인프라, 백엔드 포팅 메뉴얼

1. 빌드 및 배포

1) 프로그램 버젼

```
git version 2.25.1
Docker version 25.0.
nginx version: nginx/1.18.0 (Ubuntu)
Intellij Ultimate: 2023.3.2
VsCode: 1.85.2

JAVA: 17
MySQL: 8.3

FastAPI: 0.110.0
Python: 3.10.14

Nginx:1.18.0
Jenkins 2.440.1
Sonarqube 10.4.1
```

2) 환경변수

1. back-end application.yml

```
server:
  port: 8080
  servlet:
    context-path: /api
    encoding:
      charset: UTF-8
      enabled: true
      force: true
```

```yaml
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
    username:
    password:



ASSEMBLY_IMG_KEY:
ASSEMBLY_IMG_URL:
ASSEMBLY_INFO_KEY:
ASSEMBLY_INFO_URL:
BILL_INFO_KEY:
BILL_INFO_URL:
BODY_INFO_KEY:
BODY_INFO_URL:
CMIT_INFO_KEY:
CMIT_INFO_URL:
POLY_INFO_URL:
POLY_INFO_KEY:
SNS_INFO_KEY:
SNS_INFO_URL:
22_URL:
22_KEY:
CAND_URL:
CAND_KEY:

PYTHON_URL:

logging:
  config: classpath:logback-spring.xml

aws:
  s3:
    accessKey:
    secretKey:
    bucket:
```

```
cloud_front:
  url:
```

2. logbakc-spring.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
    <!-- base.xml default.xml 에 존재하는 Log 메시지의 Color 설정
    <conversionRule conversionWord="clr" converterClass="org.

    <!-- 콘솔에 출력되는 로그 패턴 -->
    <property name="CONSOLE_LOG_PATTERN"
              value="%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %c
    <!-- Log파일에 기록되는 로그 패턴 -->
    <property name="FILE_LOG_PATTERN" value="%d{yyyy-MM-dd HH

    <!-- 콘솔로그 Appender -->
    <appender name="CONSOLE" class="ch.qos.logback.core.Conso
        <encoder>
            <pattern>${CONSOLE_LOG_PATTERN}</pattern>
        </encoder>
    </appender>

    <!-- 파일로그 Appender -->
    <appender name="FILE" class="ch.qos.logback.core.rolling.
        <encoder>
            <pattern>${FILE_LOG_PATTERN}</pattern>
        </encoder>
        <!-- RollingPocliy: 로그가 길어지면 가독성이 떨어지므로 로그를
        <!-- 로그파일을 크기, 시간 기반으로 관리하기 위한 SizeAndTimeB
        <rollingPolicy class="ch.qos.logback.core.rolling.Siz
            <!-- 로그파일명 패턴 -->
            <!-- 날짜별로 기록되며 maxFileSize를 넘기면 인덱스(i)를
            <fileNamePattern>/logs/%d{yyyy-MM}/%d{yyyy-MM-dd}
            <!-- 로그파일 최대사이즈 -->
            <maxFileSize>100MB</maxFileSize>
            <!-- 생성한 로그파일 관리 일수 -->
```

```
            <maxHistory>30</maxHistory>
        </rollingPolicy>
    </appender>

    <!-- local Profile에서의 로그 설정 -->
    <springProfile name="security">
        <!-- 해당 패키지의 로그는 DEBUG 레벨 부터 출력 -->
        <logger name="com.honey.backend" level="DEBUG" />

        <!-- 전체적인 로그는 INFO 레벨 부터 출력 -->
        <root level="INFO">
            <!-- CONSOLE 로그 Apender를 로그 Appender로 등록 -->
            <appender-ref ref="CONSOLE" />
            <appender-ref ref="FILE" />
        </root>
    </springProfile>

</configuration>
```

3) 배포시 특이사항 기재

```
docker network create --gateway 172.20.0.1 --subnet 172.20.0.
```

2. Ubuntu Linux 세팅

```
sudo add user test
```

3. Docker 세팅

```
1. 우분투 시스템 패키지 업데이트
sudo apt-get update
2. 필요한 패키지 설치
sudo apt-get install apt-transport-https ca-certificates curl
```

```
3. Docker의 공식 GPG키를 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sud


4. Docker의 공식 apt 저장소를 추가
sudo add-apt-repository "deb [arch=amd64] https://download.do


5. 시스템 패키지 업데이트
sudo apt-get update
6. Docker 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

docker group 추가

```
sudo usermod -aG docker user1
```

4. Nginx 세팅

```
sudo apt update
sudo apt install nginx
```

ufw 허용

```
sudo ufw allow 'Nginx Full'
```

구동 확인

```
systemctl status nginx

● nginx.service - A high performance web server and a reverse
```

```
     Loaded: loaded (/lib/systemd/system/nginx.service; enabl
     Active: active (running) since Wed 2022-04-20 09:03:47 U
       Docs: man:nginx(8)
   Main PID: 20596 (nginx)
      Tasks: 2 (limit: 1147)
     Memory: 5.4M
     CGroup: /system.slice/nginx.service
             ├─20596 nginx: master process /usr/sbin/nginx -g
             └─20597 nginx: worker process
```

certbot 활용한 SSL 인증서 발급

```
apt-get update
apt-get upgrade
apt-get install python3-certbot-nginx

certbot certonly --nginx -d example.com
```

Nginx 설정파일 생성

/etc/nginx/site-available

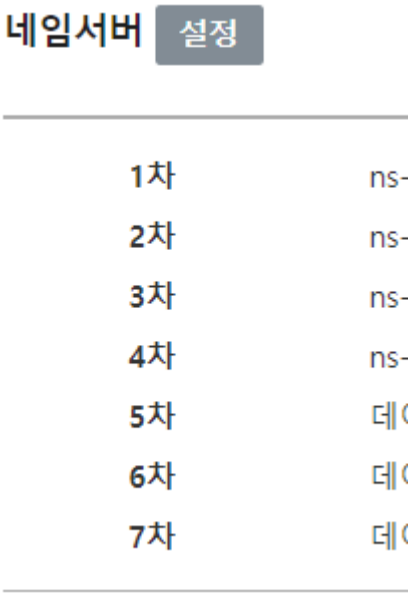
```
server {
    server_name example.com;
    listen 443 ssl;

     ssl_certificate /etc/letsencrypt/live/example.com/fullch
     ssl_certificate_key /etc/letsencrypt/live/example.com/pr

    location /api {
        proxy_pass http://spring;
        proxy_set_header X-Real-IP $remote_addr;
        error_page 405 = $uri;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
        proxy_set_header Origin http://localhost:port;
        proxy_set_header Authorization $http_authorization;
```

```
        proxy_next_upstream error timeout invalid_header http

    }

     location / {
          proxy_pass http://localhost:port;
        }

}

upstream spring{
        least_conn;
        server localhost:port weight=2 max_fails=5 fail_timeo
        server localhost:port max_fails=5 fail_timeout=30s;
        server localhost:port backup;
}

server {
    if ($host = yeouido-honeypot.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
  listen 80;
  server_name yeouido-honeypot;
    return 404; # managed by Certbot
}
```

심볼릭 링크 연결

```
ln -s /etc/nginx/site-available/yeoido-hoenypot.com /etc/ngin

nginx -s reload
```

5. 가비아 도메인 & route 53 연결



가비아의 네임서버 route 53에 등록

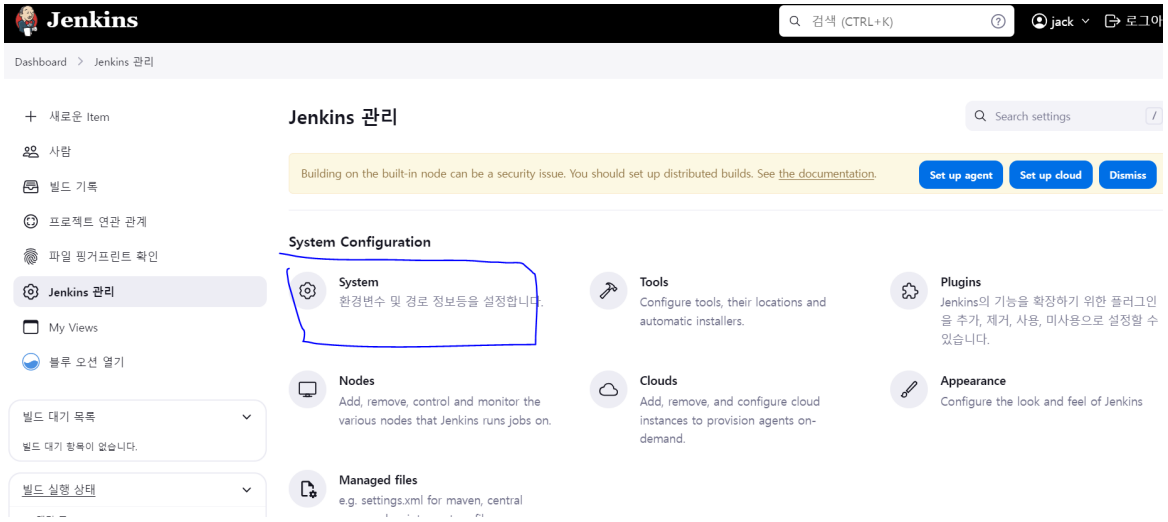
6. jenkins 세팅

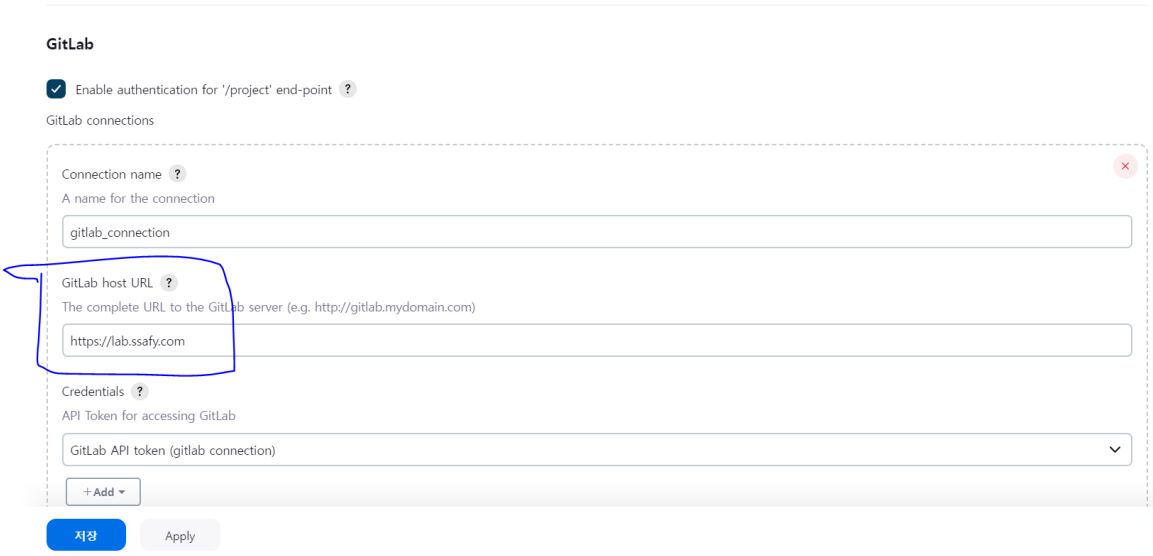
```
sudo docker run -d -p 8082:8080 -p 50000:50000 --restart=on-f
```

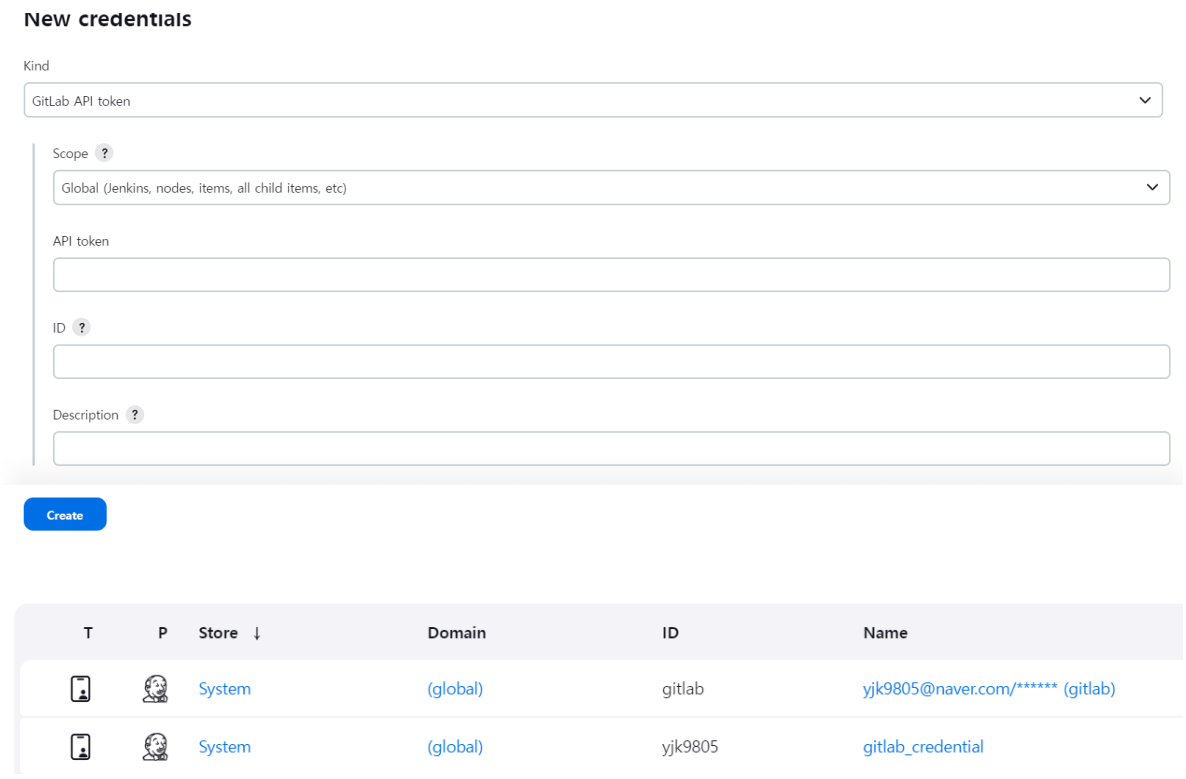
plugin install

```
gitlab, nodejs, publish over ssh, mattermost notification, ma
```

ec2 ssh credential





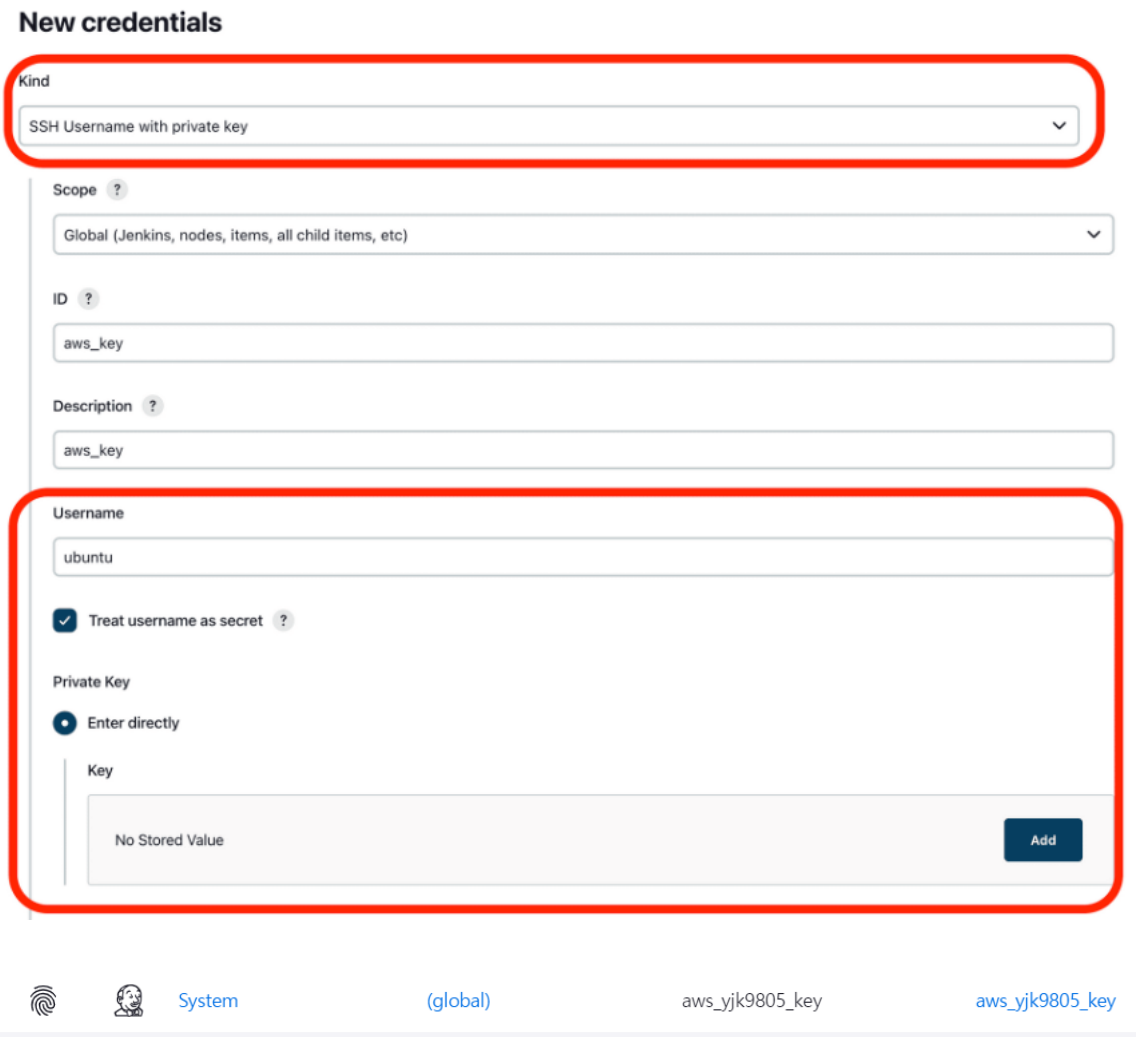


ssh 4096 key 생성

1. local 컴퓨터에서 키 생성(ssh-keygen -t rsa -b 4086)
2. aws ec2 접속 후 authorized_keys에 a에서 만든 pub키 쓰기

jenkins ssh credentials 추가

1. 2.a에서 만든 private key 등록



Dockerfile

```
FROM openjdk:17-jdk-alpine

COPY app.jar app.jar

ENTRYPOINT ["java","-jar" ,"app.jar"]
```

deploy.sh

```
cd /home/user/log-back-end-application

sudo docker ps -a -q --filter "name=log-spring-cd" | grep -
```

```
sudo docker rmi log-spring-cd


sudo docker build -t log-spring-cd:latest -f /home/user/log

sudo docker run -d -p 8081:8080 --name log-spring-cd --netw

 #####
sudo docker ps -a -q --filter "name=log-spring-cd-1" | grep

sudo docker rmi log-spring-cd-1

 sudo docker build -t log-spring-cd-1:latest -f /home/user/

sudo docker run -d -p 8083:8080 --name log-spring-cd-1 --ne

#####

sudo docker ps -a -q --filter "name=log-spring-cd-2" | grep

sudo docker rmi log-spring-cd-2

 sudo docker build -t log-spring-cd-2:latest -f /home/user/

sudo docker run -d -p 8084:8080 --name log-spring-cd-2 --ne
```

back-end pipeline

```
pipeline {
    agent any
    tools {
        maven 'maven'
    }
    stages {
        stage('Git Clone') {
            steps {
                git branch: 'be-develop', credentialsId: 'git
```

```
            }
        }
        stage('Copy application') {
            steps {
                sshagent(credentials: ['aws_key']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no user@

                        scp user@ec2ip:/home/user/back-end-ap

                    '''
                }
            }
        }

        stage('BE-Build') {
            steps {
                dir("./back-end") {
                    sh 'mvn clean compile package'
                }
            }
        }

         stage('SonarQube analysis') {
             steps{
                withSonarQubeEnv('sonarqube'){
                     dir("./back-end") {
                    sh "mvn sonar:sonar -Dsonar.projectKey=wi
                    }


                }
            }
         }


        stage('BE-Deploy') {
            steps {
```

```
                sshagent(credentials: ['aws_key']) {

                        sh '''
                        ssh -o StrictHostKeyChecking=no user@

                        scp /var/jenkins_home/workspace/back-


                        ssh -o StrictHostKeyChecking=no user@
                        ssh -o StrictHostKeyChecking=no user@
                        ssh -o StrictHostKeyChecking=no user@

                    '''

                }
            }
        }

    }
     post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend (color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUI
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend (color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUI

                )
            }
```

```
        }
        }

}
```

7. docker ufw 설정

/etc/ufw/after.rules

```
#
# rules.input-after
#
# Rules that should be run after the ufw command line added r
# rules should be added to one of these chains:
#   ufw-after-input
#   ufw-after-output
#   ufw-after-forward
#

# Don't delete these required lines, otherwise there will be
*filter
:ufw-after-input - [0:0]
:ufw-after-output - [0:0]
:ufw-after-forward - [0:0]
# End required lines

# don't log noisy services by default
-A ufw-after-input -p udp --dport 137 -j ufw-skip-to-policy-i
-A ufw-after-input -p udp --dport 138 -j ufw-skip-to-policy-i
-A ufw-after-input -p tcp --dport 139 -j ufw-skip-to-policy-i
-A ufw-after-input -p tcp --dport 445 -j ufw-skip-to-policy-i
-A ufw-after-input -p udp --dport 67 -j ufw-skip-to-policy-in
-A ufw-after-input -p udp --dport 68 -j ufw-skip-to-policy-in

# don't log noisy broadcast
-A ufw-after-input -m addrtype --dst-type BROADCAST -j ufw-sk
```

```
# don't delete the 'COMMIT' line or these rules won't be proc
COMMIT
~
~
```