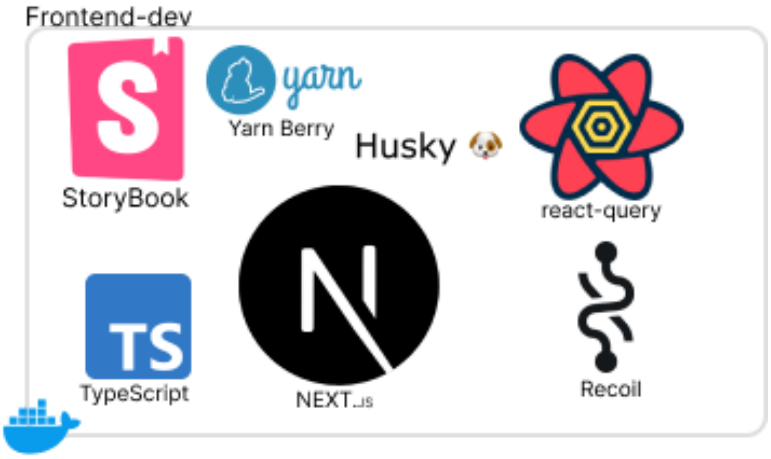
포팅메뉴얼

소유자	H4R1B0
태그	

목차

Develop Environment

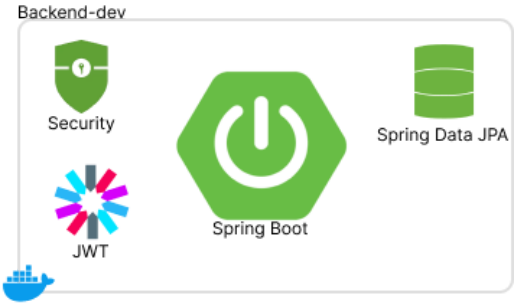
Front-End



- yarn berry 3.6.4
- node 18.17.1 LTS
- 기본 설정

    - Next.js 13 (App Router)
    - React 18
    - TypeScript
    - ESLint — 코드의 문제점을 찾고 고치기 위함
    - Prettier — 일관적인 코드 포맷 스타일
    - Husky — 커밋하기 전에 ESLint와 Prettier Scripts를 실행하기 위해서
    - Commitlint — 커밋 메세지가 규칙을 따르는지 확인하기 위함
    - Renovate — 의존성을 관리하고 최신 상태로 유지하기 위함
    - lint-staged — Staged 된 git files들에 대해 EsLint를 실행하고 Prettier를 실행
    - Path Mapping — 컴포넌트 혹은 이미지들을 Import 해올때 절대경로를 사용할 수 있음
- Tool: VSCode

Back-end

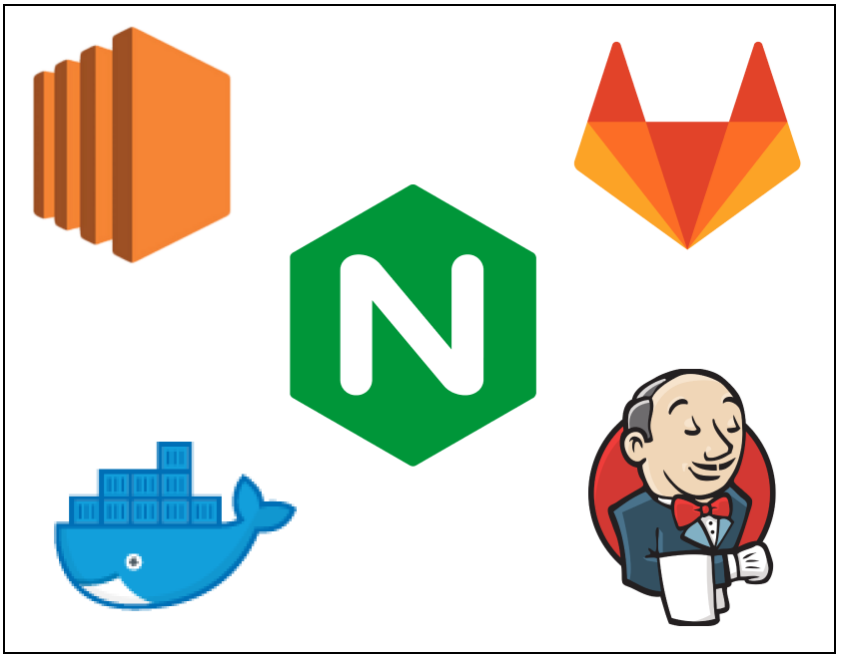


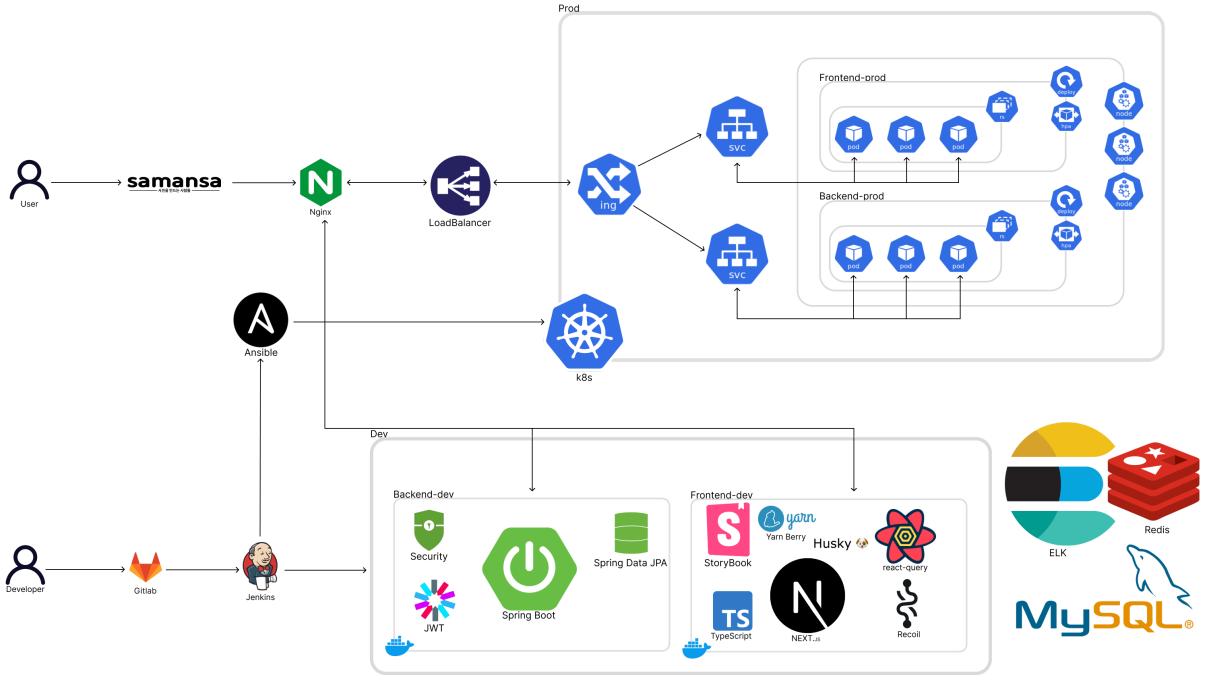
- Azul Zulu 17 (latest)
- Spring
    - Project Build: Gradle - Groovy
    - Language: Java
    - Spring Boot: 3.1.4
    - Packaging: Jar
    - Java: 17
    - Dependencies:
        - Spring Web
        - Spring Boot DevTools
        - Lombok
        - Spring Data JPA
        - Security
        - Oauth2
        - Google Cloud
        - gson
        - build.gradle

```
//common
implementation 'org.springframework.boot:spring-boot-starter-web'
compileOnly 'org.projectlombok:lombok'
developmentOnly 'org.springframework.boot:spring-boot-devtools'
annotationProcessor 'org.projectlombok:lombok'
testImplementation 'org.springframework.boot:spring-boot-starter-test'
implementation 'jakarta.xml.bind:jakarta.xml.bind-api:4.0.0'
annotationProcessor "org.springframework.boot:spring-boot-configuration-processor"

//Spring Data JPA
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

//Spring Security
implementation 'org.springframework.boot:spring-boot-starter-security'
testImplementation 'org.springframework.security:spring-security-test'

//Oauth2 Client
implementation 'org.springframework.boot:spring-boot-starter-oauth2-client'

//DB
implementation 'org.springframework.boot:spring-boot-starter-data-redis'
runtimeOnly 'com.mysql:mysql-connector-j'

//elasticsearch
implementation 'org.springframework.boot:spring-boot-starter-data-elasticsearch:3.1.1'
implementation 'co.elastic.clients:elasticsearch-java:8.7.1'
implementation 'com.fasterxml.jackson.core:jackson-databind:2.15.1'

//jwt
implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'

// Querydsl 추가
implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'
annotationProcessor "com.querydsl:querydsl-apt:5.0.0:jakarta"
annotationProcessor "jakarta.annotation:jakarta.annotation-api"
annotationProcessor "jakarta.persistence:jakarta.persistence-api"
```

- Tool: IntelliJ Ultimate (2023.1.3)

Infra Structure





- AWS EC2 Instance
- Jenkins
  - 2.422
- Ansible
  - 2.12.10
- kubernetes
  - 1.27.6
- Webhook
- Docker
  - 24.0.6
- Nginx
  - 1.18
- SSL
  - letsencrypt

DataBase

- ELK 8.7.1
  - ec2에 직접 설치
- Redis 7.2.1
  - Docker로 띄워서 사용
- MySQL 8.0.33
  - ec2에 직접 설치

Local Application Setting

Front-End

1. 빠른 실행
   - `yarn create next-app -e https://github.com/jpedroschmitz/typescript-nextjs-starter`
2. Yarn Berry 설정
   - yarn을 패키지 매니저로 사용하기 위해 `pnpm-lock.yaml` 을 지우고 `husky`, `ts.config`, `package.json` 에서 관련 설정을 모두 pnpm 에서 yarn으로 바꾼다.
3. Path Mapping

   기본적으로 파일이나 컴포넌트를 Import할 때는 @를 붙여 절대경로를 사용할 수 있다.

   ```
   import { Button } from '@/components/Button';

   // public foldelr에서 image나 다른 파일들을 import 해올 때
   import avatar from '@/public/avatar.png';
   ```

4. 폴더 구조

   ~~.husky - Husky 설정과 hooks들을 모아 놓음~~

   public - 정적인 파일들 (예 : robots.tsxt, 이미지, favicon 등등)

   src - Application의 source 코드들이 있음 (예 : pages, components, styles)

   vendor : Third-party api와 관련된 것을 넣으면 된다.

   ~~server-actions : server-actions(nextjs 13)~~

   hooks : custom-hook을 넣을 수 있음

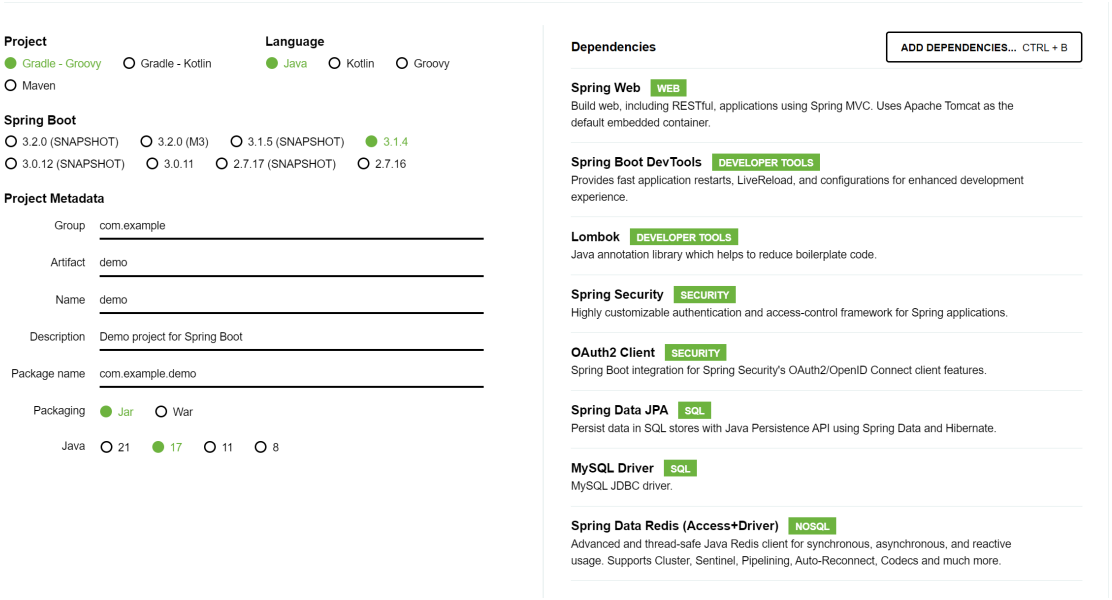
   config : 프로젝트 configuration 관련

   components : client-components

Back-End

1. Zulu 17 다운로드
   - Azul Zulu 17 msi 다운로드
   - msi는 환경변수 자동 세팅
2. IntelliJ Ultimate 다운로드
   - 학교 계정 연결하면 Ultimate 사용 가능
3. start.spring.io 에서 Spring 프로젝트 세팅
   - Gradle - Groovy
     - xml의 구조적인 틀을 벗어나 간결한 정의 가능
   - Spring Boot 3.1.4
     - 2.x 버전은 23년 11월 지원 종료로 인한 3.x 사용

- Java 17
  - 3.x 부터는 JDK 17부터 지원
- Jar
  - Spring Boot 안에 Tomcat을 내장하고 있어서, 코드만 패키징하는 Jar 형식 선택
- Dependencies
  - Spring Web
  - Spring Boot DevTools
  - Lombok
  - Security
  - Oauth2
  - Spring Data JPA
  - MysqlDB Driver
  - Redis
- Generate 클릭



4. 프로젝트를 열어 build.gradle이 의존 라이브러리를 가져옴
5. "File" -> "Settings"
   - "Plugins" 에서 Lombok Install 되어있는지 확인
   - "Editor" -> "File Encodings" -> Encoding 설정들 UTF-8로 변경 -> "Apply"
   - "Build, Execution, Deployment" -> "Build Tools" -> "Gradle" -> "Build and Run" 에서 Gradle로 되어있는 것 IntelliJ로 변경 -> "Apply"
   - "Build, Execution, Deployment" -> "Compiler" -> "Annotation Processors" -> "Enable annotation Processing" 체크 -> "Apply", "OK"
   1. build.gradle에 추가로 사용할 Dependency를 적용한다.

```
//jwt
implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'

// Querydsl 추가
implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'
annotationProcessor "com.querydsl:querydsl-apt:5.0.0:jakarta"
annotationProcessor "jakarta.annotation:jakarta.annotation-api"
```

```
annotationProcessor "jakarta.persistence:jakarta.persistence-api"

//elasticsearch
implementation 'org.springframework.boot:spring-boot-starter-data-elasticsearch:3.1.1'
implementation 'co.elastic.clients:elasticsearch-java:8.7.1'
implementation 'com.fasterxml.jackson.core:jackson-databind:2.15.1'
```

2. application.yml 파일을 세팅한다.

```
spring:

  # MySQL setting
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://[DB_HOST]:[DB_PORT]/[SCHEMA]?useUniCode=yes&characterEncoding=UTF-8
    username: [USERNAME]
    password: [PASSWORD]

  jpa:
    show-sql: true
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        format_sql: true
      open-in-view: true
  jwt:
    prefix: 'Bearer'
    secret: [SECRET_KEY]
    token:
      access-expiration-time: [ACCESS_EXPIRE_TIME]
      refresh-expiration-time: [REFRESH_EXPIRE_TIME]
  data:
    redis:
      host: [DB_HOST]
      port: [DB_PORT]
      password: [PASSWORD]

  elasticsearch:
    uris: [ELK_HOST]
    username: [ELK_USERNAME]
    password: [ELK_PASSWORD]
    index:
      word: [ELK_INDEX]

  security:
    oauth2:
      client:
        registration:
          google:
            clientId: [GOOGLE_CLIENT_ID]
            clientSecret: [GOOGLE_CLIENT_SECRET]
            scope:
              - email
              - profile
          naver:
            client-id: [NAVER_CLIENT_ID]
            client-secret: [NAVER_CLIENT_SECRET]
            client-authentication-method: client_secret_post
            authorization-grant-type: authorization_code
            redirect-uri: '{baseUrl}/{action}/oauth2/code/{registrationId}'
            client-name: Naver
            scope:
              - nickname
              - email
          kakao:
            client-id: [KAKAO_CLIENT_ID]
            client-secret: [KAKAO_CLIENT_SECRET]
            redirect-uri: '{baseUrl}/{action}/oauth2/code/{registrationId}'
            authorization-grant-type: authorization_code
            client-authentication-method: client_secret_post
            client-name: Kakao
            scope:
              - profile_nickname
              - account_email
        provider:
          naver:
            authorizationUri: https://nid.naver.com/oauth2.0/authorize
            tokenUri: https://nid.naver.com/oauth2.0/token
            userInfoUri: https://openapi.naver.com/v1/nid/me
            userNameAttribute: response
          kakao:
```

```
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id

  cloud:
    gcp:
      storage:
        bucket-name: [BUCKET_NAME]
        project-id: [PROJECT_ID]
        credentials:
          location: [JSON_NAME]
  file:
    negative-crawling: [TEXT_NAME]
server:
  port: [SERVER_PORT]
  servlet:
    context-path: [PREFIX]

logging:
  level:
    com.com.vegetable: debug
```

DataBase

MySQL

1. ubuntu 업데이트 및 설치

```
sudo apt-get update

sudo apt-get install mysql-server
```

2. 외부 포트 열기

```
sudo ufw allow mysql
```

3. mysql 실행 및 설정 변경

```
sudo systemctl start mysql

# ubuntu 서버가 재시작 되더라도 mysql이 자동 시작
$ sudo systemctl enable mysql
```

4. 외부 접속 허용

```
sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
# #bind-address = 127.0.0.1 의 부분을
# bind-address = 0.0.0.0 으로 수정
```

5. mysql 재시작

```
sudo service mysql restart
```

6. 서버 비밀번호 설정

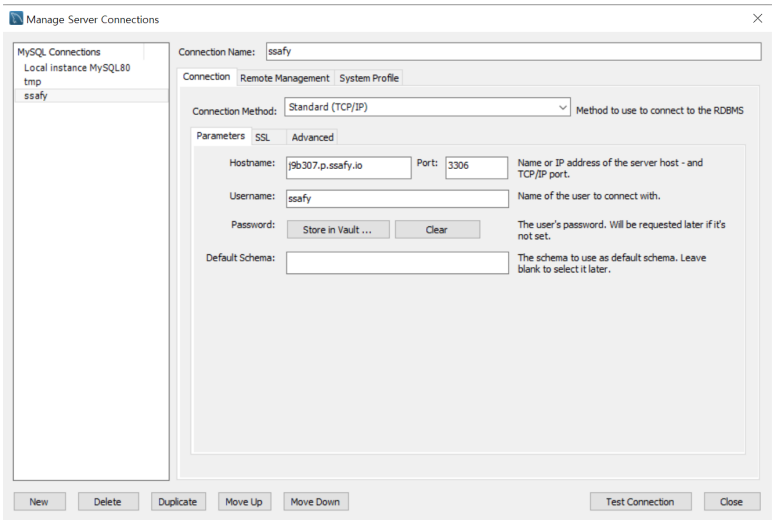
```
sudo /usr/bin/mysql -u root -p
# 암호 입력
```

7. 사용자 생성

```
# mysql 접속 후
CREATE USER '사용자명'@'%' IDENTIFIED BY '비밀번호';
```

```
GRANT ALL PRIVILEGES ON * . * TO '사용자 명'@'%' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

8. mysql workbench 연결 확인



Redis

1. Redis docker 이미지 띄우기

```
docker run -p 6379:6379 --name redis -d redis:latest --requirepass [PASSWORD]
```

2. 비밀번호 사용하여 접속

```
docker exec -i -t redis redis-cli -a [PASSWORD]
```

EC2 Setting

Docker

1. ca-certificates, curl, gnupg 설치

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
```

2. 도커의 공식 GPG 키 추가

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

3. repository 설정

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

4. 설치

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

Potainer

> Docker를 웹에서 관리 도와주는 툴

1. 업데이트

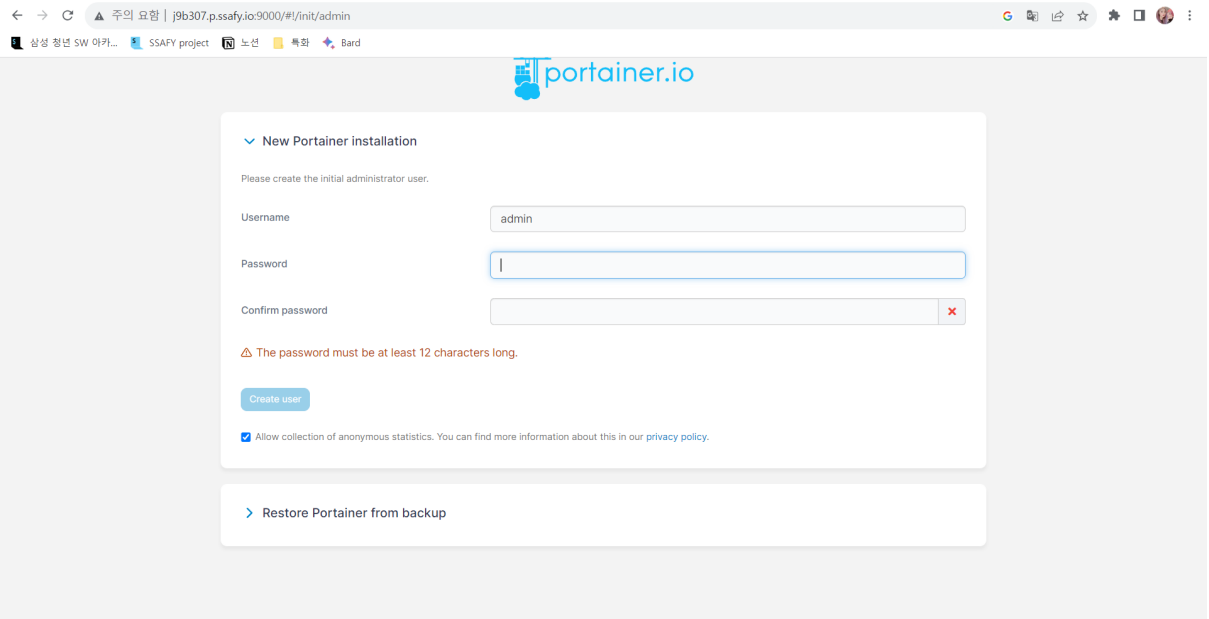
```
sudo apt update
```

2. portainer 설치

```
sudo docker run --name portainer -p 9000:9000 -d --restart always -v /data/portainer:/data -v /var/run/docker.sock:/var/run/docker.soc
```

3. 계정 생성

[서버 IP]:9000 접속



비밀번호 생성 ⇒ https://www.expressvpn.com/kr/password-generator

Jenkins

1. jdk17이 설치된 Jenkins 설치

```
$ docker run -d -p 8080:8080 -v /var/run/docker.sock:/var/run/docker.sock --name jenkins
  jenkins/jenkins:jdk17
```

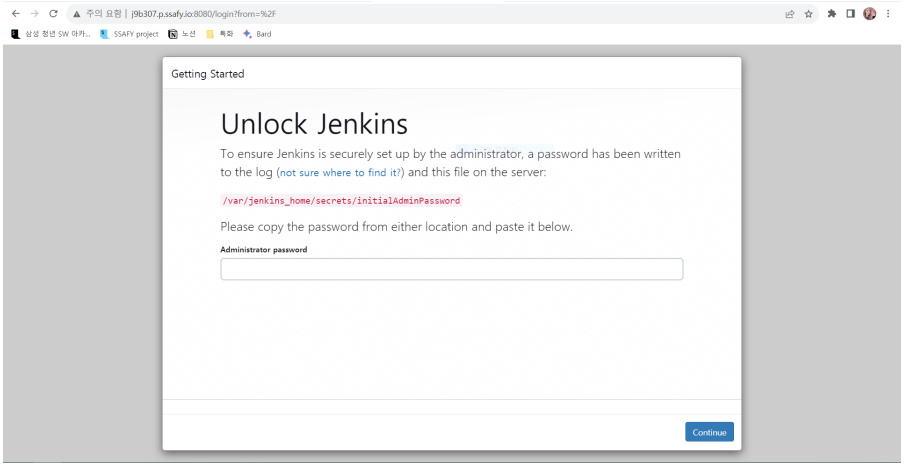
2. 초기 비밀번호 확인 (2가지 방법)

```
# jenkins 컨테이너에 접속해 초기 비밀번호 확인
$ docker exec -it jenkins bash
$ cat /var/jenkins_home/secrets/initialAdminPassword

# log 확인
$ sudo docker logs jenkins
```

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

01e51af85e99488aaba66657bafffb62

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
```

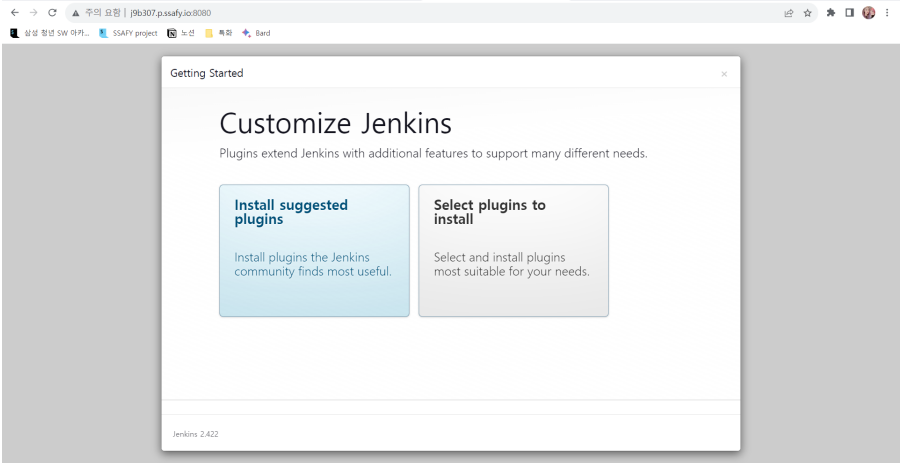
3. 세팅

   a. [서버 IP]:8080 접속



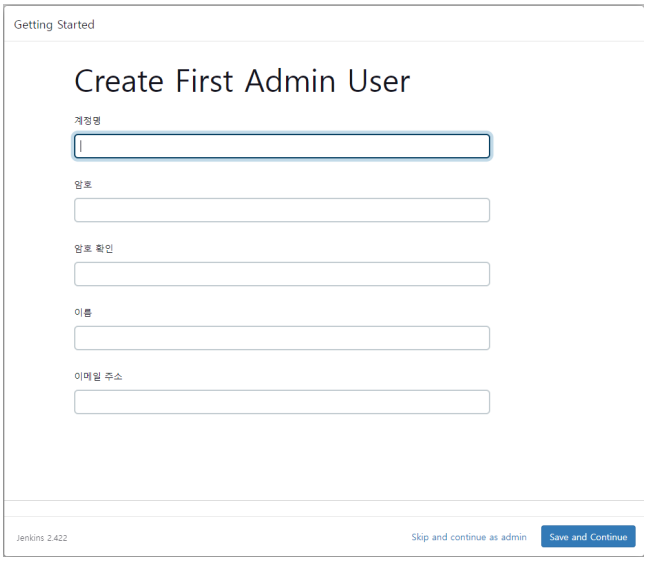
   b. 초기 비밀번호 입력

   c. Install suggested plugins 선택



   d. 계정 생성

   - 비밀번호 생성 ⇒ https://www.expressvpn.com/kr/password-generator



e. 시간 설정

- 사용자 - 설정 - User Defined Time Zone
  - Asia/Seoul

f. docker 명령어 설치

- https://velog.io/@chang626/docker-container에서-docker-image-빌드-진행-과정-jenkins-host-docker.sock을-연결 참고

```
# root 권한으로 jenkins 접속
sudo docker exec -it -u root jenkins bash

# 공식 docker apt repository 구성 및 docker ce 바이너리 설치
apt-get update && \
apt-get -y install apt-transport-https \
     ca-certificates \
     curl \
     gnupg2 \
     software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
   $(lsb_release -cs) \
   stable" && \
apt-get update && \
apt-get -y install docker-ce

# docker jenkins에서 host docker 접근권한을 부여
service docker start
groupadd -f docker
usermod -aG docker jenkins
chown root:docker /var/run/docker.sock
```

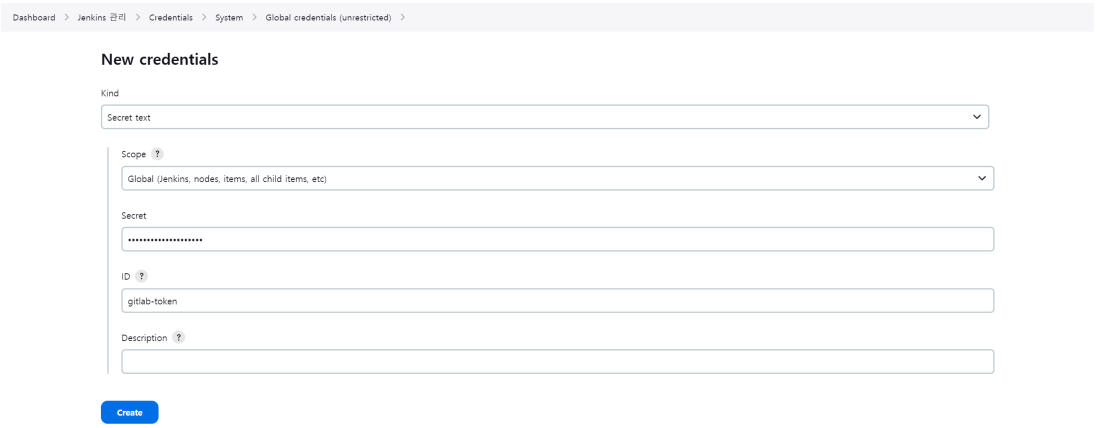
g. Plugins

- Gitlab
- Publish Over SSH
- NodeJS
- Docker
- Docker Pipeline

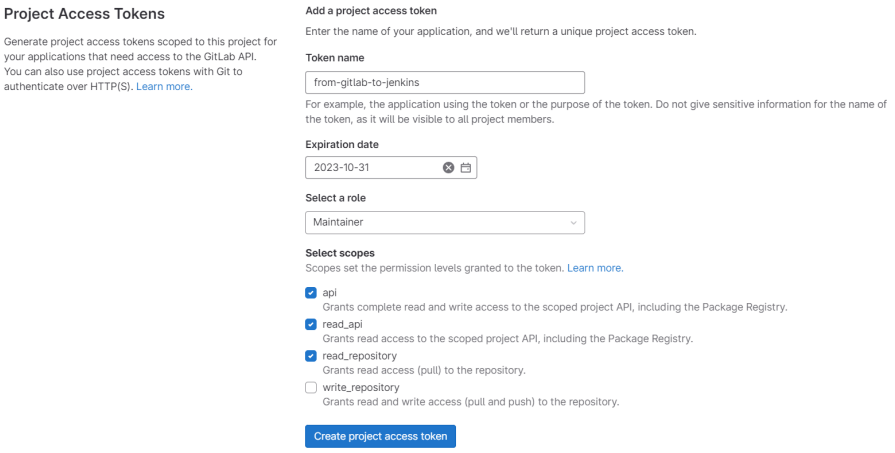
h. Credentials

- Jenkins 관리 - Credentials - Stores scoped to Jenkins(global)

- gitlab 액세스 토큰



  - kind
    - Secret-text
  - secret
    - gitlab에서 생성한 토큰
      - 프로젝트 - Settings - Access Tokens



  - ID
    - gitlab-token
    - dockerhub-token

i. System

- Dashboard - Jenkins 관리- System
  - GitLab
    - Connection name

            - gitlab-connection
        - GitLab host URL
            - https://lab.ssafy.com
        - Credentials
            - Credentials에서 생성한 `gitlab-token`
    - SSH Server (ec2 서버)
        - Name
            - ubuntu-server
        - Hostname
            - `hostname -i` 로 확인
        - Username
            - ubuntu
        - 고급
            - Use password authentication, or use a different key 체크
            - Path to key
                - /var/jenkins_home/.ssh/id_rsa
                - 우분투 서버에 젠킨스 퍼블릭 키 추가

j. Tools

- JDK installations

```
# jenkins 컨테이너 접속
$ sudo docker exec -it jenkins bash

# 환경 변수 확인
$ env
```

    - Name
        - Java17
    - JAVA_HOME
        - /opt/java/openjdk
- Gradle

> 프로젝트에서 쓰는 gradle버전과 같게 설정

    - name
        - Gradle8.2.1
    - version (Install automatically)
        - Gradle 8.2.1 선택
- NodeJS installations

> 프로젝트에서 쓰는 nodejs 버전과 같게 설정

    - Global npm packages to install
        - yarn

k. Item

frontend-develop

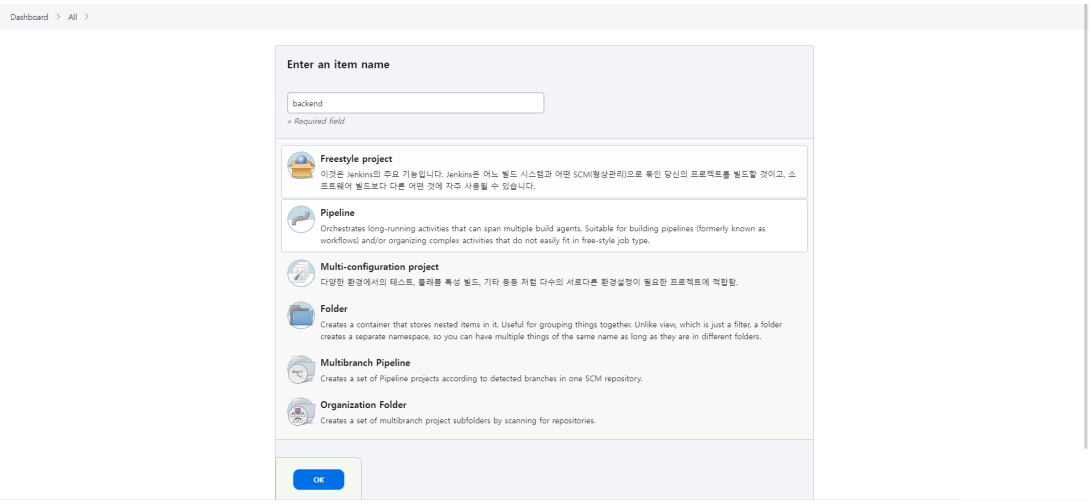
backend-develop

samansa

samansa-backend

samansa-frontend

- 아이템 추가



- Build Triggers - Build when a change is pushed to GitLab. GitLab webhook URL: http://호스트:8080/project/backend 체크
  - 고급의 secret token 생성
- gitlab 프로젝트 - Settings - Webhooks에 입력

Nginx Setting

> Nginx 설치 및 SSL 적용

1. nginx 설치

```
# 설치
sudo apt-get install nginx

# 설치 확인 및 버전 확인
nginx -v
```

2. letsencrypt 설치를 위해 다음과 같은 순서로 명령어를 입력

```
sudo apt-get install letsencrypt

sudo systemctl stop nginx

sudo letsencrypt certonly --standalone -d [도메인]
sudo letsencrypt certonly --standalone -d [서브 도메인]
```

   - 서브 도메인 확장

```
sudo letsencrypt certonly --standalone -d samansa.kr -d *.samansa.kr --expand
```

```
letsencrypt certonly --manual --preferred-challenges dns -d samansa.kr -d *.samansa.kr
```

3. conf 작성

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

sudo vi proxy-setting.conf
```

- proxy-setting.conf

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    server_name samansa.kr;

    # HTTP에서 HTTPS로 리다이렉션 설정
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;

    server_name samansa.kr;

    location /api {
#        proxy_pass http://localhost:5000/api;
        proxy_pass http://158.247.242.117/back/api;
#        proxy_redirect http://localhost:5000/ https://samansa.kr/;
        proxy_redirect http://localhost https://samansa.kr/;
        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;
    }

    location / {
#        proxy_pass http://localhost:3000;
        proxy_pass http://158.247.242.117/front$request_uri;
#        proxy_redirect http://localhost:3000/ https://samansa.kr/;
        proxy_redirect http://localhost https://samansa.kr/;
        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;
    }

    ssl_certificate /etc/letsencrypt/live/samansa.kr/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/samansa.kr/privkey.pem; # managed by Certbot
    # include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    # ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

# sub domain

server {
    listen 80;
        listen [::]:80;
    server_name test.samansa.kr;


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

server {
    listen 443 ssl;
    listen [::]:443 ssl;

    server_name test.samansa.kr;

#    root /var/www/html;

#    error_page 404 /404.html;

    location / {
```

```
        proxy_pass http://localhost:3000;
        proxy_redirect http://localhost https://test.samansa.kr;
        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;
    }
    location /api {
        proxy_pass http://localhost:5000/api;
        proxy_redirect http://localhost https://test.samansa.kr;
        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;
    }


#    location / {
#       internal;
#        return 404; # 여기에 404 대신 실제 리소스 위치를 설정하십시오.
#    }

    ssl_certificate /etc/letsencrypt/live/test.samansa.kr/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/test.samansa.kr/privkey.pem; # managed by Certbot
    # include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    # ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}
```

4. 기존 nginx port 변경

```
cd /etc/nginx/sites-enabled

vi default

# 아래와 같이 변경
```

```
server {
        listen 180 default_server;
        listen [::]:180 default_server;

        # SSL configuration
        #
        # listen 443 ssl default_server;
        # listen [::]:443 ssl default_server
```

```
sudo ln -s /etc/nginx/sites-available/proxy-setting.conf /etc/nginx/sites-enabled/proxy-setting
```

- nginx 테스트

```
sudo nginx -t
```

```
ubuntu@ip-172-26-4-119:/etc/nginx/sites-enabled$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

- 재시작

```
sudo systemctl restart nginx
```

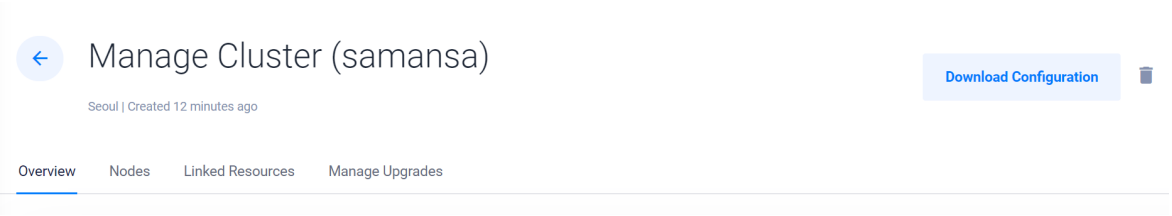
Kubernetes Setting

1. k8s 생성

1. Vultr.com 접속 후 가입하면 250 크레딧 받을 수 있음
2. Add Kubernetes
   - 클러스터 이름 입력
   - **Kubernetes Version**
     - k8s 버전 선택
     - 1.27.6+1
   - **Cluster Location**
     - 서울
   - **Cluster Capacity**
     - Label (required)
       - 노드 라벨 명 입력
     - Number of Nodes
       - 노드 기본 3개
     - 사양은 본인 원하는 것

2. k8s 접속

1. k8s 정보가 담긴 파일 다운로드



2. kubectl (1.27.6 ver) 설치 (선택)

> 서버에 kubectl 명령어가 존재하지 않는 경우
>
> - root으로 접속했음.

```
# kubectl 다운로드
curl -LO https://dl.k8s.io/release/v1.27.6/bin/linux/amd64/kubectl

# 권한 설정
chmod +x ./kubectl

# root가 아닌 일반 사용자로 접속한 경우
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH

# kubectl 명령어 확인
kubectl version --short --client
```

3. 클러스터 접속 위한 서버 실행

> 다른 플랫폼과 달리 Configuration 파일로 클러스터에 접근 가능.
> 모든 OS에서 접속 가능하지만 우분투에서 실행하였음.

```
# kubectl이 저장될 폴더 생성
mkdir ~/.kube

# ~/.kube 폴더에 클러스터에 접속하기 위한 설정 파일 저장
cd ~/.kube
vi k8s-config.yaml
# 이전에 다운받은 Configuration 내용 복붙

# KUBECONFIG export
export KUBECONFIG=$KUBECONFIG:$HOME/.kube/config:$HOME/.kube/k8s-config.yaml

# config 스위칭
# vultr에서 받은 config 파일 명
kubectl config use-context vke-7bbf5023-f14b-4b2a-80e3-b80eae85d16e
```

3. 외부 노출

1. Ingress-nginx 설치

> 하나의 로드밸런서로 클러스터 내부의 서비스로 포워딩해주기위함

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.7.0/deploy/static/provider/aws/deploy.ya

# ingress-nginx가 떠 있음 확인
kubectl get ns
```

4. k8s 접속

1. Ansible 설치

```
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible

# ansible 버전 확인
ansible --version

# 파이썬 path에 등록
# 설치된 파이썬 위치 확인
which python
sudo apt update
sudo apt install python-pip
pip install docker
```

2. 접속 테스트

- `deploy-k8s.yml`
  - 서버에 저장된 kube-config로 접속 가능 확인
- `ansible-playbook deploy-k8s.yml -v`로 확인

```
- hosts: localhost
  connection: local
  tasks:

  - name: switch config
    command: kubectl config use-context vke-7bbf5023-f14b-4b2a-80e3-b80eae85d16e
    environment:
      KUBECONFIG: "{{ lookup('env', 'HOME') }}/.kube/config:{{ lookup('env', 'HOME') }}/.kube/k8s-config.yaml"
```

5. 실행

브랜치

- samansa-prod
  - merge backend-develop

- merge frontend-develop

CI

Jenkins

samansa pipepline

- git clone samansa-prod.git
- secret 파일 복사

▼ pipeline

```
pipeline {
    agent any

    environment{
        backendContainerName = "API-Server"
        frontendContainerName = "React-Server"
        backendImageName = "h4r1b0/samansa-api"
        frontendImageName = "h4r1b0/samansa-react"
    }

    stages {
        stage('Delete Directory') {
            steps {
                script {
                    // 폴더가 있는 경우만 삭제
                    deleteDir(dir: '/var/jenkins_home/workspace/samansa/S09P31B207')
                }
            }
        }
        stage('Checkout Samansa Prod') {
            steps {
                withCredentials([string(credentialsId: 'gitlab-token', variable: 'ACCESS_TOKEN'), file(credentialsId: 'applicat
ion-yml', variable: 'APPLICATION_YML'), file(credentialsId: 'env', variable: 'ENV_DEVELOPMENT')]) {
          // GitLab 레포지토리를 클론하는 단계
          sh '''
            # checkout
            git clone -b samansa-prod https://gitlab-ci-token:${ACCESS_TOKEN}@lab.ssafy.com/s09-final/S09P31B207.git

            # cp backend secret file
            mkdir S09P31B207/backend/memetionary/src/main/resources || true
            cp ${APPLICATION_YML} S09P31B207/backend/memetionary/src/main/resources/application.yml

            # cp frontend secret file
            cp ${ENV_DEVELOPMENT} S09P31B207/frontend/.env

            pwd
            ls
          '''
                }
            }
        }
    }
}
```

samansa-backend

- samansa의 hook으로 발생

▼ pipeline

```
pipeline {
    agent any

    environment{
        backendImageName = "h4r1b0/samansa-api"
    }

    stages {
        stage('Delete Directory') {
            steps {
                script {
```

```
                    // 폴더가 있는 경우만 삭제
                    deleteDir(dir: '/var/jenkins_home/workspace/samansa-backend')
                }
            }
        }
        stage('Build API Image'){
            tools {
                gradle 'Gradle8.3'
            }
            steps {
                sh '''
                    cd /var/jenkins_home/workspace/samansa/S09P31B207/backend/memetionary
                    pwd
          chmod +x gradlew
          ./gradlew build
                    sudo docker build -t ${backendImageName} .
                '''
            }
        }
        stage('Push Backend Image To DockerHub'){
            steps {
                script {
                    // 도커 허브 인증 정보
                    docker.withRegistry('https://registry.hub.docker.com', 'docker-hub') {
                        // 이미지 태그
                        def customImage = docker.image(backendImageName)
                        // 이미지 푸시
                        customImage.push()
                    }
                }
            }
        }
    }
}
```

samansa-frontend

- samansa의 hook으로 발생

▼ pipeline

```
pipeline {
    agent any

    environment{
        frontendImageName = "h4r1b0/samansa-react"
    }

    stages {
        stage('Delete Directory') {
            steps {
                script {
                    // 폴더가 있는 경우만 삭제
                    deleteDir(dir: '/var/jenkins_home/workspace/samansa-frontend')
                }
            }
        }
    stage('Build React Image'){
            tools {
                nodejs 'NodeJS'
            }
            steps {
        script{
          try{
            // yarn build
            sh '''
              pwd
              ls
              cd /var/jenkins_home/workspace/samansa/S09P31B207/frontend
              yarn set version 3.x
              yarn cache clean
              yarn install
              yarn build
              sudo docker build -t ${frontendImageName} .
            '''
          } catch (e) {
            // 실패 사유를 변수에 할당
                        env.FAILURE_REASON = e
                        throw e
          }
        }
```

```
            }
        }
    stage('Push Frontend Image To DockerHub'){
            steps {
                script {
                    // 도커 허브 인증 정보
                    docker.withRegistry('https://registry.hub.docker.com', 'docker-hub') {
                        // 이미지 태그
                        def customImage = docker.image(frontendImageName)
                        // 이미지 푸시
                        customImage.push()
                    }
                }
            }
        }
    }
}
```

CD

Ansible

- yml으로 k8s를 관리

설치

```
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible

# ansible 버전 확인
ansible --version

# 파이썬 path에 등록
# 설치된 파이썬 위치 확인
which python
sudo apt update
sudo apt install python-pip
pip install docker
```

k8s

ansible

- 여러 명령어를 통해 한번에 관리

```
- hosts: localhost
  connection: local
  tasks:

  - name: switch config
    command: kubectl config use-context vke-7bbf5023-f14b-4b2a-80e3-b80eae85d16e
    environment:
      KUBECONFIG: "/home/ubuntu/.kube/config:/home/ubuntu/.kube/k8s-config.yaml"

  - name: check kubectl
    command: kubectl get no -o wide
    environment:
      KUBECONFIG: "/home/ubuntu/.kube/config:/home/ubuntu/.kube/k8s-config.yaml"

  - name: apply deployment
    command: kubectl apply -f /home/ubuntu/ansible/samansa-deployment.yml
    environment:
      KUBECONFIG: "/home/ubuntu/.kube/config:/home/ubuntu/.kube/k8s-config.yaml"

  - name: apply service
    command: kubectl apply -f /home/ubuntu/ansible/samansa-service.yml
    environment:
      KUBECONFIG: "/home/ubuntu/.kube/config:/home/ubuntu/.kube/k8s-config.yaml"

  - name: apply ingress
    command: kubectl apply -f /home/ubuntu/ansible/samansa-ingress.yml
```

```
    environment:
      KUBECONFIG: "/home/ubuntu/.kube/config:/home/ubuntu/.kube/k8s-config.yaml"
```

Deployment

- samansa-deployment.yml
- `kubectl apply -f samansa-deployment.yml` 로 실행
- `kubectl get deploy -o wide` 로 결과 확인

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: samansa-api-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: samansa-api
  template:
    metadata:
      labels:
        app: samansa-api
    spec:
      containers:
      - name: api-server
        image: h4r1b0/samansa-api:1.3
        ports:
        - containerPort: 5000
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: samansa-react-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: samansa-react
  template:
    metadata:
      labels:
        app: samansa-react
    spec:
      containers:
      - name: react-server
        image: h4r1b0/samansa-react:1.21
        ports:
        - containerPort: 3000
        resources:
          limits:
            cpu: 1024m
            memory: 2048Mi
          requests:
            cpu: 512m
            memory: 1024Mi
```

Service

- samansa-service.yml
- `kubectl apply -f samansa-service.yml` 로 실행
- `kubectl get svc` 로 결과 확인

```
apiVersion: v1
kind: Service
metadata:
  name: samansa-api-service
  annotations:
    service.beta.kubernetes.io/vultr-loadbalancer-proxy-protocol: 'true'
spec:
  selector:
    app: samansa-api
  ports:
  - protocol: TCP
    port: 80
```

```
    targetPort: 5000
  type: ClusterIP
---
apiVersion: v1
kind: Service
metadata:
  name: samansa-react-service
  annotations:
    service.beta.kubernetes.io/vultr-loadbalancer-proxy-protocol: 'true'
spec:
  selector:
    app: samansa-react
  ports:
  - protocol: TCP
    port: 80
    targetPort: 3000
  type: ClusterIP
```

Ingress

- samansa-ingress.yml
- `kubectl apply -f samansa-ingress.yml` 로 실행
- `kubectl get ingress` 로 결과 확인

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: samansa-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/configuration-snippet: proxy_set_header X-Forwarded-For "$proxy_add_x_forwarded_for";
spec:
  ingressClassName: "nginx"
  rules:
  - http:
      paths:
      - path: /back(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: samansa-api-service
            port:
              number: 80
      - path: /front(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: samansa-react-service
            port:
              number: 80
```

hpa

- samansa-hpa.yml

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: samansa-react-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: samansa-react-deployment
  minReplicas: 3
  maxReplicas: 6
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```