



포팅메뉴얼

1. 개발환경
 - 1.1 Frontend
 - 1.2 Backend
 - 1.3 Server
 - 1.4 Database
 - 1.5 AI
 - 1.5 UI/UX
 - 1.6 IDE
 - 1.7 형상관리
2. 환경변수
 - 2.1 Frontend
 - 2.2 Backend
3. EC2 환경설정
 - 3.1 SSL 발급
 - 3.2 Nginx 설정
 - 3.3 UFW(방화벽) 설정
 - 3.4 EC2 사용 포트
4. Docker Container
 - 4.1 docker, docker compose 설치
 - 4.2 mysql
 - 4.3 redis
 - 4.4 openvidu
5. Jenkins 설정
 - 5.1 Jenkins 도커 이미지 pull
 - 5.2 Docker compose 파일 작성
 - 5.3 Jenkins 컨테이너 생성
 - 5.4 Jenkins 설정 + gitlab 연동 설정
6. 빌드 및 배포
 - 6.1 backend
 - 6.2 frontend

개발환경

1.1 Frontend

- node.js 20.15.0

- pnpm 9.6.0
- react 18.3.1
- redux 9.1.2
- tensorflow-models/posenet 2.2.2
- tensorflow/tfjs 4.20.0
- axios 1.7.3
- stomp/stompjs 7.0.0
- openvidu-browser 2.30.1
- socket.io-clinet 4.7.5
- sockjs-clinet 1.6.1

1.2 Backend

- JAVA OpenJDK 1.7.0
- Spring Boot 3.3.1
 - Spring Data JPA 3.2.1
 - Spring Security 6.2.1
 - Junit5
 - Lombok 1.18.30
 - Openvidu 2.30.0
 - Spring WebSocket: 6.1.2

1.3 Server

- Ubuntu 20.04.6 LTS
- Docker 27.1.1
- Docker-compose 1.27.4
- Nginx 1.18.0

1.4 Database

- mysql 8.4.3
- redis 1.4.3

1.5 AI

- Python 3.9
- TensorFlow 2.15.0
- Keras: TensorFlow에 내장된 Keras API 사용
- Scikit-learn 1.0.2
- TensorFlow.js 4.20.0
- MediaPipe 0.8.10
- OpenCV 4.5.5.64

1.6 UI/UX

- Figma

1.7 IDE

- visual studio code
- IntelliJ IDEA
- Mysql Workbench

1.8 형상관리

- Gitlab
- Jira

환경변수

Frontend

Backend

- datasource
 - driver-class-name
 - url
 - username
 - password
- redis
 - port
 - host
 - password
- jwt
 - key
 - live
 - atk
 - rtk
- mail
 - host
 - id
 - password
 - name
 - email
- openvidu
 - url
 - secret

EC2 설정

3.1 SSL 발급

```
#Encrypt 설치
sudo apt-get install letsencrypt

# Certbot 설치
sudo apt-get install certbot python3-certbot-nginx

sudo systemctl stop nginx
sudo certbot --nginx
sudo certbot --nginx -d [도메인 혹은 ip 주소]

# nginx 설정 적용
sudo service nginx restart
sudo systemctl reload nginx
```

3.2 Nginx 설정

```
# Nginx 설치
sudo apt update
sudo apt upgrade
sudo apt install nginx
sudo service nginx start
sudo service nginx status
```

```
cd etc/nginx/sites-available
vi configure
```

```
# 시간 당 request 비율 제한 (클라이언트 IP에 대한 요청 1초에 최대 5개)
limit_req_zone $binary_remote_addr zone=ddos_req:10m rate=5r/s;

server {
    location /jenkins/ { # jenkins는 https로 포워딩 안 되도록 막음
        proxy_pass http://localhost:9005/jenkins/;
        proxy_redirect off;
        limit_req zone=ddos_req burst=10;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host:$server_port;
        proxy_set_header X-Forwarded-Proto http;
        proxy_set_header X-Forwarded-Port "443";
        proxy_set_header X-Forwarded-Host $http_host;
    }

    location / {
        proxy_pass http://localhost:3000;
        limit_req zone=ddos_req burst=10;
    }

    location /api {
        proxy_pass http://localhost:8081/api;
        limit_req zone=ddos_req burst=10;
    }

    location /ws { # websocket reverse proxy 설정
        proxy_pass http://localhost:8081/ws;
    }
}
```

```

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;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;

# WebSocket 연결 유지를 위한 타임아웃 설정
proxy_read_timeout 300s;
proxy_send_timeout 300s;
}

location /actuator { # 모니터링 시스템을 위한 reverse proxy
    proxy_pass http://localhost:8081/actuator;
}

listen 443 ssl;
ssl_certificate /etc/letsencrypt/live/i11e104.p.ssafy.io/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/i11e104.p.ssafy.io/privkey.pem;
}

server {
    if ($host = i11e104.p.ssafy.io) {
        return 301 https://$host$request_uri;
    }
    listen 80;
    server_name i11e104.p.ssafy.io;
    return 404;
}

```

3.3 UFW(방화벽) 설정

```

sudo ufw allow 8080 # 포트 개방

sudo ufw enable # 방화벽 활성화
sudo ufw status # 방화벽 상태 확인
sudo ufw deny 8080 # 방화벽 닫기

```

3.4 EC2 사용 포트

22	ALLOW	Anywhere
8989	ALLOW	Anywhere
443	ALLOW	Anywhere
9005	ALLOW	Anywhere
80	ALLOW	Anywhere
3000	ALLOW	Anywhere
3478	ALLOW	Anywhere
8443	ALLOW	Anywhere
6379	ALLOW	Anywhere
22/tcp	ALLOW	Anywhere
8442	ALLOW	Anywhere
3306	ALLOW	Anywhere
8080	ALLOW	Anywhere

8081	ALLOW	Anywhere
8081/tcp	ALLOW	Anywhere
8084	ALLOW	Anywhere
9090	ALLOW	Anywhere
3030	ALLOW	Anywhere
9104	ALLOW	Anywhere

Docker Container

4.1 Docker 설치

```
# 만약 도커가 이미 설치돼 있다면
sudo apt remove docker docker-engine docker.io containerd runc

sudo apt install apt-transport-https ca-certificates curl software-properties-common
sudo wget -qO- https://get.docker.com/ | sh

#도커 서비스 실행하기 및 부팅 시 자동 실행 설정
sudo systemctl start docker
sudo systemctl enable docker

#docker 설치 확인
docker -v
```

4.2 Mysql

```
# mysql image pull
docker pull mysql
```

docker-compose.yml

```
services:
  mysql:
    image: mysql:8
    command : --lower_case_table_names=1
    environment:
      MYSQL_ROOT_PASSWORD: e104_taffy_best
      MYSQL_DATABASE: myapp
      MYSQL_USER: user_taffy
      MYSQL_PASSWORD: e104_taffy_test
    volumes:
      - mysql-data:/var/lib/mysql
    ports:
      - "3306:3306"
    networks:
      - app-network:
          driver: bridge
          volumes:
            mysql-data:
```

4.3 Redis

```
# redis image pull
docker pull redis
```

docker-compose.yml

```
redis:
  image: redis:latest
  command: redis-server --requirepass e104_redis_taffy
  ports:
    - "6379:6379"
  hostname: redis
  volumes:
    - redis-data:/data
  networks:
    - app-network

networks:
  app-network:
    driver: bridge

volumes:
  redis-data:
```

4.4 Openvidu

```
# OpenVidu 서버 생성
sudo su
cd /opt
curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/
install_openvidu_latest.sh | bash
# OpenVidu 서버 설정
cd openvidu
vi .env
./openvidu start
# .env 파일 설정
# DOMAIN_OR_PUBLIC_IP=도메인명
# OPENVIDU_SECRET=내 비밀번호
# CERTIFICATE_TYPE=letsencrypt
# (이렇게 해야 SSL 인증서 생성가능; 하지만 도메인이 있어야함, IP주소 안됨)
# LETSENCRYPT_EMAIL=내 이메일
# HTTPS_PORT=8443
```

```
# 테스트
# https://내도메인:8443/dashboard/ 접속
# user: OPENVIDUAPP / pass: 설정한 OPENVIDU_SECRET 값
# 4. HTTP_PORT 변경 (최초 SSL 인증서 발급 후 가능)
# vi .env
# HTTP_PORT=8442 (기본 http(80), https(443) 포트는 향후
# nginx reverse proxy server를 위해 비워줘야 함)
```

Jenkins 설정

5.1 Jenkins 도커 이미지 pull

```
sudo docker pull jenkins/jenkins:lts
# 내려받은 이미지 확인
sudo docker images
```

5.2 Docker compose 파일 작성

docker-compose.yml

```
version: "3"
services:
  jenkins:
    image: jenkins/jenkins:lts //내가 받은 image 이름
    container_name: jenkins //image를 통해 띄울 container 이름
    user: root
    environment: //나중에 jenkins 접속할 때 url에 붙일 prefix
      - JENKINS_OPTS="--prefix=/jenkins"
      - JENKINS_ARGS="--prefix=/jenkins"
    volumes:
      - /var/jenkins_home:/var/jenkins_home //host의 var/jenkins_home 과
        // jenkins container의 var/jenkins_home 디렉토리를 마운트한다.
      - /var/run/docker.sock:/var/run/docker.sock //소켓마운트
    ports:
      - 9005:8080 //host로부터 9005로 들어오는 요청을 container의 8080으로 보내겠다.
```

volumes 를 설정하는 이유는 container와 호스트 간에 sock 연결을 통해 container 안에서도 docker 명령어를 사용할 수 있도록 함. 그리고 디렉토리를 마운트함으로써 컨테이너에 접속하지 않아도 host 시스템의 디렉토리 구조에 동일한 파일, 디렉토리가 만들어짐(복사)

5.3 Jenkins 컨테이너 생성

```
# docker container 생성
docker compose up -d
# 실행중인 컨테이너 확인 (뒤에 -a 붙이면 모든 컨테이너 확인)
docker ps [-a]
```

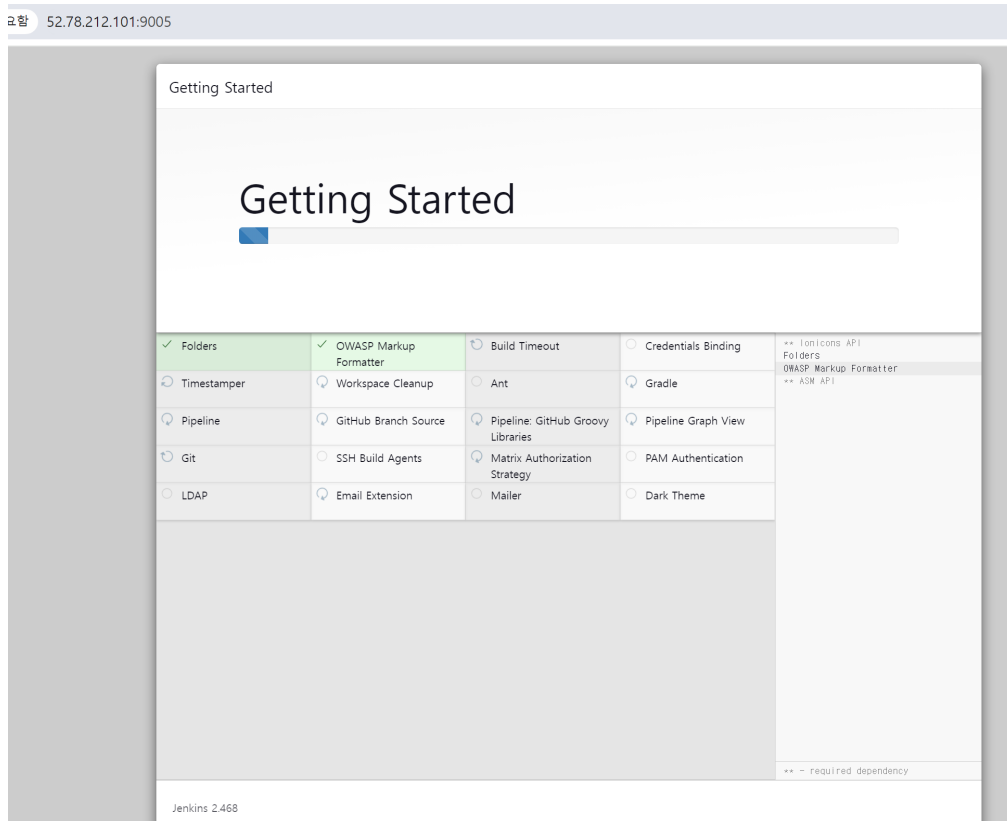
5.3.1 Jenkins 접속

```
# 젠킨스 컨테이너 비밀번호 확인 명령어
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword

# 젠킨스 컨테이너로 접속해서 도커 명령어 실행 여부 확인 명령어
docker exec -it <container_name_or_id> /bin/bash
docker exec -it jenkins /bin/bash

# 젠킨스 컨테이너에 접속해서 Docker 명령어 되는지 확인
docker
```

초기 비밀번호로 Jenkins 에 접속 후 플러그인 표준으로 설치



5.3.2 계정 등록

Getting Started

Create First Admin User

계정명

암호

암호 확인

이름

이메일 주소

Jenkins 2.462.1

[Skip and continue as admin](#) [Save and Continue](#)

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 2.462.1

Not now

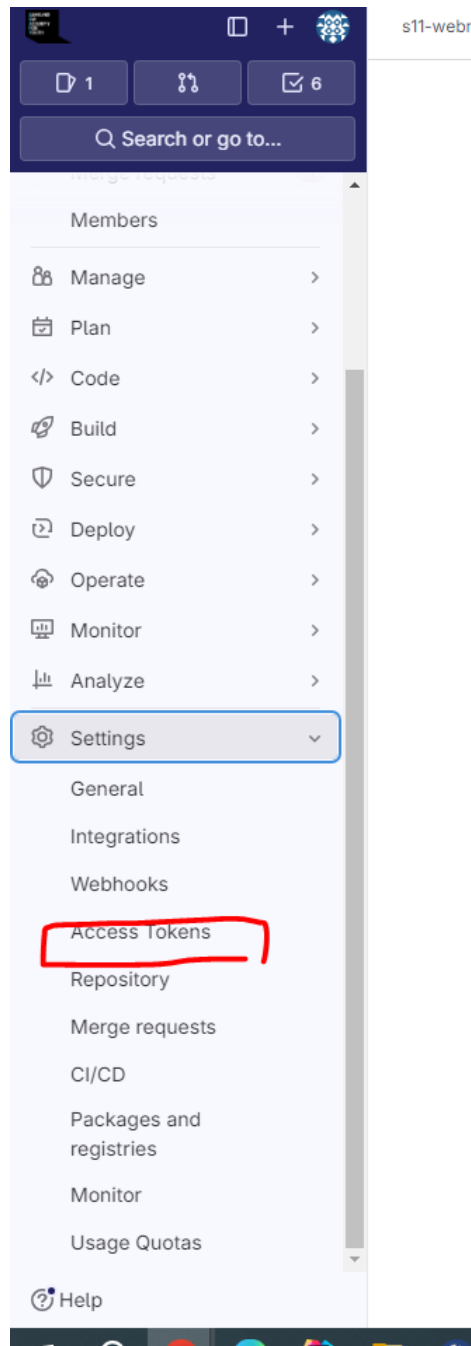
Save and Finish

5.4 Jenkins - gitlab 연동

- 우리는 젠킨스가 git push를 감지할 때 마다 해당 파이프라인을 실행시킬 것이다. 그러기 위해서는 gitlab의 동작을 캐치할 수 있는 Webhook이라는 것이 필요하다.

1. gitlab token 발급

gitlab의 project에 들어오면 이렇게 project setting이 있다. 여기에서 **Access Tokens** 로 들어간다. **만약 이 항목이 표시되지 않는다?** **Manage → Member** 에서 자신의 권한을 확인한 후 maintainer 이상 권한을 가진 사람에게 maintainer 권한 요청을 한다.



add token해서 해당 project에 접근할 수 있는 토큰을 생성한다.

Project Access Tokens

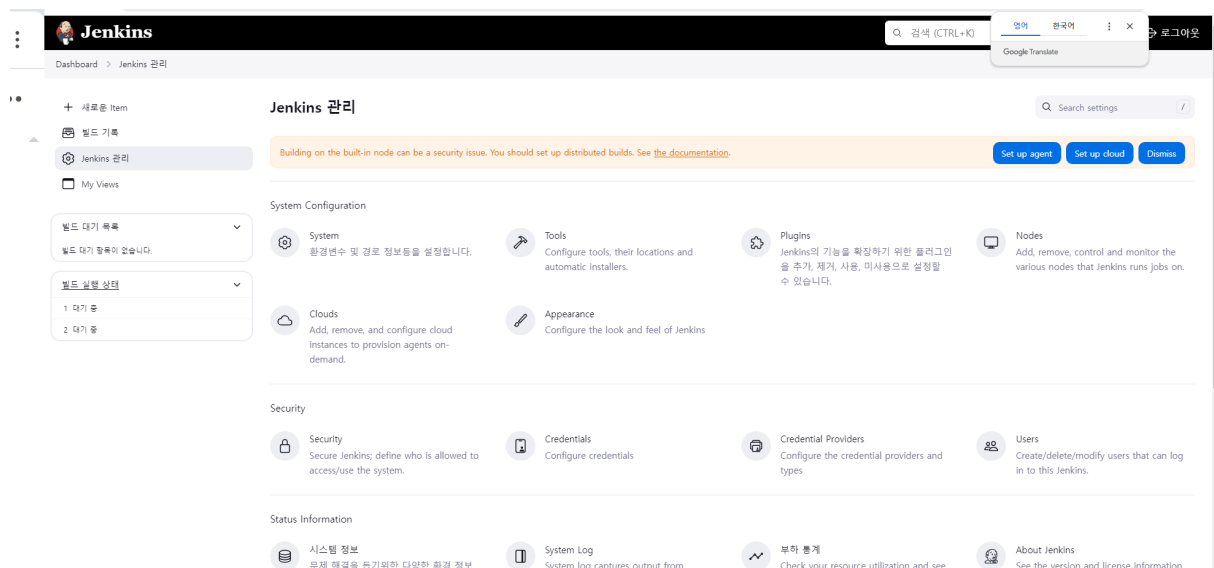
Generate project access tokens scoped to this project for your applications that need access to the GitLab API. You can also use project access tokens with Git to authenticate over HTTP(S). [Learn more.](#)

Active project access tokens 2 Add new token

Token name	Scopes	Created	Last Used	Expires	Role	Action
gitlab-jenkins-token	api_read_api, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Jul 22, 2024	3 weeks ago	in 1 week	Maintainer	
gitlab_token	api_read_api, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Aug 01, 2024	1 week ago	in 2 weeks	Maintainer	

2. jenkins gitlab plugin설치

이번에는 jenkins에서 이 gitlab token을 인식할 수 있도록 하는 plugin을 설치한다. jenkins 관리 → plugins



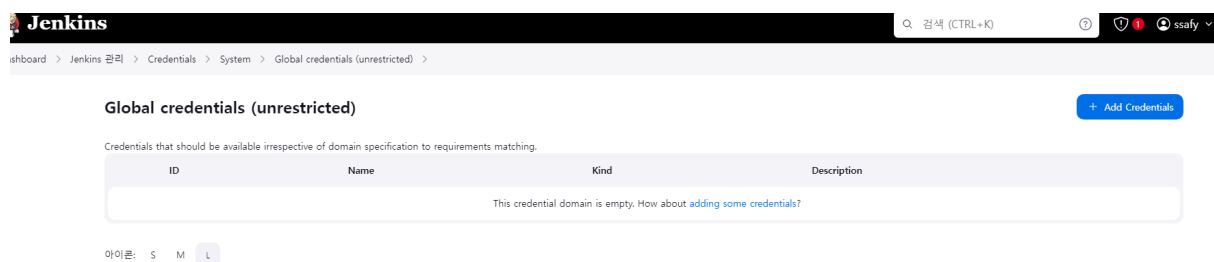
Installed plugins 에서 **gitlab** 검색해서 설치



3. gitlab api token credentials 등록

token을 이용해서 gitlab에 접근할 수 있는 권한을 생성

jenkins 관리 → credentials



add credentials를 이용하여 권한을 추가

New credentials

Kind
 GitLab API token

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

API token

ID ?
 access_token

Description ?

Create

kind : GitLab API token

API token : 이전에 발급받았던 gitlab access token

ID : 각자 알아서 사용할 값

gitlab access token 발급 : user Settings의 Access Tokens 에서 추가 가능

User settings

- Profile
- Account
- Applications
- Chat
- Access Tokens**
- Emails
- Password
- Notifications
- SSH Keys
- GPG Keys
- Preferences
- Comment Templates
- Active Sessions
- Authentication Log
- Usage Quotas
- Help

Search settings

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API. You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Active personal access tokens 10 [Add new token](#)

Token name	Scopes	Created	Last Used	Expires	Action
jenkins-2t	api, read_api, read_user, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Jul 16, 2024	3 weeks ago	in 1 day	
gitlab-jenkins-test	api, read_api, read_user, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Jul 16, 2024	Never	in 1 day	
IntelliJ IDEA GitLab Integration Plugin	api, read_api, read_user, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Jul 23, 2024	3 days ago	in 1 week	
personal_token	api, read_api, read_user, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Jul 25, 2024	24 minutes ago	in 1 week	
taffy-jenkins-personal	api, read_api, read_user, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Jul 24, 2024	2 weeks ago	in 2 weeks	
ssh-token	api, read_api, read_user, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Aug 01	1 week ago	in 2 weeks	

4. gitlab 계정 등록

Dashboard > Jenkins 관리 > Credentials > System > Global credentials (unrestricted) >

New credentials

Kind: Username with password

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

Username:

☐ Treat username as secret

Password:

ID:

Description:

Create

username : 본인의 gitlab 계정(이메일)

password : 깃랩 계정의 access token

5. gitlab 계정 연결 테스트

jenkins 관리 → System → 밑에 쪽 내리다보면 **GitLab** 부분

GitLab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name: A name for the connection

taffy

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

https://lab.ssfy.com/

Credentials: API Token for accessing GitLab

GitLab API token

+ Add

고급

Success

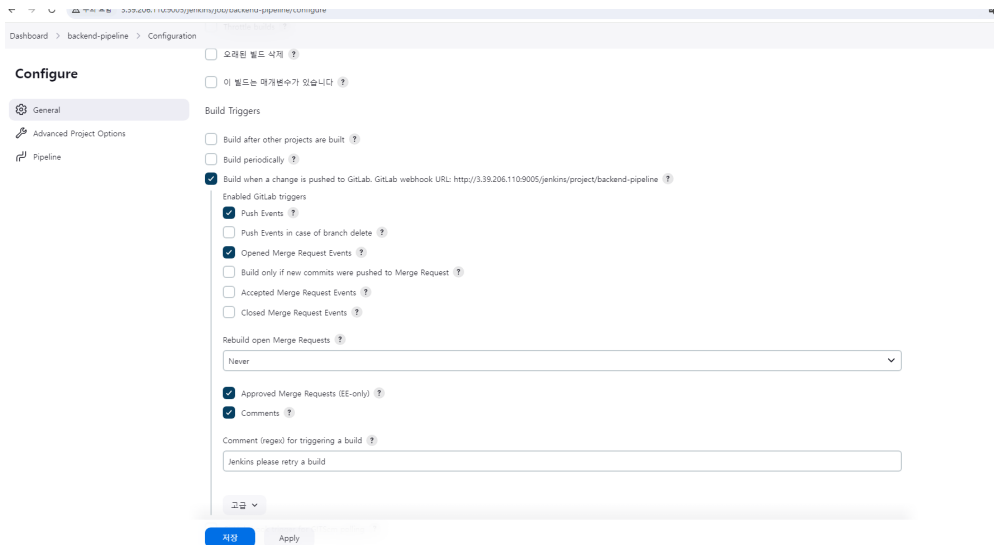
Test Connection

gitlab host URL은 본인이 사용하고 있는 gitlab 주소, Credentials 는 방금 설정한 GitLab API token 을 설정해서 **test connection** 시 Success 가 나오면 성공

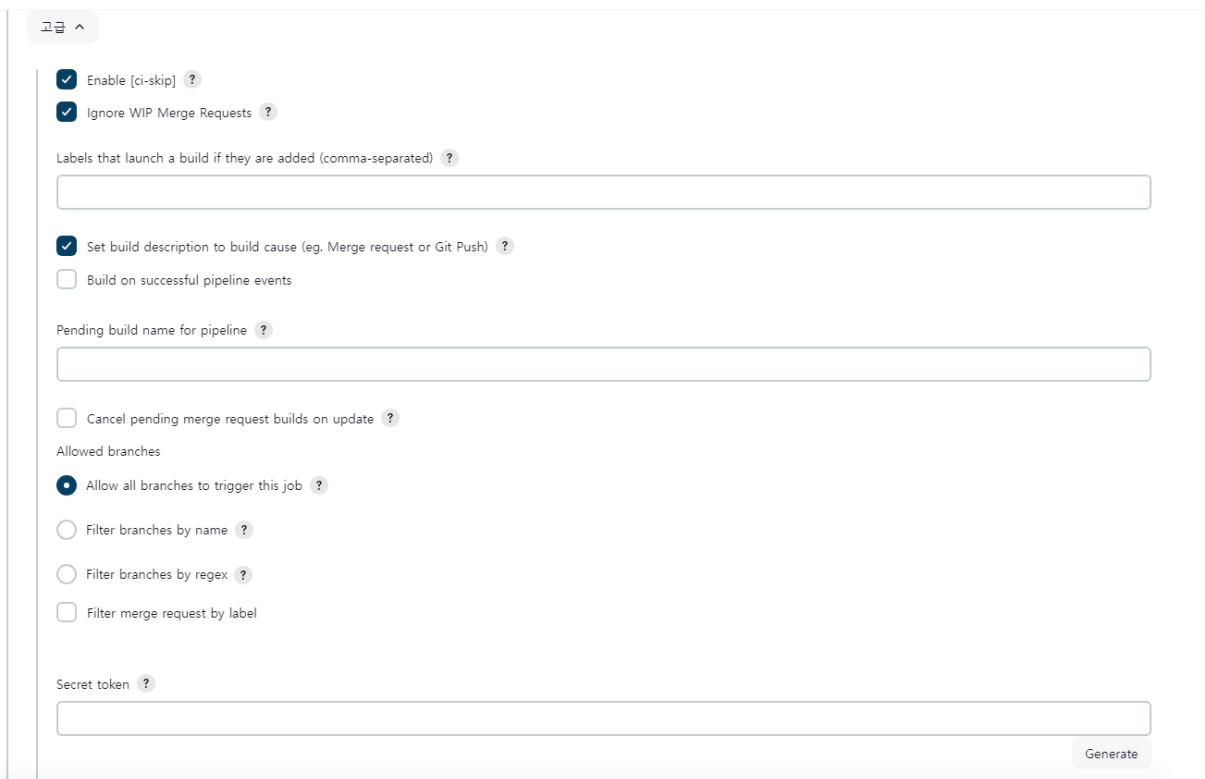
6. pipeline 생성하기

dashboard → All → New Item 에서 item name 설정하고 pipeline 클릭

build triggers에서 push 될 때마다 pipeline이 실행되도록 설정



고급 에서 secret token 발급 → 복사해두기



7. gitlab과 연동하기

복사한 secret token 을 gitlab webhook에다가 사용
gitlab의 setting → webhook → add new webhook

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://3.39.206.110:9005/jenkins/project/backend-pipeline> ?
 - 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 ?

요거를 URL 로 사용하고, 발급받은 secret token을 붙여넣기

Webhooks

⚠ Secret token will be cleared on save unless token is updated.

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

☒ Show full URL
☐ Mask portions of URL
Do not show sensitive data such as tokens in the UI.

Custom headers </> 0 Add custom header

No custom headers configured.

Name (optional)

Description (optional)

Secret token

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

Trigger
☒ Push events

trigger push events 선택해주고 add webhook

8. 간단한 pipeline 작성

Pipeline

Definition
Pipeline script

Script ?

```
1- pipeline {
2-   agent: any
3-   stages {
4-     stage('Git Clone'){
5-       steps {
6-         git branch: 'master', credentialsId: 'gitlab-account', url: 'https://lab.ssfy.com/dongji_11/gitlab-jenkins.git'
7-       }
8-     }
9-   }
10- }
11- }
```

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

9. webhook이 잘 동작하는지 테스트

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

Project Hooks 1

[Add new webhook](#)

[http://3.39.206.110:9005/jenkins/project/backend-pipeline](#)

[Push events](#) [SSL Verification: enabled](#)

Test Edit Delete

Push events

Tag push events

Issues events

Confidential issues events

Comments

Confidential comments

Merge request events

Job events

Pipeline events

Wiki page events

gitlab webhooks 에서 등록된 project hook에 trigger를 줄 수 있다.

Dashboard > backend-pipeline > #1

Status

Changes

Console Output

Edit Build Information

Delete build '#1'

Polling Log

Timings

Git Build Data

Pipeline Overview

Pipeline Console

Restart from Stage

Replay

Pipeline Steps

Workspaces

Build #1 (2024. 8. 13. 오전 7:42:09)

Started by GitLab push by 김재경

Triggered by GitLab Webhook

This run spent:

- 5.7 sec waiting;
- 10 sec build duration;
- 16 sec total from scheduled to completion.

git

Revision: a19e67efee4a3a7889fc3a67c18d6df583c5e04d

Repository: https://lab.ssafy.com/dongji_11/gitlab-jenkins.git

refs/remotes/origin/master

성공

이렇게 git clone 된 프로젝트는 `/var/jenkins_home/workspace` 에서 확인할 수 있다.

빌드 및 배포

6.1 Backend

- 전체코드

```
pipeline {
  agent any
  stages {
```

포팅메뉴얼

17

```

stage('Git Clone'){
    steps {
        git branch: 'develop-be', credentialsId: 'gitlab-account',
            url: 'https://lab.ssafy.com/s11-webmobile1-sub2/S11P12E104.git'
    }
    post {
        failure {
            echo 'Repository clone failure !'
        }
        success {
            echo 'Repository clone success !'
        }
    }
}

stage('build with gradle') {
    steps {
        sh 'pwd'
        sh 'mkdir -p backend/src/main/resources'
        sh 'cp ../settings/application.yml backend/src/main/resources/application.yml'
        sh 'chmod +x ./backend/gradlew'
        sh "cd backend && ./gradlew clean build -x test"
    }
}

stage('Docker Hub Login'){
    steps{
        withCredentials([usernamePassword(credentialsId: 'DOCKER_USER',
            passwordVariable: 'DOCKER_PASSWORD', usernameVariable: 'DOCKER_USERNAME')]) {
            sh '''
            echo "$DOCKER_PASSWORD" | docker login -u
            $DOCKER_USERNAME --password-stdin
            '''
        }
    }
}

stage('Docker Build and Push') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'DOCKER_HUB', passwordVariable: 'DOCKER_PASSWORD', usernameVariable: 'DOCKER_USERNAME')]) {
            sh '''
            cd ./backend && docker build -f Dockerfile -t
            $DOCKER_REPO/$DOCKER_PROJECT .
            '''
            sh 'cd ./backend && docker push $DOCKER_REPO/$DOCKER_PROJECT'
            echo 'docker push Success!!'
        }
        echo 'docker push Success!!'
    }
}

stage('Deploy') {
    steps {
        sh 'docker rm -f backend-app || true'
        sh '''
        docker stop backend-app || true
        docker rm backend-app || true
        '''
    }
}

```

```

        docker run -d --name backend-app -p 8081:8080 dongji11/taffy
    },
    },
    },
}

```

1. 프로젝트 clone 받기

```

stage('Git Clone'){
    steps {
        git branch: 'develop-be', credentialsId: 'gitlab-account',
        url: 'https://lab.ssafy.com/s11-webmobile1-sub2/S11P12E104.git'

        post {
            failure {
                echo 'Repository clone failure !'
            }
            success {
                echo 'Repository clone success !'
            }
        }
    }
}

```

내 gitlab repo에서 project 를 clone 한다. 만약 실패하면 repository clone failure ! 를, 성공하면 Repository clone success ! 를 출력

2. build

```

stage('build with gradle') {
    steps {
        sh 'pwd'
        sh 'mkdir -p backend/src/main/resources'
        sh 'cp ../settings/application.yml backend/src/main/resources/application.yml'
        sh 'chmod +x ./backend/gradlew'
        sh "cd backend && ./gradlew clean build -x test"
    }
}

```

gradle 을 통해서 build를 하는 코드

gitignore 에 application.yml을 포함 시켜 놓았기 때문에, ec2 서버 내에 application.yml 을 따로 기술하고, build 시에 해당 project로 copy하는 방법을 사용

docker hub에 spring boot build image를 올리기 때문에 추가적인 설정이 더 필요

3. docker hub login

```

stage('Docker Hub Login'){
    steps{
        withCredentials([usernamePassword(credentialsId: 'DOCKER_USER',
            passwordVariable: 'DOCKER_PASSWORD', usernameVariable: 'DOCKER_USERNAME

```

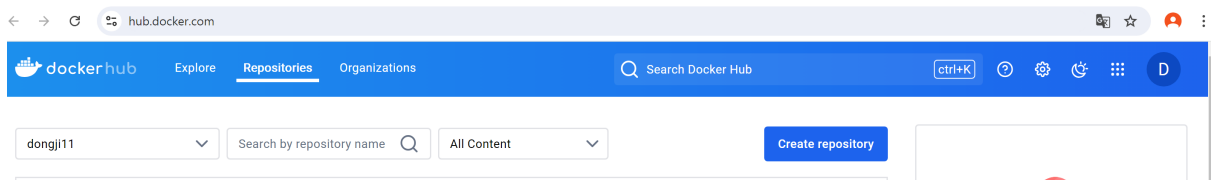
```

sh '''
echo "$DOCKER_PASSWORD" | docker login -u $DOCKER_USERNAME
--password-stdin
'''
}
}
}

```

docker hub에 회원가입, 로그인 후에 project image 를 올릴 dockerhub repository를 만듦.

Create repository → new setting → create



create repository → namespace와 repository name 을 설정합니다. public으로 설정한 후 create

이제 jenkins에서 해당 docker hub에 접근할 수 있도록 user credentials 를 설정

New credentials

Kind
Username with password

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

Username
dongji11

☐ Treat username as secret

Password

ID
DOCKER_USER

Description

Create

username 에는 그냥 본인의 id만 적으면 됨(email 아님)

docker hub의 특정 repository에 image를 push & pull 할 수 있도록 설정

```

username : Docker Hub 의 namespace
password: Docker Hub REpository name
ID : DOCKER_REPO

```

```

username : Docker Hub 의 namespace
password: Docker Hub REpository name
ID : DOCKER_HUB

```

Jenkins에서도 이에 대한 pipeline 을 구성

```

stage('Docker Build and Push') {
    steps {

```

```

        withCredentials([usernamePassword(credentialsId: 'DOCKER_HUB', passwordV
        sh '''
        cd ./backend && docker build -f Dockerfile -t
        $DOCKER_REPO/$DOCKER_PROJECT .
        '''
        sh 'cd ./backend && docker push $DOCKER_REPO/$DOCKER_PROJECT'
        echo 'docker push Success!!'
    }
    echo 'docker push Success!!'
}
}
}

```

3. 배포

```

stage('Deploy') {
    steps {
        sh 'docker rm -f backend-app || true'
        sh '''
        docker stop backend-app || true
        docker rm backend-app || true

        docker run -d --name backend-app -p 8081:8080 dongji11/taffy
        '''
    }
}

```

backend-app 이라는 container가 존재한다면 remove

docker run 명령어를 이용해서 host 포트 8081 과 container 내부 포트 8080을 mount 한 후, docker container 이름을 dongji11/taffy 로 지정

6.2 Frontend

- 전체코드

```

pipeline {
    agent any
    stages {
        stage('Git Clone'){
            steps {
                git branch: 'develop-fe', credentialsId: 'gitlab-account',
                url: 'https://lab.ssafy.com/s11-webmobile1-sub2/S11P12E104.git'
            }
            post {
                failure {
                    echo 'Repository clone failure !'
                }
                success {
                    echo 'Repository clone success !'
                }
            }
        }
    }
}

```

```

stage('Build - PRE SETTING'){
    steps{
        sh "chmod +x -R /var/jenkins_home/workspace/front/frontend"
    }
}

stage('Build - FE'){
    steps{
        dir("/var/jenkins_home/workspace/front/frontend"){
            nodejs(nodeJSInstallationName: 'NodeJS'){
                script {
                    try {
                        // 이전 빌드 파일 삭제
                        sh 'rm -rf dist'

                        // 의존성 설치 및 빌드
                        // sh 'npm ci'
                        sh 'npm install'
                        sh 'npm run build'

                        // 빌드 성공 시 처리
                        currentBuild.result = 'SUCCESS'
                        echo "Frontend build succeeded"
                    } catch (Exception e) {
                        // 빌드 실패 시 처리
                        currentBuild.result = 'FAILURE'
                        echo "Frontend build failed: ${e.message}"
                        error "Frontend build failed"
                    }
                }
            }
        }
    }
    post {
        success {
            echo 'Frontend build was successful!'
        }
        failure {
            echo 'Frontend build failed!'
        }
    }
}

stage('Build - Docker'){
    steps{
        dir("/var/jenkins_home/workspace/front/frontend/deploy"){
            sh 'docker compose build --no-cache'
        }
    }
}

stage('Deploy'){
    steps{
        dir("/var/jenkins_home/workspace/front/frontend/deploy"){
            sh ''
        }
    }
}

```

```
docker stop taffy_fe  
docker rm taffy_fe  
docker compose up -d  
docker image prune -f  
'''  
}  
}  
}
```

6.2.1 git clone

```
stage('Git Clone'){
    steps {
        git branch: 'develop-fe', credentialsId: 'gitlab-account',
        url: 'https://lab.ssafy.com/s11-webmobile1-sub2/S11P12E104.git'
    }
    post {
        failure {
            echo 'Repository clone failure !'
        }
        success {
            echo 'Repository clone success !'
        }
    }
}
```

6.2.2 build 전 권한 설정

```
stage('Build - PRE SETTING'){
    steps{
        sh "chmod +x -R /var/jenkins_home/workspace/front/frontend"
    }
}
```

6.2.3 build

```
stage('Build - FE'){
    steps{
        dir("/var/jenkins_home/workspace/front/frontend"){
            nodejs(nodeJSInstallationName: 'NodeJS'){
                script {
                    try {
                        // 이전 빌드 파일 삭제
                        sh 'rm -rf dist'

                        // 의존성 설치 및 빌드
                        // sh 'npm ci'
                        sh 'npm install'
                        sh 'npm run build'
                    }
                }
            }
        }
    }
}
```

```

        // 빌드 성공 시 처리
        currentBuild.result = 'SUCCESS'
        echo "Frontend build succeeded"
    } catch (Exception e) {
        // 빌드 실패 시 처리
        currentBuild.result = 'FAILURE'
        echo "Frontend build failed: ${e.message}"
        error "Frontend build failed"
    }
}
}
}
}
}
}
}

```

6.2.4 docker image로 만들기

```

stage('Build - Docker'){
    steps{
        dir("/var/jenkins_home/workspace/front/frontend/deploy"){
            sh 'docker compose build --no-cache'
        }
    }
}

```

docker image를 만들 때 사용하는 Dockerfile 파일

Dockerfile (프론트 프로젝트 루트에 존재)

```

FROM nginx
RUN mkdir /deploy
WORKDIR /deploy
RUN mkdir ./build
ADD ./dist ./build
RUN rm /etc/nginx/conf.d/default.conf
COPY ./nginx.conf /etc/nginx/conf.d
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

Nginx 설정 파일(프론트 프로젝트 루트에 존재)

nginx.conf

```

server{
    listen 80;
    location / {
        root /deploy/build;
        index index.html;
        try_files $uri $uri/ /index.html;
    }
}

```

```

ubuntu@ip-172-31-16-10:~$ cd /var/jenkins_home/workspace/front/frontend
ubuntu@ip-172-31-16-10:~/workspace/front/frontend$ ls
Dockerfile  deploy      dist        nginx.conf  package-lock.json  pnpm-lock.yaml  src
README.md   deploy@tmp  index.html  node_modules package.json        public           vite.config.js
ubuntu@ip-172-31-16-10:~/workspace/front/frontend$ cat nginx.conf

```

docker-compose.yml


```

version: "3.8"
services:
  fe:
    container_name: taffy_fe
    image: taffy_fe:0.1
    build:
      context: ../
    ports:
      - 3000:80

```

```

ubuntu@ip-172-31-0-14:/var/jenkins_home/workspace/front/frontend/deploy$ ls
docker-compose.yml
ubuntu@ip-172-31-0-14:/var/jenkins_home/workspace/front/frontend/deploy$ cat docker-compose.yml
version: "3.8"
services:
  fe:
    container_name: taffy_fe
    image: taffy_fe:0.1
    build:
      context: ../
    ports:
      - 3000:80

```

6.2.5 deploy

```

stage('Deploy'){
  steps{
    dir("/var/jenkins_home/workspace/front/frontend/deploy"){
      sh '''
        docker stop taffy_fe
        docker rm taffy_fe
        docker compose up -d
        docker image prune -f
      '''
    }
  }
}

```