CI/CD

📌 **EC2 서버**

도메인 `k10s106.p.ssafy.io`

🐳 컨테이너 정리

컨테이너	포트	URL	
jenkins	`8080:8080`	http://k10s106.p.ssafy.io:8080	
portainer	`9443:9443`	https://k10s106.p.ssafy.io:9443/#!/home	
nginx	`443:443` | `80:80`		
frontend(nginx)	`8081:81`	https://k10s106.p.ssafy.io	
springboot	`:8080`	https://k10s106.p.ssafy.io/api	
mysql	`3306:3306`		
elastiksearch	`9200:9200`	http://k10s106.p.ssafy.io:9200/	
kibana	`5601:5601`	http://k10s106.p.ssafy.io:5601/	
logstash	`50000:50000`		

서버 접속

1. 서버 접속

```
ssh -i k10s106.p.ssafy.io.pem ubuntu@k10s106.p.ssafy.io
```

서버 설정

1. 서버 시간 설정

```
sudo timedatectl set-timezone Asia/Seoul
# 확인
date
```

2. 미러 서버 변경

```
sudo vi /etc/apt/sources.list
# %s : 문자 대체
# %s/기존 문자열/변경할 문자열/
:%s/kr.archive.ubuntu.com/mirror.kakao.com/
```

3. 방화벽 설정

```
sudo ufw status
sudo ufw allow {port}
```

```
sudo ufw allow 9443 # portainer
sudo ufw allow 8080/tcp # jenkins
sudo ufw allow 9000/tcp # sonarqube
```

4. 사용하는 jdk 설치
   - jdk21을 사용할것이기 때문에 21버전으로 설치

```
sudo apt update
sudo apt install openjdk-21-jre-headless -y
java -version
```

도커화

Docker & Docker Compose 설치

1. Docker 설치

Install Docker Engine on Ubuntu

Jumpstart your client-side server applications with Docker Engine on Ubuntu. This guide details prerequisites and multiple methods to install Docker Engine on Ubuntu.

https://docs.docker.com/engine/install/ubuntu/

2. Docker Compose 설치

Install the Compose plugin

Download and install Docker Compose on Linux with this step-by-step handbook. This plugin can be installed manually or by using a repository.

https://docs.docker.com/compose/install/linux/

EC2 디렉토리 구조

```
mkdir docker-compose
mkdir docker-compose/volumns
mkdir deploy
mkdir deploy/backend
mkdir deploy/frontend
mkdir deploy/ai
```

deploy shell 스크립트

폴더 구조

```
deploy
-backend
--deploy.sh
-frontend
--deploy.sh
-ai
--deploy.sh
```

backend → deploy.sh

```
docker cp docker-compose-jenkins-1:/var/jenkins_home/workspace/backend-develop/Backe
docker cp /home/ubuntu/deploy/backend/app.jar docker-compose-springboot-1:/
docker restart docker-compose-springboot-1
```

frontend → deploy.sh

```
rm -rf /home/ubuntu/deploy/frontend/dist
docker cp docker-compose-jenkins-1:/var/jenkins_home/workspace/frontend-develop/Fron
docker exec -it docker-compose-nginx-1 rm -rf /usr/share/nginx/html
docker cp /home/ubuntu/deploy/frontend/dist/. docker-compose-frontend-1:/usr/share/n
docker restart docker-compose-frontend-1
```

ai → deploy.sh

```
sudo rm -rf /home/ubuntu/deploy/ai/app
docker cp docker-compose-jenkins-1:/var/jenkins_home/workspace/ai-develop/Ai/app /home/
docker exec docker-compose-fastapi-1  rm -rf /usr/src/app
docker cp /home/ubuntu/deploy/ai/app docker-compose-fastapi-1:/usr/src
docker cp /home/ubuntu/deploy/ai/.env docker-compose-fastapi-1:/usr/src/app
docker cp /home/ubuntu/deploy/ai/xlsx docker-compose-fastapi-1:/usr/src/app
docker cp /home/ubuntu/deploy/ai/ai_models docker-compose-fastapi-1:/usr/src/app
sudo docker restart docker-compose-fastapi-1
```

컨테이너 띄우기

1. docker-compose.yml 작성

▼ `docker-compose.yml`

```
version: '3.7'

services:
  nginx:
    networks:
      - appnet
    ports:
      - '80:80'
      - '443:443'
    image: nginx

  frontend:
    networks:
      - appnet
    ports:
      - '8081:81'
    image: nginx

  springboot:
    networks:
      - appnet
    command: sh -c 'if [ -e /app.jar ]; then java -jar /app.jar --spring.profiles
    image: openjdk:21

  fastapi:
    networks:
      - appnet
    image: fastapi
    command: sh -c 'cd app && uvicorn app:app  --host 0.0.0.0 --port 8080 --reloa
    ports:
```

```
      - '8087:8087'

  mysql:
    ports:
      - '3306:3306'
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - appnet
    environment:
      MYSQL_ROOT_PASSWORD: r1234!
      MYSQL_DATABASE: semento
      MYSQL_USER: dfg
      MYSQL_PASSWORD: dfg123
      TZ: Asia/Seoul
    command: --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
    image: mysql:8.0.34

  jenkins:
    ports:
      - '8080:8080'
    user: root
    networks:
      - jenkinsnet
    image: jenkins/jenkins:jdk17

  portainer:
    ports:
      - '9443:9443'
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data
    image: portainer/portainer-ce:latest


  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELASTIC_VERSION: 8.11.4
    volumes:
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/
      - elasticsearch:/usr/share/elasticsearch/data:Z
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      node.name: elasticsearch
      ES_JAVA_OPTS: -Xms512m -Xmx512m
      # Bootstrap password.
      # Used to initialize the keystore during the initial startup of
      # Elasticsearch. Ignored on subsequent runs.
      ELASTIC_PASSWORD: dfg123
      # Use single node discovery in order to disable production mode and avoid b
```

```
      # see: https://www.elastic.co/guide/en/elasticsearch/reference/current/boot
      discovery.type: single-node
      TZ: Asia/Seoul
    networks:
      - kafka-elk
    restart: unless-stopped

  logstash:
    build:
      context: logstash/
      args:
        ELASTIC_VERSION: 8.11.4
    volumes:
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro
      - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
      - ./logstash/mysql-connector-j-8.3.0.jar:/usr/share/logstash/logstash-core/
      - ./logstash/config/pipelines.yml:/usr/share/logstash/config/pipelines.yml
    ports:
      - 5044:5044
      - 50000:50000/tcp
      - 50000:50000/udp
      - 9600:9600
    environment:
      LS_JAVA_OPTS: -Xms1000m -Xmx2000m
      LS_HEAP_SIZE: 2048m
      LOGSTASH_INTERNAL_PASSWORD: dfg123
      TZ: Asia/Seoul
    logging:
      driver: json-file
      options:
        max-size: "200m"
        max-file: "10"
    networks:
      - kafka-elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: 8.11.4
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    ports:
      - 5601:5601
    environment:
      KIBANA_SYSTEM_PASSWORD: dfg123
      TZ: Asia/Seoul
    networks:
      - kafka-elk
    depends_on:
      - elasticsearch
```

```
    restart: unless-stopped

volumes:
  mysql_data: {}
  portainer_data: {}
  elasticsearch: {}

networks:
    jenkinsnet: {}
    appnet: {}
    kafka-elk:
      driver: bridge
```

2. docker compose 실행

```
sudo docker compose up -d // 백그라운드 실행
sudo docker compose up --build -d // 재빌드 후 백그라운드 실행
sudo docker compose down
```

Nginx SSL 설정

1. nginx 컨테이너 접속

```
sudo docker exec -it {docker ID} /bin/bash
```

2. Certbot 패키지 설치

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

3. nginx 설정 파일 수정

```
vi /etc/nginx/conf.d/default.conf
```

▼ `default.conf`

```
// default.conf

server {
    listen       80;

    # 도메인으로 수정
    server_name  j10s106.p.ssafy.io;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    # 백엔드 서버로 이동
    location /api {
```

```
        proxy_pass http://docker-compose-springboot-1:8080;
    }

        # vue 배포용 nginx로 이동
    location / {
        proxy_pass http://docker-compose-frontend-1:81/; # /붙여야 vue 배포용 nginx
    }

    # 이 부분 추가
    location /.well-known/acme-challenge/ {
        allow all;
        root /var/www/certbot;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

4. Certbot 인증서 발급

```
certbot --nginx -d k10s106.p.ssafy.io
```

- 관리자 email 주소 입력 `ratatou2gpt@gmail.com`
- 이메일 서버 등록 `y`
- 캠페인 소식 여부 `y`

```
Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/j10s005.p.ssafy.io/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/j10s005.p.ssafy.io/privkey.pem
This certificate expires on 2024-06-09.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for j10s005.p.ssafy.io to /etc/nginx/conf.d/default.conf
Congratulations! You have successfully enabled HTTPS on https://j10s005.p.ssafy.io

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
If you like Certbot, please consider supporting our work by:
 * Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
 * Donating to EFF:                    https://eff.org/donate-le
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

5. 인증서 자동 설정

- 인증이 성공적으로 완료되었다면 `/etc/nginx/conf.d/default.conf` 에 자동 설정됨

▼ `default.conf`

```
# /etc/nginx/conf.d/default.conf

server {
    server_name  j10s005.p.ssafy.io;

    location / {
        proxy_pass 프록시를 적용할 서버;
    }
```

```
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/도메인/fullchain.pem; # managed by Certb
    ssl_certificate_key /etc/letsencrypt/live/도메인/privkey.pem; # managed by Cer
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}

server {
    # 80 포트로 접속 시 https로 리다이렉트
    if ($host = j10s005.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen       80;
    listen  [::]:80;
    server_name  j10s005.p.ssafy.io;
    return 404; # managed by Certbot
}
```

6. nginx 리로드

```
service nginx reload
```

Portainer 접속

`https://k10s106.p.ssafy.io:9443/`

▼ `To re-enable your Portainer instance, you will need to restart Portainer.` 에러

```
sud docker restart {portainer 컨테이너}
```

- 계정
  - 아이디 : `admin`
  - 비밀번호 : `sementodfg123!`

Jenkins

jenkins 접속

Jenkins 초기 비밀번호 확인

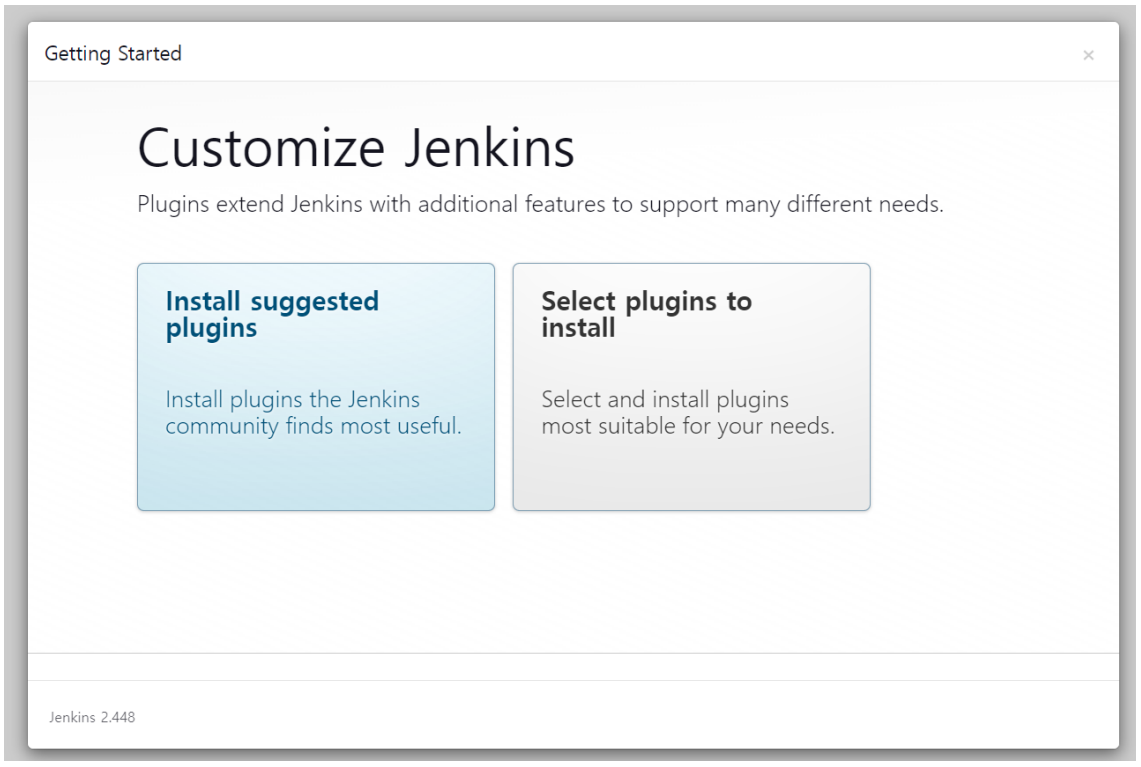
```
sudo docker logs {docker container name}
```

`63dfbd7cef63448bbe1a599d5d956705`

Jenkins URL 접속 및 초기 설정

`http://k10s106.p.ssafy.io:8080/`

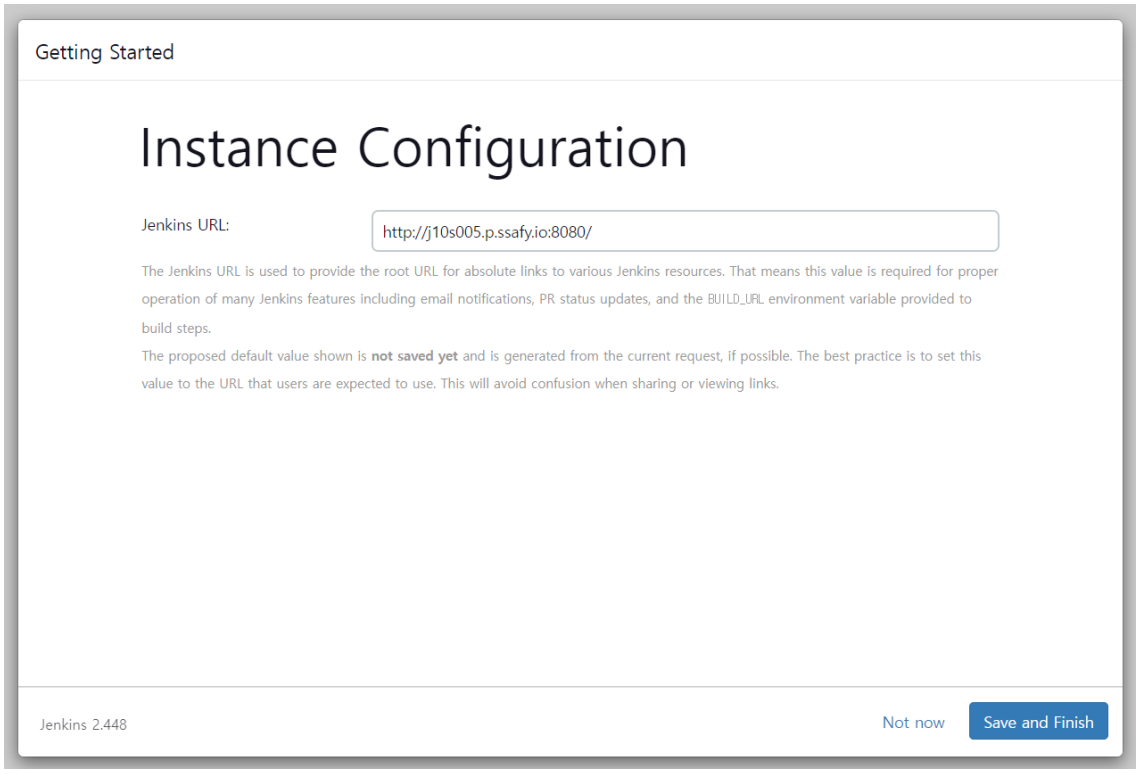
1. 비밀번호 입력
2. 시작



3. 계정 생성

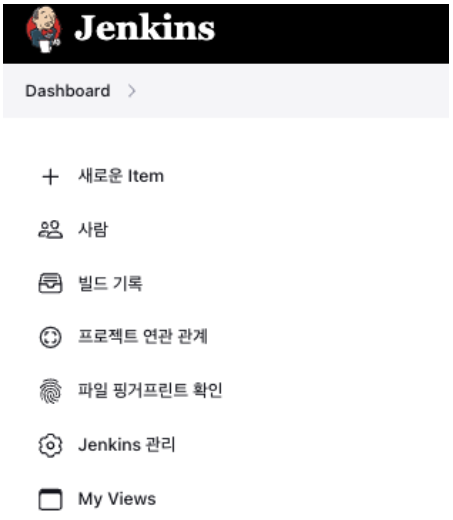
계정명	dfg
암호	dfg123!
이름	dfg
이메일 주소	ratatou2gpt@gmail.com

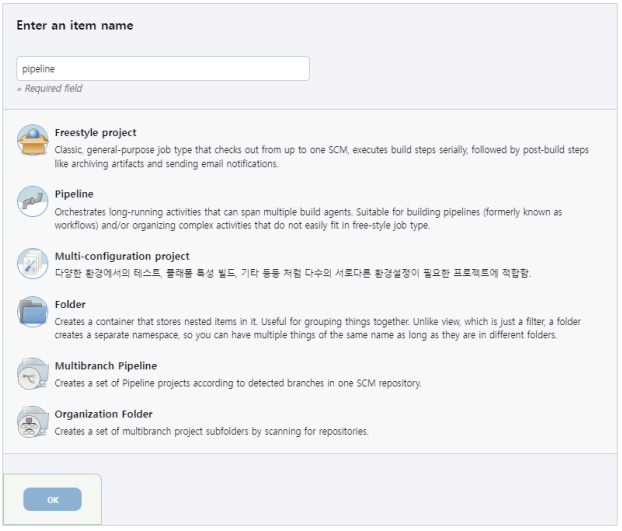
4. jenkins url 설정



jenkins 파이프라인 구축

1. item 생성



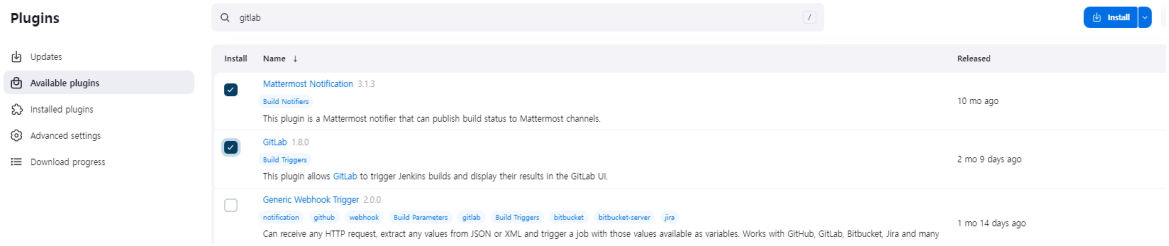


- pipeline 이름 적기
- 방식 선택
  - freestyle
    - 사용하기 쉽지만 파이프라인 커스텀이 어려움
  - pipeline

- 사용하기 어렵지만 파이프라인 커스텀 가능

2. plugin 설정

`Dashboard` → `Jenkins 관리` → `Plugins` 이동

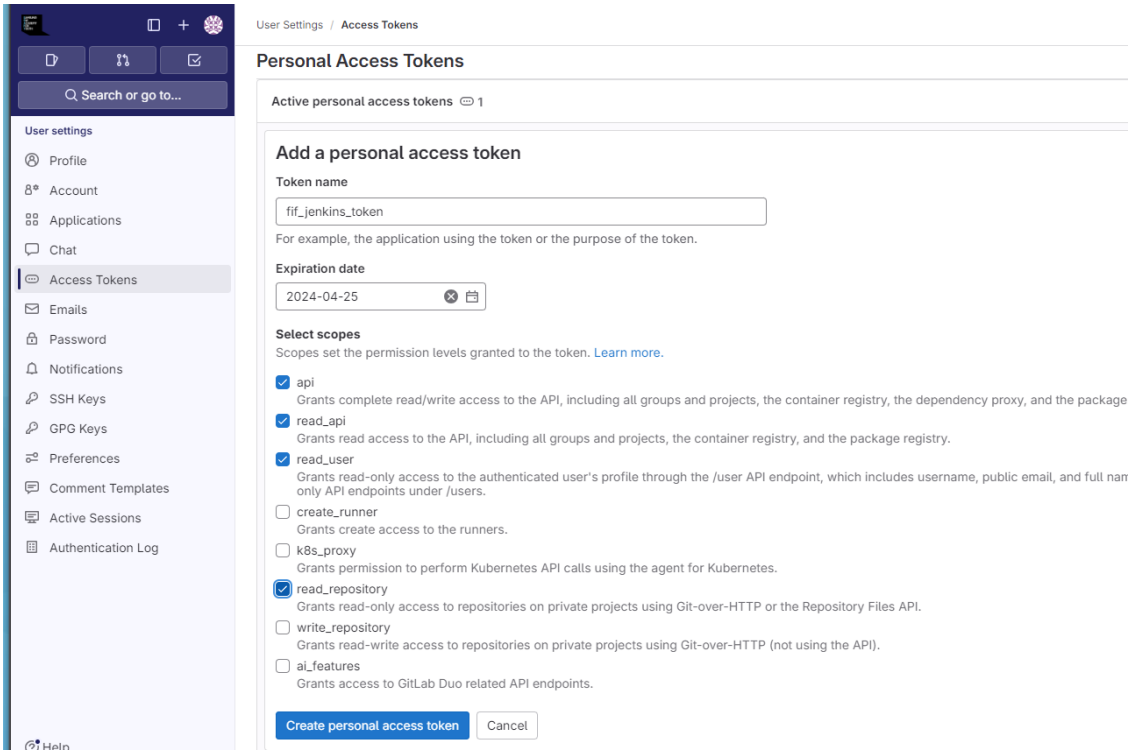


- Mattermost Notification
- GitLab
- SSH Agent
- Pipeline: stage view : build 확인 용 플러그인

3. 플러그인 설정

GitLab

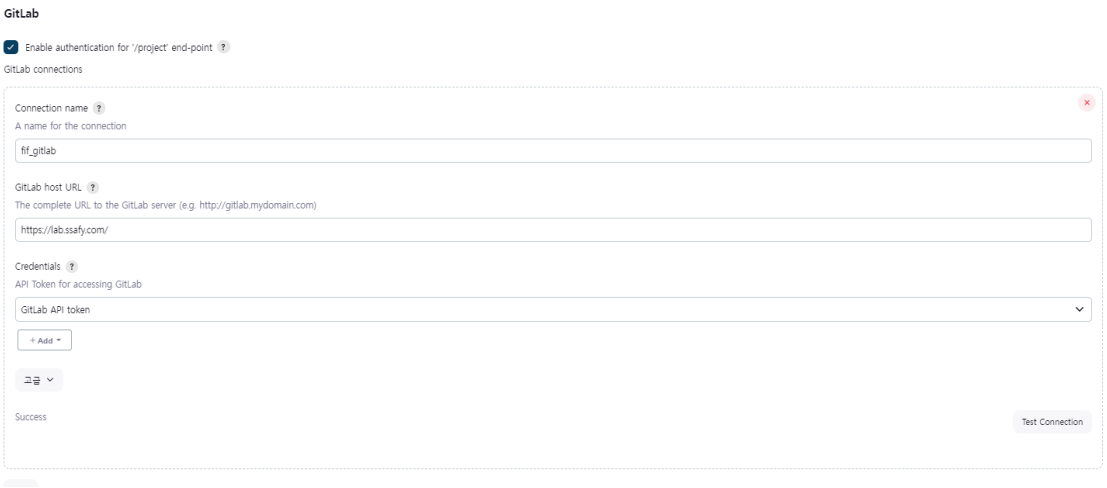
- GitLab 탭에서 Connection name과 host URL을 적어주고 Credential은 Add를 선택
  - host URL은 자신이 이용하고 있는 GitLab 서버 URL을 적으면 됨
- GitLab의 Profile → Preferences → Access Token 에서 새로운 토큰을 발급
  - 기본적인 읽기 권한은 모두 포함



`Dashboard` → `Jenkins 관리` → `System` 이동

- token과 ID 작성 후 credential 생성

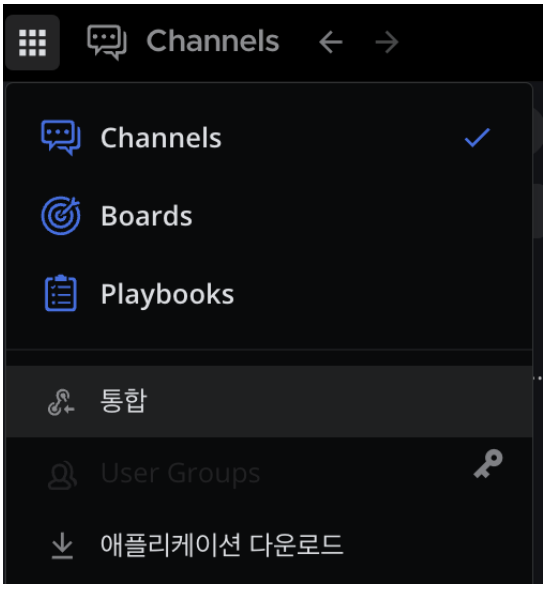
- Test Connection을 통해 정상적인지 확인

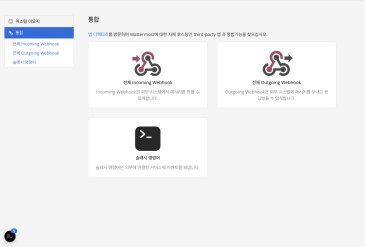


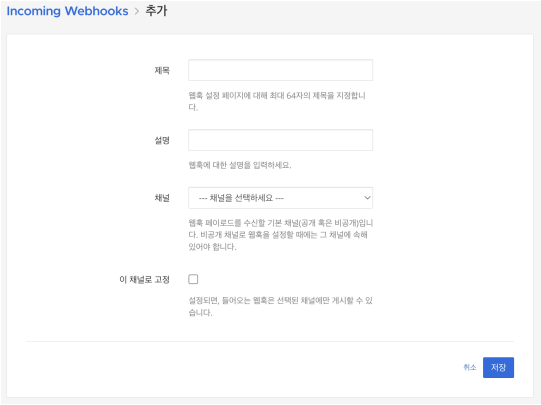
Mattermost

- 제일 하단 Global Mattermost Notifier Settins 로 이동
- mattermost 웹 훅 생성

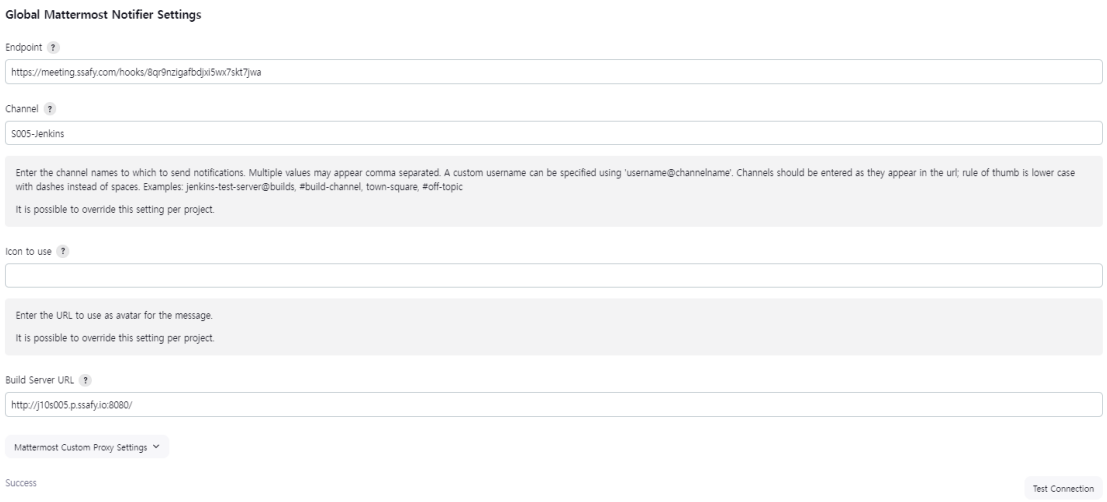
  `통합` → `전체 Incomming Webhooks` → `incomming webhook 추가` → `생성`







- 저장 후 생기는 웹 훅 기억해두기
  - `https://meeting.ssafy.com/hooks/efxboeuoh7rrby59oc1wefrtfa`
- Endpoint와 Channel 작성 후 Test Connection



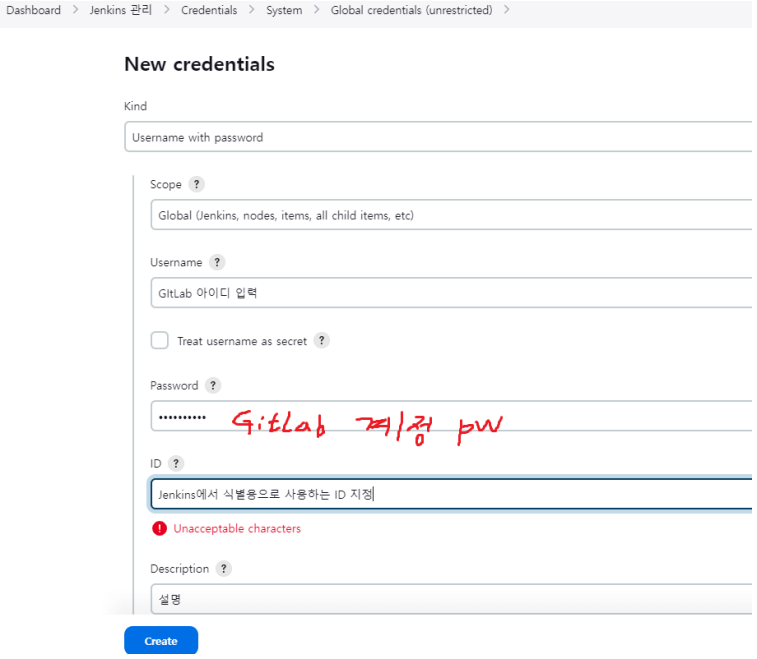
4. 권한 설정

- pipeline에 필요한 키들을 등록하는 과정
  - GitLab repo clone을 위해 필요한 키
  - ec2 접속을 위한 pem 키

`Dashboard` → `Jenkins 관리` → `Credentials` 이동 → `(global) 클릭` → `Add Credentials`

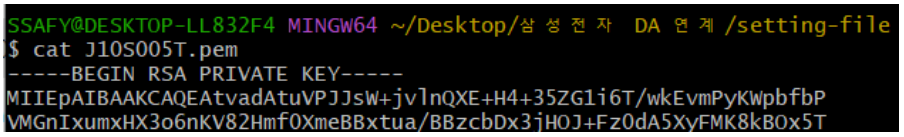
GitLab

- pipeline에서 Gitlab에 접근하기 위한 Gitlab 로그인용 Credential 생성
- Username with password 방식의 권한으로 생성
- Username에 아이디, Password에 비밀번호 입력

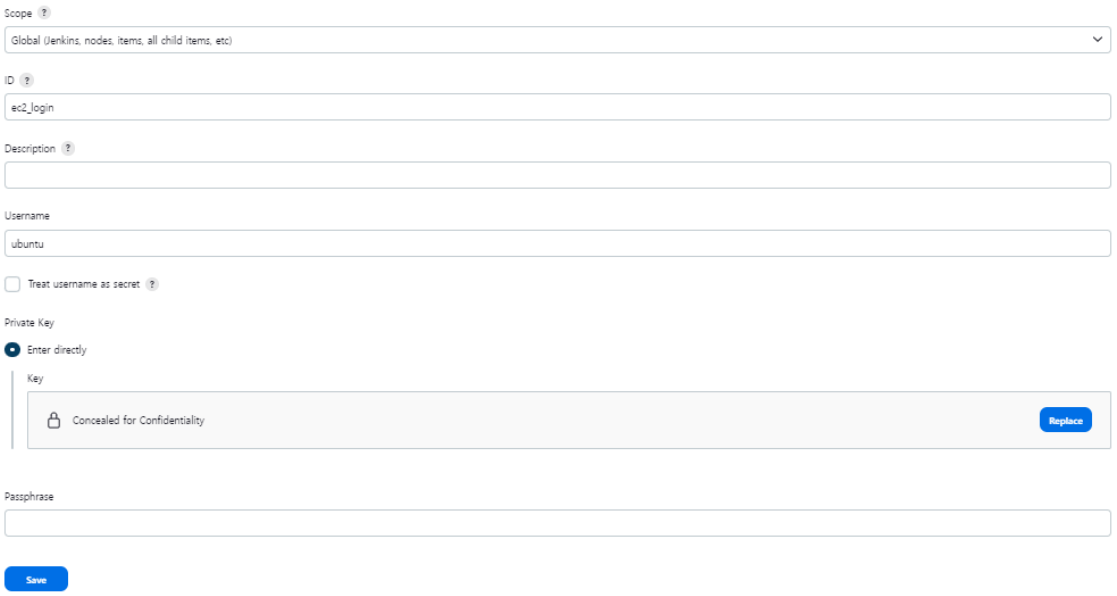


EC2

- SSH Username with private key 방식의 권한으로 생성
- ID는 고유한 값을 입력하고 Enter directly를 선택
  - 터미널에 `cat {pem key}` 로 나온 문자열을 입력



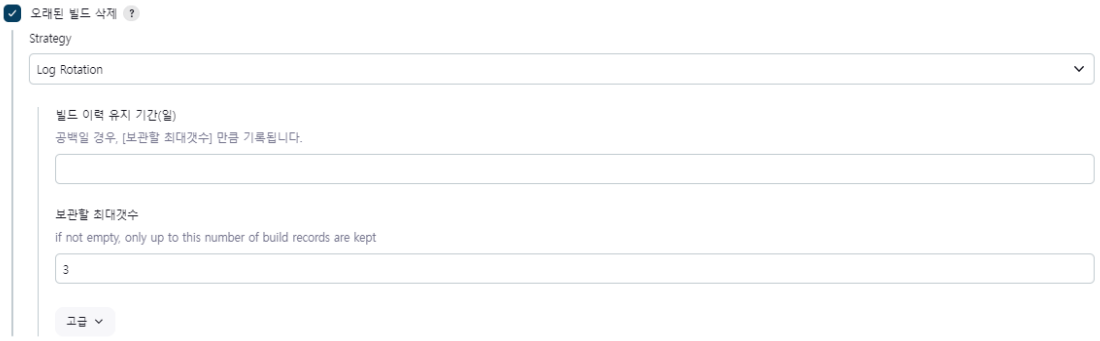
- username은 서버 접속 사용자명
- passphase는 해당 키로 서버 접속할 때 쓰는 비밀번호



5. pipeline 구성

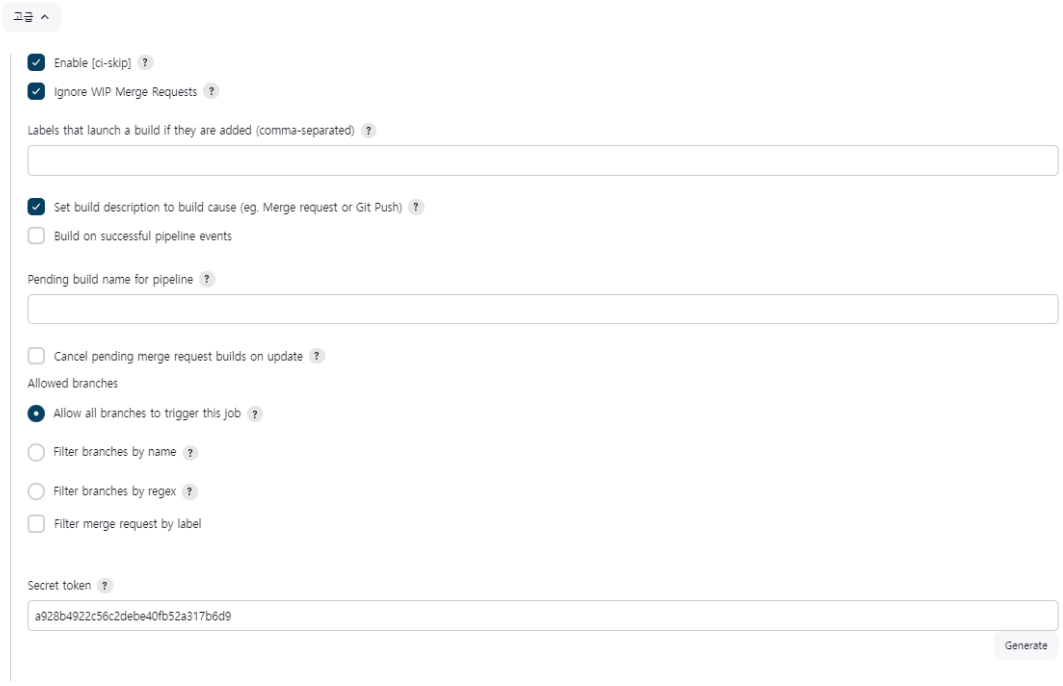
`Dashboard` → `{{자신의 Item}}` → `Configuretion` 이동

- 빌드 유지

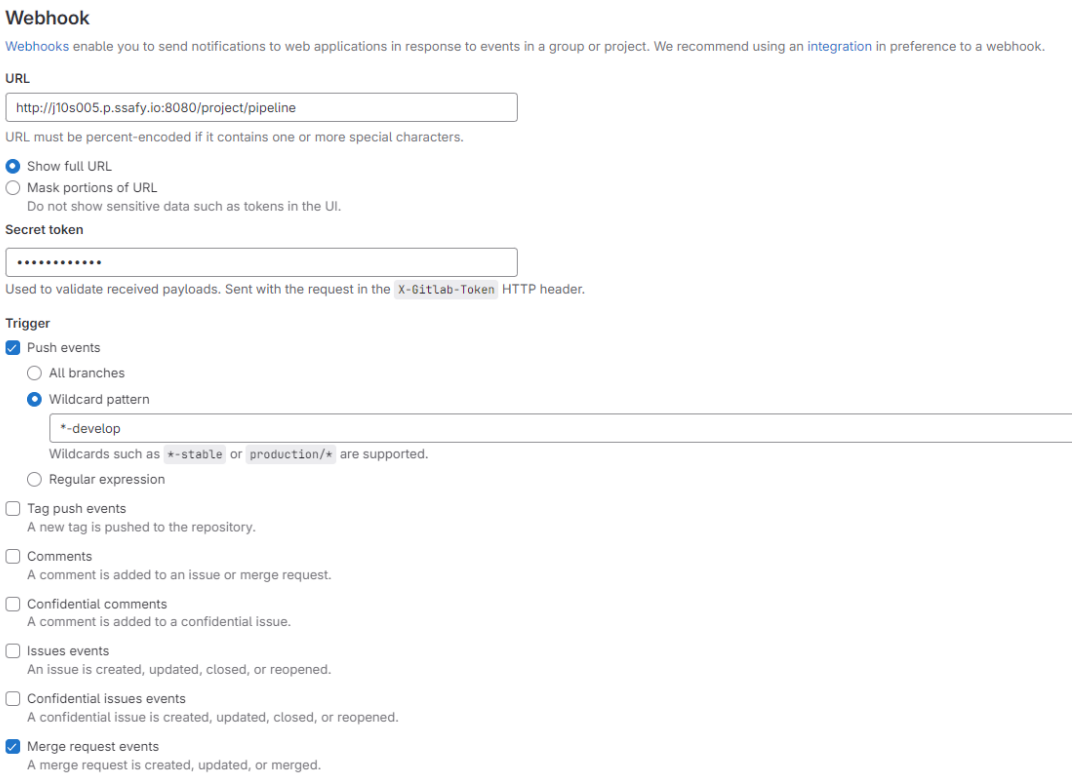


- Build Triggers
  - GitLab webhook URL : `http://k10s106.p.ssafy.io:8080/project/backend-develop`
  - push와 merge 시 트리거 작동

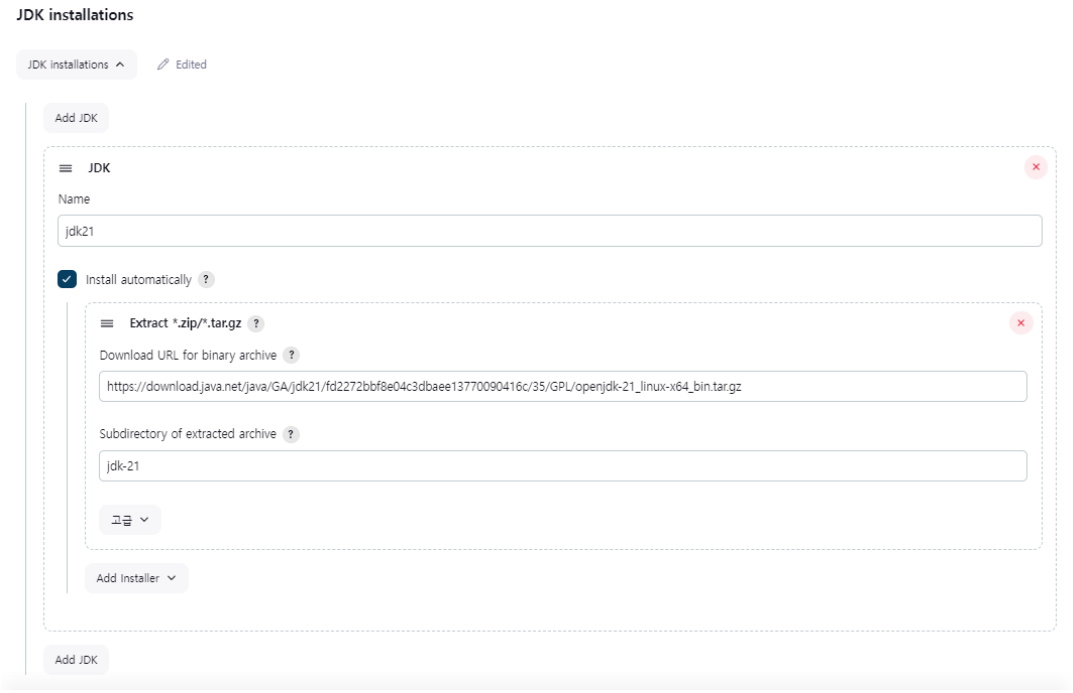
- 고급 → generate 를 통해 생성된 토큰 가지고 있기
  - Secret Token : `57da4e342df3205a053b4d71e24ea6e6`



- `GitLab` → `Repo` → `Settings` → `Webhooks` → `Add new webhooks`
  - 이전에 저장했던 url과 token을 작성
  - 언제 trigger를 발생할지 선택



- Pipeline 작성
  - 트리거가 작동했을 때 하는 행동을 설정
  - tools에 JDK 21 추가



  - git credentialsId ⇒ credential에 저장한 gitlab 로그인
  - sshagent ⇒ credential에 저장한 ec2 키

```
// be-develop

pipeline{
    agent any

    tools {
        jdk 'jdk21'
    }

    environment {
        JAVA_HOME = "tool jdk21"
    }

    stages{
        stage("Clone Repository"){
            steps{
                git credentialsId: "gitlab_login",
                url: 'https://lab.ssafy.com/s10-final/S10P31S106.git',
                branch: 'backend-develop'
            }
        }
        stage("Build"){
            steps{
                dir("Backend"){
                    sh 'chmod +x ./gradlew; ./gradlew clean bootJar'
                }
            }
            post {
                failure{
                    mattermostSend (
                        color: "danger",
                        message: ":face_with_symbols_on_mouth: [백엔드 개발 서버] 빌
                    )
                }
            }
        }
        stage("Deploy"){
            steps {
                sshagent(['ec2_login']){
                    sh 'ssh ubuntu@k10s106.p.ssafy.io "sudo sh ~/deploy/backend/d
                }
            }
            post {
                failure{
                    mattermostSend (
                        color: "danger",
                        message: ":face_with_symbols_on_mouth: [백엔드 개발 서버] 배
                    )
                }
            }
        }
    }

    post{
```

```
        success{
            mattermostSend (
                    color: "good",
                    message: ":sparkling_heart: [백엔드 개발 서버] 배포 성공!! #${env.
                )
        }
    }
}
```

Jenkins EC2 접속

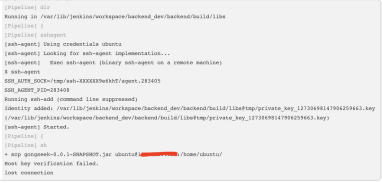
- pipeline을 통해 jenkins에서 ec2에 접근할 수 있도록 SSH Agent를 사용했다.
- 하지만, `host key verification failed.` 라는 에러가 발생할 수 있다.
- 이유는 jenkins에서 ec2로 최초 한번 ssh를 통한 접근이 필요하기 때문

  ▼ 참고

  Jenkins Host key verification failed

  문제점 Jenkins 파이프라인을 작성하다가 Host key verification failed 문제를 맞이했다. 처음에 작성한 파이프라인은 아래와 같다. pipeline { agent any stages { stage('Checkout SCM') { steps { checkout([$class: 'GitSCM', branches: [[name: '*/develop']], extensions:

  https://hellorennon.tistory.com/20

- 이를 위해 공개키/비밀키를 통해 접속했다.

  ▼ 참고

  SSH 인증키 생성 및 서버에 등록 & 간편하게 접속하기

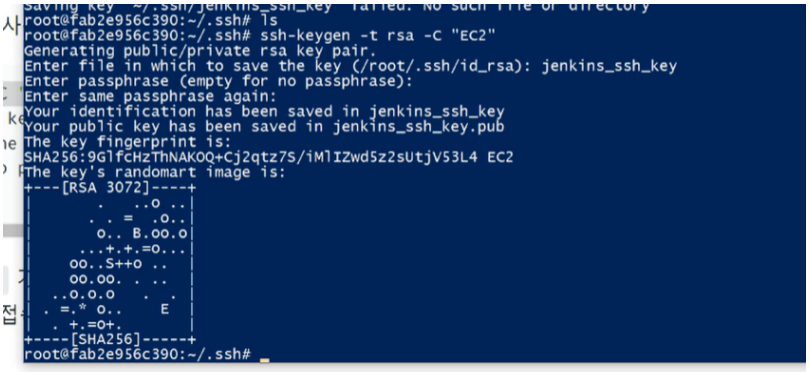
  로컬에서 ssh key를 생성하고, 생성된 ssh key를 서버에 등록하면 해당 서버에 접속하려는 계정의 비밀번호 입력없이 ssh 접속이 가능하다.클라이언트는 비밀키를 가지고 있고, 서버에 공개키를 가지고 있도록 하여 접속하는 방식이다.ssh-kegen으로 공개키/비밀

  https://velog.io/@solar/SSH-인증키-생성-및-서버에-등록-간편하게-접속하기

  velog

1. Jenkins 컨테이너에 접속하여 키를 생성한다.

```
ssh-keygen -t rsa -C "EC2"
```



2. Jenkins 컨테이너에서 생성된 키를 확인하고 복사한다.

```
cat jenkins_ssh_key.pub
```

3. EC2에 공개키 정보 저장
   - 다음을 입력하고 공개키를 붙여넣은 후 줄바꿈 한다.
   - ctrl + D를 누르면 저장됨

```
cat >> ~/.ssh/authorized_keys
```

4. Jenkins 컨테이너에서 생성된 키를 이용해 ec2에 접속한다.

```
ssh -i jenkins_ssh_key ubuntu@k10s106.p.ssafy.io
```

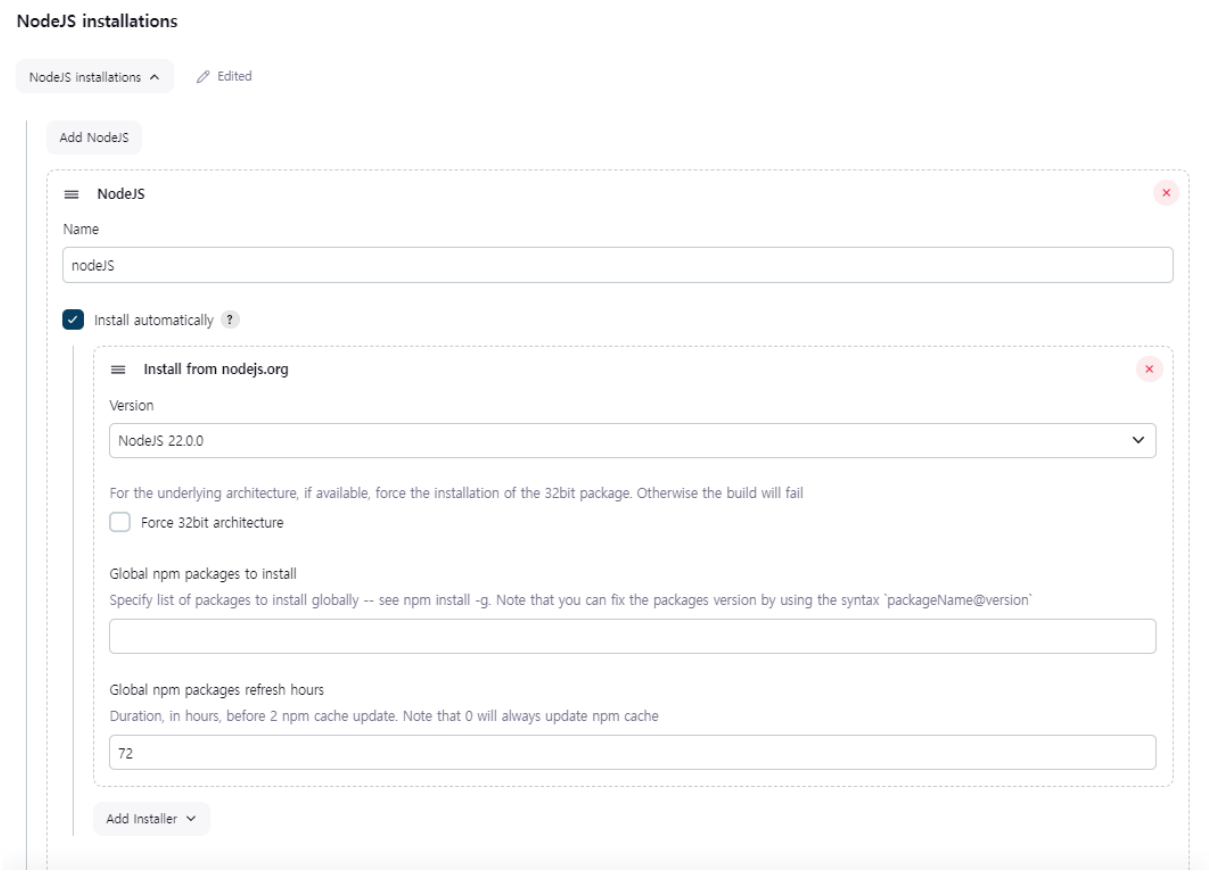
Frontend Pipeline 구축

Node JS 등록

`Jenkins 관리` → `Plugins`

- NodeJS 설치

`Jenkins 관리` → `Tools`



Pipeline 생성

Build Triggers

- Gitlab webhook : `http://k10s106.p.ssafy.io:8080/project/frontend-develop`
- generate token : `697b804af6c8dd05014ffe872d5d439e`

Pipeline

```
pipeline {
    agent any

    tools {
        nodejs "nodeJS"
    }

    stages {
        stage('Clone Repository') {
            steps{
                git credentialsId: "gitlab_login",
                url: 'https://lab.ssafy.com/s10-final/S10P31S106.git',
                branch: 'frontend-develop'
            }
        }

        stage('Install Dependencies') {
            steps {
                dir('Frontend') {
                    sh 'npm install'
                }
            }
        }

        stage('Build') {
            steps {
                dir('Frontend') {
                    sh 'npm run build'
                }
            }
        }

        stage("Deploy"){
            steps {
                sshagent(['ec2_login']){
                    sh 'ssh ubuntu@k10s106.p.ssafy.io "sudo sh ~/deploy/frontend/dep
                }
            }
            post {
                failure{
                    mattermostSend (
                        color: "danger",
```

```
                        message: ":face_with_symbols_on_mouth: [프론트 개발 서버] 배포 실
                    )
                }
            }
        }
    }
    post{
        success{
            mattermostSend (
                    color: "good",
                    message: ":sparkling_heart: [프론트 개발 서버] 배포 성공!! #${env.BUI
                )
        }
    }
}
```

Nginx Proxy Pass 설정

1. nginx 컨테이너 접속
2. /etc/nginx/conf.d/default.conf 이동
3. default.conf

```
server {
    listen       81; # 내부 포트
    listen  [::]:81; # 내부 포트
    server_name  localhost;

    #access_log  /var/log/nginx/host.access.log  main;

    #### SPA이기 때문에 index.html 외 다른 페이지 새로고침시 404에러 ####
    #### 모든 페이지 호출시 무조건 index.html을 거쳐 호출되도록 설정 ####
    location / {
        try_files $uri $uri/ /index.html = 404;
        root   /usr/share/nginx/html/dist;
    }
```

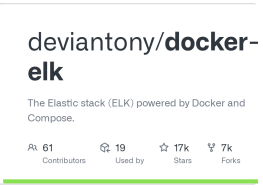
ELK

ELK + Kafka 컨테이너 띄우기

1. ELK 컨테이너 구조 clone
   - ELK 설치 시 에러가 많이 발생하므로 이미 잘 구성해놓은 repo를 clone받아 사용

Docker Kafka + ELK 연동
Kafka + ELK 연동
https://tries1.github.io/spring/2021/01/17/kafka_elk.html

GitHub - deviantony/docker-elk: The Elastic stack (ELK) powered by Docker and Compose.
The Elastic stack (ELK) powered by Docker and Compose. - deviantony/docker-elk
https://github.com/deviantony/docker-elk



   - docker-compose 폴더 안에 docker-elk 레포 내의 모든 파일을 이동시킴
     - 주의 : docker-compose.yml이 겹쳐 기존 yml 파일이 사라질 수 있으니 조심해야함(이름 바꿔놓기)

```
mv docker-elk/* docker-compose/
```

2. docker-compose.yml 수정하기
   - repo안에 있던 docker-compose.yml 파일에서 services에 해당하는 내용들을 이동

▼ docker-compose-with-kafka.yml

```
version: '3.7'

services:
  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELASTIC_VERSION: 8.11.0
    volumes:
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/conf
      - elasticsearch:/usr/share/elasticsearch/data:Z
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      node.name: elasticsearch
      ES_JAVA_OPTS: -Xms512m -Xmx512m
      # Bootstrap password.
      # Used to initialize the keystore during the initial startup of
      # Elasticsearch. Ignored on subsequent runs.
      ELASTIC_PASSWORD: dfg123
      # Use single node discovery in order to disable production mode and avoi
      # see: https://www.elastic.co/guide/en/elasticsearch/reference/current/b
      discovery.type: single-node
    networks:
      - kafka-elk
    restart: unless-stopped
```

```
  logstash:
    build:
      context: logstash/
      args:
        ELASTIC_VERSION: 8.11.0
    volumes:
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml
      - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
      - ./logstash/mysql-connector-j-8.3.0.jar:/usr/share/logstash/logstash-co
    ports:
      - 5044:5044
      - 50000:50000/tcp
      - 50000:50000/udp
      - 9600:9600
    environment:
      LS_JAVA_OPTS: -Xms1000m -Xmx2000m
      LS_HEAP_SIZE: 2048m
      LOGSTASH_INTERNAL_PASSWORD: dfg123
    networks:
      - kafka-elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: 8.11.0
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    ports:
      - 5601:5601
    environment:
      KIBANA_SYSTEM_PASSWORD: dfg123
    networks:
      - kafka-elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  zookeeper:
    container_name: zookeeper
    image: confluentinc/cp-zookeeper:latest
    ports:
      - "9900:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    networks:
      - kafka-elk

  kafka:
```

```
    container_name: kafka
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://loc
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST
            KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_CREATE_TOPICS: "test-topic:1:1"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - kafka-elk

networks:
  kafka-elk:
    driver: bridge

volumes:
  elasticsearch:
```

- ELK 버전의 경우 JDK 버전에 맞춰 선택해야함

```
JDK21 에서는 8.11.0 으로 사용
```

- 수정한 docker-compose.yml
  - 환경변수를 모두 바꿔줘야함에 주의

▼ docker-compose.yml

```
version: '3.0'

services:
  nginx:
    networks:
      - appnet
    ports:
      - '80:80'
      - '443:443'
    image: nginx

  frontend:
    networks:
      - appnet
    ports:
      - '8081:81'
    image: nginx

  springboot:
    networks:
```

```
      - appnet
    command: sh -c 'if [ -e /app.jar ]; then java -jar /app.jar --spring.profi
    image: openjdk:21

  mysql:
    ports:
      - '3306:3306'
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - appnet
    environment:
      MYSQL_ROOT_PASSWORD: r1234!
      MYSQL_DATABASE: semento
      MYSQL_USER: dfg
      MYSQL_PASSWORD: dfg123
      TZ: Asia/Seoul
    command: --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode
    image: mysql:8.0.34

  jenkins:
    ports:
      - '8080:8080'
    user: root
    networks:
      - jenkinsnet
    image: jenkins/jenkins:jdk17

  portainer:
    ports:
      - '9443:9443'
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data
    image: portainer/portainer-ce:latest


  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELASTIC_VERSION: 8.11.0
    volumes:
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/conf
      - elasticsearch:/usr/share/elasticsearch/data:Z
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      node.name: elasticsearch
      ES_JAVA_OPTS: -Xms512m -Xmx512m
      # Bootstrap password.
      # Used to initialize the keystore during the initial startup of
      # Elasticsearch. Ignored on subsequent runs.
```

```
      ELASTIC_PASSWORD: dfg123
      # Use single node discovery in order to disable production mode and avoi
      # see: https://www.elastic.co/guide/en/elasticsearch/reference/current/b
      discovery.type: single-node
      TZ: Asia/Seoul
    networks:
      - kafka-elk
    restart: unless-stopped

  logstash:
    build:
      context: logstash/
      args:
        ELASTIC_VERSION: 8.11.0
    volumes:
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml
      - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
      - ./logstash/mysql-connector-j-8.3.0.jar:/usr/share/logstash/logstash-co
          - ./logstash/config/pipelines.yml:/usr/share/logstash/config/pipel
    ports:
      - 5044:5044
      - 50000:50000/tcp
      - 50000:50000/udp
      - 9600:9600
    environment:
      LS_JAVA_OPTS: -Xms1000m -Xmx2000m
      LOGSTASH_INTERNAL_PASSWORD: dfg123
      TZ: Asia/Seoul
    networks:
      - kafka-elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: 8.11.0
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    ports:
      - 5601:5601
    environment:
      KIBANA_SYSTEM_PASSWORD: dfg123
      TZ: Asia/Seoul
    networks:
      - kafka-elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  zookeeper:
    container_name: zookeeper
```

```
    image: confluentinc/cp-zookeeper:latest
    ports:
      - "9900:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    networks:
      - kafka-elk

  kafka:
    container_name: kafka
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://loc
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_CREATE_TOPICS: "test-topic:1:1"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - kafka-elk

volumes:
  mysql_data: {}
  portainer_data: {}
  elasticsearch: {}

networks:
    jenkinsnet: {}
    appnet: {}
    kafka-elk:
      driver: bridge
```

X-path 설정 지우기

- ELK의 모니터링 유료 서비스라고함
- 그래서 관련 설정을 모두 지워줘야함
- but! elasticksearch에서도 지워버릴 경우 다음과 같은 에러가 발생할 수 있으니 주의
  - `exception in thread "main" java.nio.file.filesystemexception: /usr/share/elasticsearch/config/elasticsearch.yml: device or resource busy`

1. `elasticsearch` → `config` → `elasticsearch.yml`

   ▼ elasticsearch.yml

```
---
## Default Elasticsearch configuration from Elasticsearch base image.
## https://github.com/elastic/elasticsearch/blob/main/distribution/docker/src/doc
#
cluster.name: docker-cluster
network.host: 0.0.0.0
path.data: /usr/share/elasticsearch/data

## X-Pack settings
## see https://www.elastic.co/guide/en/elasticsearch/reference/current/security-s
#
xpack.license.self_generated.type: basic
xpack.security.enabled: false
```

2. `kibana` → `config` → `kibana.yml`

▼ kibana.yml

```
---
## Default Kibana configuration from Kibana base image.
## https://github.com/elastic/kibana/blob/main/src/dev/build/tasks/os_packages/do
#
server.name: kibana
server.host: 0.0.0.0
elasticsearch.hosts: [ http://elasticsearch:9200 ]

monitoring.ui.container.elasticsearch.enabled: true
monitoring.ui.container.logstash.enabled: true

## X-Pack security credentials
#
elasticsearch.username: kibana_system
elasticsearch.password: dfg123

## Encryption keys (optional but highly recommended)
##
## Generate with either
##  $ docker container run --rm docker.elastic.co/kibana/kibana:8.6.2 bin/kibana-
##  $ openssl rand -hex 32
##
## https://www.elastic.co/guide/en/kibana/current/using-kibana-with-security.html
## https://www.elastic.co/guide/en/kibana/current/kibana-encryption-keys.html
#
#xpack.security.encryptionKey:
#xpack.encryptedSavedObjects.encryptionKey:
#xpack.reporting.encryptionKey:

## Fleet
## https://www.elastic.co/guide/en/kibana/current/fleet-settings-kb.html
#
#xpack.fleet.agents.fleet_server.hosts: [ http://fleet-server:8220 ]

#xpack.fleet.outputs:
```

```
#  - id: fleet-default-output
#    name: default
#    type: elasticsearch
#    hosts: [ http://elasticsearch:9200 ]
#    is_default: true
#    is_default_monitoring: true

#xpack.fleet.packages:
#  - name: fleet_server
#    version: latest
#  - name: system
#    version: latest
#  - name: elastic_agent
#    version: latest
#  - name: docker
#    version: latest
#  - name: apm
#    version: latest

#xpack.fleet.agentPolicies:
#  - name: Fleet Server Policy
#    id: fleet-server-policy
#    description: Static agent policy for Fleet Server
#    monitoring_enabled:
#      - logs
#      - metrics
#    package_policies:
#      - name: fleet_server-1

#        package:
#          name: fleet_server
#      - name: system-1
#        package:
#          name: system
#      - name: elastic_agent-1
#        package:
#          name: elastic_agent
#      - name: docker-1
#        package:
#          name: docker
#  - name: Agent Policy APM Server
#    id: agent-policy-apm-server
#    description: Static agent policy for the APM Server integration
#    monitoring_enabled:
#      - logs
#      - metrics
#    package_policies:
#     - name: system-1
#       package:
#         name: system
#     - name: elastic_agent-1
#       package:
#         name: elastic_agent
#     - name: apm-1
```

```
#       package:
#         name: apm
        # See the APM package manifest for a list of possible inputs.
        # https://github.com/elastic/apm-server/blob/v8.5.0/apmpackage/apm/manife
        #       inputs:
        # - type: apm
        #   vars:
        #     - name: host
        #       value: 0.0.0.0:8200
         #     - name: url
         #      value: http://apm-server:8200
```

3. `logstash` → `config` → `logstash.yml`

▼ logstash.yml

```
---
## Default Logstash configuration from Logstash base image.
## https://github.com/elastic/logstash/blob/main/docker/data/logstash/config/logs
#
http.host: 0.0.0.0

node.name: logstash
```

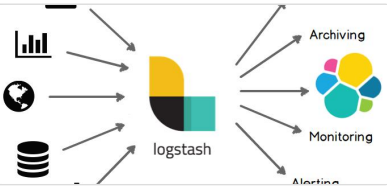
Logstash 설정

1. mysql connector jar 파일 다운로드
   - logstash 컨테이너에 volume 연결이 되어 있음
   - `logstash` 내에 jar 파일 위치 시키기

Maven Repository: com.mysql » mysql-connector-j » 8.3.0
https://mvnrepository.com/artifact/com.mysql/mysql-connector-j/8.3.0

2. logstash.conf 파일 설정

[elasticsearch] 로그스태시로 엘라스틱서치 MySQL 동기화하기
[elasticsearch] 로그스태시로 엘라스틱서치 mysql 동기화하기 ELK Stack은 Elastic Search, Logstash, Kibana를 통칭하는 말인데, 데이터 로그들이 logstash를 거쳐 엘라스틱서치에 저장되고, 이를 시각화하여 Kibana에서 보여주게 된다. 그중에서도 로그스태시는 다양한 소스에서 데이터를 수집해
https://devuna.tistory.com/94



ELK, Mysql, Kafka 구축 및 연동
0. 개요 ELK란 Elsaticsearch + Logstash + Kibana를 통합해서 부르는 말입니다. 이 글에서는 ELK, Mysql, Kafka를 구축해서 연동할 것입니다. 프로젝트 구조도는 아래와 같습니다. 아래는 ELK를 구축하기 위한 프로젝트 구조입니다. ES-springBoot:. └─es ├─docker-compose.yml │
https://ksb-dev.tistory.com/320

▼ logstash → pipeline → logstash.conf

```
input {
        jdbc {
                jdbc_driver_library => "/usr/share/logstash/logstash-core/lib/jar
                jdbc_driver_class => "com.mysql.cj.jdbc.Driver"
                jdbc_connection_string => "jdbc:mysql://k10s106.p.ssafy.io:3306/o
                jdbc_user => "dfg"
                jdbc_password => "dfg123"
                schedule => "*/1 * * * * *"
                statement => "select * from log where curr_time > :sql_last_value
                tracking_column => "curr_time"
                use_column_value => true
                tags => ["mysql"]
                codec => plain {
                        charset=>"UTF-8"
                }
        }
        stdin {
                codec => plain {
                        charset=>"UTF-8"
                }
        }
}


filter{
                # 임시저장용 필드 생성
        mutate {
                add_field => { "temp_date" => "%{curr_time}" }
        }
        # UTC 시간을 읽어 Asia/Seoul로 변경 후 문자열 저장
        ruby {
                code => "
                        require 'time'
                        utc_time = Time.parse(event.get('temp_date')).utc
                        korea_time = utc_time.getlocal('+09:00')  # Seoul은 UTC+9
                        event.set('temp_date', korea_time.strftime('%Y-%m-%dT%H:%
                "
        }
        # T를 기준으로 문자열을 나눠 년월일 분리
        mutate {
                split => { "temp_date" => "T"}
        }
        # 년월일에 해당하는 문자열을 저장
        ruby {
                code => "event.set('formatted_curr_date', event.get('[temp_date][
        }
        # 임시저장용 필드 삭제
        mutate {
                remove_field => "temp_date"
        }
        # 시간 변수 timezone 변경
        mutate {
                convert => { "curr_time" => "string" }
        }
```

```
        ruby {
                code => "
                        require 'time'
                        utc_time = Time.parse(event.get('curr_time')).utc
                        korea_time = utc_time.getlocal('+09:00')  # Seoul은 UTC+9
                        event.set('curr_time', korea_time.strftime('%Y-%m-%dT%H:%
                "
        }

        mutate {
                convert => { "start_time" => "string" }
        }
        ruby {
                code => "
                        require 'time'
                        utc_time = Time.parse(event.get('start_time')).utc
                        korea_time = utc_time.getlocal('+09:00')  # Seoul은 UTC+9
                        event.set('start_time', korea_time.strftime('%Y-%m-%dT%H:
                "
        }
        mutate {
                convert => { "timestamp" => "string" }
        }
        ruby {
                code => "
                        require 'time'
                        utc_time = Time.parse(event.get('@timestamp')).utc
                        korea_time = utc_time.getlocal('+09:00')  # Seoul은 UTC+9
                        event.set('@timestamp', korea_time.strftime('%Y-%m-%dT%H:
                "
        }

}

output {
        elasticsearch {
                hosts => "elasticsearch:9200"
                user => "elastic"
                index => "semento-mysql-logs-%{formatted_curr_date}"
                password => "dfg123"
                document_id => "%{doc_id}"
        }
        stdout {
                codec => rubydebug
        }
}
```

pipeline 생성

1. logstash → config → pipelines.yml

   ▼ pipelines.yml

```
- pipeline.id: semento_logstash
  path.config: "/usr/share/logstash/pipeline/logstash.conf"
```

Backend 파이프라인 변경

환경변수 파일 추가

- src → main → resouces 밑에 중요 정보를 담은 config.properties 위치
- gitlab에 올라가지 않으므로 따로 빌드 시에 포함해줘야함

1. `deploy` → `backend` → `config.properties` 위치 시키기

▼ config.properties

```
DATASOURCE_URL=jdbc:mysql://k10s106.p.ssafy.io:3306/oht_log?serverTimezone=Asia/S
DATASOURCE_USERNAME=root
DATASOURCE_PASSWORD=r1234!

# host에 http는 제외해야함
ELASTICSEARCH_HOST=k10s106.p.ssafy.io:9200
#ELASTICSEARCH_USERNAME=elastic
#ELASTICSEARCH_PASSWORD=dfg123
```

환경변수 스크립트 작성

- 환경변수 파일을 EC서버 → Jenkins 컨테이너로 이동시킬 스크립트 파일 작성
- `deploy` → `backend` → `config.sh`

▼ config.sh

```
docker cp /home/ubuntu/deploy/backend/config.properties docker-compose-jenkins-1:
```

Backend pipeline 변경

- 기존 backend 파이프라인에서 빌드 전 config.sh를 실행시키는 단계 추가

▼ pipeline

```
// be-develop

pipeline{
    agent any

    tools {
        jdk 'jdk21'
    }
```

```
    environment {
        JAVA_HOME = "tool jdk21"
    }

    stages{
        stage("Clone Repository"){
            steps{
                git credentialsId: "gitlab_login",
                url: 'https://lab.ssafy.com/s10-final/S10P31S106.git',
                branch: 'backend-develop'
            }
        }
        stage("Build"){
            steps{
                sshagent(['ec2_login']){
                    sh 'ssh ubuntu@k10s106.p.ssafy.io "sudo sh ~/deploy/backend/conf
                }

                dir("Backend"){
                    sh 'chmod +x ./gradlew; ./gradlew clean bootJar'
                }
            }
            post {
                failure{
                    mattermostSend (
                        color: "danger",
                        message: ":blue_heart: [**BE** 개발 서버] 빌드 실패... :loopy-sa
                    )
                }
            }
        }
        stage("Deploy"){
            steps {
                sshagent(['ec2_login']){
                    sh 'ssh ubuntu@k10s106.p.ssafy.io "sudo sh ~/deploy/backend/depl
                }
            }
            post {
                failure{
                    mattermostSend (
                        color: "danger",
                        message: ":blue_heart: [**BE** 개발 서버] 배포 실패... :loopy-sa
                    )
                }
            }
        }
    }

    post{
        success{
            mattermostSend (
                    color: "good",
                    message: ":blue_heart: [**BE** 개발 서버] 배포 성공!!! :loopy_hahaha
                )
```

```
        }
    }
}
```

FastAPI 서버 구동을 위한 파일

app.zip

`/home/ubuntu/deploy/ai` **에 압축 해제**

참고용 자료 모음

Nginx Proxy Pass 설정 팁

- 같은 네트워크 상에 묶인 서버라면 접근 가능
- proxy pass에서는 컨테이너명 또는 네트워크 ip로 접근가능함
- 내부 ip 확인

```
# 네트워크 확인
sudo docker network ls

# 특정 네트워크 목록 확인
sudo docker network inspect {네트워크명}
```

2. nginx default.conf 변경

▼ `default.conf`

```
server {
    listen       80;

    # 도메인으로 수정
    server_name  j10s005.p.ssafy.io;

    location /api/ {
                proxy_pass http://172.18.0.3:8080/api/;
    }

    location /api/ai/ {
                proxy_pass http://172.18.0.5:8087/api/ai/;
    }
...
```

Docker

▼ `docker-compose.yml`

```
version: '3.7'

services:
  nginx:
    networks:
      - appnet
    ports:
      - '80:80'
      - '443:443'
    image: nginx

  frontend:
    networks:
      - appnet
    ports:
      - '8081:81'
    image: nginx

  springboot:
    networks:
      - appnet
    command: sh -c 'if [ -e /app.jar ]; then java -jar /app.jar --spring.profiles.ac
    image: openjdk:21

  fastapi:
    networks:
      - appnet
    image: hj-fastapi:latest
    command: sh -c 'cd /usr/src/app && uvicorn app:app  --host 0.0.0.0 --port 8087 -
    ports:
      - '8087:8087'

  mysql:
    ports:
      - '3306:3306'
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - appnet
    environment:
      MYSQL_ROOT_PASSWORD: r1234!
      MYSQL_DATABASE: semento
      MYSQL_USER: dfg
      MYSQL_PASSWORD: dfg123
      TZ: Asia/Seoul
    command: --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
    image: mysql:8.0.34
```

```
  jenkins:
    ports:
      - '8080:8080'
    user: root
    networks:
      - jenkinsnet
    image: jenkins/jenkins:jdk17

  portainer:
    ports:
      - '9443:9443'
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data
    image: portainer/portainer-ce:latest


  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELASTIC_VERSION: 8.11.4
    volumes:
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/ela
      - elasticsearch:/usr/share/elasticsearch/data:Z
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      node.name: elasticsearch
      ES_JAVA_OPTS: -Xms512m -Xmx512m
      # Bootstrap password.
      # Used to initialize the keystore during the initial startup of
      # Elasticsearch. Ignored on subsequent runs.
      ELASTIC_PASSWORD: dfg123
      # Use single node discovery in order to disable production mode and avoid boot
      # see: https://www.elastic.co/guide/en/elasticsearch/reference/current/bootstr
      discovery.type: single-node
      TZ: Asia/Seoul
    networks:
      - kafka-elk
    restart: unless-stopped

  logstash:
    build:
      context: logstash/
      args:
        ELASTIC_VERSION: 8.11.4
    volumes:
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro,Z
      - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
      - ./logstash/mysql-connector-j-8.3.0.jar:/usr/share/logstash/logstash-core/lib
      - ./logstash/config/pipelines.yml:/usr/share/logstash/config/pipelines.yml
```

```
    ports:
      - 5044:5044
      - 50000:50000/tcp
      - 50000:50000/udp
      - 9600:9600
    environment:
      LS_JAVA_OPTS: -Xms1000m -Xmx2000m
      LS_HEAP_SIZE: 2048m
      LOGSTASH_INTERNAL_PASSWORD: dfg123
      TZ: Asia/Seoul
    logging:
      driver: json-file
      options:
        max-size: "200m"
        max-file: "10"
    networks:
      - kafka-elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: 8.11.4
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    ports:
      - 5601:5601
    environment:
      KIBANA_SYSTEM_PASSWORD: dfg123
      TZ: Asia/Seoul
    networks:
      - kafka-elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

volumes:
  mysql_data: {}
  portainer_data: {}
  elasticsearch: {}

networks:
    jenkinsnet: {}
    appnet: {}
    kafka-elk:
      driver: bridge
```

▼ `Dockerfile`

```
# miniconda 이미지
FROM continuumio/miniconda3:latest
```

```
# python 환경 설정
RUN conda install -y python==3.9.19
RUN conda update conda

WORKDIR /usr/src/app

COPY ./app /app
COPY ./.env /app/.env
COPY ./ai_models /app/ai_models
COPY ./xlsx /app/xlsx

RUN pip install -r /app/requirements.txt
```

▼ `docker-compose/ai/.env`

.env

▼ 주의할점

- .env 파일의 경우 우분투에서는 변수의 대소문자 구분
- pywin32는 윈도우에서 필요한 모듈로 requirements.txt에서 삭제함

▼ 참고

[패스트캠퍼스 챌린지 44일차] Python 기반 Jenkins CI Pipeline Build

이번 글에서는 Python ML backend app이 있다고 가정하고 Jenkins Pipeline을 생성하여 application을 배포해보도록 하겠습니다. jenkins pipeline 하위 폴더에 app폴더를 생성하고 main.py에 아래와 같은 코드를 작성합니다. 이제 Jenkins CI Pipeline 생성을 python application에 적용하기

https://hotorch.tistory.com/187

Refs.

미래 & 서현 & 희중 CI/CD정리

SpringBoot, Vue3 프로젝트 CI/CD (1)

아키텍처 구조 본 글에서는 SpringBoot와 Vue3 사용한 '날숨' 프로젝트의 자동 빌드, 배포하는 과정을 설명한다. 아래는 필자가 진행한 '날숨' 프로젝트의 아키텍처 구조이다. Docker를 사용하며, Docker를 개념적으로 두 개의 영역으로 구분했다. 왼쪽은 배포한 웹 애플리케이션이 동작하는 영역이며, 클라이언트 요청에 대해

https://gunjoon.tistory.com/198