드림멍즈
DREAM MUNGZ

PORTING MANUAL

목 차

1. 개발 환경

Server : AWS EC2 Ubuntu 20.04 LTS

Visual Studio Code : 1.70.1

IntelliJ IDEA : 2022.1.3(Ultimate Edition) 11.0.15 + 10-b2043.56 amd64

JVM : 11.0.16+8-post-Ubuntu-0ubuntu120.04

Docker : 20.10.12

Node.js : 18.9.0

MySQL : 8.0.30-1.el8

Nginx : 1.23.1

Jenkins : 2.361.1

IPFS: 0.16.0

Truffle: 5.5.28

Ganache: 7.4.0

2. 설정 파일 목록

React

- **Dockerfile** : /jenkins/workspace/frontend/frontend
- **Web3Config.js** : /jenkins/workspace/frontend/frontend

SpringBoot

- **application.properties** : /jenkins/workspace/backend/backend/src/main/resources
- **Dockerfile** : /jenkins/workspace/backend/backend
- **deploy.sh** : /jenkins/workspace/backend/backend

Docker

- docker-compose.yml(MySQL, Jenkins, Nginx) : /home/ubuntu
- docker-compose.yml(IPFS) : /home/ubuntu/ipfs

Nginx

- app.conf : /home/ubuntu/nginx/conf.d

Truffle

- truffle-config.js
- .secret

Ipfs

- config : /home/ubuntu/ipfs/compose/ipfs0

3. 설정 파일 및 환경 변수 정보

React

- dockerfile

```
FROM node:alpine
WORKDIR /usr/src/app
COPY ./package* /usr/src/app/
RUN npm install
COPY ./ /usr/src/app/
CMD ["npm","run","start"]
```

- Web3Config.js

```
import Web3 from “web3”

const MUNGAbi = {ERC-20 토큰 컨트랙트 ABI}
```

```
const MFTAbi = {ERC-721 토큰 컨트랙트 ABI}
const MFTSaleFactoryAbi = {토큰 거래 컨트랙트 ABI}
export const web3 = new Web3(window.ethereum)
export const chainId = {블록체인 네트워크 Chain ID}

export const OwnerAddress = {컨트랙트를 배포한 Owner 주소}
export const MUNGContractAddress = {ERC-20 토큰 CA}
export const MFTContractAddress = {ERC-721 토큰 CA}
export const MFTSaleFactoryContractAddress = {토큰 거래 CA}

export const MUNGContract = new web3.eth.Contract(MUNGAbi,
MUNGContractAddress)
export const MFTContract = new web3.eth.Contract(MFTAbi,
MFTContractAddress)
export const MFTSaleFactoryContract = new web3.eth.Contract(
  MFTSaleFactoryAbi,
  MFTSaleFactoryContractAddress
)
```

Springboot

- application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://mysql:3306/dreammungz?useSSL=fal
se&useUnicode=true&serverTimezone=Asia/Seoul
spring.datasource.username={MySQL 사용자 이름}
spring.datasource.password={MySQL 패스워드}

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true

server.port=8081
```

- Dockerfile

```
FROM openjdk:11-jdk
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","-Duser.timezone=Asia/Seoul","/app.jar"]
```

- deploy.sh

```
echo '실행 시작'

echo 'git pull'

echo 'jar 파일 삭제'
rm build/libs/*.jar

echo '빌드 전 cleanQuery'
./gradlew cleanQuerydslSourceDir

echo '빌드 시작'
./gradlew build

echo '도커파일 이미지 빌드'
docker build -t springbootapp .

echo '컨테이너 중지'
docker stop springbootapp

echo '컨테이너 삭제'
docker rm springbootapp

echo '컨테이너 실행'
docker run -p 8081:8081 --name springbootapp --network
ubuntu_default -d springbootapp
```

Docker

- docker-compose.yml(MySQL, Jenkins, Nginx)

```
version: "3"
services:
  mysql:
    image: mysql

    container_name: mysql
    environment:
      MYSQL_DATABASE: {scheme명}
      MYSQL_ROOT_PASSWORD: {root 계정 패스워드}
    volumes:
      - /mysql:/var/lib/mysql
    ports:
       - 3306:3306

  nginx:
    image: nginx
    container_name: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt
      - ./nginx/conf.d:/etc/nginx/conf.d

  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /jenkins:/var/jenkins_home
    ports:
      - 9090:8080
    privileged: true
    user: root
```

- docker-compose.yml(IPFS)

```
version: '3'

services:
  ipfs0:
    container_name: ipfs0
    image: ipfs/go-ipfs:latest
    ports:
      - "4001:4001" # ipfs swarm - expose if needed/wanted
      - "5001:5001" # ipfs api - expose if needed/wanted
      - "8080:8080" # ipfs gateway - expose if needed/wanted
    volumes:
      - ./compose/ipfs0:/data/ipfs

  cluster0:
    container_name: cluster0
    image: ipfs/ipfs-cluster:latest
    depends_on:
      - ipfs0
    environment:
      CLUSTER_PEERNAME: cluster0
      CLUSTER_SECRET: # From shell variable if set
      CLUSTER_IPFSHTTP_NODEMULTIADDRESS: /dns4/ipfs0/tcp/5001
      CLUSTER_CRDT_TRUSTEDPEERS: '*' # Trust all peers in Cluster
      CLUSTER_RESTAPI_HTTPLISTENMULTIADDRESS:
/ip4/0.0.0.0/tcp/9094  # Expose API
      CLUSTER_MONITORPINGINTERVAL: 2s # Speed up peer
discovery
    ports:
          - "127.0.0.1:9094:9094"
          # - "9096:9096"  # Cluster IPFS Proxy endpoint
    volumes:
      - ./compose/cluster0:/data/ipfs-cluster

  ipfs1:
    container_name: ipfs1
    image: ipfs/go-ipfs:latest
    volumes:
      - ./compose/ipfs1:/data/ipfs
```

```
  cluster1:
    container_name: cluster1
    image: ipfs/ipfs-cluster:latest
    depends_on:
      - ipfs1
    environment:
      CLUSTER_PEERNAME: cluster1
      CLUSTER_SECRET:
      CLUSTER_IPFSHTTP_NODEMULTIADDRESS: /dns4/ipfs1/tcp/5001
      CLUSTER_CRDT_TRUSTEDPEERS: '*'
      CLUSTER_MONITORPINGINTERVAL: 2s # Speed up peer
discovery
    volumes:
      - ./compose/cluster1:/data/ipfs-cluster

  ipfs2:
    container_name: ipfs2
    image: ipfs/go-ipfs:latest
    volumes:
      - ./compose/ipfs2:/data/ipfs

  cluster2:
    container_name: cluster2
    image: ipfs/ipfs-cluster:latest
    depends_on:
      - ipfs2
    environment:
      CLUSTER_PEERNAME: cluster2
      CLUSTER_SECRET:
      CLUSTER_IPFSHTTP_NODEMULTIADDRESS: /dns4/ipfs2/tcp/5001
      CLUSTER_CRDT_TRUSTEDPEERS: '*'
      CLUSTER_MONITORPINGINTERVAL: 2s # Speed up peer
discovery
    volumes:
      - ./compose/cluster2:/data/ipfs-cluster
```

Nginx

- app.conf

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name j7a605.p.ssafy.io  www.j7a605.p.ssafy.io;

    return 301 https://$server_name$request_uri;
}

server {
          listen 443 ssl;
          listen [::]:443 ssl;
          server_name j7a605.p.ssafy.io;
          access_log off;

          ssl_certificate
/etc/letsencrypt/live/j7a605.p.ssafy.io/fullchain.pem;
          ssl_certificate_key
/etc/letsencrypt/live/j7a605.p.ssafy.io/privkey.pem;

          location /ipfs {
          proxy_pass http://j7a605.p.ssafy.io:5001/api/v0;
          proxy_set_header  Host $host;
          proxy_redirect  off;

          }

          location /api/ {
         proxy_pass http://j7a605.p.ssafy.io:8081/;
         proxy_set_header  Host $host;
         proxy_redirect  off;
         }

          location / {
          proxy_pass http://j7a605.p.ssafy.io:3000;
```

```
        proxy_set_header Host $host;
         proxy_redirect off;
         proxy_set_header Upgrade $http_upgrade;
         proxy_set_header Connection "upgrade";
         }
}
```

Truffle

- truffle-config.js

```
  (...)

const HDWalletProvider = require("@truffle/hdwallet-provider")
const fs = require("fs")
const mnemonic = fs.readFileSync(".secret").toString().trim()

module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*",
    },
    ropsten: {
      provider: () =>
        new HDWalletProvider(
          mnemonic,
          `https://ropsten.infura.io/v3/{프로젝트 ID}`
        ),
      network_id: 3, // Ropsten's id
      gas: 5500000,
      confirmations: 2,
      timeoutBlocks: 200,
      skipDryRun: true,
    },
  },
```

```
  (...)

  compilers: {
    solc: {
      version: "0.8.16",
      settings: {
        optimizer: {
          enabled: true,
          runs: 1000,
        },
      },
    },
   },

   (...)
}
```

- .secret

```
{컨트랙트 Owner 주소 복구 구문(Nemonic)}
```

Ipfs

- config

```
{
  (...)
"API": {
    "HTTPHeaders": {
        "Access-Control-Allow-Credentials": [
                "true"
        ],
        "Access-Control-Allow-Methods": [
        "PUT",
        "GET",
        "POST",
```

```
        "OPTIONS"
        ],
        "Access-Control-Allow-Origin": [
                "*"
        ]
    }
},
(...)
```

4. Docker 설치

- Docker Install

```
sudo apt-get update

sudo apt-get install ca-certificates curl gnupg lsb-release

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

- Docker Compose Install

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-
```

```
compose

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

5. SSL 인증서 발급

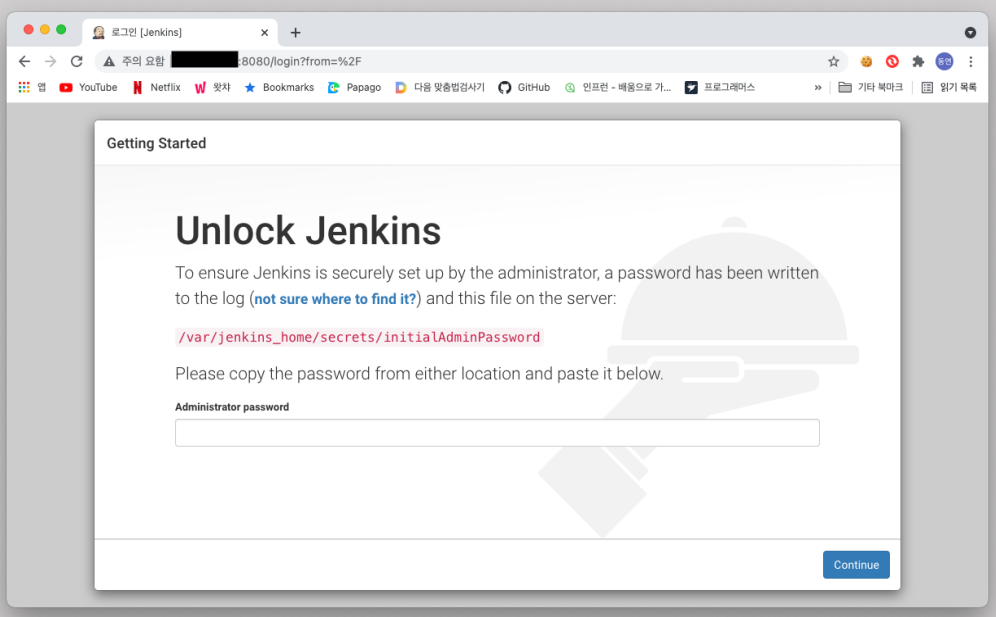
```
sudo apt-get install letsencrypt

sudo letsencrypt certonly --standalone -d www제외한 도메인 이름

이메일 작성 후 Agree

뉴스레터 수신 여부 Yes/No

해당 경로에 Key 생성 여부 확인
ssl_certificate /etc/letsencrypt/live/{도메인}/fullchain.pem
ssl_certificate_key /etc/letsencrypt/live/{도메인}/privkey.pem
```

6. Infra 배포

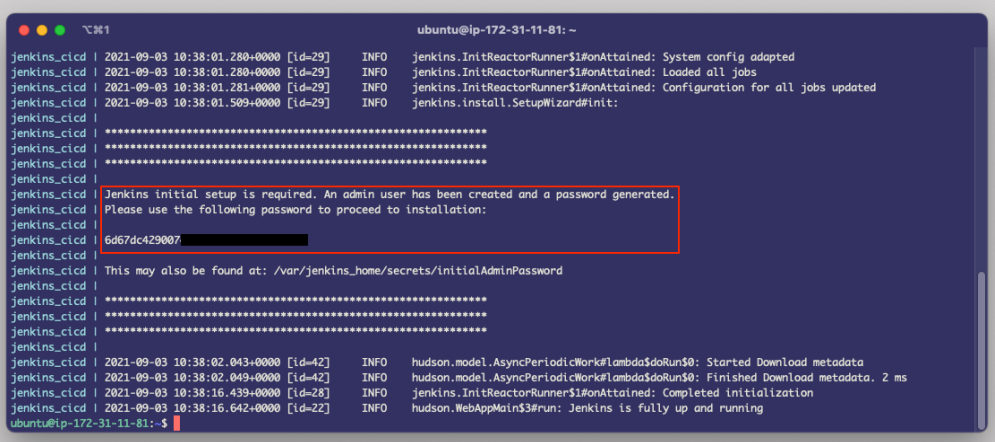
- docker-compose 실행

```
cd /home/ubuntu/mysql,jenkins,nginx (docker-compose.yml 경로에서)

sudo docker-compose up --build -d

cd /home/ubuntu/ipfs (docker-compose.yml 경로에서)

sudo docker-compose up --build -d
```

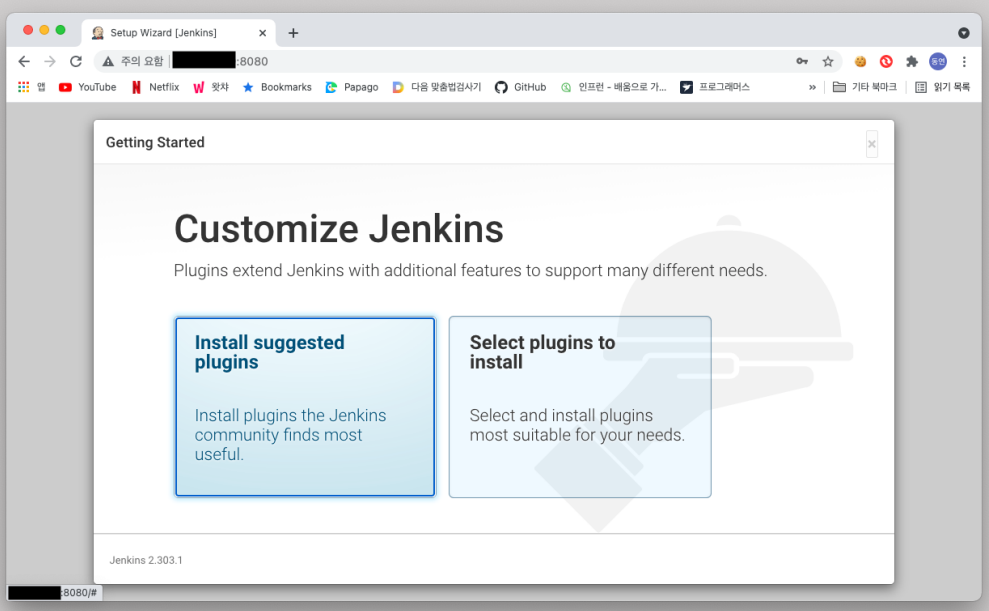
- Jenkins 플러그인 및 docker-in-docker 설치



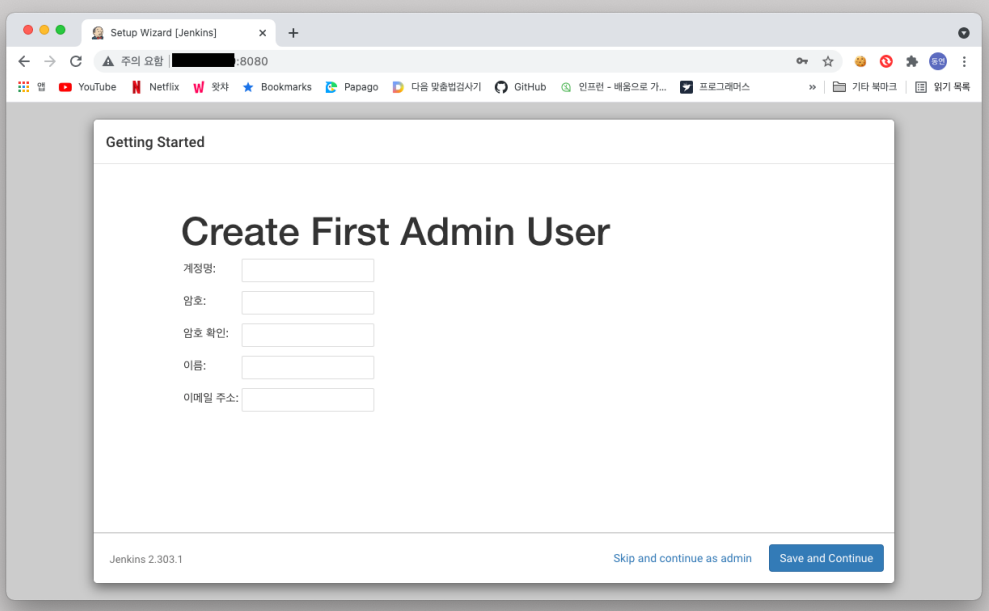
1. http://서비스도메인:9090 으로 접속하여 jenkins 페이지 진입



2. 서버 콘솔에서 sudo docker logs [Jenkins 컨테이너 이름]으로 Administrator password를 확인하고 입력

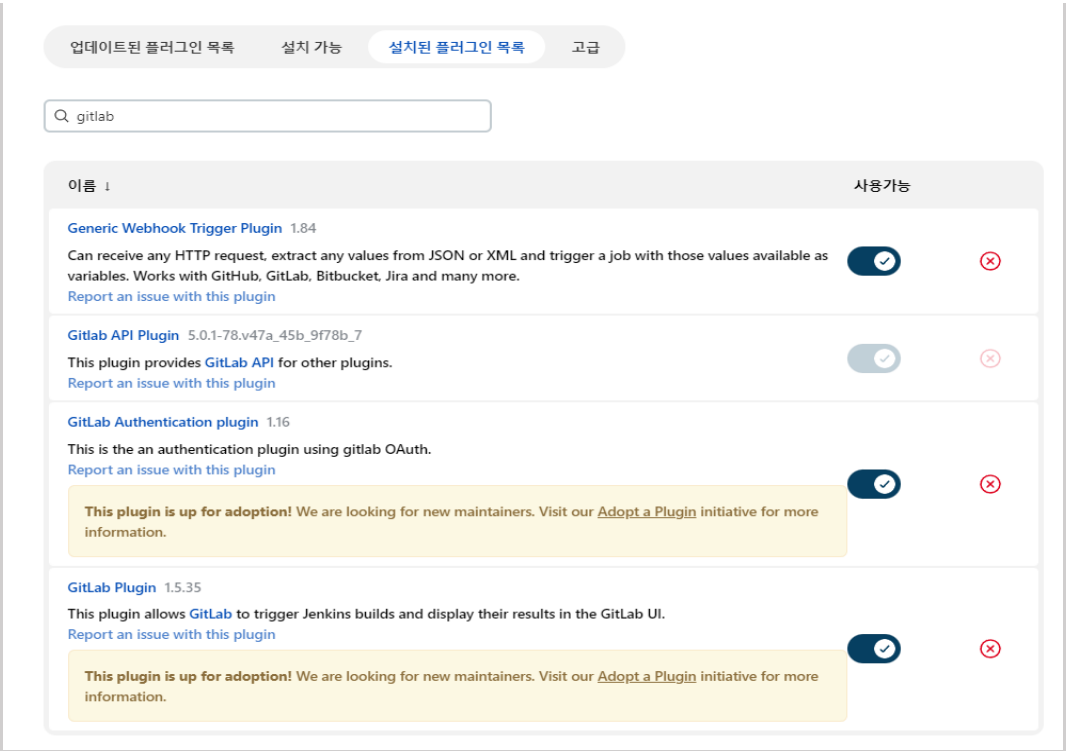


3. Install suggested plugins을 선택하여 플러그인들을 설치



4. 생성할 관리자 계정 정보를 입력하고 Save and Continue

5. Jenkins 접속 URL확인 후 Save and Finish



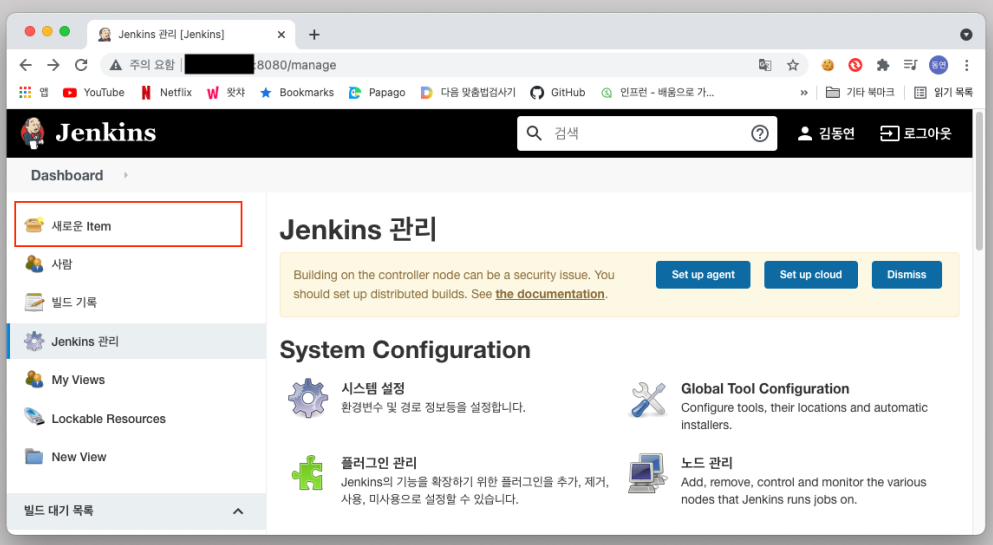
6. 메인 화면에서 DashBoard -> Manager JenKins -> Plugin Manager에서 gitlab, docker에 대해서 플러그인을 설치

7. 서버 콘솔로 돌아가 Jenkins 내부에 docker를 설치

```
sudo docker exec -it {jenkins 컨테이너 이름} /bin/bash
apt-get update -y
apt-get install -y
apt-get install docker.io -y
docker -v
```

7. Backend CI/CD

- Jenkins Job 설정



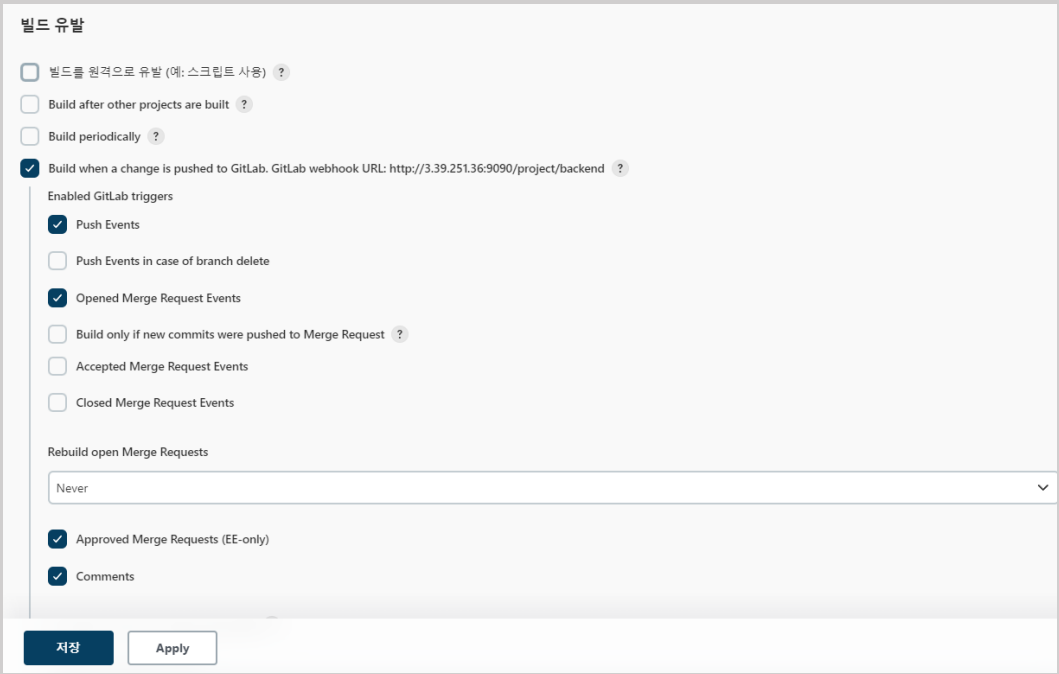
1. Jenkins 메인 화면 -> Dashboard -> 새로운 Item 클릭

2. FreeStyle project를 선택하고 item name은 backend로 설정 후 OK

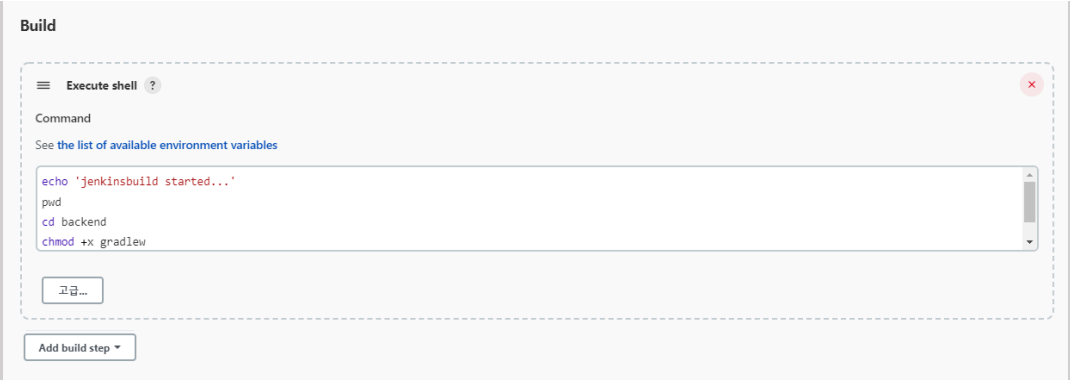
3. Repositories URL에는 프로젝트 레포지토리의 HTTPS Clone 주소 입력 Credentials의 Add를 클릭,

Domain -> Global credentials
Kind -> Username with password
Username -> 레포지토리 접근 권한이 있는 Gitlab 계정 아이디
Password -> Username에 작성한 Gitlab 계정 비밀번호
순서대로 입력하고 드롭박스에서 생성된 Credential을 선택

4. backend 브랜치의 내용만을 받아와서 빌드하기 위한 설정



5. 빌드 유발을 다음과 같이 설정
하단의 고급 기능을 눌러 Secret Token을 Generate하여 기록



6. Build 탭에 Excute shell을 선택하고
cd backend
chmod +x gradlew
chmod +x deploy.sh
./deploy.sh

- Jenkins에 sudo 권한 부여

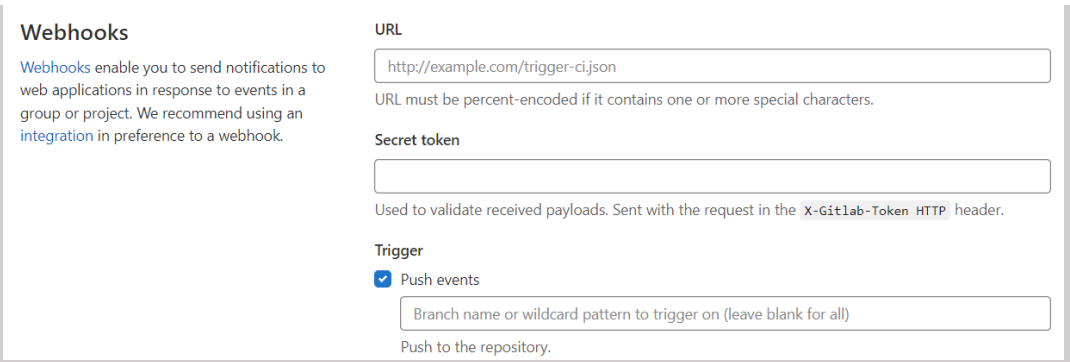
sudo vim /etc/sudoers

```
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
jenkins ALL=(ALL) NOPASSWD: ALL
```

해당 부분에 jenkins ALL=(ALL) NOPASSWO: ALL 작성

‘

- Gitlab Webhook 설정

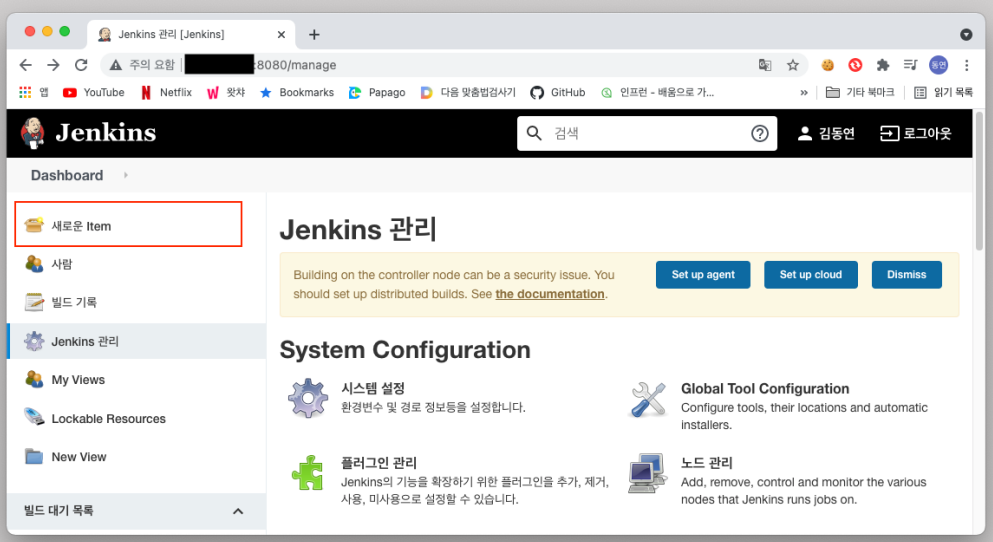


Gitlab 프로젝트 Repository의 Setting > Webhooks 로 이동

URL에는 Jenkins의 빌드 유발의 Webhook URL을 입력
Secret Token에는 빌드 유발에서 생성했던 Secret Token을 입력
Trigger의 Push events를 체크하고 backend 입력 후 Add webhook

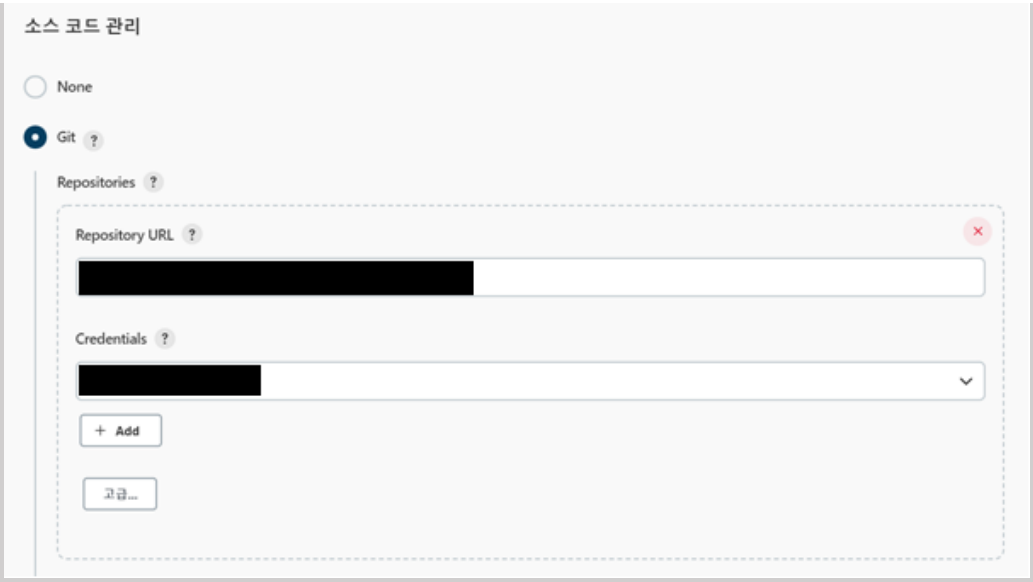
8. Frontend CI/CD

- Jenkins Job 설정

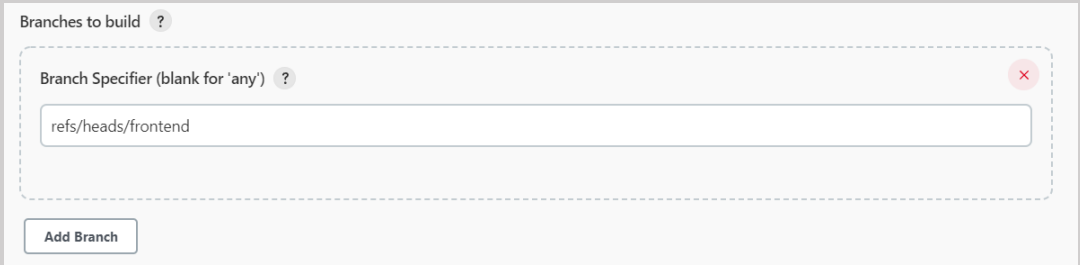


1. Jenkins 메인 화면 -> Dashboard -> 새로운 Item 클릭

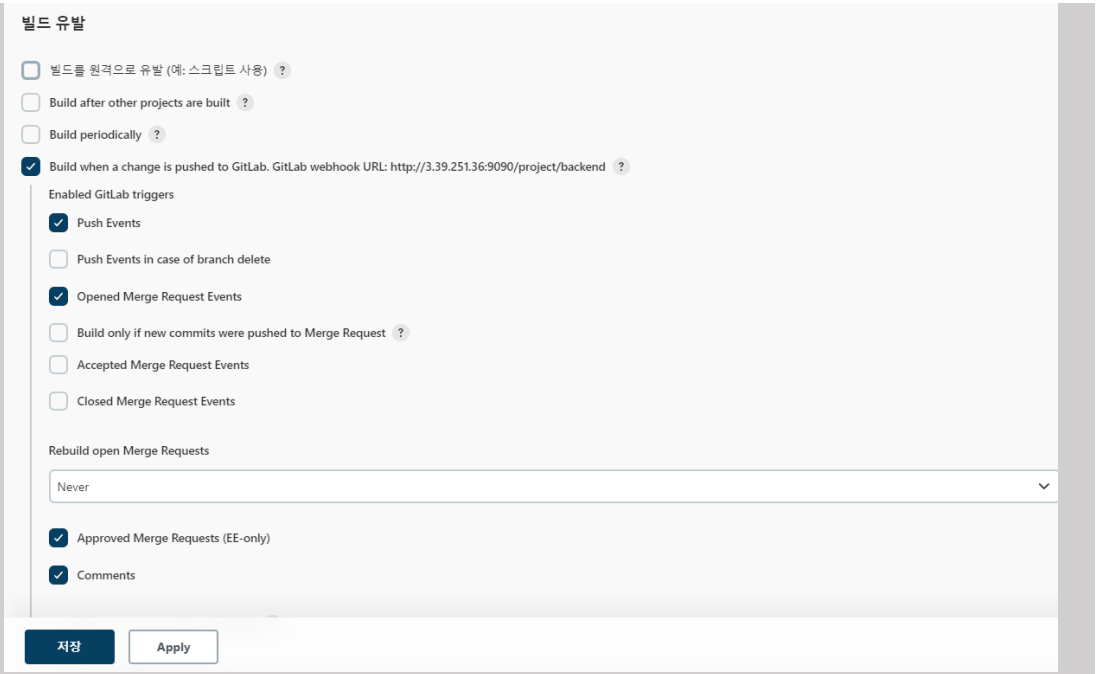
2. FreeStyle project를 선택하고 item name은 backend로 설정 후 OK



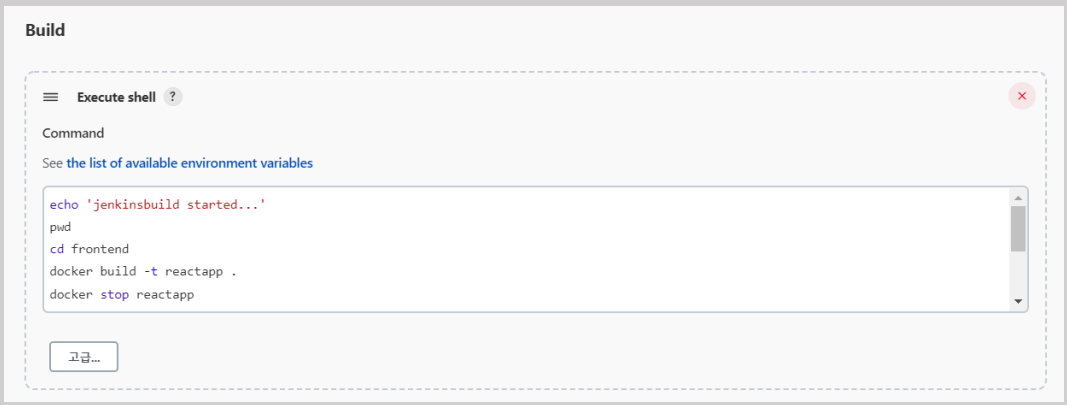
3. Repositories URL에는 프로젝트 레포지토리의 HTTPS Clone 주소 입력
Credentials의 Add를 클릭,
Domain -> Global credentials
Kind -> Username with password
Username -> 레포지토리 접근 권한이 있는 Gitlab 계정 아이디
Password -> Username에 작성한 Gitlab 계정 비밀번호
순서대로 입력하고 드롭박스에서 생성된 Credential을 선택



4. Frontend 브랜치의 내용만을 받아와서 빌드하기 위한 설정

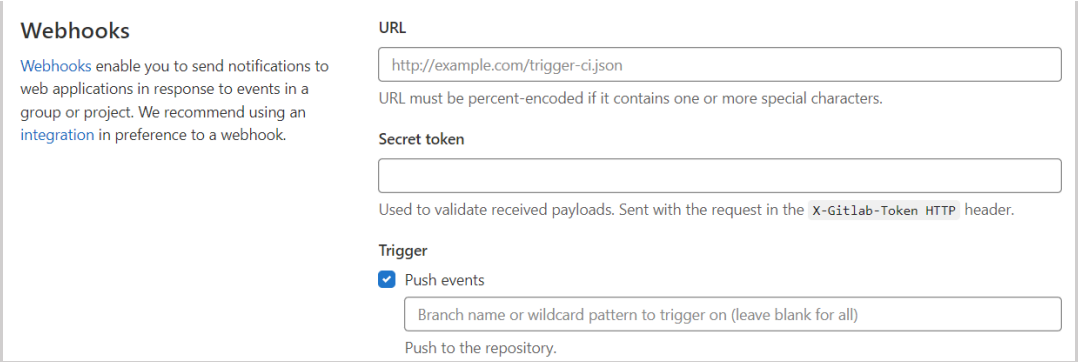


5. 빌드 유발을 다음과 같이 설정
하단의 고급 기능을 눌러 Secret Token을 Generate하여 기록



6. Build 탭에 Excute shell을 선택하고
cd frontend
docker build -t reactapp .
docker stop reactapp
docker rm reactapp
docker run -p 3000:3000 --name reactapp --network ubuntu_default -d reactapp

- Gitlab Webhook 설정



Gitlab 프로젝트 Repository의 Setting > Webhooks 로 이동
URL에는 Jenkins의 빌드 유발의 Webhook URL을 입력
Secret Token에는 빌드 유발에서 생성했던 Secret Token을 입력
Trigger의 Push events를 체크하고 backend 입력 후 Add Webhook

9. Smart Contract 배포

- 1_intial_migration.js 작성

```
const SSFToken = artifacts.require("SSFToken");
const MFT = artifacts.require("MFT");
const MFTSaleFactory = artifacts.require("MFTSaleFactory");

module.exports = async function (deployer) {
  await deployer.deploy(SSFToken, "MUNG", "M");
  await deployer.deploy(MFT);
  await deployer.deploy(MFTSaleFactory, SSFToken.address,
MFT.address);
};
```

- Truffle 배포 명령어 입력

```
truffle compile
truffle deploy --network {네트워크이름} --reset --compile-none
```