포팅 매뉴얼

▼ 0.a 각 버전

▼ 백엔드 버전

- JDK: 17
- Spring Boot: 3.3.1
- Apache Tomcat: 10.1.25
- MySQL: 9.0.1
- IntelliJ IDEA ultimate 2024.2.0.1
- redis: 7.2.5

▼ 프론트엔드 버전

- IDE: vscode 1.90 + / webstorm 2024.1
- Typescript: 5
- React.js: 18.3.1
- Next.js: 14.2.5

▼ CICD 버전

- 도커: 27.1.1(API: 1.46)
- 도커 컴포즈: 2.29.0
- 젠킨스: 2.469
- nginx: 1.27.0

▼ 도커 올릴 때 환경변수(**도커 컴포즈 파일)**

```
services:
  mysql:
    image: mysql
```

```
    container_name: mysql-saturi
    restart: always
    environment:
      MYSQL_DATABASE: saturi
      MYSQL_ROOT_PASSWORD: Hhl0422!@
      MYSQL_USER: ssafy
      MYSQL_PASSWORD: ssafy
      LANG: ko_KR.utf-8
      LANGUAGE: ko_KR:ko
      LC_ALL: ko_KR.utf-8
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    volumes:
      - /home/ubuntu/docker/mysql-data:/var/lib/mysql
      - /home/ubuntu/docker/mysql-my.cnf:/etc/mysql/conf

  nginx:
    volumes:
      - /home/ubuntu/docker/default.conf:/etc/nginx/conf
      - /etc/letsencrypt/live:/etc/letsencrypt/live
      - /etc/letsencrypt/archive:/etc/letsencrypt/archiv

jenkins:
    - /home/ubuntu/docker/jenkins-data:/var/jenkins_home
    - /var/run/docker.sock:/var/run/docker.sock
```

▼ MySQL 접속 정보

- root 계정 비밀번호 : Hhl0422!@
- 일반 계정 id : ssafy
- 일반 계정 패스워드 : ssafy

▼ AI 서버 환경변수

```
{
  "type": "service_account",
  "project_id": "aesthetic-frame-430503-d3",
  "private_key_id": "abbeb208b89aa9e576ab70e313ea31d7cf9
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEuwIBA
  "client_email": "saturi-storage-access@aesthetic-frame
  "client_id": "111830742792573343589",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis
  "client_x509_cert_url": "https://www.googleapis.com/ro
  "universe_domain": "googleapis.com"
}
```


▼ 환경변수 정의 파일 리스트

- /backend/src/main/resources/application.yml
- /frontend/.env
- /AI/AIpjt/requirements.txt
- /AI/AIpjt/AIpjt/settings.txt
- /AI/AIpjt/.env

▼ 0.b 외부 서비스 정보

1) 카카오 api 서비스 등록 방법

1. 카카오 디벨로퍼스 앱 생성
2. 비즈앱 전환
3. 내가 만든 앱의 **테스트 앱** 생성
4. 테스트 앱에서 이메일 등 개인 정보를 한시적으로 얻을 수 있다.
5. 테스트 앱에서 아래 정보 등록

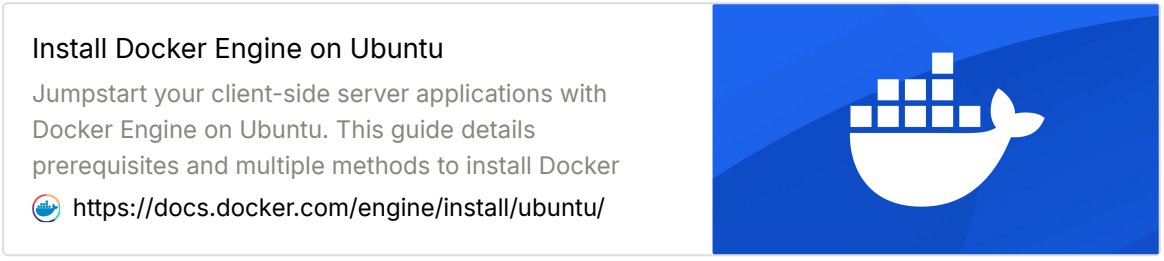
```
1. 카카오 로그인 활성화
```

```
2. 내 프로젝트 URL 등록
    - '플랫폼' 탭에서 내 프로젝트 URL을 등록한다.

3. redirect URI 등록
    - '카카오 로그인' 탭에서 카카오 로그인 요청 이후에 이동할 페이지
    - redirect는 인증 서버에서 코드를 주는 URI이다.
    - localhost:3000 에서 로그인 요청을 하고 redirect를 local
    - 했다면 카카오 로그인 창이 뜨고, 유저가 로그인을 하면 위에 주소

4. 애플리케이션에 필요한 동의항목 설정
    - 카카오 로그인 안에 '동의항목'에서 원하는 정보를 선택한다.
    - 만약 활성화가 안되어있다면 테스트 앱이 아닌거임

5. 필요에 따라 보안 강화 설정
    - 카카오 로그인 안에 '보안' 탭에서 보안 강화용 client_secret

6. 서비스 키 확인
```

▼ 1. 도커, 도커 컴포즈, 도커 이미지 다운로드 받기

Docker 설치

아래 페이지를 참조해 ec2 내에서 도커를 설치해줍니다.



젠킨스 이미지 받기

```
FROM jenkins/jenkins:jdk17
```

```
USER root

RUN apt-get update && \
    apt-get install -y apt-transport-https ca-certificates
    curl -fsSL https://download.docker.com/linux/debian/gpg
    add-apt-repository "deb [arch=amd64] https://download.c
    apt-get update && \
    apt-get install -y docker-ce-cli iputils-ping netcat-op
    apt-get clean

RUN groupadd -f docker
RUN usermod -aG docker jenkins
USER jenkins
```

DooD를 사용하려면 젠킨스에 Docker-cli가 깔려있어야 합니다. 이미지에 포함시켜버립니다. 위 코드를 Dockerfile이란 이름으로 저장합니다.

```
docker build -t jenkins .
```

이미지를 빌드합니다. Dockerfile이 있는 위치에서 실행합니다.

Docker-compose 설치 및 mysql, redis, nginx 이미지 받기

```
//docker-compose 설치
sudo curl -SL https://github.com/docker/compose/releases/do
sudo chmod +x /usr/local/bin/docker-compose

//mysql,redis,nginx 설치
sudo docker pull mysql
sudo docker pull redis
sudo docker pull nginx
```

젠킨스 말고도 백엔드에서 사용할 `mysql`, `redis`와 리버스 프록시 기능을 지원하는 `nginx`도 도커 이미지로 다운받습니다. 컨테이너 여러 개를 한번에 관리하기 위한 docker-compose도 같이 받습니다.

▼ 2. 젠킨스 설정하기(백엔드)

젠킨스 컨테이너 띄우기

젠킨스 안에서 플러그인 이라는 것들을 또 다운받아야 하는데 이게 상당히 오래 걸립니다.

```
version: "3.8"
services:
  jenkins:
    image: jenkins
    container_name: jenkins-saturi
    restart: always
    ports:
      - "9090:8080"
    volumes:
      - /home/ubuntu/docker/jenkins-data:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
networks:
  default:
    name: saturi_network
    driver: bridge
```

젠킨스를 컨테이너로 올리기 위한 docker-compose.yml 파일을 작성합니다.

```
docker-compose up -d
docker-compose down
```

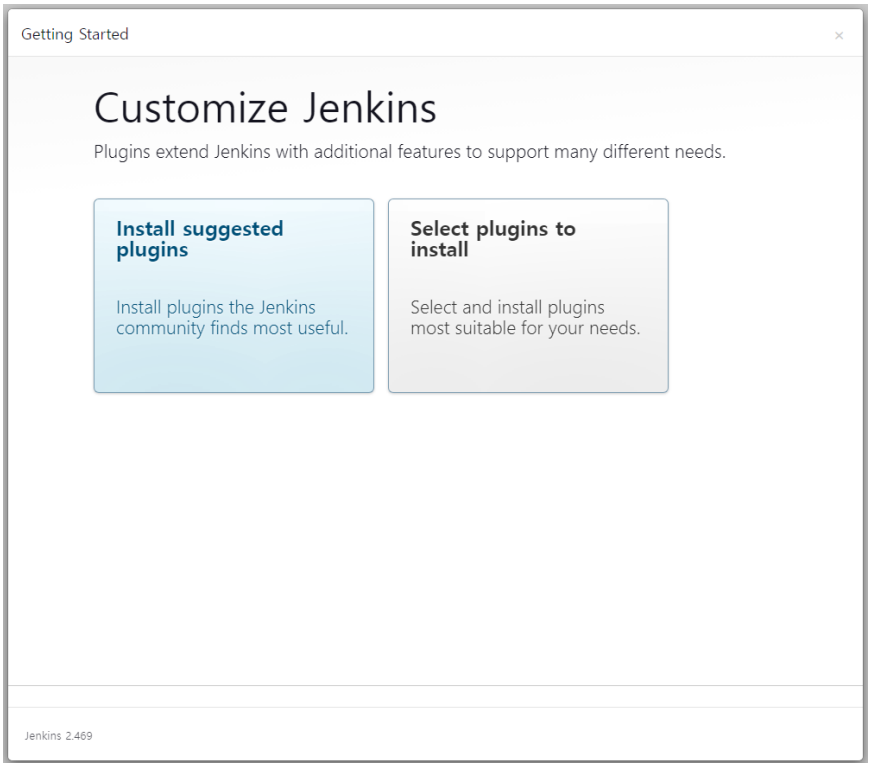
up 명령어로 컨테이너들을 올리고 down 명령어로 내립니다.

▼ 권한 에러

젠킨스의 데이터를 저장하기 위해 ec2 디렉토리와 마운트 시켜놨는데 ec2 디렉토리에 대해 jenkins 사용자가 권한이 없을 수 있습니다. 그럴때는 jenkins 사용자에게 chown이나 chmod 명령어를 통하여 ec2 로컬 디렉토리의 권한을 부여해야 합니다.

젠킨스 플러그인 설치하기

젠킨스 컨테이너를 올리셨으면 퍼블릭 IP 주소:9090 을 쳐서 젠킨스 홈페이지로 들어갑니다. 초기 비밀번호를 입력하는 칸이 나오는데 `docker logs jenkins-saturi` 를 입력해 초기 비밀번호를 얻을 수 있습니다.



이후 어떤 플러그인 세트를 설치할 지 물어보는데 왼쪽 **권장 플러그인**들 설치를 해줍니다. 그리고 추가적으로 **ssh-agent와 gitlab, Nodejs** 플러그인 또한 설치해줍니다. 플러그인을 모두 설치했다면 한번 재부팅 해줍시다.

`Jenkins 관리 > Plugins > Available plugins`

Jenkins Credentials 설정하기

젠킨스에서 제 깃랩에 있는 코드를 클론해와서 빌드/ 배포할 예정입니다. 그럴려면 제 깃랩에 접근할 권한이 있어야 합니다.

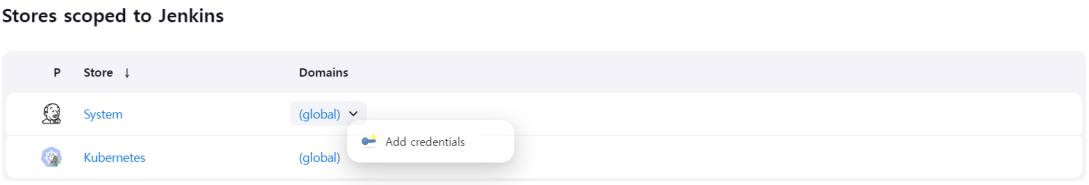
깃랩 AccessToken 발급받기

유저 아이콘 클릭 > Preferences > Access Token > Add new token

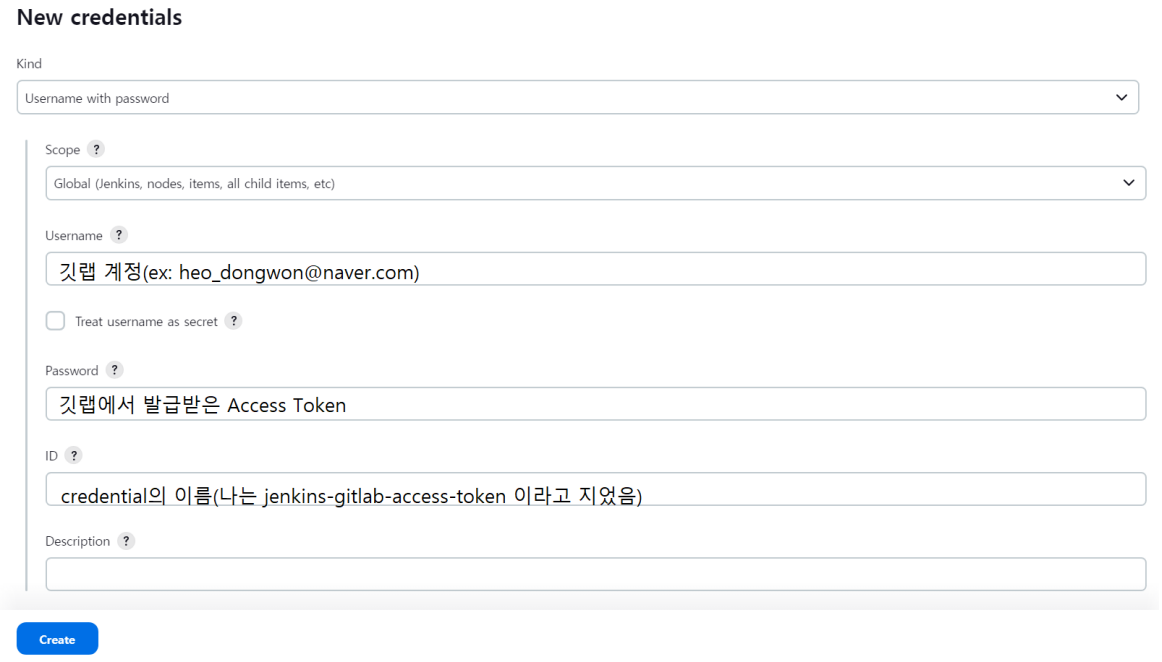
이름넣고 권한, 유효 기간 설정하면 됩니다.

젠킨스에 깃랩 AccessToken 등록하기

Jenkins > Jenkins 관리 > Credentials 로 이동합니다.



global 도메인에서 Add credential 클릭



사진대로 작성해주고 Create 클릭

Jenkins에 EC2 ssh key 등록하기

마찬가지로 Jenkins > Jenkins 관리 > Credentials 로 이동해서 add credentials

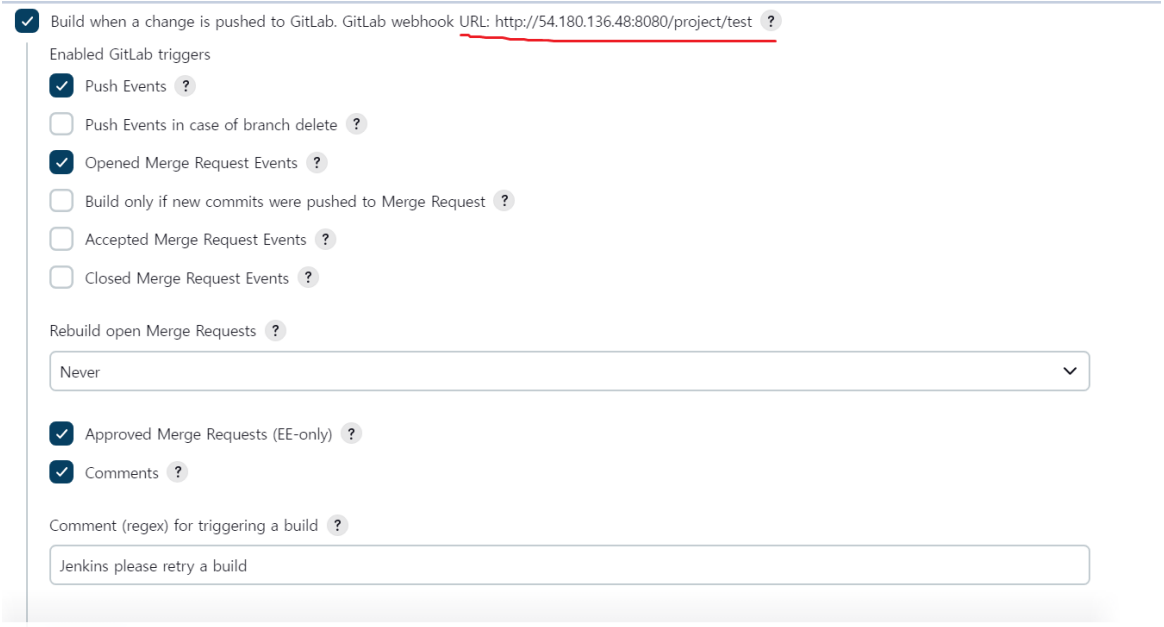
요렇게 하면 젠킨스에서 제 프로젝트와 제 EC2 서버에 접근할 수 있게 됩니다.

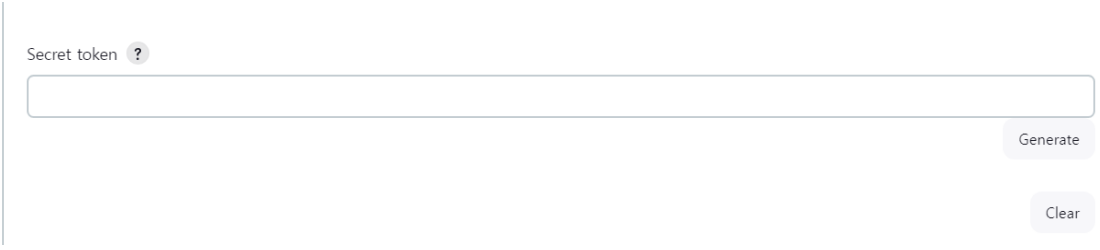
Jenkins 배포 셋팅하기

이제 젠킨스가 제 깃에 있는 코드를 클론해서 빌드/ 배포할 겁니다.

Jenkins Dashboard > 새로운 아이템 > **프로젝트 이름** 입력하고 Pipeline 선택해서 OK

이제 아이템 설정에서 Build Trigger 옵션은 build when a change is pushed.....를 선택합니다. 깃랩에 어떤 변화가 푸쉬되면 빌드하겠다라는 옵션입니다. 이 옵션을 체크했을 때 URL이 뜨는데 이를 잘 기록해둡니다. 이따가 깃랩에서 푸쉬 이벤트가 발생했을 때 저희 젠킨스로 알림 비스무리한걸 보내야 하는데 방금 복사한 url이 그 알람을 받을 url입니다. 또한 아래로 조금 내려서 고급, Generate 버튼을 눌러 key도 저장해둡니다.





Webhook 설정하기

깃에 코드가 푸쉬가 되면 깃랩에서 젠킨스로 웹훅이라는 알람을 보냅니다.

깃랩 > 본인 프로젝트 > Settings > Webhooks로 이동해서 Add new webhook 클릭

url : 트리거 설정할 때 복사한 URL

name : 웹훅 이름 설정

secret token : 마찬가지로 트리거 설정할 때 복사한 secret token 넣기

push event 체크

이걸로 설정한 프로젝트에 푸쉬가 일어날 때마다 젠킨스로 알람이 가게 되고, 아래 보이는 파이프라인 스크립트로 알람이 왔을 때 어떤 동작을 할 지 정의해줄 수 있습니다.

파이프라인 작성하기

푸쉬가 됐을 때 젠킨스가 할 일을 정의합니다.

```
pipeline {
    agent any

    environment {
        // 환경 변수 설정
        BACK_IMAGE_NAME = "backend/saturi" // 도커 이미지 이름
        BACK_CONTAINER_NAME = "backend-saturi" // 도커 컨테이
    }

    stages {
        stage('Clone Repository') {
            steps {
                script {
                    // 오래된 브랜치 삭제
                    sh 'git remote prune origin || true'
                }

                // GitLab에서 프로젝트를 클론
                git branch: 'deploy', credentialsId: 'gitla
            }
        }

        stage('Build Project') {
            steps {
                // 프로젝트 빌드
                // backend 디렉토리에서 아래 명령어 실행
                dir('backend'){
                    sh '''
                        chmod +x ./gradlew
                        ./gradlew clean build -x test
                    '''
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                // Docker 이미지 빌드
```

```
                // Dockerfile이 /backend 디렉토리에 있어야 한다.
                // 빌드한 백엔드 jar 파일로 도커 이미지를 생성하는
                dir('backend'){
                    sh "docker build -t ${BACK_IMAGE_NAME}
                }
            }
        }

        stage('Stop and Remove Old Container') {
            steps {
                script {
                    // 기존 컨테이너 중지 및 삭제
                    // 만약 saturi-backend 라는 이름의 컨테이너가
                    def containerExists = sh(script: "docke

                    if (containerExists) {
                        sh "docker stop ${BACK_CONTAINER_NA
                        sh "docker rm ${BACK_CONTAINER_NAME
                    } else {
                        echo "Container ${BACK_CONTAINER_NA
                    }
                }
            }
        }

        stage('Run New Container') {
            steps {
                script {
                    // 새 이미지로 컨테이너 실행, 불필요 이미지 제거
                    sh '''
                    docker run -d --name=${BACK_CONTAINER_N
                    -e GOOGLE_APPLICATION_CREDENTIALS="/hom
                     -v /home/ubuntu/docker/backend-data/go
                    -v /home/ubuntu/docker/backend-data/sat
                    docker rmi -f $(docker images -f "dangl
                    '''
                }
            }
```

```
        }
    }

    post {
        success {
            echo 'Pipeline completed successfully!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

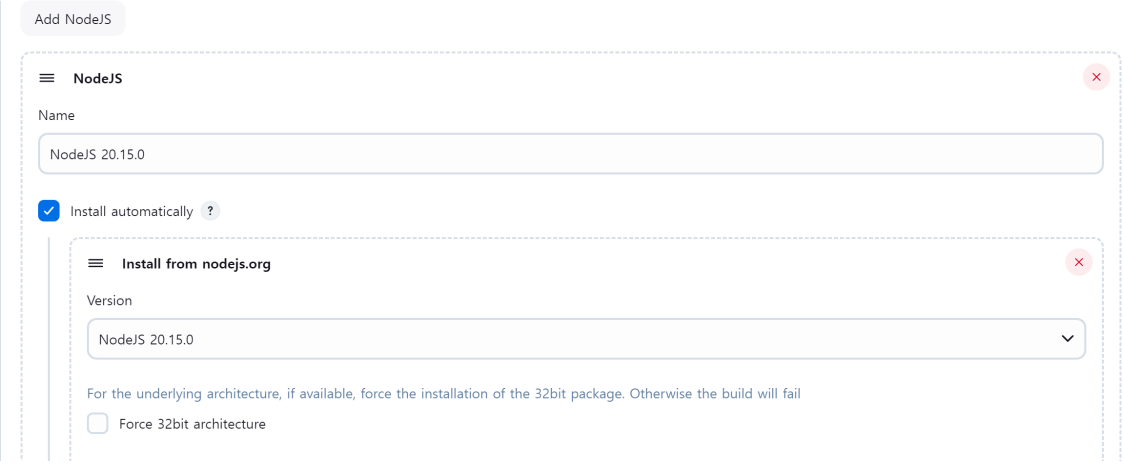
푸쉬가 감지되면 스프링부트 프로젝트를 빌드하고 그 빌드 결과물을 이용하여 image 파일을 만들고 최종적으로 컨테이너로 올립니다.

▼ 3. 젠킨스 설정하기(프론트엔드)

Node.js 빌드 설정

Jenkins Plugins에서 Nodejs 플러그인을 깔고 온다음에 빌드 설정을 해줍니다.

Dashboard > Jenkins 관리 > Tools > NodeJS 에서 아래와 같이 내용 입력



Name : 본인이 원하는 이름

Version : 본인 프로젝트에 맞는 버전

프론트 전용 파이프라인 스크립트 작성하기

백엔드 스크립트에 잘 넣어도 되고 아이템을 새로 만들어도 됩니다. 프론트엔드 프로젝트를 도커 컨테이너로 올립니다. 일반적으로 React 프로젝트의 경우 빌드시 정적파일이 생성돼서 그 파일을 nginx에 등록해줍니다. 하지만 저희 프로젝트는 Next.js를 사용했습니다. Next.js는 SSR 기반이므로 빌드 시 정적파일이 생성되지 않습니다. 그래서 컨테이너로 올려서 배포하고 Nginx는 프록시만 해주도록 했습니다.

```
pipeline {
    agent any

    environment {
        FRONT_IMAGE_NAME = "frontend/saturi"
        FRONT_CONTAINER_NAME = "frontend-saturi"
    }

    stages {
        stage('Clone Repository') {
            steps {
                script {
                    // 오래된 브랜치 삭제
                    sh 'git remote prune origin || true'
                }

                // GitLab에서 프로젝트를 클론
                git branch: 'release/FE', credentialsId: 'g
            }
        }

        stage('Stop and Remove Old Container') {
            steps {
                script {
                    script {
                        def containerExistsFront = sh(scrip
                        if (containerExistsFront) {
                            sh "docker stop ${FRONT_CONTAIN
                            sh "docker rm ${FRONT_CONTAINER
                        } else {
                            echo "Container ${FRONT_CONTAIN
                        }
```

```
                    }
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                dir('frontend'){
                    sh "docker build -t ${FRONT_IMAGE_NAME}
                }
            }
        }

        stage('Run New Container') {
            steps {
                script {
                    // 새 이미지로 컨테이너 실행, 불필요 이미지 제
                    sh '''
                    docker run -d --name=${FRONT_CONTAINER_
                    docker rmi -f $(docker images -f "dangl
                    '''

                }
            }
        }
    }

    post {
        success {
            echo 'Pipeline completed successfully!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

▼ 4. 젠킨스 설정하기(ai)

위와 똑같은 방식으로 ai용 아이템을 만들거나 아래 스크립트 코드를 원래 있던 아이템과 잘 섞어서 적습니다.

```
pipeline {
    agent any

    environment {
        // 환경 변수 설정
        AI_IMAGE_NAME = "django/saturi" // 도커 이미지 이름
        AI_CONTAINER_NAME = "django-saturi" // 도커 컨테이너
    }

    stages {
        stage('Clone Repository') {
            steps {
                script {
                    // 오래된 브랜치 삭제
                    sh 'git remote prune origin || true'
                }

                // GitLab에서 프로젝트를 클론
                git branch: 'deploy', credentialsId: 'gitla
            }
        }

        stage('Build Docker Image') {
            steps {
                // Docker 이미지 빌드
                // Dockerfile이 /backend 디렉토리에 있어야 한다
                // 빌드한 백엔드 jar 파일로 도커 이미지를 생성하는
                dir('AI/AIpjt'){
                    sh "docker build -t ${AI_IMAGE_NAME} ."
                }
            }
        }

        stage('Stop and Remove Old Container') {
            steps {
```

```
                script {
                    // 기존 컨테이너 중지 및 삭제
                    // 만약 saturi-backend 라는 이름의 컨테이너가
                    def containerExists = sh(script: "docke

                    if (containerExists) {
                        sh "docker stop ${AI_CONTAINER_NAME
                        sh "docker rm ${AI_CONTAINER_NAME}"
                    } else {
                        echo "Container ${AI_CONTAINER_NAME
                    }
                }
            }
        }

        stage('Run New Container') {
            steps {
                script {
                    // 새 이미지로 컨테이너 실행, 불필요 이미지 제거
                    sh '''
                    docker run -d --name=${AI_CONTAINER_NAM
                    docker rmi -f $(docker images -f "dangl
                    '''
                }
            }
        }
    }

    post {
        success {
            echo 'Pipeline completed successfully!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

▼ 5. Docker-compose로 컨테이너 관리하기

```
version: "3.8"

services:
  mysql:
    image: mysql
    container_name: mysql-saturi
    restart: always
    environment:
      MYSQL_DATABASE: saturi
      MYSQL_ROOT_PASSWORD: Hhl0422!@
      MYSQL_USER: ssafy
      MYSQL_PASSWORD: ssafy
      LANG: ko_KR.utf-8
      LANGUAGE: ko_KR:ko
      LC_ALL: ko_KR.utf-8
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    volumes:
      - /home/ubuntu/docker/mysql-data:/var/lib/mysql
      - /home/ubuntu/docker/mysql-my.cnf:/etc/mysql/conf.d/

  redis:
    container_name: redis-saturi
    image: redis

  jenkins:
    image: jenkins
    container_name: jenkins-saturi
    restart: always
    ports:
      - "9090:8080"
    volumes:
      - /home/ubuntu/docker/jenkins-data:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
```

```
  nginx:
    image: nginx
    container_name: nginx-saturi
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/docker/default.conf:/etc/nginx/conf.d/
      - /etc/letsencrypt/live:/etc/letsencrypt/live
      - /etc/letsencrypt/archive:/etc/letsencrypt/archive
networks:
  default:
    name: saturi_network
    driver: bridge
```

아까 적은 jenkins만 있는게 아니라 다른 mysql 등 또한 추가해줍니다. 특히 nginx의 마운트를 잘 확인하셔야 합니다. 5번에서 만들 default.conf를 ec2 어디든 만들어도 상관없지만 왼쪽에 본인의 default.conf가 어디 있는지를 꼭 잘 적어주셔야 합니다. 또한 발급받을 ssl 인증서의 위치도 적어줍니다.

▼ 6. Nginx 설정하기

```
server {
    listen 443 ssl;
    server_name i11d104.p.ssafy.io;

    location / {
        proxy_pass http://frontend-saturi:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }


    location /saturi-api/ {
        proxy_pass http://backend-saturi:8080;
```

```
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwa
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    location /saturi-ai/ {
        proxy_pass http://django-saturi:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwa
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }


    location ^~ /.well-known/acme-challenge/ {
        default_type "text/plain";
        root /var/www/letsencrypt;
    }

    ssl_certificate /etc/letsencrypt/live/i11d104.p.ssafy.i
    ssl_certificate_key /etc/letsencrypt/live/i11d104.p.ssa
}

server {
    listen 80;
    server_name i11d104.p.ssafy.io;

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

default.conf 파일을 위와 같이 작성해줍니다. SSL 인증서를 포함한 코드이므로 인증서를 발급받아야 합니다.

▼ 7. HTTPS 설정하기

SSL 인증서 발급

```
sudo apt update
sudo apt-get install letsencrypt -y

sudo apt upgrade -y
sudo apt install certbot python3-certbot-nginx

# certbot 실행 시뮬레이션
sudo certbot certonly --standalone --dry-run

# certbot 실행
sudo certbot certonly --standalone
```

letsencrypt 인증서를 발급받기 위해 certbot을 이용할 수 있습니다. 특히 저는 certbot의 standalone 방식으로 인증서를 발급받습니다. 본인의 도메인 명 등을 적어주면 간단하게 인증서를 발급받을 수 있습니다.

▼ 8. 각 프로젝트에 Dockerfile 작성하기

각 프로젝트 루트 디렉토리에 아래와 같이 작성해줍니다.

백엔드

```
FROM openjdk:17-jdk
LABEL maintainer="heo_dongwon@naver.com"
ARG JAR_FILE=build/libs/backend-0.0.1-SNAPSHOT.jar
ADD ${JAR_FILE} backend-springboot.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom
```

프론트엔드

```
# 가져올 이미지 정의
FROM node:20.15.0 as build-stage

# 경로 설정
WORKDIR /app

# package.json 워킹 디렉토리에 복사
COPY package*.json ./

# 의존성 설치
RUN npm install

# 결과물 복사
COPY . .

# 빌드하기
RUN npm run build

#FROM nginx:stable-alpine
#COPY --from=build-stage ./app/.next /usr/share/nginx/html/
#CMD ["nginx", "-g", "daemon off;"]

# 포트 노출
EXPOSE 3000

# 실행 명령어 정의
CMD ["npm", "start"]
```

Next.js를 사용하므로 nginx 이미지를 올리는 게 아니라 프론트엔드 자체를 컨테이너로 올려서 nginx로 프록싱해줍니다.

AI

```
# 베이스 이미지로 Python 3 사용
FROM python:3

# 작업 디렉토리 설정
```

```
WORKDIR /AIpjt

# 필요 패키지 복사 및 설치
COPY requirements.txt .
RUN pip install -r requirements.txt

# 소스 파일 복사
COPY . .

# 포트 8000 노출
EXPOSE 8000

# Django 서버 실행 명령어
CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
```