포팅 메뉴얼

작성일: 2025.05.19.
작성자: 김휘동

목차

---

1. 버전 정보

1.1. Backend

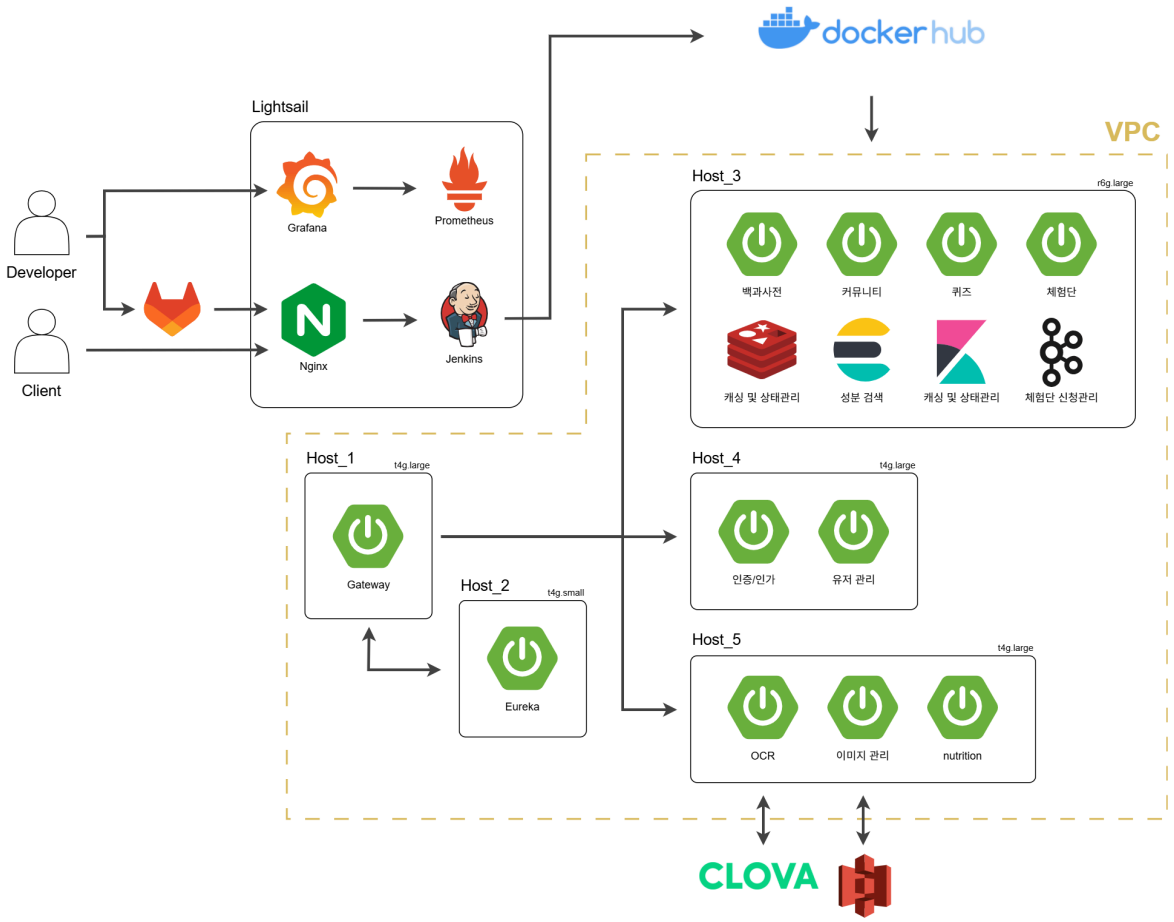
- EC2 : Ubuntu 24.04-arm
- Spring Boot : 3.4.5
- JDK : Corretto 17
- Gradle : 8.13
- FastAPI : 0.115.12
- Python : 3.10

- Nginx : 1.27.5
- Docker : 28.1.1
- Jenkins : 2.504.1
- Spring Cloud(Eureka, Gateway) : 2024.0.1
- Kafka : 7.5.3
- MySQL : 8.0
- Redis : 8.0
- MongoDB : 6.0
- ElasticSearch, Kibana : 8.13.4

1.2. Frontend

- React : 19.1.0
- Nextjs : 15.3.1
- npm : 11.0.0

2. 서비스 구조



외부 서비스: 네이버 클로바 OCR, AWS S3, AWS Cloudfront

3. 백엔드

공통사항

1. **도커 및 도커컴포즈 설치**

```
# 도커 설치에 필요한 패키지 설치
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl

# GPG 키를 저장할 디렉토리(키링) 생성 및 권한설정
sudo install -m 0755 -d /etc/apt/keyrings

# GPG 키(패키지 검증을 위해 사용) 다운로드
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt
```

```
# 도커 레포지토리 추가
# $(dpkg --print-architecture)는 현재 시스템 아키텍쳐를 감지해서 넣음.
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" \
> /etc/apt/sources.list.d/docker.list
# 권한 오류 발생 시
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | suc

# 패키지 목록 업데이트(반드시 실행)
sudo apt-get update

# 도커 설치
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

# 도커 컴포즈 설치
sudo apt-get install -y docker-buildx-plugin docker-compose-plugin

# 현재 사용자를 docker 그룹에 추가
sudo usermod -aG docker $USER

#변경사항 적용
newgrp docker

#권한 확인(docker가 있으면 됨)
groups
```

2. **Jenkins 서버 Public Key 등록**
   a. Jenkins 서버에서 각 서비스 연결을 위한 키 생성
   젠킨스 실행 시 볼륨마운트가 되었기 때문에, 호스트에서 아래 명령어를 실행해도 무관함.

   ```
   ssh-keygen
   ```

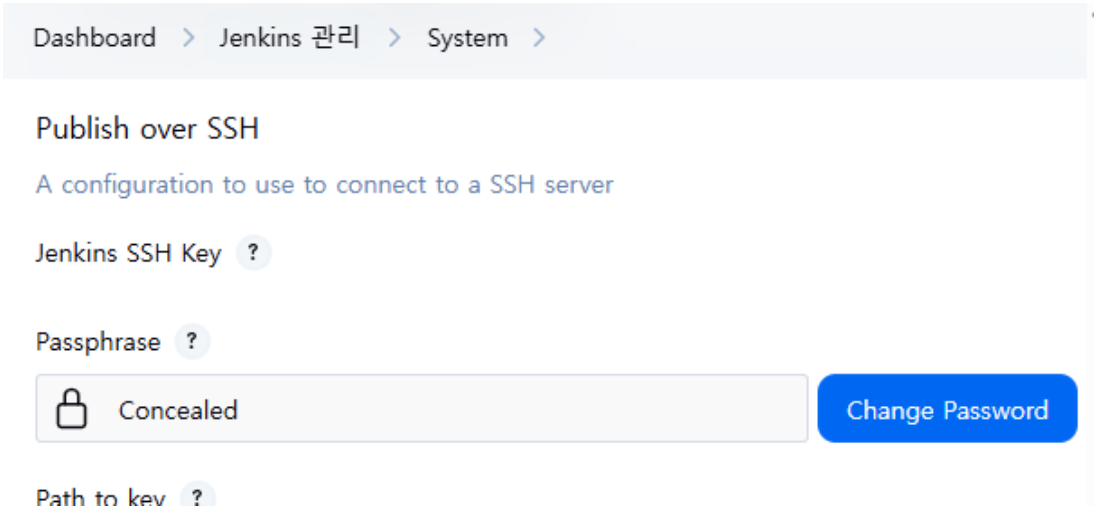
   b. 공개키를 복사하여 각 서비스에 등록
   Jenkins 호스트에서 공개키 복사(
   `ssh` 부터 아이피를 포함하여 복사)

```
$ cat home/ubuntu/.ssh/id_rsa.pub
ssh-rsa AAAAB3Nza...t8yFF+ysc8k= ubuntu@ip-172-***-***-***
```

서비스 호스트에서 아래 경로의 파일 아래에 추가

```
$ vim home/ubuntu/.ssh/authorized_keys
```

3. **Jenkins Publish over SSH 설정**
   a. Jenkins 관리 - System - Publish over SSH 에서 '추가' 클릭



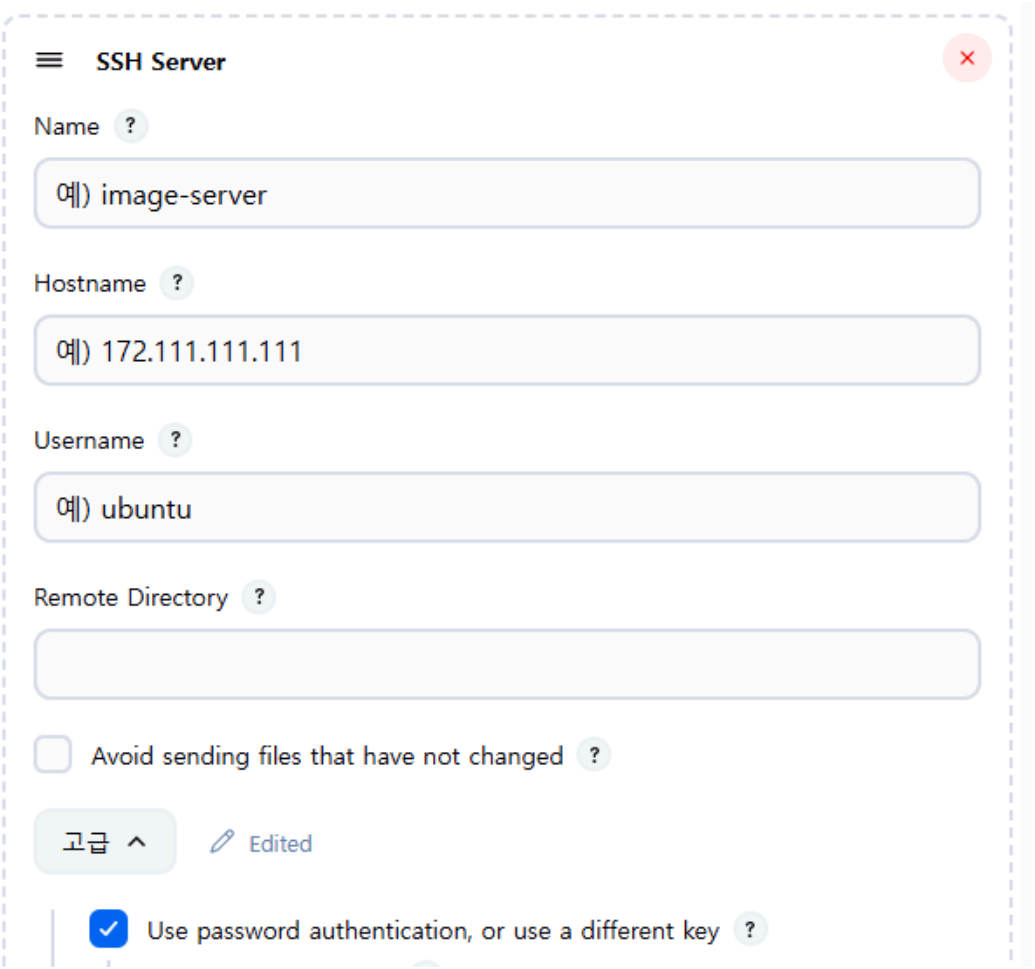
   b. 원격 호스트 정보 입력

      Name: pipeline에서 식별할 이름

      Hostname: 서비스 호스트 ip(같은 VPC인 경우, 사설 ip 사용 가능)

      Username: 호스트 계정 이름(보통 ubuntu)

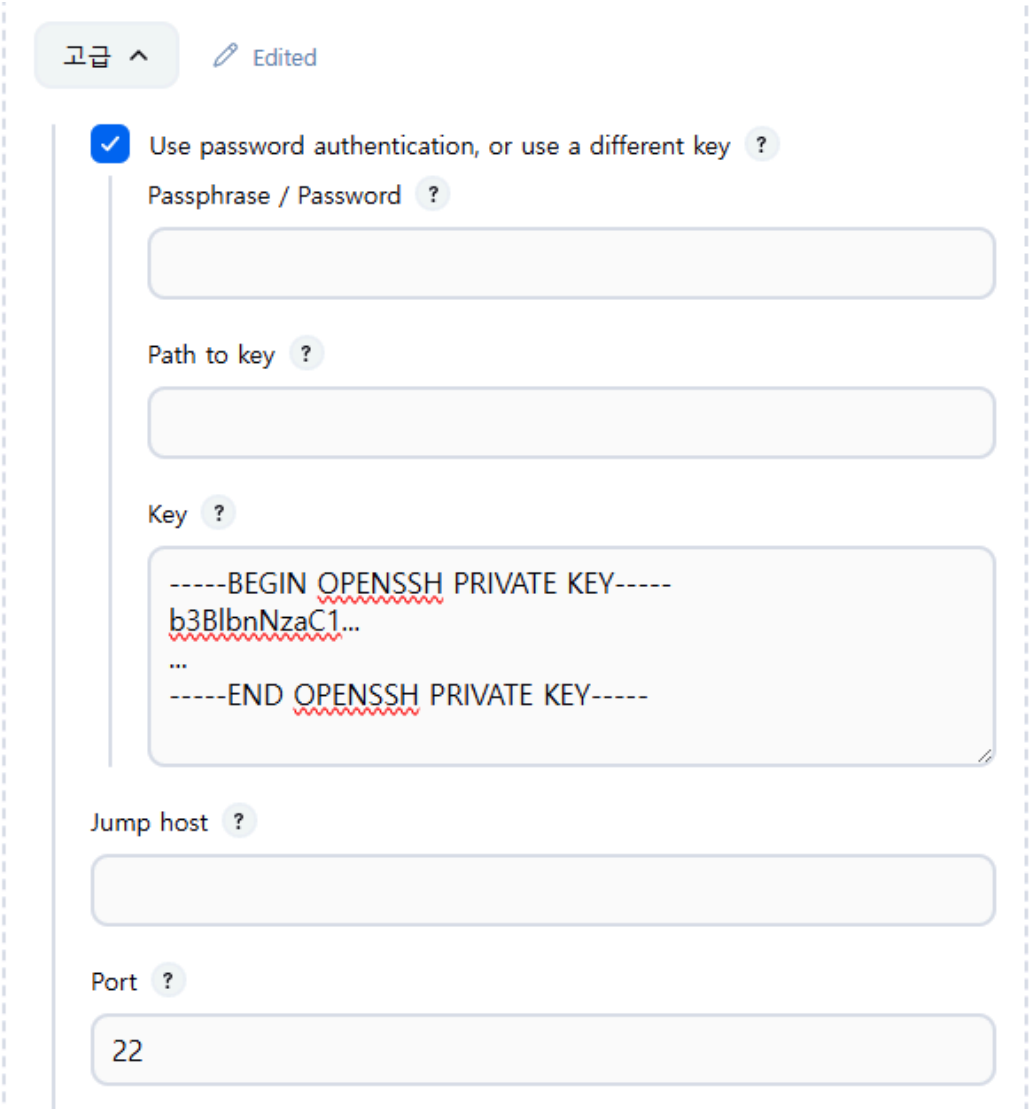
      Remote Directory(선택): 작업할 디렉토리(기본은 `/home/ubuntu`)



c. Jenkins 호스트에서 비밀키 복사 **(!! 유출되면 인생 망함 !!)**

```
$ cat home/ubuntu/.ssh/id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1...
...
-----END OPENSSH PRIVATE KEY-----
```

d. 고급 설정의 `Use password authentication, or use a different key` 선택, 비밀키 입력



3.1. Jenkins

1. jdk 설치

```
sudo apt update
sudo apt install openjdk-17-jdk
```

2. 젠킨스 폴더 생성

```
# 폴더 생성(sudo를 붙이면 소유자가 root가 되어 추후에 권한 오류 발생할 수도 있음
cd /home/{사용자명} && mkdir jenkins-data
```

3. 네트워크 생성
   아직 Nginx가 구성되지 않았지만, 미리 설정함

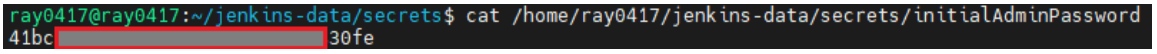
```
docker network create nginx-network
```

4. 설치

```
docker run -d \
  -v /home/{사용자명}/jenkins-data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /home/{사용자명}/docker/proxy:/proxy \
  -v /home/ubuntu/.ssh:/root/.ssh \
  -p 8080:8080 \
  -e JENKINS_OPTS="--prefix=/jenkins" \
  --group-add $(getent group docker | cut -d: -f3) \
  -e TZ=Asia/Seoul \
  --restart=on-failure \
  --network nginx-network \
  --name jenkins \
  jenkins/jenkins:lts-jdk17
```

5. 계정 설정

```
cat /home/{사용자명}/jenkins-data/secrets/initialAdminPassword
```



위와 같이 `41bc****30fe` 라는 초기 비밀번호를 확인할 수 있음.

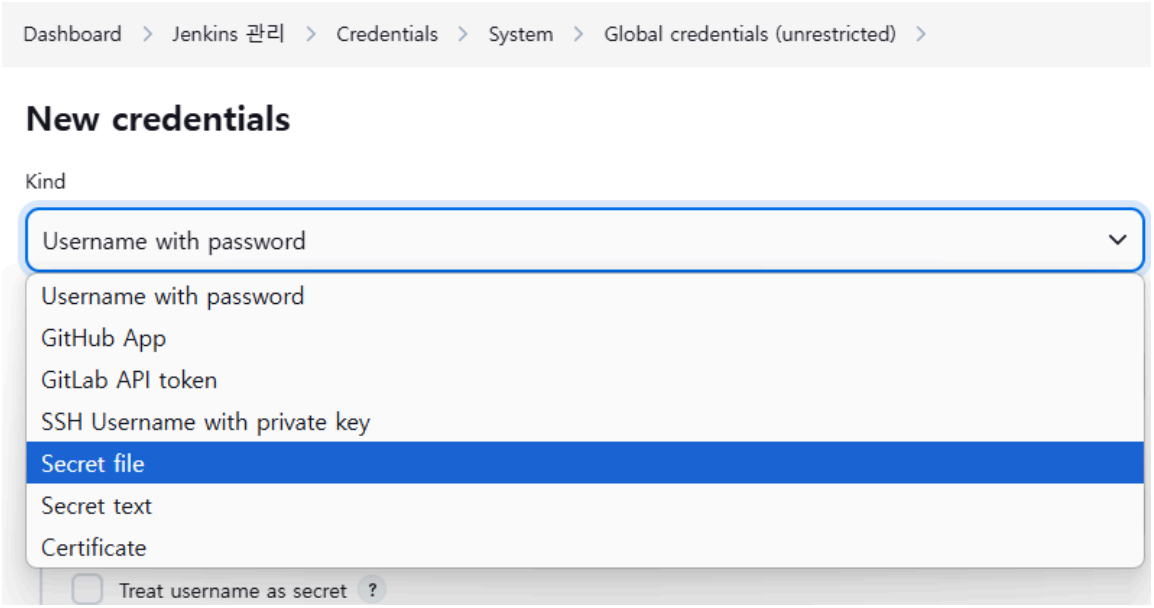
http://[호스트]:8080/jenkins 로 접속 후, 비밀번호 입력

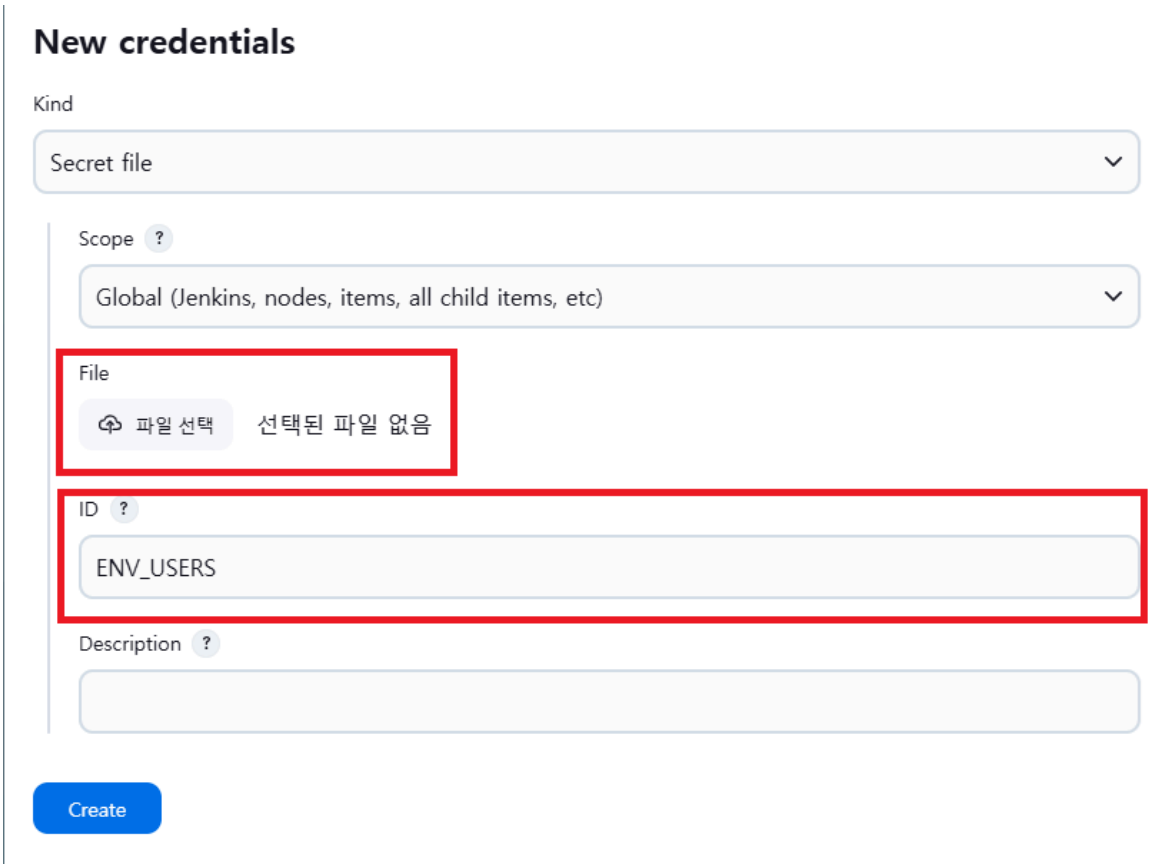
이후 **Install suggested plugins** 클릭.

6. Jenkins 컨테이너에 도커 설치

```
# root로 접속해야 모든 작업 실행 가능

docker exec -it -u root jenkins bash

# 패키지 업데이트 및 필요한 패키지 설치
apt-get update
apt-get install -y ca-certificates curl lsb-release

# 도커 GPG 키 설정
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | tee /etc/apt/ke

# 도커 리포지토리 설정(debian)
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings
https://download.docker.com/linux/debian bookworm stable" > /etc/apt/so

# 패키지 업데이트 및 도커 CLI 설치
apt-get update
apt-get install -y docker-ce-cli
```

7. 환경변수 설정





3.2. Nginx

1. 기본 파일 생성

```
nginx/
├── conf.d/
```

```
├── docker-compose.yml
└── nginx.conf
```

```
events {
    worker_connections 1024;
}

http {
    server {
        listen 80;
        listen [::]:80;
        server_name [호스트];

        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
        }

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

    server {
        listen 443 ssl;
        server_name [호스트];

        ssl_certificate /etc/letsencrypt/live/$server_name/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/$server_name/privkey.pem;

        # 젠킨스
        location /jenkins {
            proxy_pass http://jenkins:8080/jenkins;
            proxy_http_version 1.1;
            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;
```

```
        # Jenkins 관련 추가 설정
        proxy_set_header X-Jenkins-Context "/jenkins";
        proxy_redirect http:// https://;
    }

    # 게이트웨이
    location /api/ {
        proxy_pass http://[호스트]:8000/; # 기본값이 8000포트
        proxy_http_version 1.1;
        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;
        proxy_pass_request_headers on;

    }

    # react
    location / {
        proxy_pass http://[호스트]:3000/; # 기본값이 3000 포트
        proxy_http_version 1.1;
        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;
        proxy_pass_request_headers on;
    }

  }
}
```

2. 인증서 발급 인증서 권한 설정

```
docker run -it --rm \
  -v letsencrypt_certs:/etc/letsencrypt \
  -v letsencrypt_data:/var/lib/letsencrypt \
  -p 80:80 \
  certbot/certbot certonly --standalone -d [server-domain]
```

```
docker run --rm \
  -v letsencrypt_certs:/etc/letsencrypt \
  alpine sh -c "chmod -R 755 /etc/letsencrypt/live && chmod -R 755 /etc/le
```

3. docker-compose.yml 작성

```
services:
  nginx:
    image: nginx:latest
    container_name: nginx-proxy
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - letsencrypt_certs:/etc/letsencrypt:ro
      - letsencrypt_data:/var/lib/letsencrypt
    restart: always
    networks:
      - nginx-network

  certbot:
    image: certbot/certbot
    container_name: certbot
    volumes:
      - letsencrypt_certs:/etc/letsencrypt
      - letsencrypt_data:/var/lib/letsencrypt
      - webroot:/var/www/html
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew --webr
    restart: always

volumes:
  letsencrypt_certs:
    external: true
  letsencrypt_data:
    external: true
```

```
  webroot:

networks:
  nginx-network:
    external: true
```

4. 실행
   Nginx docker-compose.yml이 있는 위치에서 실행

```
docker compose up -d
```

3.3. Spring Boot

1. pipeline

```
pipeline {
    agent any
    environment {
        BRANCH_NAME = '[브랜치 이름]'
        ENV_FILE_NAME = '[env 파일 이름]'
            ENV_CRED_ID = '[Credentials에 저장된 env ID]'
            DIR = '[프로젝트 경로]'
            DOCKER_USER = '[도커허브사용자명]'
        DOCKER_REPO = '[도커허브 레포지토리명]'
        SERVICE_NAME = '[서비스명]'
        IMAGE_NAME = "${DOCKER_USER}/${DOCKER_REPO}:${SERVICE_N
        SSH_REMOTE_NAME = '[publish over ssh 설정 id]'
    }
    stages {
        stage('Checkout') {
            steps {
                git credentialsId: 'GITLAB_ACCOUNT',
                    url: 'https://lab.*****.com/****/***.git',
                    branch: "${BRANCH_NAME}"
            }
        }
        stage('Build') {
```

```
        steps{
            dir("${SERVICE_DIR}"){
                withCredentials([file(credentialsId: "${ENV_CRED_ID}", variable
                    sh "touch ./$ENV_FILE_NAME || true"
                    sh "chmod +w ./$ENV_FILE_NAME"
                    sh "cp \$ENV_FILE ./$ENV_FILE_NAME"
                }
                sh "chmod +x ./gradlew"
                sh "./gradlew clean build -x test"
            }
        }
    }
    stage('Build & Push Docker Image') {
        steps {
            dir('${SERVICE_DIR}') {
                script {
                    // 도커 허브 로그인
                    withCredentials([usernamePassword(credentialsId: 'DOCKE
                        sh "docker login -u \$DOCKER_USERNAME -p \$DOCKER
                    }

                    // 도커 이미지 빌드(arm64) 및 push - 명시적으로 빌더 사용
                    sh "docker buildx build --builder mybuilder --platform linux/
                }
            }
        }
    }
    stage('Deploy') {
        steps {
            script {
                sshPublisher(
                    failOnError: true,
                    publishers: [
                        sshPublisherDesc(
                            configName: '${SSH_REMOTE_NAME}',
                            verbose: true,
                            transfers: [
                                sshTransfer(
```

```
                        cleanRemote: false,
                        sourceFiles: '${DIR}/docker-compose.*.yml',
                        removePrefix: '${DIR}',
                        remoteDirectory: './${SERVICE_NAME}/', // 컨테이너
                    ),
                    sshTransfer(
                        cleanRemote: false,
                        sourceFiles: '${DIR}/${ENV_CRED_ID}',
                        removePrefix: '${DIR}',
                        remoteDirectory: './${SERVICE_NAME}/',
                    ),
                    sshTransfer(
                        execCommand: """
                            docker stop ${SERVICE_NAME} || true
                            docker rm ${SERVICE_NAME} || true
                            docker login
                            docker rmi  ${IMAGE_NAME} || true
                            docker pull ${IMAGE_NAME}
                            docker compose -f ${SERVICE_NAME}/docker-
                        """
                    )
                ]
            )
        ]
    )
  }
 }
 }
 post {
  cleanup {
   cleanWs()
  }
 }
}
```

3.4. FastAPI(OCR)

1. pipeline

```
pipeline {
    agent any
    environment {
        BRANCH_NAME = '[브랜치 이름]'
        ENV_FILE_NAME = '[env 파일 이름]'
        ENV_CRED_ID = '[Credentials에 저장된 env ID]'
        DIR = '[프로젝트 경로]'
        DOCKER_USER = '[도커허브사용자명]'
    DOCKER_REPO = '[도커허브 레포지토리명]'
    SERVICE_NAME = '[서비스명]'
    IMAGE_NAME = "${DOCKER_USER}/${DOCKER_REPO}:${SERVICE_N
    SSH_REMOTE_NAME = '[publish over ssh 설정 id]'
    }
    stages {
    stage('Checkout') {
        steps {
        git credentialsId: 'GITLAB_ACCOUNT',
            url: 'https://lab.*****.com/****/***.git',
            branch: "${BRANCH_NAME}"
        }
    }
    stage('Build') {
        steps{
        dir("${SERVICE_DIR}"){
            withCredentials([file(credentialsId: "${ENV_CRED_ID}", variabl
            sh "touch ./$ENV_FILE_NAME || true"
            sh "chmod +w ./$ENV_FILE_NAME"
            sh "cp \$ENV_FILE ./$ENV_FILE_NAME"
            }
            sh "chmod +x ./gradlew"
            sh "./gradlew clean build -x test"
        }
        }
    }
    stage('Build & Push Docker Image') {
```

```
        steps {
            dir('${SERVICE_DIR}') {
                script {
                    // 도커 허브 로그인
                    withCredentials([usernamePassword(credentialsId: 'DOCKE
                        sh "docker login -u \$DOCKER_USERNAME -p \$DOCKER
                    }

                    // 도커 이미지 빌드(arm64) 및 push - 명시적으로 빌더 사용
                    sh "docker buildx build --builder mybuilder --platform linux/
                }
            }
        }
    }
    stage('Deploy') {
        steps {
            script {
                sshPublisher(
                    failOnError: true,
                    publishers: [
                        sshPublisherDesc(
                            configName: '${SSH_REMOTE_NAME}',
                            verbose: true,
                            transfers: [
                                sshTransfer(
                                    cleanRemote: false,
                                    sourceFiles: '${DIR}/docker-compose.*.yml',
                                    removePrefix: '${DIR}',
                                    remoteDirectory: './${SERVICE_NAME}/', // 컨테이너
                                ),
                                sshTransfer(
                                    cleanRemote: false,
                                    sourceFiles: '${DIR}/${ENV_CRED_ID}',
                                    removePrefix: '${DIR}',
                                    remoteDirectory: './${SERVICE_NAME}/',
                                ),
                                sshTransfer(
                                    cleanRemote: false,
```

```
                        sourceFiles: '${DIR}/init-mongo.js',
                        removePrefix: '${DIR}',
                        remoteDirectory: './${SERVICE_NAME}/',
                    ),
                    sshTransfer(
                        execCommand: """
                            docker stop ${SERVICE_NAME} || true
                            docker rm ${SERVICE_NAME} || true
                            docker login
                            docker rmi  ${IMAGE_NAME} || true
                            docker pull ${IMAGE_NAME}
                            docker compose -f ${SERVICE_NAME}/docker-
                        """
                    )
                ]
            )
        ]
    )
  }
 }
 }
 post {
    cleanup {
        cleanWs()
    }
 }
}
```

4. 프론트엔드

4.1. pipeline

```
pipeline {
    agent any
```

```
    environment {
        DOCKER_IMAGE = 'my-nextjs-app'
        DOCKER_TAG = 'latest'
        CONTAINER_NAME = 'nextjs-container'
        PATH = "/opt/nodejs/bin:$PATH"
    }

    stages {
        stage('Checkout') {
            steps {
                git credentialsId: 'GITLAB_ACCOUNT',
                    url: 'https://lab.*****.com/***/***.git',
                    branch: 'fe/develop'
            }
        }

        stage('Docker Build') {
            steps {
                dir("./frontend/boindang") {
                    sh "docker build -t ${DOCKER_IMAGE}:${DOCKER_TAG} ."
                }
            }
        }

        stage('Stop & Remove Old Container') {
            steps {
                sh """
                    docker stop ${CONTAINER_NAME} || true
                    docker rm ${CONTAINER_NAME} || true
                """
            }
        }

        stage('Run New Container') {
            steps {
                sh """
                    docker run -d \
                    --name ${CONTAINER_NAME} \
```

```
            -p 3000:3000 \
            --network nginx-network \
            --restart unless-stopped \
            ${DOCKER_IMAGE}:${DOCKER_TAG}
          """
        }
      }
    }
}
```

5. 환경변수

아래 항목 중 각 서비스에 해당하는 항목만 선택하여 작성

```
// 공통
EUREKA_URL
HOST_IP

// MySQL
DB_ROOT_PASSWORD
DB_USERNAME
DB_PASSWORD
DB_NAME
DB_URL

// auth
JWT_SECRET

// Campaign
BOOTSTRAP_SERVER

// Encyclopedia
ELASTICSEARCH_URL
REDIS_HOST

// Image
```

```
S3_ACCESS_KEY
S3_SECRET_ACCESS_KEY
S3_BUCKET_NAME
S3_REGION

// OCR
CLOVA_OCR_API_URL
CLOVA_OCR_SECRET_KEY
OPENAI_API_URL
OPENAI_API_KEY
MONGODB_URI

// User
REDIS_HOST
```