



포팅 매뉴얼

⚙️ SSAFY 8th 자율프로젝트_B105 FOREST 포팅 매뉴얼 ⚙️

목차

- ✓ 기술 스택 버전
- ✓ CI/CD
- ✓ 아키텍처
- ✓ ERD
- ✓ Java
- ✓ Python
- ✓ Nginx
- ✓ Docker
- ✓ MySQL
- ✓ MongoDB
- ✓ Jenkins
- ✓ k8s
- ✓ 방화벽 설정
- ✓ React 배포
- ✓ SpringBoot 배포
- ✓ Flask 배포
- ✓ 외부 문서
 - S3
 - Google Cloud Vision
 - Jitpack

기술 스택 버전



프론트엔드

- Next.js 13.3.0
- React 18.2.0
- React-query 3.39.3
- Redux 4.2.1
- Styled-components 5.3.9
- TypeScript 5.0.4



백엔드

- openJDK 11.0.18

- Python 3.11.1
- SpringBoot 2.7.10
- Spring Swagger: 2.9.2
- queryDSL: 5.0.0
- Flask 2.3.2
- Werkzeug 2.3.3

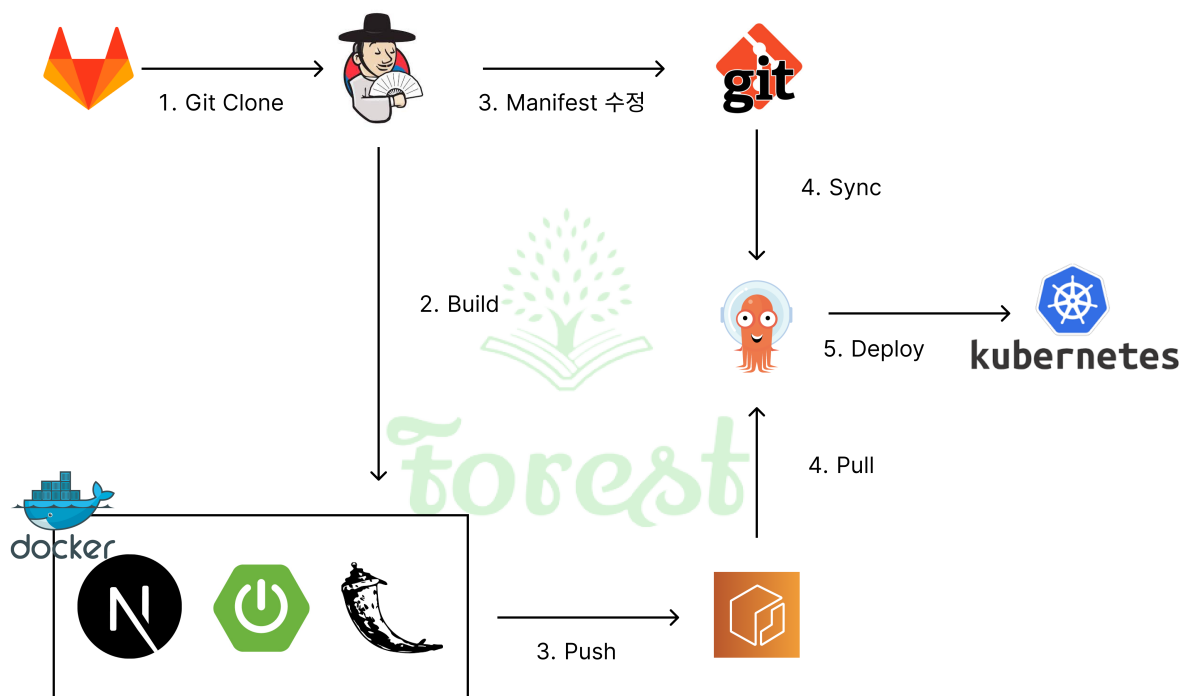
DB

- MySQL: 8.0.31
- Mongodb : 4.4

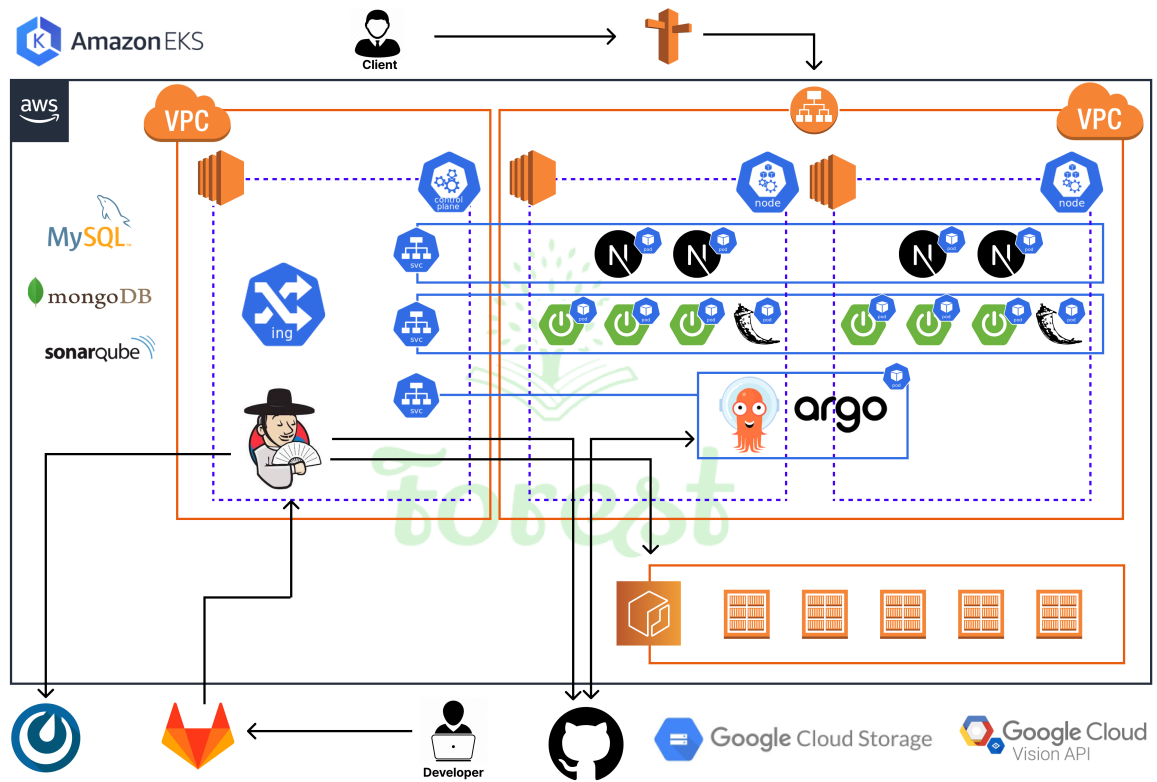
인프라

- Ubuntu 20.04
- docker 23.0.1
- docker-compose 1.29.2
- Jenkins 2.387.1
- AWS EC2
- AWS EKS
- AWS ECR
- AWS ROUTE53
- ArgoCD

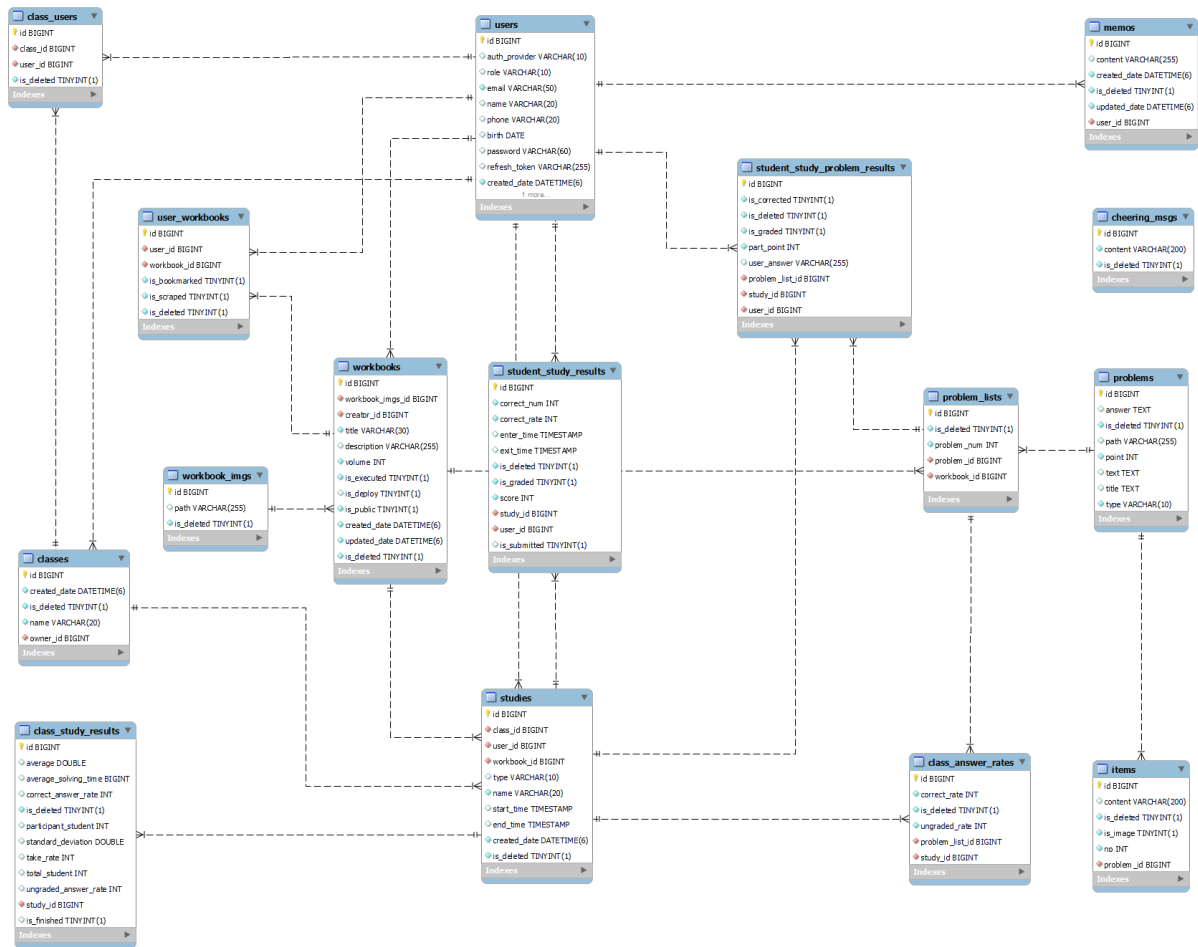
CI/CD



아키텍처



ERD



Java 설치

```

sudo apt-get update && sudo apt-get upgrade
sudo apt-get install openjdk-11-jdk
java -version
  
```

Python 설치

```

sudo apt-get update -y
sudo apt-get install python3 pip3 -y
python3 --version
pip3 --version
  
```

Nginx 설치

- Nginx 설치 및 버전 확인

```

apt-get install nginx
nginx -v
  
```

- Nginx 중지

```
systemctl stop nginx
```

- SSL 인증서 발급

```
apt-get install letsencrypt # Let's Encrypt 설치

sudo letsencrypt certonly --standalone -d [도메인] # 인증서 적용 및 pem키 발급

cd /etc/letsencrypt/live/[도메인] # 발급 경로 확인

vim custom.conf # conf 파일 생성
```



```
:/etc/letsencrypt/live/[redacted].p.ssafy.io# ls
README  cert.pem  chain.pem  fullchain.pem  keystore.p12  privkey.pem
```

- .conf 파일 작성

```
server {
    # 프론트 연결(포트 번호는 본인의 프론트 포트번호를 입력)
    location /{
        proxy_pass http://localhost:3000;
        add_header 'Access-Control-Allow-Origin' '*';
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
        add_header 'Access-Control-Allow-Headers' 'DNT, User-Agent, X-Requested-With, If-Modified-Since, Cache-Control, Content-Type';
        add_header 'Access-Control-Expose-Headers' 'Content-Length, Content-Range';

        # https websocket
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
    }

    location /api{
        proxy_pass https://localhost:9010;
    }

    location /workbook{
        proxy_pass https://localhost:9011;
    }

    location /study{
        proxy_pass https://localhost:9012;
    }

    listen 443 ssl http2; # managed by Certbot
    # 도메인 이름을 써줘야함
    ssl_certificate /etc/letsencrypt/live/k8b105.p.ssafy.io/fullchain.pem; # managed by Certbot
    # 도메인 이름을 써줘야함
    ssl_certificate_key /etc/letsencrypt/live/k8b105.p.ssafy.io/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
}

server {
    # 도메인 이름을 입력
    if ($host = k8b105.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    server_name {도메인 주소};
    return 404; # managed by Certbot
}
```

- 심볼릭 연결 테스트 및 상태 확인

```
# 심볼릭 링크 연결
sudo ln -s /etc/nginx/sites-available/[파일명].conf /etc/nginx/sites-enabled/[파일명].conf

sudo nginx -t # nginx 테스트

sudo systemctl restart nginx # nginx 재시작
```

```
sudo systemctl status nginx # nginx 상태 확인
```

Docker 설치

- apt-transport-https : 패키지 관리자가 https를 통해 데이터 및 패키지에 접근할 수 있도록 한다.
- ca-certificates : certificate authority에서 발행하는 디지털 서명. SSL 인증서의 PEM 파일이 포함되어 있어 SSL기반 앱이 SSL 연결이 되어 있는지를 확인할 수 있다.
- curl : 특정 웹사이트에서 데이터를 다운로드 받을 때 사용
software-properties-common : PPA(Personal Package Archive)를 추가하거나 제거할 때 사용한다

```
sudo apt update && sudo apt-get upgrade
sudo apt install apt-transport-https ca-certificates
sudo apt install curl gnupg-agent software-properties-common
```

- Docker 공식 GPG 키 추가

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
```

- stable repository를 세팅하기 위한 명령어 실행
- add-apt-repository: PPA저장소를 추가해준다. apt 리스트에 패키지를 다운로드 받을 수 있는 경로가 추가됨

```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
```

- 가장 최신 버전의 Docker 엔진을 설치한 후, 버전 확인

```
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io
docker -v
```

- docker-compose 설치

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

MySQL 설치

- MySQL Docker Image 다운로드, 태그 버전 생략하면 최신 버전 다운로드

```
docker pull mysql:8.0.31
```

- MySQL Docker 컨테이너 생성 및 실행

```
docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=B209HERE -v mysql-volume:/var/lib/mysql -d -p 3306:3306 mysql:8.0.31
```

- MySQL docker 컨테이너 접속

```
docker exec -it mysql-container bash

mysql -u root -p
Enter password: B209HERE
```

MongoDB 설치

- 설치

```
docker pull mongo:4.4

docker run -i -t --name MongoDB -p 27017:27017 -e MONGO_INITDB_ROOT_USERNAME=root -e MONGO_INITDB_ROOT_PASSWORD=B105FOREST -v /my/mong
```

- 접속

```
docker exec -it -u root MongoDB /bin/bash

mongo -u {name} -p ${password}
```

- MongoDB Compass 접속

```
mongodb://{name}:{pw}@k8b105.p.ssafy.io:27017/?authMechanism=DEFAULT
```

Jenkins 설치

- 폴더 생성

```
sudo mkdir -p /home/ubuntu/jenkins
```

- Docker에 Jenkins 설치

```
sudo docker run --name jenkins -d -p 9999:9999 -p 50000:50000 -v /home/ubuntu/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/r
```

- jenkins container 접속

```
sudo docker exec -it jenkins /bin/bash
```

- jenkins 접속

```
http://<your-aws-domain>:<jenkins port> 접속 후 admin password 입력
ex) http://j8b209.p.ssafy.io:9999
```

- jenkins admin password 확인

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

- jenkins 내에 docker 설치

```
# Old Version Remove
apt-get remove docker docker-engine docker.io containerd runc

## Setup Repo
apt-get update

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

mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/debian/gpg | 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/debian \
    $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

## - Install Docker Engine
apt-get update

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

Kubernetes 설정

- Bastion Host에 AWS CLI 관리툴인 aws 설치

```
sudo apt-get install -y unzip
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
aws --version
```

- Bastion Host에 k8s 관리툴인 kubectl 설치

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/kubectl
chmod +x ./kubectl
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
kubectl version --short --client
```

- AWS IAM 생성하기

```
1. root 로그인후 IAM 생성
2. 사용자이름(User name*) : <user-name>
3. 프로그래밍 방식(Programmatic access) 선택
4. 기존 정책(Attach existing policies directly) : AdministratorAccess
5. 태그 추가(Add tags (optional)) - <SKIP>
```

- Bastion Host에서 aws 관리할 수 있도록 aws계정 등록

```
aws configure
AWS Access Key ID [None]: AKIASJ...E37V
AWS Secret Access Key [None]: XLzhAqt...7g
Default region name [None]: ap-northeast-2
Default output format [None]: <ENTER>
```

- EKS 구성

```
eksctl create cluster \
  --name <demo-name> \
  --region ap-northeast-2 \
  --with-oidc \
```



```
--ssh-access \  
--ssh-public-key <aws--login-key> \  
--nodes 3 \  
--node-type t3.medium \  
--node-volume-size=20 \  
--managed
```

- 설치 결과 확인 및 CLI 명령어 자동 완성 기능 추가

```
kubectl get nodes  
  
source <(kubectl completion bash)  
echo "source <(kubectl completion bash)" >> ~/.bashrc
```

- 워커 노드 정보 보기

```
kubectl get nodes -o wide
```

- EKS 삭제

```
eksctl delete cluster --name k8s-demo
```

방화벽 설정

- ufw 상태 확인

```
sudo ufw status verbose
```

- ufw 활성화/비활성

```
sudo ufw enable # 활성화  
sudo ufw disable # 비활성화
```

- ufw 기본 룰 확인

```
sudo ufw show raw
```

- ufw 포트 허용/거부

```
# 포트  
sudo ufw allow 8080  
sudo ufw allow 8080/tcp  
sudo ufw allow 8080/udp  
  
sudo ufw deny 8080  
sudo ufw deny 8080/tcp  
sudo ufw deny 8080/udp  
  
# 서비스 이름  
sudo ufw allow ssh  
sudo ufw deny ssh
```

- 특정 IP 방화벽 허용

```
sudo ufw allow from ${IP_ADDRESS}
```

- 특정 IP 주소와 프로토콜, 포트 허용

```
sudo ufw allow from ${IP_ADDRESS} to any port 22
```

- Allowed Ports

- 22, SSH
- 80, HTTP
- 443, HTTPS
- 3000, React, Nginx
- 3306, MySQL
- 9010, auth
- 9011, workbook
- 9012, study
- 5000, flask
- 9999, jenkins
- 27017, mongodb
- 9000, sonarQube

React 배포

- Dockerfile

```
FROM node:16-alpine AS base

# 만약 컨테이너 안의 이미지의 경로가 /app 이런식으로 되어있다면 작업할 div 경로를 설정할 수도 있다.
# 설정해주면 COPY 의 두번째 경로를 ./ 이것으로 했을 때 자동으로 /app 경로가 된다.
WORKDIR /app

# package.json 파일을 복사한다. 만약 다시 빌드할 때 변경사항이 없을 경우 npm install까지 그냥 넘어간다.
COPY package.json /app

# 이미지를 받으면 npm install을 자동으로 해줌
RUN npm install

# 어떤 파일이 이미지에 들어가야 하는지
# 첫 번째 .은 이 프로젝트의 모든 폴더 및 파일들 (Dockerfile을 제외한)
# 두 번째 .은 파일을 저장할 컨테이너 내부 경로 (ex /app)
COPY . /app

RUN npm run build
EXPOSE 3000

ENV PORT 3000

CMD ["npm", "run", "start"]
```

- Jenkinsfile

```

def previousBuildNumber = currentBuild.previousBuild?.number

pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout 1 hour
    }
    environment {
        DOCKER = 'sudo docker'
        TIME_ZONE = 'Asia/Seoul'
        PROFILE = 'local'
        REGION = 'ap-northeast-2'
    }
    stages{
        stage('Clone Repository') {
            steps {
                checkout scm
                echo 'Checkout Scm'
            }
        }

        stage('env setting') {
            steps{
                sh 'cp /var/jenkins_home/env/.env /var/jenkins_home/workspace/prod-front/forest-front'
            }
        }

        stage('dockerizing project by dockerfile') {
            steps {
                dir('forest-front') {
                    sh '''
                    ls -al
                    npm -v
                    node -v
                    docker build -t ${IMAGE_NAME_FRONT}:${BUILD_NUMBER} .
                    docker tag ${IMAGE_NAME_FRONT}:${BUILD_NUMBER} ${IMAGE_NAME_FRONT}:latest
                    '''
                }
            }
            post {
                success {
                    echo 'success dockerizing project'
                }
                failure {
                    error 'fail dockerizing project' // exit pipeline
                }
            }
        }
    }
    stage('upload aws ECR') {
        steps {
            script{
                // cleanup current user docker credentials
                sh 'rm -f ~/.dockercfg ~/.docker/config.json || true'

                sh 'echo ${ECR_PATH_FRONT}'
                sh 'echo ${REGION}'
                sh 'echo ${AWS_CREDENTIAL_NAME}'

                docker.withRegistry("https://${ECR_PATH_FRONT}", "ecr:${REGION}:${AWS_CREDENTIAL_NAME}") {
                    docker.image("${IMAGE_NAME_FRONT}:${BUILD_NUMBER}").push()
                    docker.image("${IMAGE_NAME_FRONT}:latest").push()

                    sleep 10
                }
            }
        }
        post {
            success {
                echo 'success upload image'
                sh 'docker rmi ${IMAGE_NAME_FRONT}:${BUILD_NUMBER}'
                sh 'docker rmi ${IMAGE_NAME_FRONT}:latest'
            }
            failure {
                error 'fail upload image' // exit pipeline
                sh 'docker rmi ${IMAGE_NAME_FRONT}:${BUILD_NUMBER}'
                sh 'docker rmi ${IMAGE_NAME_FRONT}:latest'
            }
        }
    }
}

stage('Manifest Update') {
    steps {

```

```
git credentialsId: '${GIT_CREDENTIALS_ID}',  
    url: '${GIT_URL}',  
    branch: 'main'  
  
dir('front') {  
    sh """  
        ls -al  
        sed -i 's|image: .*|image: "${IMAGE_NAME_FRONT}:${currentBuild.number}"|' ${EKS_DEPLOY_REACT_YML}  
        git status  
        git add ${EKS_DEPLOY_REACT_YML}  
        git commit -m '[UPDATE] forest-front ${currentBuild.number} image versioning'  
        git push origin main  
    """  
}  
}  
}
```

Spring Boot 배포

- Dockerfile

```
FROM adoptopenjdk/openjdk11
COPY build/libs/here-auth-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=prod", "app.jar"]
```

- Jenkinsfile

```

pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout 1 hour
    }
    environment {
        DOCKER = 'sudo docker'
        TIME_ZONE = 'Asia/Seoul'
        PROFILE = 'local'
        REGION = 'ap-northeast-2'
    }
    stages{

        stage('Clone Repository') {
            steps {
                checkout scm
                echo 'Checkout Scm'
            }
        }

        stage('env setting') {
            steps{
                sh 'cp /var/jenkins_home/env/auth/application-prod.yml /var/jenkins_home/workspace/prod-back-auth/forest-back/forest-auth'
            }
        }

        stage('dockerizing project by dockerfile') {
            steps {
                dir('forest-back/forest-auth') {
                    sh '''
                        ls -al
                        chmod +x ./gradlew
                        ./gradlew build
                        docker build -t ${IMAGE_NAME_AUTH}:${BUILD_NUMBER} .
                        docker tag ${IMAGE_NAME_AUTH}:${BUILD_NUMBER} ${IMAGE_NAME_AUTH}:latest
                    '''
                }
            }
        }
        post {
            success {
                echo 'success dockerizing project'
            }
            failure {
                error 'fail dockerizing project' // exit pipeline
            }
        }
    }
}

```

```

stage('upload aws ECR') {
    steps {
        script{
            // cleanup current user docker credentials
            sh 'rm -f ~/.dockercfg ~/.docker/config.json || true'

            docker.withRegistry("https://${ECR_PATH_AUTH}", "ecr:${REGION}:${AWS_CREDENTIAL_NAME}") {
                docker.image("${IMAGE_NAME_AUTH}:${BUILD_NUMBER}").push()
                docker.image("${IMAGE_NAME_AUTH}:latest").push()

                sleep 10
            }
        }
    }
    post {
        success {
            echo 'success upload image'
            sh 'docker rmi ${IMAGE_NAME_AUTH}:${BUILD_NUMBER}'
            sh 'docker rmi ${IMAGE_NAME_AUTH}:latest'
        }
        failure {
            error 'fail upload image'
            sh 'docker rmi ${IMAGE_NAME_AUTH}:${BUILD_NUMBER}'
            sh 'docker rmi ${IMAGE_NAME_AUTH}:latest'
        }
    }
}

stage('Manifest Update') {
    steps {
        git credentialsId: '${GIT_CREDENTIALS_ID}',
            url: '${GIT_URL}',
            branch: 'main'

        dir('back') {
            sh """
            ls -al

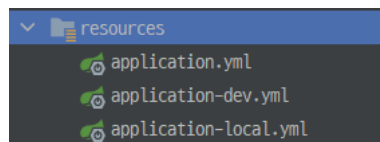
            cat ${EKS_DEPLOY_AUTH_YML} | grep image

            sed -i 's|image: .*|image: "${IMAGE_NAME_AUTH}:${currentBuild.number}"|' ${EKS_DEPLOY_AUTH_YML}
            cat ${EKS_DEPLOY_AUTH_YML} | grep image

            git status
            git add ${EKS_DEPLOY_AUTH_YML}
            git commit -m '[UPDATE] forest-back-auth ${currentBuild.number} image versioning'
            git push origin main
            """
        }
    }
}
}
}
}

```

- gitignore로 업로드 되지 않는 env 파일 및 기타 파일들은 jenkins 서버에 저장되어 있어서, clone 받은 이후 cp 명령어를 통해 배포 시점에 해당 파일들을 프로젝트에 복사하여 적용
- application.yml
- 개발 환경에 맞춰 yml 파일을 분리하여, local과 dev버전에 맞는 yml파일을 사용하도록 설정함
- prod버전은 배포 시점에 yml파일 삽입되도록 설정



Flask 배포

- Dockerfile

```

FROM python:3.11.1

WORKDIR /test
COPY . .
RUN pip install -r requirements.txt

```

EXPOSE 5000

CMD python ./app.py

- requirements.txt

Flask==2.3.2
Werkzeug==2.3.3

- docker-compose.yml

```
version: "3"
services:
  application:
    image: ${FOREST_FLASK_IMAGE}
    container_name: ${FOREST_FLASK_CONTAINER}
    ports:
      - 5000:5000
    restart: always
```

외부 문서

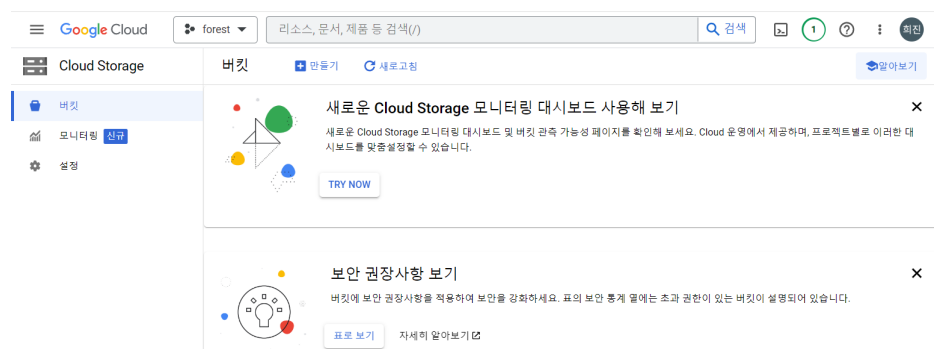
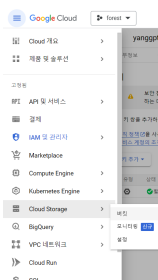
AWS S3

- application.yml

```
cloud:
  aws:
    credentials:
      accessKey: ${S3_ACCESS_KEY}
      secretKey: ${S3_SECRET_KEY}
    s3:
      bucket: ${S3_BUCKET}
    stack:
      auto: false
    region:
      static: ap-northeast-2
```

Google Cloud Vision

- Google Storage 생성



- API 사용 설정

API에 대한 액세스 사용 설정

- 1 프로젝트 확인
- 2 API 사용 설정

- IAM 계정 생성 및 권한 설정

역할 지정
역할은 권한 집합으로 구성되며, 주 구성원이 이 리소스로 할 수 있는 작업을 결정합니다.
[자세히 알아보기](#)

역할	권한	비고
서비스 계정 생성자	서비스 계정을 생성할 수 있는 액세스 권한입니다.	IAM 권한 추가
서비스 계정 관리자	서비스 계정을 관리할 수 있는 액세스 권한입니다.	IAM 권한 추가
서비스 계정 사용자	서비스 계정을 사용하여 리소스에 액세스할 수 있는 액세스 권한입니다.	IAM 권한 추가

- 서비스 계정 생성 및 비공개 키 파일(.JSON) 다운로드

세부정보 권한 키 측정항목 로그

키

⚠ 보안 침해 시 서비스 계정 키로 인해 보안 위험이 발생할 수 있습니다. 서비스 계정 키를 다운로드하는 대신 [워크로드 아이덴티티 제휴](#)를 사용하는 데 적합한 방법은 [여기](#)에서 자세히 알아볼 수 있습니다.

새 키 쌍을 추가하거나 기존 키 쌍의 공개키 인증서를 업로드하세요.

[조직 정책](#)을 사용하여 서비스 계정 키 생성을 차단합니다.
[서비스 계정의 조직 정책 설정 자세히 알아보기](#)

[키 추가](#)

유형	상태	키	키 생성일	키 만료일
	활성	[REDACTED]	2023. 4. 26.	10000. 1. 1.

- IntelliJ에서 *Environment variables* 설정

```
GOOGLE_APPLICATION_CREDENTIALS=D:\final-auto\S08P31B105\forest-back\forest-workbook\src\main\resources\{key}.json
```

Edit Configuration Settings

Name: ForestWorkbookApplication ☐ Store as project file

Run on: ☒ Local machine [Manage targets...](#)

Build and run [Modify options](#) [Alt+M](#)

Java 11 SDK of 'forest-workbook' -cp forest-workbook.main

com.ssafy.forestworkbook.ForestWorkbookApplication

Press Alt for field hints

Active profiles: local

Comma separated list of profiles

Shorten command line: JAR manifest - java -cp classpath.jar className [args]

Environment variables: GOOGLE_APPLICATION_CREDENTIALS=D:\final-auto\S08P31B105\forest-l

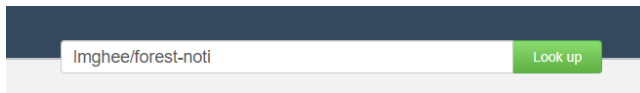
Separate variables with semicolon: VAR=value; VAR1=value1

☐ Open run/debug tool window when started ☐ Add dependencies with "provided" scope to classpath

[Run](#) [Cancel](#) [Apply](#)

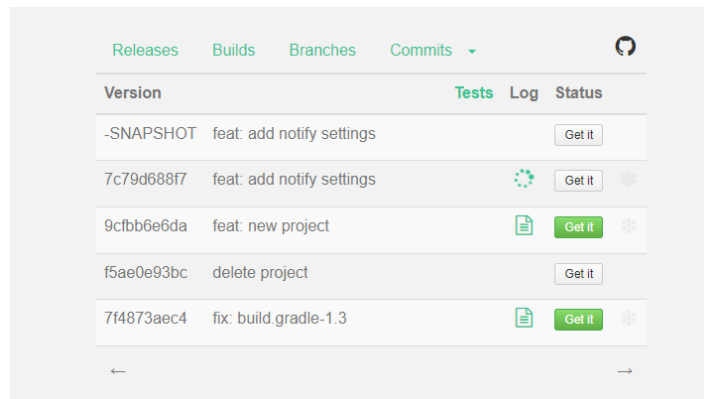
Jitpack

- <https://jitpack.io/>
- [github username]/[repository name] 검색



lmghee/forest-noti Look up

- commit version 혹은 Releases 확인 후 Get It 클릭 → 빌드 성공 여부 확인



Version	Tests	Log	Status
-SNAPSHOT feat: add notify settings			Get it
7c79d688f7 feat: add notify settings			Get it
9cfbb6e6da feat: new project			Get it
f5ae0e93bc delete project			Get it
7f4873aec4 fix: build.gradle-1.3			Get it

- 라이브러리 추가하기
 - /build.gradle

```
allprojects {
    repositories {
        ...
        maven { url 'https://jitpack.io' }
    }
}

dependencies {
    implementation 'com.github.lmghee:forest-noti:Tag'
}
```