    restart: on-failure
    ports:
      - 27017:27017
    volumes:
      - ./mongodb:/data/db
    environment:
      - MONGO_INITDB_ROOT_USERNAME={root username}
      - MONGO_INITDB_ROOT_PASSWORD={root password}
networks:
  my-network:
    external: true
```

▼ 5️⃣ 백엔드 빌드

각 Spring 프로젝트 레포지토리의 root 위치에 `Dockerfile` , `docker-compose.yml` 파일이 위치.

📂 src/main/resources 위치에 필요한 파일(아래 서버별 파일 참고) 배치한 후, 아래 명령어를 통해 프로젝트 빌드 및 실행

```
chmod +x gradlew
./gradlew build -x test
docker compose up
```

▼ Spring Cloud Eureka 🌱

📄 Dockerfile

```
FROM openjdk:17-alpine

#build된 jar 파일
ARG JAR_FILE_PATH=build/libs

# 호스트의 JAR 파일을 컨테이너로 복사
COPY ${JAR_FILE_PATH}/service-discovery-0.0.1-SNAPSHOT.jar se
rvice-discovery.jar

# 실행시 사용할 환경 변수 설정 (예: 프로파일 설정)
# ENV SPRING_PROFILES_ACTIVE=dev,oauth

ENTRYPOINT ["java", "-jar", "./service-discovery.jar"]
```

📄 docker-compose.yml

```
services:
  spring-app:
    container_name: discovery-app
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 8761
    networks:
      - my-network

networks:
  my-network:
    external: true
```

▼ Spring Cloud Gateway 🌱

📄 Dockerfile

```
FROM openjdk:17-alpine

#build된 jar 파일
ARG JAR_FILE_PATH=build/libs

# 호스트의 JAR 파일을 컨테이너로 복사
COPY ${JAR_FILE_PATH}/gateway-service-0.0.1-SNAPSHOT.jar gate
way-service.jar

# 실행시 사용할 환경 변수 설정 (예: 프로파일 설정)
# ENV SPRING_PROFILES_ACTIVE=dev,oauth

ENTRYPOINT ["java", "-jar", "./gateway-service.jar"]
```

📄 docker-compose.yml

```
services:
  gateway-app:
    container_name: gateway-app
    build:
      context: .
      dockerfile: Dockerfile
    expose :
      - 8000
    networks:
      - my-network
networks:
  my-network:
    external: true
```

📂 src/main/resources

📄 env.yml

```
token:
  secret: {JWT secret key}

eureka:
```

```
  host:
    url: http://{server IP}/eureka
```

▼ User Service 🌱

📄 Dockerfile

```
FROM openjdk:17-alpine

#build된 jar 파일
ARG JAR_FILE_PATH=build/libs

# 호스트의 JAR 파일을 컨테이너로 복사
COPY ${JAR_FILE_PATH}/user-service-0.0.1-SNAPSHOT.jar user-se
rvice.jar

ENTRYPOINT ["java", "-jar", "./user-service.jar"]
```

📄 docker-compose.yml

```
services:
  user-app:
    container_name: user-app
    build:
      context: .
      dockerfile: Dockerfile
    ports:
     - "8080:8080"
    networks:
      - my-network
networks:
  my-network:
    external: true
```

▼ 📂 src/main/resources

📄 env.yml

```
redis:
  host: redis-user-service
  port: 6379

mysql:
  host: mysql-user-service
```

```
  port: 3306
  database: edge
  username: {root username}
  password: {root password}

EUREKA_SERVER_URL: http://{eureka server IP}:8761/eureka
SERVER_HOST: {eureka server IP}
# Kafka
KAFKA_BOOTSTRAP_SERVER_1: kafka.edgescheduler.co.kr:19092
KAFKA_BOOTSTRAP_SERVER_2: kafka.edgescheduler.co.kr:19093
KAFKA_BOOTSTRAP_SERVER_3: kafka.edgescheduler.co.kr:19094
kafka:
  topic:
    timezone-configured: timezone-configured
    member-created: member-created
```

📄 application-oauth.yml

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: {google client ID}
            client-secret: {google client secret}
            scope: profile,email
            redirect-uri: https://user-service.edgeschedul
er.co.kr/login/oauth2/code/google
server:
  forward-headers-strategy: native
```

▼ Schedule Service 🌱

📄 Dockerfile

```
FROM openjdk:17-alpine

#build된 jar 파일
ARG JAR_FILE_PATH=build/libs

# 호스트의 JAR 파일을 컨테이너로 복사
COPY ${JAR_FILE_PATH}/schedule-service-0.0.1-SNAPSHOT.jar sch
edule-service.jar
```

```
# 실행시 사용할 환경 변수 설정 (예: 프로파일 설정)
# ENV SPRING_PROFILES_ACTIVE=dev,oauth

ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=prod",
"./schedule-service.jar"]
```

📄 docker-compose.yml

```
services:
  schedule-app:
    container_name: schedule-app
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 8301:8301
    networks:
      - my-network

networks:
  my-network:
    external: true
```

📂 src/main/resources

📄 env.yml

```
# MySQL
MYSQL_HOSTNAME: mysql
MYSQL_PORT: 3306
MYSQL_DATABASE: {database name}
MYSQL_ROOT_PASSWORD: {root password}

# Kafka
KAFKA_BOOTSTRAP_SERVER_1: kafka1:9092
KAFKA_BOOTSTRAP_SERVER_2: kafka2:9093  # 안되면 9092 포트로
KAFKA_BOOTSTRAP_SERVER_3: kafka3:9094  # 안되면 9092 포트로
KAFKA_GROUP_ID: schedule-service
kafka:
  topic:
    timezone-configured: timezone-configured
    meeting-created: meeting-created
```

```
    meeting-updated: meeting-updated
    meeting-deleted: meeting-deleted
    attendee-response: attendee-response
    attendee-proposal: attendee-proposal

#Eureka
EUREKA_SERVER_URL: http://{eureka server IP}/eureka
SERVER_HOST: k10s201.p.ssafy.io
```

▼ Notification Service 🌱

📄 Dockerfile

```
FROM openjdk:17-alpine

#build된 jar 파일
ARG JAR_FILE_PATH=build/libs

# 호스트의 JAR 파일을 컨테이너로 복사
COPY ${JAR_FILE_PATH}/notification-service-0.0.1-SNAPSHOT.jar
notification-service.jar

# 실행시 사용할 환경 변수 설정 (예: 프로파일 설정)
# ENV SPRING_PROFILES_ACTIVE=dev,oauth

ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=prod",
"./notification-service.jar"]
```

📄 docker-compose.yml

```
services:
  notification-app:
    container_name: notification-app
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 8201:8201
    networks:
      - my-network

networks:
```

```
  my-network:
    external: true
```

📂 src/main/resources

📄 env.yml

```
# MongoDB
MONGO_HOST: mongo
MONGO_PORT: 27017
MONGO_INITDB_ROOT_USERNAME: {root username}
MONGO_INITDB_ROOT_PASSWORD: {root password}

# Kafka
KAFKA_BOOTSTRAP_SERVER_1: kafka.edgescheduler.co.kr:19092
KAFKA_BOOTSTRAP_SERVER_2: kafka.edgescheduler.co.kr:19093
KAFKA_BOOTSTRAP_SERVER_3: kafka.edgescheduler.co.kr:19094
KAFKA_GROUP_ID: notification-service
kafka:
  topic:
    timezone-configured: timezone-configured
    meeting-created: meeting-created
    meeting-updated: meeting-updated
    meeting-deleted: meeting-deleted
    attendee-response: attendee-response
    attendee-proposal: attendee-proposal
    member-created: member-created

# Mail
MAIL_USERNAME: {개인 이메일}
MAIL_PASSWORD: {Gmail 앱 비밀번호}

#Eureka
EUREKA_SERVER_URL: http://{eureka server IP}/eureka
SERVER_HOST: notification.edgescheduler.co.kr
```

▼ 6️⃣ 프론트엔드 빌드

```
//tsconfig.json

{
  "compilerOptions": {
    "lib": ["dom", "dom.iterable", "esnext"],
```

```
    "allowJs": true,
    "skipLibCheck": true,
    "strict": true,
    "noEmit": true,
    "esModuleInterop": true,
    "module": "esnext",
    "moduleResolution": "bundler",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "jsx": "preserve",
    "incremental": true,
    "plugins": [
      {
        "name": "next"
      }
    ],
    "paths": {
      "@/*": ["./src/*"]
    }
  },
  "include": ["next-env.d.ts", "**/*.ts", "**/*.tsx", ".next/types
  "exclude": ["node_modules"]
}
```

```
//package.json
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "test": "jest",
    "test:coverage": "jest --coverage",
    "test:bail": "jest --bail"
  },
  "dependencies": {
    "@emotion/react": "^11.11.4",
```

```
    "@types/react-beautiful-dnd": "^13.1.8",
    "date-fns": "^3.6.0",
    "eslint-config-react-app": "^7.0.1",
    "eslint-plugin-jest-dom": "^5.4.0",
    "event-source-polyfill": "^1.0.31",
    "next": "14.2.2",
    "react": "^18",
    "react-dom": "^18",
    "react-icons": "^5.1.0",
    "styled-components": "^6.1.8",
    "zustand": "^4.5.2"
  },
  "devDependencies": {
    "@babel/preset-env": "^7.24.5",
    "@babel/preset-react": "^7.24.1",
    "@babel/preset-typescript": "^7.24.1",
    "@testing-library/jest-dom": "^6.4.2",
    "@testing-library/react": "^15.0.6",
    "@testing-library/user-event": "^14.5.2",
    "@types/event-source-polyfill": "^1.0.5",
    "@types/jest": "^29.5.12",
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "eslint": "^8",
    "eslint-config-next": "14.2.2",
    "jest": "^29.7.0",
    "jest-environment-jsdom": "^29.7.0",
    "msw": "^2.2.14",
    "ts-jest": "^29.1.2",
    "typescript": "^5"
  }
}
```

```
//next.config.mjs

/** @type {import('next').NextConfig} */
const nextConfig = {
  compiler: {
    styledComponents: true,
  },
  reactStrictMode: false,
```

```
};

export default nextConfig;
```

```
// jest.config.js

const { pathsToModuleNameMapper } = require("ts-jest");
const { compilerOptions } = require("./tsconfig");

module.exports = {
  transform: {
    "^.+\\.(js|jsx|ts|tsx)$": ["babel-jest", { configFile: "./babe
  },
  testEnvironment: "jest-environment-jsdom",
  moduleNameMapper: {
    ...pathsToModuleNameMapper(compilerOptions.paths, { prefix: "<
    "^@next/font/google$": "<rootDir>/__mocks__/@next/font/google.
  },
};
```

```
// babel.config.jest.js

module.exports = {
  presets: ["@babel/preset-env", "@babel/preset-react", "@babel/pr
};
```

▼ 7️⃣ CI / CD

▼ 1. Jenkins

▼ 🔒KEY

🔑Jenkins admin 비밀번호

```
${JENKINS_ADMIN}
```

🔑GitLab Access Token

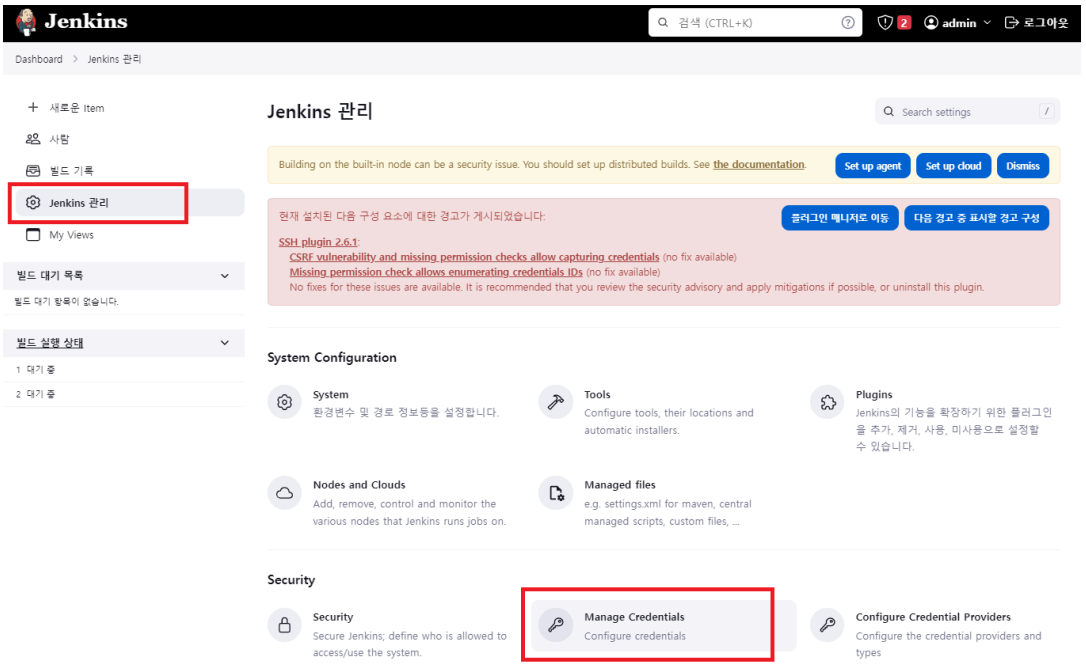
```
${GITLAB_ACCESS_TOKEN}
```

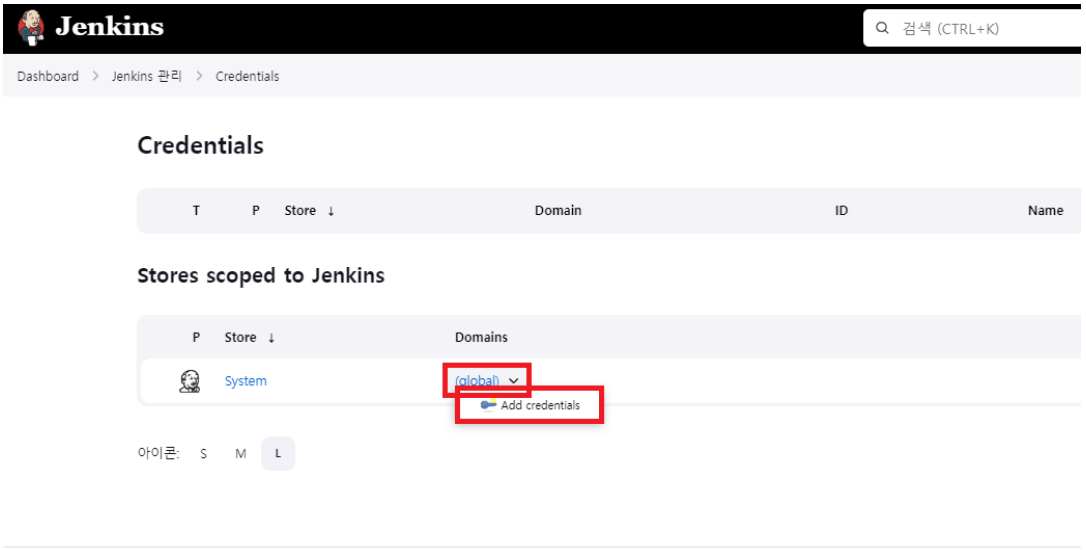
🔑Jenkins Spring 파이프 라인 secret key

```
${JENKINS_PIPELINE_SECRET_KEY}
```

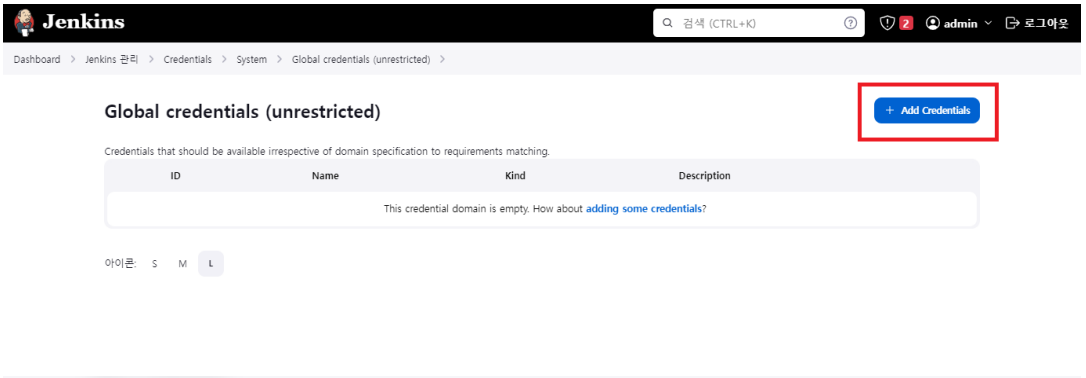
▼ (1) GitLab Credential

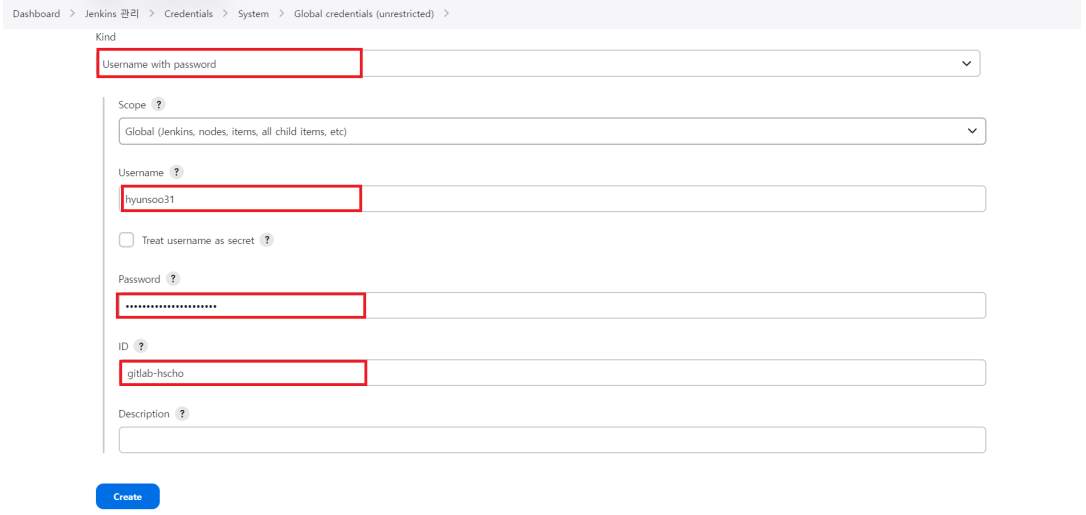
✅ Credential 등록





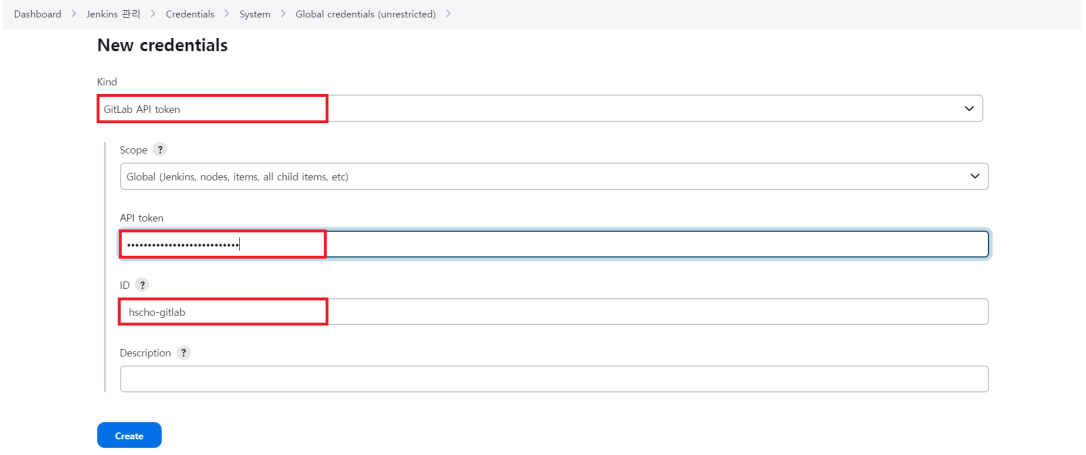
📍 GitLab (Username with password)





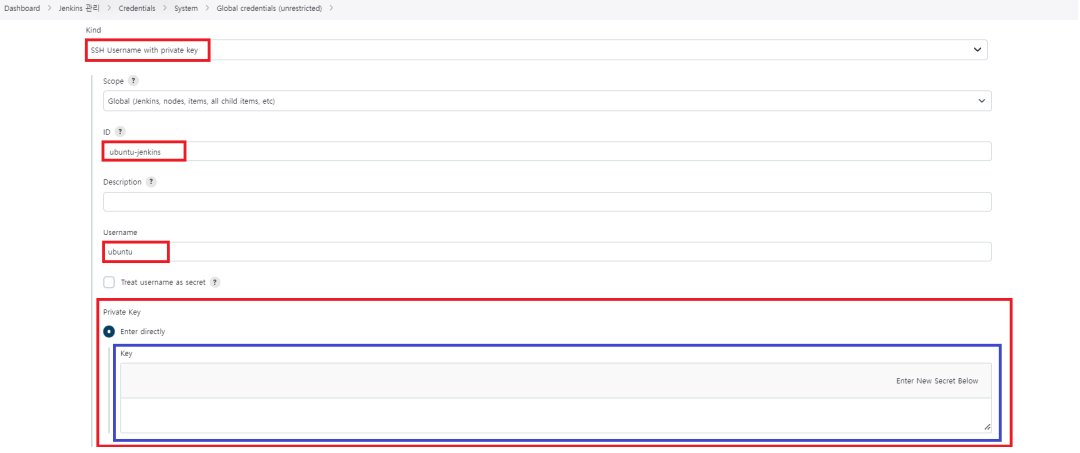
- **username : GitLab ID(로그인 아이디)**
- **username : Gitlab password**

📍 GitLab (API token)



- **API token: GitLab Access Token**

📍Ubuntu (SSH)



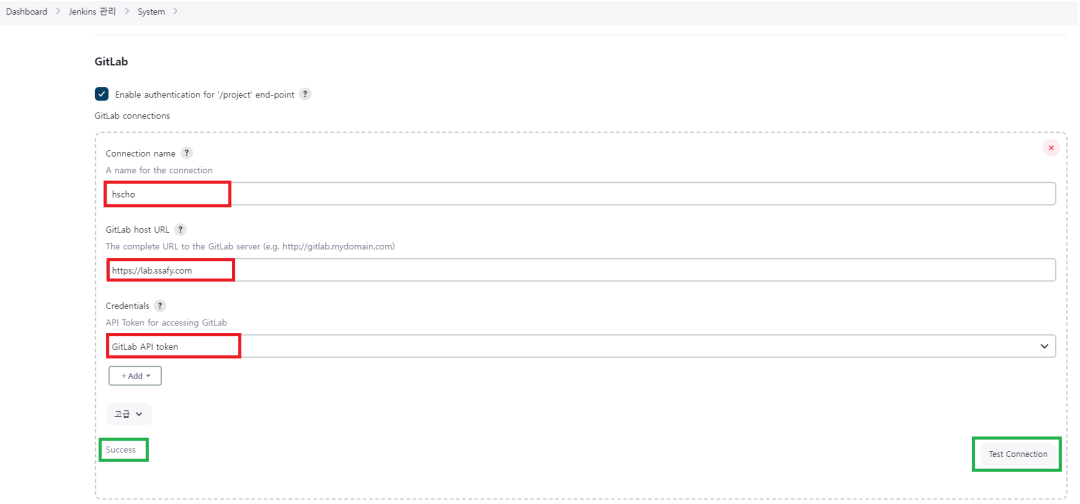
- **ID: Jenkins에서 Credential에 지정할 별칭**
- **Username: SSH 원격 서버 호스트에서 사용하는 계정이름**
- **Key: *.pem 키의 내용을 메모장으로 복사후 붙여넣기**

📍Credentials 등록 확인

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	ubuntu-jenkins	ubuntu
		System	(global)	gitlab-hscho	hyunsoo31/******
		System	(global)	hscho-gitlab	GitLab API token

✅GitLab과 연결



- **Connection name:** connection 이름 설정
- **GitLab host URL:** gitlab URL 작성
- **GitLab API token** 사용
- **Test Connection** 눌러서 Success 출력되는지 확인

▼ (2) Jenkins pipline

▼ FE

▼ Pipeline Script

```
    pipeline {
         agent any
         stages {

             stage('gitlab Connect'){
                 steps{
                     git branch: 'develop',
                     credentialsId: 'gitlab-hscho-id',
                     url: 'https://lab.ssafy.com/s10-f
inal/S10P31S201.git'
                 }
             }


             stage('deploy'){
                 steps{
                     sh 'docker stop next-app || true'
                     sh 'docker rm next-app || true '
                     sh 'docker rmi frontend-next-app
```

```
|| true'

                    dir('frontend/'){
                        script{

                            // 도커 컴포즈 파일 경로 지정
                            def dockerComposeFile =
'docker-compose.yml'

                            // 도커 컴포즈 실행 명령어
                            def dockerComposeCmd = "
docker compose up -d"

                            // 도커 컴포즈 실행
                            sh """
                                ${dockerComposeCmd}
                            """
                        }
                    }
                }

            }

        }
```

▼ BE

▼ Pipeliine Script

user-service

```
pipeline {
    agent any

    stages {

        stage('gitlab Connect'){
            steps{
                git branch: 'develop',
                credentialsId: 'gitlab-hscho-id',
                url: 'https://lab.ssafy.com/ms-s201/use
            }
```

```
        }

        stage('build'){
            steps{
                script{
                    buildSpring()
                }
            }
            post {
                success {
                   echo 'Successfully Jar build'
                }
                failure {
                   error 'Jar build is failed'
                }
            }
        }


        stage('deploy'){
            steps{
                script{
                    deploySpring()
                }
            }
        }
    }
}

def buildSpring() {
    sh 'cp -r /var/jenkins_home/backend/env/env.yml /va
    sh 'cp -r /var/jenkins_home/backend/env/application
    sh 'chmod +x gradlew'
    sh './gradlew build -x test'
}

def deploySpring() {
    sh 'docker stop user-app || true'
    sh 'docker rm user-app || true '
    sh 'docker rmi user-service-user-app || true'
    sh "docker compose up -d"
}
```

notification-service

```
pipeline {
    agent any

    stages {

        stage('gitlab Connect'){
            steps{
                git branch: 'develop',
                credentialsId: 'gitlab-hscho-id',
                url: 'https://lab.ssafy.com/ms-s201/n
otification-service.git'
            }
        }

        stage('build'){
            steps{
                script{
                    buildSpring()
                }
            }
            post {
                success {
                   echo 'Successfully Jar build'
                }
                failure {
                   error 'Jar build is failed'
                }
            }
        }


        stage('deploy'){
            steps{
                script{
                    deploySpring()
                }
            }
        }
    }
```

```
}

def buildSpring() {
    sh 'cp -r /var/jenkins_home/backend/env/env.yml /
var/jenkins_home/workspace/notification-service/src/m
ain/resources/'
    sh 'chmod +x gradlew'
    sh './gradlew build -x test'
}

def deploySpring() {
    sh 'docker stop notification-app || true'
    sh 'docker rm notification-app || true '
    sh 'docker rmi notification-service-notification-
app || true'
    sh "docker compose up -d"
}
```

gateway-service

```
pipeline {
    agent any

    stages {

        stage('gitlab Connect'){
            steps{
                git branch: 'develop',
                credentialsId: 'gitlab-hscho-id',
                url: 'https://lab.ssafy.com/ms-s201/g
ateway-service.git'
            }
        }

        stage('build'){
            steps{
                script{
                    buildSpring()
                }
            }
            post {
```

```
                success {
                   echo 'Successfully Jar build'
                }
                failure {
                   error 'Jar build is failed'
                }
            }
        }


        stage('deploy'){
            steps{
                script{
                    deploySpring()
                }
            }
        }
    }
}

def buildSpring() {
    sh 'pwd'
    sh 'ls -la'
    sh 'ls -ld /var/jenkins_home/'
    sh 'ls -la /var/jenkins_home/backend/'
    sh 'cp -r /var/jenkins_home/backend/env/env.yml /
var/jenkins_home/workspace/gateway-service/src/main/r
esources/'
    sh 'chmod +x gradlew'
    sh './gradlew build -x test'
}

def deploySpring() {
    sh 'docker stop gateway-app || true'
    sh 'docker rm gateway-app || true '
    sh 'docker rmi gateway-service-gateway-app || tru
e'
    sh "docker compose up -d"
}
```

schedule-service

```
pipeline {
    agent any

    stages {

        stage('gitlab Connect'){
            steps{
                git branch: 'develop',
                credentialsId: 'gitlab-hscho-id',
                url: 'https://lab.ssafy.com/ms-s201/s
chedule-service.git'
            }
        }

        stage('build'){
            steps{
                script{
                    buildSpring()
                }
            }
            post {
                success {
                   echo 'Successfully Jar build'
                }
                failure {
                   error 'Jar build is failed'
                }
            }
        }


        stage('deploy'){
            steps{
                script{
                    deploySpring()
                }
            }
        }
    }
}

def buildSpring() {
```

```
    sh 'cp -r /var/jenkins_home/backend/schedule/env/
env.yml /var/jenkins_home/workspace/schedule-service/
src/main/resources/'
    sh 'chmod +x gradlew'
    sh './gradlew build -x test'
}

def deploySpring() {
    sh 'docker stop schedule-app || true'
    sh 'docker rm schedule-app || true '
    sh 'docker rmi schedule-service-schedule-app || t
rue'
    sh "docker compose up -d"
}
```

▼ 2. GitLab Webhook

▼ 공통

▼ Secret Token

```
${GITLAB_SECRET_TOKEN}
```

▼ Trigger

▼ FE

Webhook URL

```
https://jenkins.edgescheduler.co.kr/project/frontend
```

▼ BE

▼ Gateway Service

```
https://jenkins-lg-3.edgescheduler.co.kr/project/gateway-service
```

▼ **User Service**

```
https://jenkins-lg-2.edgescheduler.co.kr/project/user-service
```

▼ **Schedule Service**

```
https://jenkins-xl.edgescheduler.co.kr/project/schedule-service
```

▼ **Notification Service**

```
https://jenkins-lg-1.edgescheduler.co.kr/project/notification-service
```