포팅매뉴얼

domain: k11a301.p.ssafy.io

▼ 사용 기술 및 버전

기술/서비스	버전	용도	포트
Nginx	nginx/1.18.0 (Ubuntu)	WAS	80 (외부공개)
Jenkins	2.462.2	CI/CD	8080 (외부공개)
Docker	27.2.1	컨테이너 관리	
Java	17	Jenkins	
Fastapi	0.115.5	형태소 분석서버	8001:8001
Fastapi	0.115.5	영상 보간 서버	8003:8003
Fastapi	0.115.5	유사어 검색 서버	8005:8005
Python	3.12	fastapi용	
MongoDB	3.6.8	단어 및 문장 URL 저장	8017:27017

▼ 계정

> MongoDB admin계정
>
> ```
> admin
> #ssafy1234
> ```

> Jenkins root계정
>
> ```
> headstone
> ```

```
#ssafy1234
```

▼ 서버 초기 설정

서버 환경 최신화

```
sudo apt update
```

방화벽(UFW) 설정

```
sudo ufw allow {사용할 포트 번호}
sudo ufw status 로 확인
```

▼ ufw 에 대한 간략한 설명

```
EC2의 ufw(우분투 방화벽)는 기본적으로 활성화(Enable) 상태이며
ssh 22번 포트만 접속 가능하게 되어 있음.

포트만 추가할 경우 6번부터 참고
처음부터 새로 세팅하는 경우 1번부터 참고


1. 처음 ufw 설정 시 실수로 ssh접속이 안되는 경우를 방지하기 위해 ssh 터미널을 여유있게 2~3개 연결해 놓는다.
$ ssh -i {pem 키 경로} {기본 사용자 계정 이름}@{public_ip}
ex) ssh -i ~/path/to/my-key.pem ubuntu@
```

```
nnn.nnn.nnn.nnn

ssh 연결 종료 시 exit 명령어 사용

2. ufw 상태 확인
$ sudo ufw status
Status : inactive

3. 사용할 포트 허용하기 (ufw inactive 상태)
$ sudo ufw allow 22

3-1 등록한 포트 조회하기 (ufw inactive 상태)
$ sudo ufw show added
Added user rules (see 'ufw status' for
running firewall):
ufw allow 22

4. ufw 활성화 하기
$ sudo ufw enable
Command may disrupt existing ssh connec
tions. Proceed with operation (y|n)? y

4.1 ufw 상태 및 등록된 rule 확인하기
$ sudo ufw status numbered
Status: active

     To                         Action
From
     --                         ------
----
[ 1] 22                         ALLOW I
```

```
N    Anywhere
[ 2] 22 (v6)                    ALLOW I
N    Anywhere (v6)

5. 새로운 터미널을 띄워 ssh 접속해 본다.
C:\> ssh -i 팀.pem ubuntu@팀.p.ssafy.io

6. ufw 구동된 상태에서 80 포트 추가하기
$ sudo ufw allow 80

6-1. 80 포트 정상 등록되었는지 확인하기
$ sudo ufw status numbered
Status: active

     To                         Action
From
     --                         ------
----
[ 1] 22                         ALLOW I
N    Anywhere
[ 2] 80                         ALLOW I
N    Anywhere
[ 3] 22 (v6)                    ALLOW I
N    Anywhere (v6)
[ 4] 80 (v6)                    ALLOW I
N    Anywhere (v6)

6-2. allow 명령을 수행하면 자동으로 ufw에 반영
되어 접속이 가능하다.

7. 등록한 80 포트 삭제 하기
```

```
$ sudo ufw status numbered
Status: active

     To                         Action
From
     --                         ------
----
[ 1] 22                         ALLOW I
N    Anywhere
[ 2] 80                         ALLOW I
N    Anywhere
[ 3] 22 (v6)                    ALLOW I
N    Anywhere (v6)
[ 4] 80 (v6)                    ALLOW I
N    Anywhere (v6)

7-1. 삭제할 80 포트의 [번호]를 지정하여 삭제하기
      번호 하나씩 지정하여 삭제한다.
$ sudo ufw delete 4
$ sudo ufw delete 2
$ sudo ufw status numbered  (제대로 삭제했
는지 조회해보기)
Status: active

     To                         Action
From
     --                         ------
----
[ 1] 22                         ALLOW I
N    Anywhere
[ 2] 22 (v6)                    ALLOW I
```

```
N    Anywhere (v6)

7-2 (중요) 삭제한 정책은 반드시 enable을 수행해야 적용된다.
$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y입력

기타
- ufw 끄기
$ sudo ufw disable
```

▼ **Docker 설치**

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

설치 준비

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture)
```

```
signed-by=/etc/apt/keyrings/docker.asc] ht
tps://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CO
DENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.
list > /dev/null
sudo apt-get update
```

설치

```
sudo apt-get install docker-ce docker-ce-c
li containerd.io docker-buildx-plugin dock
er-compose-plugin
```

▼ Docker 시스템 명령어

```
# 도커 상태 확인
sudo systemctl status docker

# 도커 엔진 시작
sudo systemctl start docker

# 도커 엔진 종료
sudo systemctl stop docker

# 자동 실행 설정
sudo systemctl enable docker
```

▼ 기본적인 Docker 명령어들

```
# 도커 이미지 확인
sudo docker images

# 도커 컨테이너 리스트 확인
```

```
sudo docker ps -a

# MySQL Docker 이미지 삭제
docker rmi (-f) mysql
-f 옵션 주면 이미지 지우면서 컨테이너 강제 삭제

# MySQL Docker 컨테이너 중지
(컨테이너는 여전히 Docker 내에 존재, 데이터 유지, 다시 실행 가능)
$ docker stop mysql-container

# MySQL Docker 컨테이너 삭제
(컨테이너 실행 중에는 삭제 불가, 선 중지 필요, 데이터 삭제)
$ docker rm mysql-container

# MySQL Docker 컨테이너 시작
$ docker start mysql-container

# MySQL Docker 컨테이너 재시작
$ docker restart mysql-container
```

▼ **Jenkins 설치**

> Jenkins를 위한 jdk17 설치
>
> ```
> sudo apt install fontconfig openjdk-17-jre
> ```

> Jenkins 설치
>
> ```
> sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
>   https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
> ```

```
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

Jenkins 실행 (관련 명령어들)

```
서버 부팅 시 jenkins 실행되게 만들기
sudo systemctl enable jenkins

jenkins 실행
sudo systemctl start jenkins

jenkins 상태 확인
sudo systemctl status jenkins
```

Unlock Jenkins, Install suggested plugins

```
초기 설정 위한 패스워드 확인
$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Create First Admin User

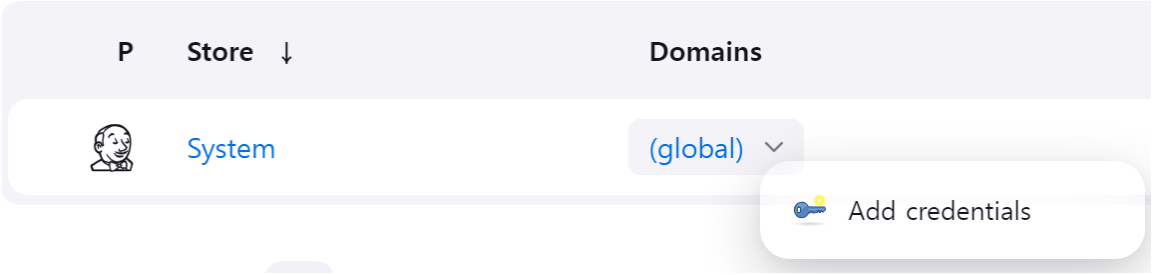
```
계정명, 비밀번호, 이름, 이메일 설정
```

> Instance Configuration
>
> ```
> http://{your_ip}:{jenkins_port}
> ```

▼ **Jenkins ↔ Gitlab 연동**

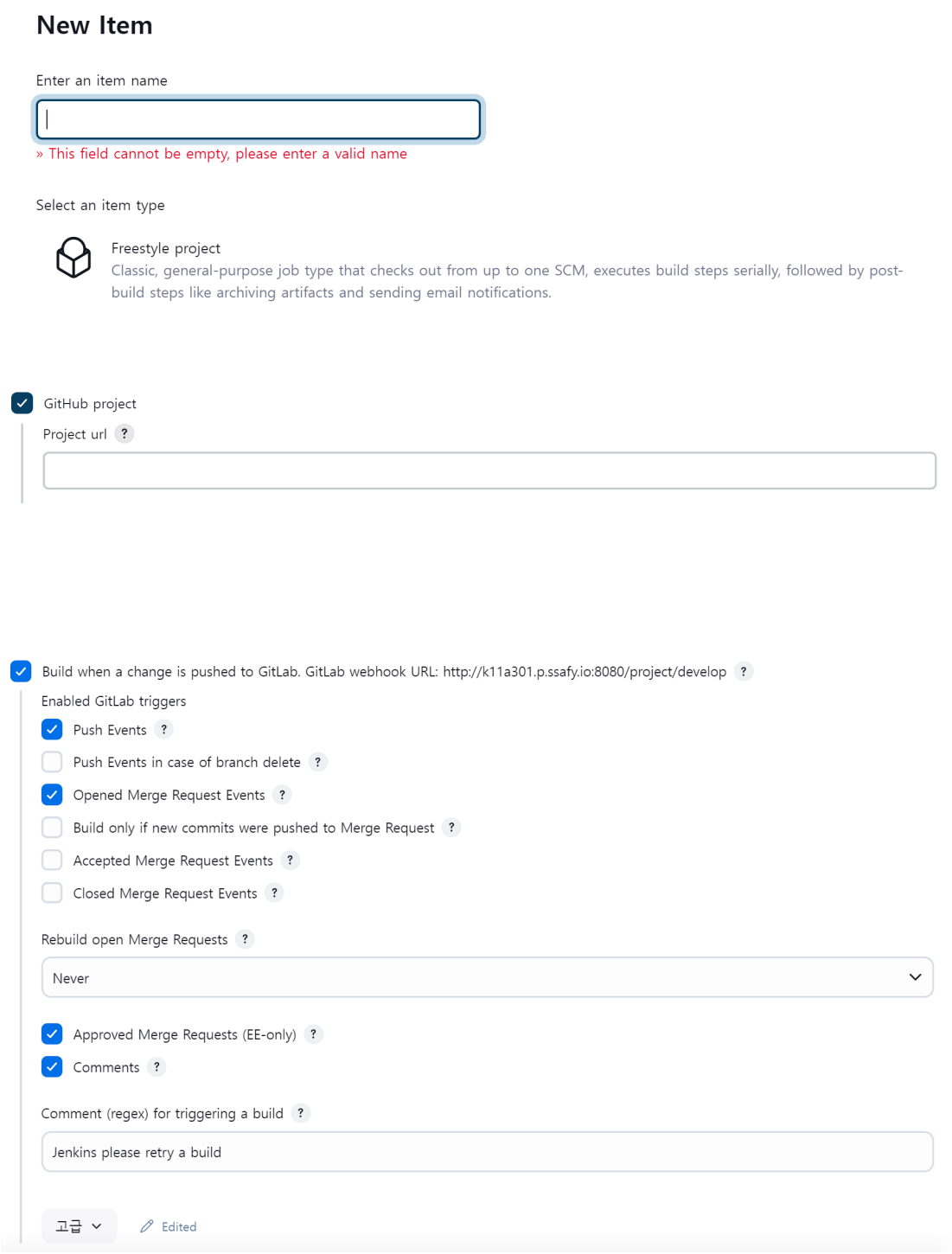
> Jenkins 에 GitLab 토큰 등록 |
> git fetch 등 원격 repository에 jenkins가 접근 시 사용할 로그인 정보를 추가해준다.

```
젠킨스 관리 → Security →  Credentials  → System → Add credentials
```



> credential 생성
>
> ```
> - kind : Username with password
> - Scope : Global (Jenkins, nodes, items, all child items, etc)
> - Username : GitLab 사용중인 아이디
> - Password : GitLab 에서 발급받은 사용자의 토큰 값
> - ID : 아무거나 이름짓기 (공백 시 자동 생성)
>
>  create 하면 credential 생성 완료
> ```

Jenkins에 새로운 아이템 생성



빌드 할 branch 선택

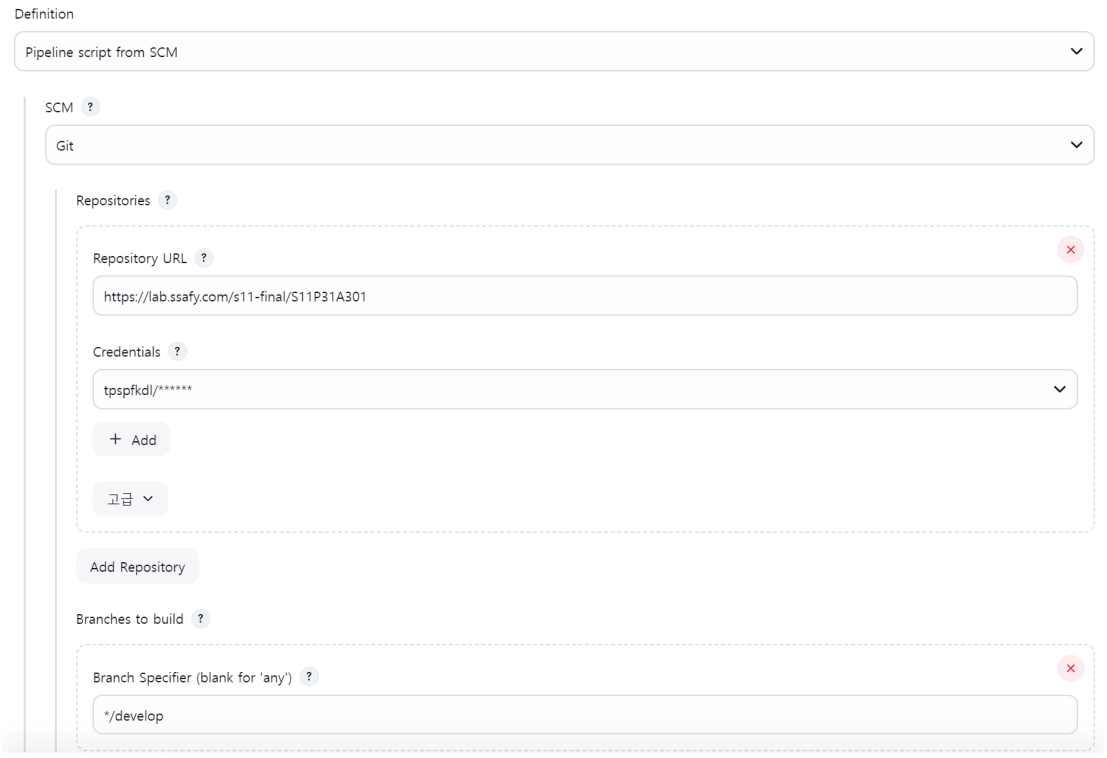
> 언제 선택한 branch를 빌드할 건지 설정 (해당 branch에 push되었을 때)
>
> - **저 url 중요하니까 복사해두기**
> - **조금 내리면 고급 - Secret token 에서 토큰도 발급하기**
>
> Approved Merge Requests (EE-only) 랑 Comments 도 체크 해제
>
> Comment (regex) for triggering a build 도 필요 없으니까 내용 지우기

> Gitlab 에서 hook 만들기
>
> ```
> Project - Settings - Webhooks - Add new webhook
> ```
>
> ```
> URL : 아까 위에서 중요하다 그랬던 URL
> Secret token : 아까 위에서 중요하다 그랬던 Secret token
> Trigger : 필요한 트리거 (push event)
> ```

▼ **Webhook으로 인한 요청 발생 시 폴더별로 Docker build 및 Docker run**

> 아마존 서버에 Git 설치
>
> ```
> sudo apt install git
> ```
>
> Jenkins 에게 경로를 알려준다.
> 이게 있어야 GIthub나 Gitlab 에서 프로젝트를 fetch 해 올 수 있다.

아래 Pipeline script from SCM을 설정해 준다.
Git같은 형상관리자로 Jenkinsfile을 관리하겠다는 의미이다.



파이프라인 작성

```
pipeline {
    agent any
    environment {
        DETERMINE_PATH = 'determine'
        ECCV_PATH = 'ECCV2022-RIFE'
        MORPHEME_PATH = 'morpheme'
    }

    stages {
        stage('Build and Run Determine') {
            when {
                changeset "${DETERMINE_PATH}/**" // determine 폴더에 변경 사항이 있을 때만 실행
            }
            steps {
                // Credentials 블록을 사용하여 ENV 파일을 환경변수로 가져오기
                withCredentials([
                    file(credentialsId: 'ENV', variable: 'ENV')
                ]) {
                    script {
                    // determine 컨테이너 중지 및 제거
                    sh 'docker stop determine_app || true && docker rm determine_app || true'
                    // 기존 determine_app 이미지 삭제
```

```
                    sh 'docker rmi determi
ne_app || true'
                    // determine 디렉토리에서
Docker 빌드 및 실행
                    dir(DETERMINE_PATH) {
                        // .env 파일 처리
                        sh '''
                        if [ -f .env ]; th
en
                            chmod u+w .env
                        fi
                        '''
                        // Jenkins에 저장한
파일 복사
                        sh 'cp ${ENV} .en
v'
                        sh 'docker build -
t determine_app .'
                        sh 'docker run -d
--name determine_app --env-file .env -p 80
01:8001 determine_app'
                        }
                    }
                }
            }
        }
        stage('Build and Run eccv') {
            when {
                changeset "${ECCV_PATH}/*
*" // ECCV2022-RIFE 폴더에 변경 사항이 있을 때만
실행
```

```
            }
            steps {
                // Credentials 블록을 사용하여
ENV 파일을 환경변수로 가져오기
                withCredentials([
                    file(credentialsId: 'E
NV', variable: 'ENV')
                ]) {
                    script {
                    // eccv_app 컨테이너 중지
및 제거
                    sh 'docker stop eccv_a
pp || true && docker rm eccv_app || true'
                    // 기존 eccv_app 이미지
삭제
                    sh 'docker rmi eccv_ap
p || true'
                    // sonnuri 디렉토리에서 D
ocker 빌드 및 실행
                    dir(ECCV_PATH) {
                        // .env 파일 처리
                        sh '''
                        if [ -f .env ]; th
en
                            chmod u+w .env
                        fi
                        '''
                        // Jenkins에 저장한
파일 복사
                        sh 'cp ${ENV} .en
v'
```

```
                        sh 'docker build -t eccv_app .'
                        sh 'docker run -d --name eccv_app --env-file .env -p 8003:8003 eccv_app'
                    }
                }
            }
        }
        stage('Build and Run Morpheme') {
            when {
                changeset "${MORPHEME_PATH}/**" // morpheme 폴더에 변경 사항이 있을 때만 실행
            }
            steps {
                // Credentials 블록을 사용하여 ENV 파일을 환경변수로 가져오기
                withCredentials([
                    file(credentialsId: 'ENV', variable: 'ENV')
                ]) {
                    script {
                    // morpheme_app 컨테이너 중지 및 제거
                    sh 'docker stop morpheme_app || true && docker rm morpheme_app || true'
                    // 기존 morpheme_app 이미지 삭제
```

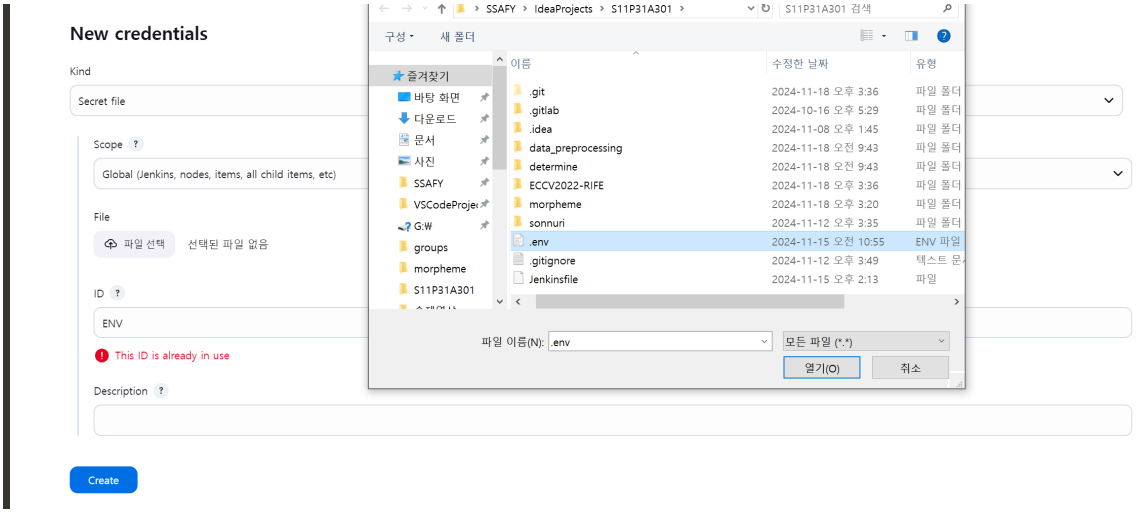
```
                    sh 'docker rmi morphem
e_app || true'
                    // morpheme 디렉토리에서
Docker 빌드 및 실행
                    dir(MORPHEME_PATH) {
                        // .env 파일 처리
                        sh '''
                        if [ -f .env ]; th
en
                            chmod u+w .env
                        fi
                        '''
                        // Jenkins에 저장한
파일 복사
                        sh 'cp ${ENV} .en
v'
                        sh 'docker build -
t morpheme_app .'
                        sh 'docker run -d
--name morpheme_app --env-file .env -p 800
5:8005 morpheme_app'
                        }
                    }
                }
            }
        }
    }

    post {
        success {
            echo 'Build, package, and cont
```

```
ainer run succeeded!'
        }
        failure {
            echo 'Build or container run f
ailed.'
        }
    }
}
```

여기서
withCredentials([
file(credentialsId: 'ENV', variable: 'ENV')
])
부분은 Jenkins에 있는 ENV라는 이름을 가진 Credentials, Secret file을 가져오겠다는 의미

이렇게 파이썬에서 쓸 .env파일을 젠킨스에 등록해준다.

- env파일을 이렇게 저장하는 이유: git에 올리면 큰일이 나버린다. AWS정보와 DB정보, OPENAI API KEY가 있는데, 노출되면 100만원^^



Dockerfile 예시

```
FROM python:3.12

# 작업 디렉토리 설정
WORKDIR /app

# requirements.txt와 main.py 복사
COPY . /app

RUN pip install -r requirements.txt

# FastAPI 애플리케이션 실행
EXPOSE 8001
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--por
t", "8001"]
```

Jenkinsfile에서 docker run을 하게 되면 실행 위치에 있는 Dockerfile을 실행한다. 이게 해당 코드로, 8001포트에 main.py를 app이라는 이름으로 0.0.0.0에 실행하겠다는 의미이다.

▼ MongoDB 설정

몽고디비 설치

```
sudo apt install -y mongodb
```

외부 접근 허용

- MongoDB 설정 파일 수정:

  ```
  sudo nano /etc/mongod.conf
  ```

  - `bindIp` 설정을 아래처럼 변경:

    ```
    yaml
    코드 복사
    bindIp: 0.0.0.0
    ```

  - 기본값 `127.0.0.1`은 로컬 연결만 허용하므로 외부에서 접근하려면 `0.0.0.0`으로 수정.

- MongoDB 서비스 시작:

  ```
  sudo systemctl start -d mongodb
  ```

- 싸피 환경이므로 port 변경:
- admin 계정 생성:

  ```
  mongo
  use admin
  db.createUser({
    user: "admin",
    pwd: "yourpassword",
    roles: [ { role: "userAdminAnyDatabase", db: "admi
  n" } ]
  })
  ```

- 인증 활성화 후 MongoDB 재시작:

```
bash
코드 복사
```

```
sudo nano /etc/mongod.conf
```

```
yaml
코드 복사
security:
  authorization: enabled
```

```
bash
코드 복사
sudo systemctl restart mongod
```

MongoDB 구성 파일 수정

MongoDB의 설정 파일을 열어 포트를 변경합니다.

```
bash
코드 복사
sudo nano /etc/mongod.conf
```

구성 파일에서 `net` **섹션**을 찾습니다. 기본적으로 이렇게 보일 것입니다:

```
yaml
코드 복사
net:
  port: 27017
  bindIp: 0.0.0.0
```

포트를 `8017`로 변경합니다:

```
yaml
코드 복사
net:
  port: 8017
  bindIp: 0.0.0.0
```

▼ Extension 설치

```
// master branch extension sonnuri 경로
// npm 설치 이후 pnpm을 설치한다.
npm i -g pnpm
// pnpm을 설치했으면, sonnuri 폴더에서 아래 명령어로 빌드한다.
pnpm build
// build 된 이후, 크롬에서 개발자 모드를 켠 뒤 압축 푼 확장 프로그램으
// 이후 익스텐션이 실행된다.
```