

DETECTO 포팅메뉴얼

본 문서는 웹으로 개발된 DETECTO 를 사용하기 위한 가이드를 분산 처리 환경의 리소스 제한으로 인해 Virtual 환경을 구축하는 과정, React, Spring Boot, FastApi, Kafka 를 배포하는 과정에 대해 서술하고 있습니다.

1. React 배포

1.1. React 개발 환경

- React 18.2
- Vite
- PWA

1.2. React 패키지

```
"scripts": {
  "dev": "vite",
  "build": "tsc && vite build",
  "preview": "vite preview",
  "lint": "eslint src/**/*.tsx",
  "prettier": "prettier --write ./src/**/*.{ts,tsx}"
},
"dependencies": {
  // CSS in JS 스타일링 라이브러리
  "@emotion/react": "^11.10.6",
  "@emotion/styled": "^11.10.6",
  // Material UI Component 라이브러리
  "@mui/icons-material": "^5.11.16",
  "@mui/material": "^5.12.0",
  "@mui/x-date-pickers": "^6.3.0",
  // 통신 라이브러리
  "axios": "^1.3.5",
  // 대시보드 페이지 차트 라이브러리
  "d3": "^7.8.4",
  "d3-cloud": "^1.2.5",
  "d3-color": "^3.1.0",
  "dayjs": "^1.11.7",
  // debounce 활용
  "lodash": "^4.17.21",
  // react 18.2
  "react": "^18.2.0",
  // 리액트 라우트 라이브러리
  "react-dom": "^18.2.0",
  "react-router-dom": "^6.10.0",
  // 상태 관리 라이브러리
  "recoil": "^0.7.7"
},
"devDependencies": {
  // 라이브러리 타입 지정
  "@types/d3": "^7.4.0",
  "@types/d3-cloud": "^1.2.5",
  "@types/lodash": "^4.14.194",
  "@types/react": "^18.0.27",
  "@types/react-dom": "^18.0.10",
  // es lint, prettier
  "@typescript-eslint/eslint-plugin": "^5.58.0",
  "@typescript-eslint/parser": "^5.58.0",
  "@vitejs/plugin-react": "^3.1.0",
  "eslint": "^8.38.0",
  "eslint-config-prettier": "^8.8.0",
  "prettier": "^2.8.7",
  "typescript": "^4.9.3",
  "vite": "^4.1.0"
},
// mock service worker
"msw": {
  "workerDirectory": "public"
}
```

1.3. React 배포과정

- Jenkins 에서 자동배포
- 하단 참고

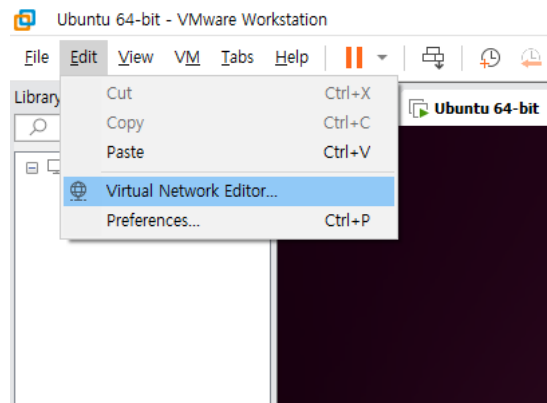
2. Virtual Machine 환경 구축 과정

2.1 과정

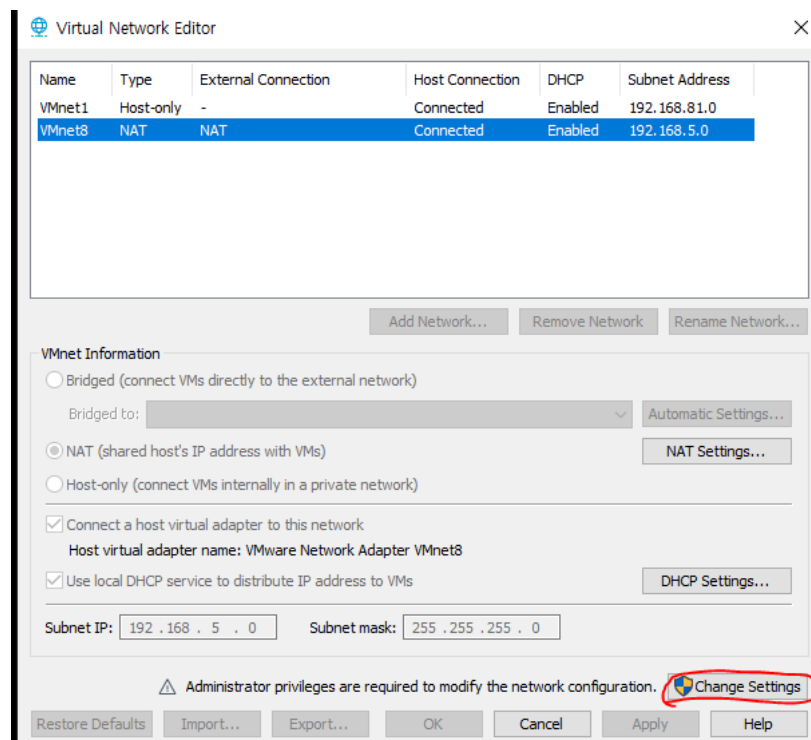
1. VMware Workstation 17 Pro 를 설치한다.
2. Ubuntu 22.04.2 LTS 의 iso를 받아 VMware에 설치한다.
3. 설치 후 다음 과정을 따른다.

▼ 네트워크 설정

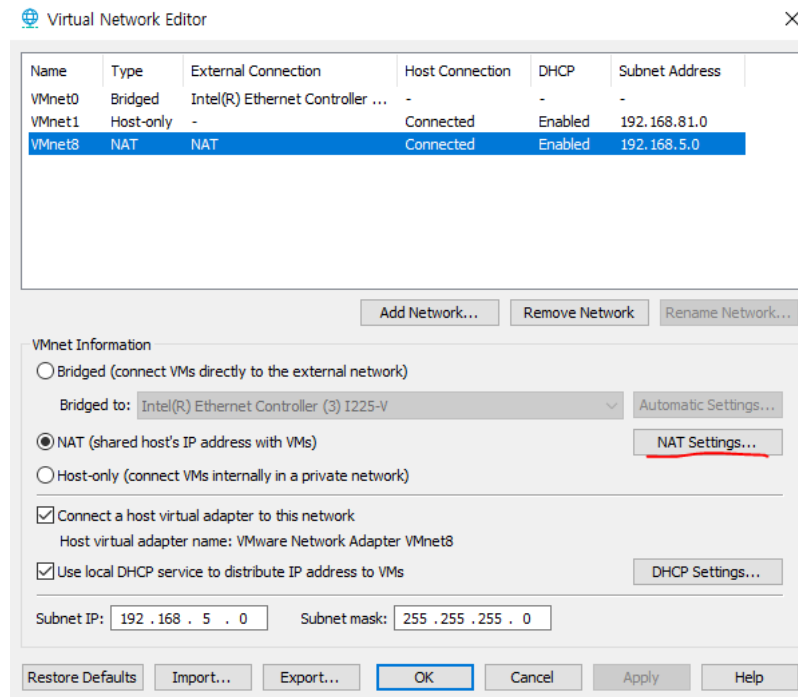
1. Edit - Virtual Network Editor 선택



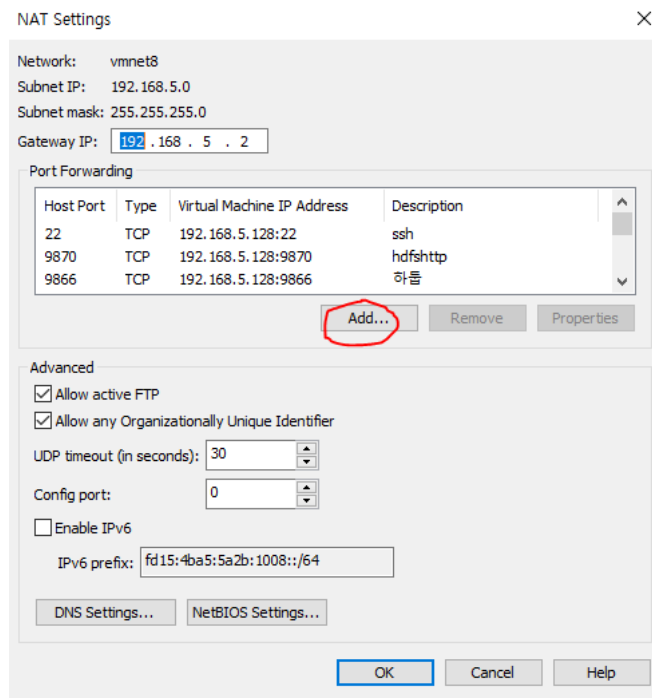
2. VMnet8 - Change Settings



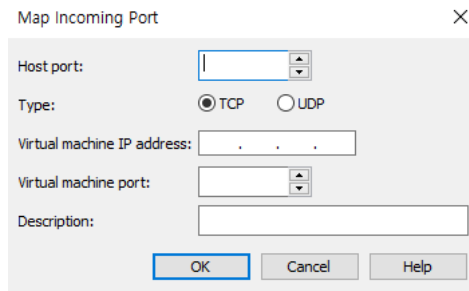
3. VMnet8 - NAT Settings...



4. Add...



5. 다음을 확인하여 적고 OK를 누른다.



Map Incoming Port dialog box with fields for Host port, Type (TCP/UDP), Virtual machine IP address, Virtual machine port, and Description. OK, Cancel, and Help buttons are at the bottom.

- a. Host port : 외부 접속 포트
 - b. Type : TCP 고정
 - c. Virtual machine IP address : 실제 인스턴스의 IP를 적는다.(192.168.5.128)
 - d. Virtual machine port : 인스턴스에서 외부로 연결 하고 싶은 포트를 적는다.
 - e. Description : 선택사항(설명 기재)
6. Rumeet 메뉴얼 (H-port, ip, V-port, dec)
- a. 22, 192.168.5.128, 22, ssh
 - b. 9870, 192.168.5.128, 9870, hdfshttp
 - c. 9866, 192.168.5.128, 9866, 하둡
 - d. 9864, 192.168.5.128, 9864, 데이터노드
 - e. 4040, 192.168.5.128, 4040, 스파크
 - f. 8080, 192.168.5.128, 8080, 스파크
 - g. 8000, 192.168.5.128, 8000, fastAPI
 - h. 8001, 192.168.5.128, 8001, fast
 - i. 8002, 192.168.5.128, 8002, fast

▼ Open SSH Server

```
sudo apt-get update
sudo apt-get install openssh-server

sudo vi /etc/ssh/sshd_config
- PermitRootLogin yes

sudo service ssh restart
```

▼ Hadoop

1. Hadoop 설치

```
sudo apt update
sudo apt install openjdk-17-jdk
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.3/hadoop-3.3.3.tar.gz
tar -xzf hadoop-3.3.3.tar.gz
sudo mv hadoop-3.3.3 /usr/local/hadoop
```

2. 환경 변수 설정

```
vi ~/.bashrc

맨 밑에
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

```
추가 후  
source ~/.bashrc
```

3. 파일 수정

- `cd $HADOOP_HOME/etc/hadoop/`

1. `hadoop-env.sh`

```
vi hadoop-env.sh  
  
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
```

2. `core-site.xml`

```
<configuration>  
  <property>  
    <name>fs.defaultFS</name>  
    <value>hdfs://0.0.0.0:9000</value>  
  </property>  
</configuration>
```

3. `yarn-site.xml`

```
<configuration>  
  
  <!-- Site specific YARN configuration properties -->  
  <property>  
    <name>yarn.nodemanager.aux-services</name>  
    <value>mapreduce_shuffle</value>  
  </property>  
  <property>  
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>  
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>  
  </property>  
  
</configuration>
```

4. `mapred.xml`

```
<configuration>  
  <property>  
    <name>mapreduce.framework.name</name>  
    <value>yarn</value>  
  </property>  
</configuration>
```

5. `hdfs namenode -format`

6. 키 생성 및 배포

```
ssh-keygen -t rsa  
ssh-copy-id rumeet@localhost  
sudo usermod -aG hadoop rumeet
```

7. 실행

```
$HADOOP_HOME/sbin/start-all.sh  
  
혹시 실행 안될시,  
  
unset HDFS_NAMENODE_USER  
unset HDFS_DATANODE_USER  
unset HDFS_SECONDARYNAMENODE_USER  
unset YARN_RESOURCEMANAGER_USER  
unset YARN_NODEMANAGER_USER  
  
export HDFS_NAMENODE_USER=rumeet  
export HDFS_DATANODE_USER=rumeet  
export HDFS_SECONDARYNAMENODE_USER=rumeet
```

```
export YARN_RESOURCEMANAGER_USER=rumeet
export YARN_NODEMANAGER_USER=rumeet
```

▼ Spark

1. Spark 설치

```
wget https://downloads.apache.org/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz
tar xvf spark-3.3.2-bin-hadoop3.tgz
```

2. 환경 변수 설정

```
vi ~/.bashrc

export SPARK_HOME=/spark-3.3.2-bin-hadoop3
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

source ~/.bashrc
```

3. 초기 설정

```
cd $SPARK_HOME/conf/
cp spark-env.sh.template spark-env.sh
vi spark-env.sh

SPARK_MASTER_HOST=192.168.5.128
SPARK_LOCAL_IP=192.168.5.128
SPARK_WORKER_CORES=6
SPARK_WORKER_MEMORY=8g
```

4. 실행

```
$SPARK_HOME/sbin/start-master.sh
$SPARK_HOME/sbin/start-workers.sh spark://localhost:7077
```

3. Spring Boot 배포 과정

3.1. 개발환경 및 배포 환경

개발 환경

- 'org.springframework.boot' version '3.0.4'
- 빌드 도구 : Gradle
- IDE : IntelliJ IDEA 2022.3.1

배포 환경

- 배포 서버 : Ubuntu 20.04.4 LTS (AWS)
- Docker : 20.10.23
- Gradle : Gradle 8.1-rc-2

3.2. API 서버활성을 위한 배포과정

- 기본적으로 CI/CD툴의 젠킨스를 이용한다.
- 위 내용은 9번(젠킨스를 통한 자동배포) 항목에서 자세히 설명한다.

4. FAST API 배포 과정

4.1. 개발환경 및 배포 환경

개발 환경

- 빌드 도구 : Python3.9
- IDE : PyCharm 2022.1.3

배포 환경

- 배포 서버 : uvicorn
 - 포트
 - 7005 : 소켓 서버
 - 구동
 - Docker 에서 대체
- ```
nohup uvicorn main:app --reload --host=0.0.0.0 --port=7005 &
```

## 4.2. API 서버활성을 위한 배포

▼ 세팅

1. Python3를 설치한다

```
apt-get update
apt-get install python3
apt-get install pip
```

2. pip를 활용하여 FastApi와 Uvicorn을 다운한다

```
pip install fastapi
pip install uvicorn
```

## 5. Kafka

### 5.1. 배포 환경

배포 환경

- Ubuntu 20.0.4

### 5.2. 개발환경 및 배포 환경

```
wget https://dldcn.apache.org/kafka/3.3.2/kafka_2.12-3.3.2.tgz
tar xvf kafka_2.12-3.3.2.tgz
bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
bin/kafka-server-start.sh -daemon config/server.properties
```

### 5.3. 토픽 생성

```
bin/kafka-topics.sh \
--create \
--bootstrap-server my-kafka:9092 \
--topic cctv.0.23 \
--partitions 365
```

## 6. Flask

## 6.1. 배포 환경

개발 환경

- 빌드 도구 : Python3.9
- IDE : PyCharm 2022.1.3

배포 환경

- Window 10

## 6.2. 패키지

- pip install ultralytics
- pip install kafka-python
- pip install pika
- pip install mysql-connector
- pip install flask
- pip install flask-sqlalchemy

# 7. Jenkins를 통한 자동배포

## 7.1. 초기 세팅

도커를 통하여 Jenkins를 설치한다.

- <https://dongle94.github.io/docker/docker-ubuntu-install/>
- 해당 사이트를 참고하여 도커 설치한다.
- 설치 후 다음 명령어를 통하여 Jenkins를 설치한다.

도커에 Ansible을 설치한다.

- 명령어를 통하여 Ansible을 설치한다.

```
docker run -itd --name ansible-server -p 20022:22 -e container=docker --tmpfs /run --tmpfs /tmp -v /sys/fs/cgroup:/sys/fs/cgroup:ro -v
```

Jenkins 플러그인을 설치한다.

- Ansible
- GitLab
- Publish Over SSH

Jenkins에 Ansible 컨테이너를 연결한다.



**Publish over SSH**

Jenkins SSH Key ?

Passphrase ?

Concealed

Path to key ?

Key ?

☐ Disable exec ?

SSH Servers

SSH Server Name ?

ansible-server

Hostname ?

172.26.5.131

Username ?

root

Remote Directory ?

.

## ▼ 젠킨스 설정

- JDK

### JDK

#### JDK installations

List of JDK installations on this system

Add JDK

#### JDK

##### Name

jdk11.0.18

##### JAVA\_HOME

/opt/java/openjdk

☐ Install automatically ?

- Gradle

Add Gradle

#### Gradle name ?

gradle

☒ Install automatically ?

#### Install from Gradle.org

##### Version

Gradle 8.1-rc-2

Add Installer ▾

## GitLab Webhook 등록

- 젠킨스 프로젝트 설정에서 Build Triggers에 webhook을 등록한다.
- Push Events을 체크한다.

### Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://i8d111.p.ssafy.io:8080/project/Back-End-Project> ?

#### Enabled GitLab triggers

- ☒ Push Events
- ☐ Push Events in case of branch delete
- ☐ Opened Merge Request Events
- ☐ Build only if new commits were pushed to Merge Request ?
- ☐ Accepted Merge Request Events
- ☐ Closed Merge Request Events

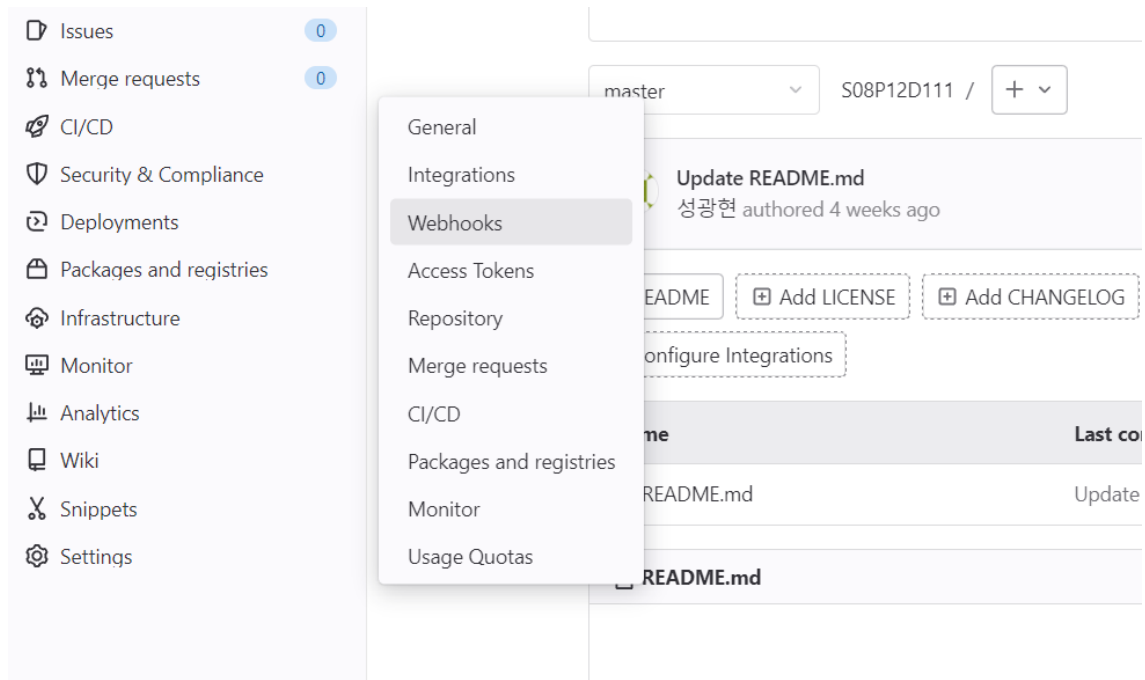
Rebuild open Merge Requests

- Secret Key를 발급한다.

Secret token ?

124c03 [REDACTED] 6bb2

- <https://lab.ssafy.com/s08-bigdata-recom-sub2/S08P22D204>에서 Settings의 Webhooks를 누른다.



- Webhooks 에 Jenkins URL을 적고 Push events에 be/develop 그리고 Jenkins에서 발급받은 Secret token을 적는다.

Q Search page

### Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

**URL**

URL must be percent-encoded if it contains one or more special characters.

**Secret token**

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

**Trigger**

☒ Push events

Push to the repository.

☐ Tag push events  
A new tag is pushed to the repository.

☐ Comments  
A comment is added to an issue or merge request.

☐ Confidential comments  
A comment is added to a confidential issue.

☐ Issues events  
An issue is created, updated, closed, or reopened.

- Add Webhooks 를 눌러 GitLab에 등록한다.

### SSL verification

- ☒ Enable SSL verification

Add webhook

## 7.3. PipeLine 설정

```
pipeline {
 agent any

 environment {
 TARGET_BRANCH = 'develop'
 BE_FOLDER = 'BE'
 FE_FOLDER = 'FE'
 FAST_FOLDER = 'Fast'
 }

 tools{
 gradle "Gradle7.6.1"
 nodejs "NodeJS18.12.1"
 }

 stages {

 stage('clone') {
 steps {
 git branch: 'develop', credentialsId: 'seok', url: 'https://lab.ssafy.com/s08-final/S08P31D201.git'
 }
 }

 stage('folder changes') {
 steps {
 script {

 def beFolder = env.BE_FOLDER
 def feFolder = env.FE_FOLDER
 def fastFolder = env.FAST_FOLDER

 // 변경된 파일목록을 가져옵니다.
 def changes = sh(returnStdout: true, script: "git diff --name-only HEAD HEAD~1").trim().split('\n')

 //특정 폴더(changedFolder)의 파일에 변경 사항이 있는지 확인합니다.
 boolean beChanged = changes.any { it.startsWith(beFolder) }
 boolean feChanged = changes.any { it.startsWith(feFolder) }
 boolean fastChanged = changes.any { it.startsWith(fastFolder)}

 if (beChanged) {
 env.beChanged = 'true'
 } else {
 env.beChanged = 'false'
 }

 if (feChanged) {
 env.feChanged = 'true'
 } else {
 env.feChanged = 'false'
 }

 if(fastChanged){
 env.fastChanged = 'true'
 } else {
 env.fastChanged = 'false'
 }
 }
 }
 }

 stage('BE Build') {
 when {
 expression { env.beChanged == 'true' }
 }
 steps {
 sh '''cd BE
 chmod +x gradlew
 ./gradlew bootJar'''
 }
 }

 stage('BE SSH') {
 when {
 expression { env.beChanged == 'true' }
 }
 steps {
```

```

 sshPublisher(publishers: [sshPublisherDesc(configName: 'ansible-server', transfers: [sshTransfer(cleanRemote: false, ex
 })
}

stage('FE Build') {
 when {
 expression { env.feChanged == 'true' }
 }
 steps {
 sh '''echo build start
 cd FE/S08P31D201
 npm install
 npm run build
 tar cvf dist.tar dist'''
 }
}

stage('FE SSH') {
 when {
 expression { env.feChanged == 'true' }
 }
 steps {
 sshPublisher(publishers: [sshPublisherDesc(configName: 'ansible-server', transfers: [sshTransfer(cleanRemote: false, ex
 })
}

stage('Fast Build') {
 when {
 expression { env.fastChanged == 'true' }
 }
 steps {
 sh '''
 cd Fast
 tar -cvf fast.tar main.py requirements.txt
 '''
 }
}

stage('Fast SSH') {
 when {
 expression { env.fastChanged == 'true' }
 }
 steps {
 sshPublisher(publishers: [sshPublisherDesc(configName: 'ansible-server', transfers: [sshTransfer(cleanRemote: false, ex
 })
}
}
}
}

```

## 7.4. Ansible 설정

### ▼ 파일 목록

- Infra의 Ansible 서버

```

[root@b2b8a7f38df1 ~]# ls
anaconda-ks.cfg anaconda-post.log Dockerfile hosts original-ks.cfg playbook.yml
[root@b2b8a7f38df1 ~]#

```

- Dockerfile

Back-End

```

FROM openjdk:17-jdk-slim

WORKDIR /app

COPY ./detecto.jar /app/detecto.jar

CMD ["java", "-jar", "/app/detecto.jar"]

```

FastAPI

```

FROM tiangolo/uvicorn-gunicorn-fastapi:python3.9

앱 디렉터리로 설정
WORKDIR /app

RUN apt-get update && apt-get install -y libgl1-mesa-glx

필요한 패키지 설치
ADD fast.tar .
RUN pip install -r requirements.txt

컨테이너 포트 설정
EXPOSE 7005

컨테이너 실행 시 실행될 명령어
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "7005"]

```

## • Playbook.yml

### Back-End

```

- hosts: all
become: true

tasks:
- name: stop current running container
 command: docker stop backend
 ignore_errors: yes

- name: remove stopped container
 command: docker rm backend
 ignore_errors: yes

- name: remove current docker image
 command: docker rmi ckd-backend-ansible
 ignore_errors: yes

- name: build a docker image with deployed jar file
 command: docker build -t ckd-backend-ansible .
 args:
 chdir: /root

- name: create a container using ckd-project-ansible image
 command: docker run -d --name backend -p 8000:7000 ckd-backend-ansible

```

### FastAPI

```

- hosts: all
become: true

tasks:
- name: remove current running container
 command: docker stop fast
 ignore_errors: yes

- name: remove stopped container
 command: docker rm fast
 ignore_errors: yes

- name: remove current docker image
 command: docker rmi ckd-fast-ansible
 ignore_errors: yes

- name: build a docker image with deployed tar file
 command: docker build -t ckd-fast-ansible -f Dockerfile2 .
 args:
 chdir: /root

- name: create a container using ckd-project-ansible image
 command: docker run -d --name fast -p 7005:7005 ckd-fast-ansible

```

### Front-End

```

- hosts: all
 tasks:
 - name: Unarchive dist.tar in the ansible-server container
 ansible.builtin.shell: "docker exec ansible-server sh -c 'tar -xvf /root/dist.tar -C /etc/test'"

 - name: Copy dist folder from ansible-server container to the host
 ansible.builtin.shell: "docker cp ansible-server:/etc/test/dist /tmp/dist"

 - name: Copy dist folder from the host to nginx container
 ansible.builtin.shell: "docker cp /tmp/dist nginx2:/var/www/frontend"

 - name: Remove dist folder from the host
 ansible.builtin.file:
 path: /tmp/dist
 state: absent

```

## • Nginx & SSL

```

//Nginx & SSL 설정

// docker로 Nginx 설치
docker pull nginx

// Nginx와 mount할 디렉토리 생성
mkdir nginx_home

// Let's Encrypt 설치
apt-get install letsencrypt

// 인증서 적용 및 .pem 키 발급
letsencrypt certonly --standalone -d [도메인]

//Nginx 도커 container 실행
docker run -d --name nginx --network host -v home/ubuntu/nginx_home:/var/nginx_home -v /etc/letsencrypt:/etc/letsencrypt -v /

// nginx container 들어가기
sudo docker exec -it --user root nginx /bin/bash

// 디렉토리 생성
cd /etc/nginx
mkdir cd sites-enabled
cd sites-enabled

// config.conf 파일 생성
vi config.conf

server {
 location /{
 root /var/nginx_home;
 try_files $uri $uri/ /index.html;
 }
 location /api {
 proxy_pass http://localhost:8888/api;
 }

 listen 443 ssl;
 ssl_certificate /etc/letsencrypt/live/[도메인]/fullchain.pem;
 ssl_certificate_key /etc/letsencrypt/live/[도메인]/privkey.pem;
}

server {
 if ($host = [도메인]){
 return 301 https://$host$request_uri;
 }

 listen 80;
 server_name [도메인];
 return 404;
}

// nginx 폴더의 default.conf에다 include

// nginx 컨테이너 안에서 reload
nginx -s reload

```

nginx container의 etc/nginx/sites-enabled/config.conf

```

server {
 listen 80;
 server_name detecto.kr;
 return 301 https://$host$request_uri;
}

server {
 listen 443 ssl;
 server_name detecto.kr;

 ssl_certificate /etc/letsencrypt/live/detecto.kr/fullchain.pem;
 ssl_certificate_key /etc/letsencrypt/live/detecto.kr/privkey.pem;

 location /fast {
 proxy_pass http://localhost:7005;
 proxy_http_version 1.1;
 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:8000;
 }

 location / {
 root /var/www/frontend/dist;
 try_files $uri $uri/ /index.html =404;
 }
}

```

- 1행 hosts에 명시된 모든 도커 컨테이너에 적용
- 2행 작업 실행
  - 1번 작업 detecto 컨테이너 종료 (에러 무시)
  - 2번 작업 현재 detecto의 이름을 가진 컨테이너 삭제 (에러 무시)
  - 3번 작업 detecto의 이름을 가진 이미지 삭제 (에러 무시)
  - 4번 작업 Dockerfile의 설정에 따라 detecto라는 이름으로 이미지 생성
  - 5번 작업 detecto의 이미지를 백그라운드로 80포트를 사용하여 컨테이너 실행
- hosts

```
172.17.0.3
```