포팅 메뉴얼

1. 개발 환경

1.1 Frontend

- Node.js 20.11.0 (LTS)
- Vue 3

1.2 Backend

- Java 17
- Spring Boot 3.2.2
- Spring Data JPA
- Lombok

1.3 Database

- MySQL 8.0.34

1.4 UI/UX

- Figma

1.5 Server

- 배포 서버
    - AWS EC2 1대
    - Ubuntu 22.04 LTS
    - Docker 25.0.4
    - Docker Compose 2.24.7
- 클러스터 서버
    - AWS EC2 5대

- Ubuntu 20.04 LTS
- Java 8
- Hadoop 3.2.3
- Spark 3.2.1
- Zookeeper 3.8.0
- Zeppelin 0.10.1

1.6 형상 / 이슈관리

- Gitlab
- Jira

2. 환경변수

2.1 Backend - Spring Boot

> server.servlet.context-path
> spring.datasource.hikari.driver-class-name
> spring.datasource.url
> spring.datasource.hikari.username
> spring.datasource.hikari.password

2.2 Backend - Fast API

> SQLALCHEMY_DATABASE_URL

3. 배포 서버 세팅

3-1 Docker 설치

```
# 1. Set up Docker's apt repository

# 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] https://download.dock
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# 2. Install the Docker packages.
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

3-2. Docker Compose 설치

```
# Install the Compose plugin

# 1. Update the package index, and install the latest version of Docker Compose
sudo apt-get update
sudo apt-get install docker-compose-plugin
```

```
# 2. Verify that Docker Compose is installed correctly by checking the version.
docker compose version
```

3-3. 컨테이너 구축

- docker-compose.yml

```yaml
version: "3.0"

services:
  nginx:
    ports:
      - "80:80"
      - "443:443"
    networks:
      - appnet
    image: nginx

  mysql:
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - appnet
    environment:
      MYSQL_ROOT_PASSWORD:
      MYSQL_DATABASE:
      MYSQL_USER:
      MYSQL_PASSWORD:
    command: --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
    image: mysql:8.0.34

  jenkins:
    build: .
    ports:
      - "8080:8080"
    networks:
      - jenkinsnet
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock

  portainer:
    ports:
      - "9443:9443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data
    image: portainer/portainer-ce:latest

volumes:
  mysql_data: {}
  portainer_data: {}

networks:
  jenkinsnet: {}
  appnet: {}
```

- Dockerfile

```
FROM jenkins/jenkins:jdk17
USER root

RUN apt-get update && \
```

```
    apt-get -y install apt-transport-https \
      ca-certificates \
      curl \
      gnupg2 \
      software-properties-common && \
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-
    add-apt-repository \
      "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
      $(lsb_release -cs) \
      stable" && \
   apt-get update && \
   apt-get -y install docker-ce

RUN groupadd -f docker
RUN usermod -aG docker jenkins
RUN chown root:docker /var/run/docker.sock
```

3.4 Nginx proxy 설정 + SSL 인증서 발급

```
# 1. 도메인 구매, 등록

# 2. certbot install
sudo snap install certbot --classic

# 3. SSL 발급 및 적용
cd /etc/nginx/conf.d
sudo certbot --nginx -d {도메인}

# 4. 설정 파일 수정 (프록시 설정)
vi /etc/nginx/conf.d/default.conf
########

server {
    server_name  도메인명

    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        proxy_pass http://vue:80;
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

        ### springboot 한번 생성 되면 주석 해제
    # location /api {
       # proxy_pass http://springboot:8080;
    # }

        ### fastapi 컨테이너 한번 생성되면 주석 해제
    # location /fapi {
        # proxy_pass http://fastapi:80;
    # }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
```

```
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/choonong.store/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/choonong.store/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 {
    if ($host = www.choonong.store) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    if ($host = choonong.store) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    listen       80;
    listen  [::]:80;
    server_name  choonong.store www.choonong.store;
    return 404; # managed by Certbot
}
```

4. CI / CD

- DooD(Docker out of Docker) 방식 채택
- GitLab WebHooks 설정으로 자동 빌드, 배포 수행

4.1 Backend PipeLine (Spring Boot)

- jenkins pipeline

```
pipeline {
    agent any

    stages {
        stage('Clone Repository') {
            steps {
                script {
```

```
                    git branch: 'back_release', credentialsId: 'gitlab-access-token', url: 'gitlab URL.
                }
            }
        }

        stage('Build') {
            steps {
                dir('backend/choonong') {
                    sh 'chmod +x gradlew'
                    sh './gradlew clean build -x test'
                }
            }
        }

        stage("Copy") {
            steps {
                dir('backend/choonong/build/libs') {
                    sh 'cp choonong-0.0.1-SNAPSHOT.jar /root/springboot/app.jar'
                }
            }
        }

        stage("Deploy") {
            steps {
                dir('/root/springboot') {
                    sh 'docker compose down'
                    sh 'docker compose build'
                    sh 'docker compose up -d --build'
                }
            }
        }
    }
}
```

- docker-compose.yml

```
version: "3.0"

services:
  springboot:
    build: .
    networks:
      - docker-compose_appnet

networks:
  docker-compose_appnet:
    external: true
```

- Dockerfile

```
FROM openjdk:17

WORKDIR /

COPY application.properties .

COPY app.jar .

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

- application.properties (스프링 서버 환경 변수)

```
# context-path
server.servlet.context-path=/api
```

```
# Mysql DataSource Configuration
spring.datasource.hikari.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url
spring.datasource.hikari.username
spring.datasource.hikari.password
```

4.2 Backend PipeLine (Fast API)

- jenkins pipeline

```
pipeline {
    agent any

    stages {
        stage('Clone Repository') {
            steps {
                script {
                    git branch: 'fastapi', credentialsId: 'gitlab-access-token', url: 'gitlab URL.git'
                }
            }
        }

        stage('Copy') {
            steps {
                dir('fast_api') {
                    sh ' if [ -d "/root/fastapi/app" ]; then rm -rf /root/fastapi/app; fi'
                    sh 'cp -r . /root/fastapi/app'
                }
            }
        }

        stage('Deploy') {
            steps {
                dir('/root/fastapi') {
                    sh 'docker compose down'
                    sh 'docker compose up -d --build'
                }
            }
        }
    }
}
```

- docker-compose.yml

```
version: "3.0"

services:
  fastapi:
    build: .
    networks:
      - docker-compose_appnet

networks:
  docker-compose_appnet:
    external: true
```

- Dockerfile

```
FROM python:3.12

WORKDIR /app

COPY app/requirements.txt .
```

```
RUN pip install --no-cache-dir --upgrade -r ./requirements.txt

COPY .env .

COPY ./app .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80"]
```

- .env (Fast API 서버 환경 변수)

```
SQLALCHEMY_DATABASE_URL
```

4.3 Frontend PipeLine (Vue)

- jenkins pipeline

```
pipeline {
    agent any

    tools {
        nodejs "nodeJS"
    }

    stages {
        stage('Clone Repository') {
            steps {
                script {
                    git branch: 'front_release', credentialsId: 'gitlab-access-token', url: 'gitlab URL
                }
            }
        }

        stage('Env copy') {
            steps {
                dir('frontend') {
                    sh 'cp /root/vue/.env .'
                }
            }
        }

        stage('Install Dependencies') {
            steps {
                dir('frontend') {
                    sh 'npm install'
                }
            }
        }

        stage('Build') {
            steps {
                dir('frontend') {
                    sh 'npm run build'
                }
            }
        }

        stage('Copy') {
            steps {
                dir('frontend') {
                    sh ' if [ -d "/root/vue/dist" ]; then rm -rf /root/vue/dist; fi'
                    sh 'cp -r dist /root/vue'
                }
            }
```

```
        }

        stage('Deploy') {
            steps {
                dir('/root/vue') {
                    sh 'docker compose down'
                    sh 'docker compose build'
                    sh 'docker compose up -d --build'
                }
            }
        }
    }
}
```

- docker-compose.yml

```
version: "3.0"

services:
  vue:
    build: .
    networks:
      - docker-compose_appnet

networks:
  docker-compose_appnet:
    external: true
```

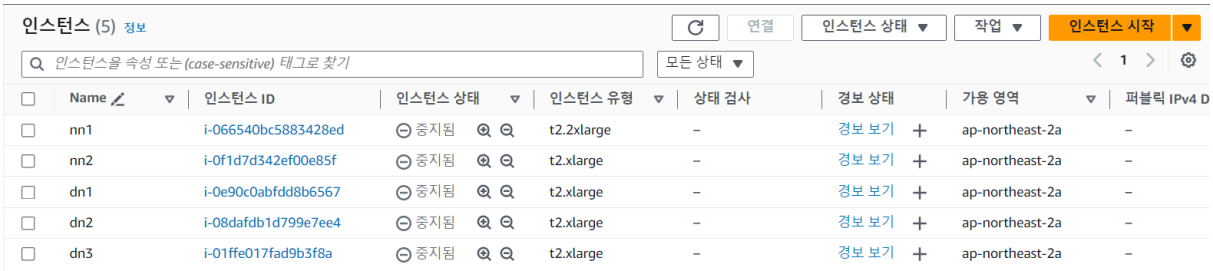
- Dockerfile

```
FROM nginx

COPY dist/. /usr/share/nginx/html
```

5. 클러스터 구축

5-1. EC2 생성

- 인스턴스 유형 : t2.xlarge
- 스토리지 : 2GiB
- 네트워크 보안 규칙 생성



5-2. Java 설치

```
# 업데이트 목록 갱신
sudo apt-get -y update
# 현재 패키지 업그레이드
sudo apt-get -y upgrade
# 신규 업데이트 설치
sudo apt-get -y dist-upgrade
# 필요 라이브러리 설치
sudo apt-get install -y vim wget unzip ssh openssh-* net-tools
```

```
# Java 8 설치
sudo apt-get install -y openjdk-8-jdk
# Java 버전 확인
java -version
# Java 경로 확인
sudo find / -name java-8-openjdk-amd64 2>/dev/null
# /usr/lib/jvm/java-8-openjdk-amd64

# Java 시스템 환경변수 등록 및 활성화
sudo vim /etc/environment

# 아래 내용 추가 후 저장
PATH 뒤에 ":/usr/lib/jvm/java-8-openjdk-amd64/bin" 추가
JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"

# 시스템 환경변수 활성화
source /etc/environment

# 사용자 환경변수 등록
sudo echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64' >> ~/.bashrc

# 사용자 환경변수 활성화
source ~/.bashrc
```

5-3. 하둡 설치

- 하둡 설치

```
# 설치파일 관리용 디렉토리 생성
sudo mkdir /install_dir && cd /install_dir
# Hadoop 3.2.2 설치
sudo wget https://dlcdn.apache.org/hadoop/common/hadoop-3.2.3/hadoop-3.2.3.tar.gz
# Hadoop 3.2.2 압축 해제
sudo tar -zxvf hadoop-3.2.3.tar.gz -C /usr/local
# Hadoop 디렉토리 이름 변경
sudo mv /usr/local/hadoop-3.2.3 /usr/local/hadoop

# Hadoop 시스템 환경변수 설정
sudo vim /etc/environment

# 아래 내용 추가 후 저장
PATH 뒤에 ":/usr/local/hadoop/bin" 추가
PATH 뒤에 ":/usr/local/hadoop/sbin" 추가
HADOOP_HOME="/usr/local/hadoop"

# 시스템 환경변수 활성화
source /etc/environment

# Hadoop 환경 사용자 환경변수 설정
sudo echo 'export HADOOP_HOME=/usr/local/hadoop' >> ~/.bashrc
sudo echo 'export HADOOP_COMMON_HOME=$HADOOP_HOME' >> ~/.bashrc
sudo echo 'export HADOOP_HDFS_HOME=$HADOOP_HOME' >> ~/.bashrc
sudo echo 'export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop' >> ~/.bashrc
sudo echo 'export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop' >> ~/.bashrc
sudo echo 'export HADOOP_YARN_HOME=$HADOOP_HOME' >> ~/.bashrc
sudo echo 'export HADOOP_MAPRED_HOME=$HADOOP_HOME' >> ~/.bashrc

# 사용자 환경변수 활성화
source ~/.bashrc
```

- hdfs-site.xml 편집

```
# hdfs-site.xml 편집
sudo vim $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

```
# 아래 내용으로 수정 후 저장
<configuration>
    <!-- configuration hadoop -->
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>/usr/local/hadoop/data/nameNode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>/usr/local/hadoop/data/dataNode</value>
    </property>
    <property>
        <name>dfs.journalnode.edits.dir</name>
        <value>/usr/local/hadoop/data/dfs/journalnode</value>
    </property>
    <property>
        <name>dfs.nameservices</name>
        <value>my-hadoop-cluster</value>
    </property>
    <property>
        <name>dfs.ha.namenodes.my-hadoop-cluster</name>
        <value>namenode1,namenode2</value>
    </property>
    <property>
        <name>dfs.namenode.rpc-address.my-hadoop-cluster.namenode1</name>
        <value>nn1:8020</value>
    </property>
    <property>
        <name>dfs.namenode.rpc-address.my-hadoop-cluster.namenode2</name>
        <value>nn2:8020</value>
    </property>
    <property>
        <name>dfs.namenode.http-address.my-hadoop-cluster.namenode1</name>
        <value>nn1:50070</value>
    </property>
    <property>
        <name>dfs.namenode.http-address.my-hadoop-cluster.namenode2</name>
        <value>nn2:50070</value>
    </property>
    <property>
        <name>dfs.namenode.shared.edits.dir</name>
        <value>qjournal://nn1:8485;nn2:8485;dn1:8485/my-hadoop-cluster</value>
    </property>
    <property>
        <name>dfs.client.failover.proxy.provider.my-hadoop-cluster</name>
        <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
    </property>
    <property>
        <name>dfs.ha.fencing.methods</name>
        <value>shell(/bin/true)</value>
    </property>
    <property>
        <name>dfs.ha.fencing.ssh.private-key-files</name>
        <value>/home/ubuntu/.ssh/id_rsa</value>
    </property>
    <property>
        <name>dfs.ha.automatic-failover.enabled</name>
        <value>true</value>
    </property>
```

```
    <property>
        <name>dfs.name.dir</name>
        <value>/usr/local/hadoop/data/name</value>
    </property>
    <property>
        <name>dfs.data.dir</name>
        <value>/usr/local/hadoop/data/data</value>
    </property>
</configuration>
```

- core-site.xml 편집

```
# core-site.xml 편집
sudo vim $HADOOP_HOME/etc/hadoop/core-site.xml

# 아래 내용으로 수정 후 저장
<configuration>
        <property>
                <name>fs.default.name</name>
                <value>hdfs://nn1:9000</value>
        </property>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://my-hadoop-cluster</value>
        </property>
        <property>
                <name>ha.zookeeper.quorum</name>
                <value>nn1:2181,nn2:2181,dn1:2181</value>
        </property>
</configuration>
```

- yarn-site.xml 편집

```
# Hadoop yarn-site.xml 파일 설정
sudo vim $HADOOP_HOME/etc/hadoop/yarn-site.xml

# 아래 내용으로 수정 후 저장
<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>
        <property>
                <name>yarn.resourcemanager.hostname</name>
                <value>nn1</value>
        </property>
        <property>
                <name>yarn.nodemanager.vmem-check-enabled</name>
                <value>false</value>
        </property>
</configuration>
```

- mapred-stie.xml 편집

```
# Hadoop mapred-site.xml 파일 설정
sudo vim $HADOOP_HOME/etc/hadoop/mapred-site.xml

# 아래 내용으로 수정 후 저장
<configuration>
```

```
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
        <property>
                <name>yarn.app.mapreduce.am.env</name>
                <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
        </property>
        <property>
                <name>mapreduce.map.env</name>
                <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
        </property>
        <property>
                <name>mapreduce.reduce.env</name>
                <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
        </property>
</configuration>
```

- hadoop-env.sh 파일 편집

```
# Hadoop hadoop-env.sh 파일 설정
sudo vim $HADOOP_HOME/etc/hadoop/hadoop-env.sh

# 아래 내용 수정 후 저장
# Java
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

# Hadoop
export HADOOP_HOME=/usr/local/hadoop
```

- hadoop workers 편집

```
# Hadoop workers 편집
sudo vim $HADOOP_HOME/etc/hadoop/workers

# 아래 내용 수정 후 저장
# localhost << 주석 처리 또는 제거
dn1
dn2
dn3
```

- hadoop master 편집

```
# Hadoop masters 편집
sudo vim $HADOOP_HOME/etc/hadoop/masters

# 아래 내용 수정 후 저장
nn1
nn2
```

5.4 Spark 설치

- spark 설치

```
# 설치 관리용 디렉토리 이동
cd /install_dir
# Spark 3.2.1 설치
sudo wget https://dlcdn.apache.org/spark/spark-3.2.1/spark-3.2.1-bin-hadoop3.2.tgz
# Spark 3.2.1 압축 해제
sudo tar -xzvf spark-3.2.1-bin-hadoop3.2.tgz -C /usr/local
# Spark 디렉토리 이름 변경
sudo mv /usr/local/spark-3.2.1-bin-hadoop3.2 /usr/local/spark
```

- python3 설치

```
# Python 설치
sudo apt-get install -y python3-pip
# Python 버전 확인
python3 -V
# PySpark 설치
sudo pip3 install pyspark findspark
```

- spark 환경 변수 설정

```
# Hadoop 시스템 환경변수 설정
sudo vim /etc/environment

# 아래 내용 추가 후 저장
PATH 뒤에 ":/usr/local/spark/bin" 추가
PATH 뒤에 ":/usr/local/spark/sbin" 추가
SPARK_HOME="/usr/local/spark"

# 시스템 환경변수 활성화
source /etc/environment

#  Spark 사용자 환경변수 설정
echo 'export SPARK_HOME=/usr/local/spark' >> ~/.bashrc

# 사용자 환경변수 활성화
source ~/.bashrc
```

- spark-env.sh 파일 편집

```
# spark-env.sh 파일 카피
cd $SPARK_HOME/conf
sudo cp spark-env.sh.template spark-env.sh

# spark-env.sh 파일 편집
sudo vim spark-env.sh

# 아래 내용 수정 후 저장
export SPARK_HOME=/usr/local/spark
export SPARK_CONF_DIR=/usr/local/spark/conf
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
export SPARK_MASTER_WEBUI_PORT=18080
```

- spark-defaults.conf 파일 편집

```
# Spark spark-defaults.conf.template 파일 복사
sudo cp /usr/local/spark/conf/spark-defaults.conf.template /usr/local/spark/conf/spark-defaults.conf

# Spark spark-defaults.conf 파일 설정
sudo vim /usr/local/spark/conf/spark-defaults.conf

# 아래 설정 후 저장
# 클러스터 매니저 정보
spark.master                yarn
# 스파크 이벤트 로그 수행 유무
# true시 spark.eventLog.dir에 로깅 경로 지정해야합니다 - 스파크 UI에서 확인 가능합니다.
spark.eventLog.enabled    true
# 스파크 이벤트 로그 저장 경로
spark.eventLog.dir        /usr/local/spark/logs
```

- spark logs 디렉터리 생성

```
sudo mkdir -p /usr/local/spark/logs && sudo chown -R $USER:$USER /usr/local/spark/
```

- workers 파일 편집

```
# Spark workers 파일 생성
sudo cp /usr/local/spark/conf/workers.template /usr/local/spark/conf/workers

# Spark workers 파일 설정
sudo vim /usr/local/spark/conf/workers

# 아래 설정 후 저장
dn1
dn2
dn3
```

- python 환경 변수 설정

```
# 시스템 환경변수 편집
sudo vim /etc/environment

# 아래 내용 추가 후 저장
PATH 뒤에 ":/usr/bin/python3" 추가

# 시스템 환경변수 활성화
source /etc/environment

#  Python & PySpark 사용자 환경변수 설정
sudo echo 'export PYTHONPATH=/usr/bin/python3' >> ~/.bashrc
sudo echo 'export PYSPARK_PYTHON=/usr/bin/python3' >> ~/.bashrc

# 사용자 환경변수 활성화
source ~/.bashrc
```

5.5 Zookeeper 설치, 설정

- 고가용성의 클러스터를 구축하기 위해 Zookeeper 사용
- zookeeper 설치

```
# 설치 관리용 디렉토리 이동
cd /install_dir
# Zookeeper 3.8.0 설치
sudo wget https://dlcdn.apache.org/zookeeper/zookeeper-3.8.0/apache-zookeeper-3.8.0-bin.tar.gz
# Zookeeper 3.8.0 압축 해제
sudo tar -xzvf apache-zookeeper-3.8.0-bin.tar.gz -C /usr/local
# Zookeeper 디렉토리 이름 변경
sudo mv /usr/local/apache-zookeeper-3.8.0-bin /usr/local/zookeeper
```

- zookeeper 환경 변수 설정

```
# Hadoop 시스템 환경변수 설정
sudo vim /etc/environment

# 아래 내용 추가 후 저장
ZOOKEEPER_HOME="/usr/local/zookeeper"

# 시스템 환경변수 활성화
source /etc/environment

#  Spark 사용자 환경변수 설정
echo 'export ZOOKEEPER_HOME=/usr/local/zookeeper' >> ~/.bashrc
```

```
# 사용자 환경변수 활성화
source ~/.bashrc
```

- zoo.cfg 파일 편집

```
# Zookeeper 설정 경로 이동
cd /usr/local/zookeeper
# Zookeeper 설정 파일 복사
sudo cp ./conf/zoo_sample.cfg ./conf/zoo.cfg

# zoo.cfg 편집
sudo vim ./conf/zoo.cfg

# 아래 내용 수정 후 저장
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/usr/local/zookeeper/data
dataLogDir=/usr/local/zookeeper/logs
clientPort=2181
maxClientCnxns=0
maxSessionTimeout=180000
server.1=nn1:2888:3888
server.2=nn2:2888:3888
server.3=dn1:2888:3888
```

- myid 설정

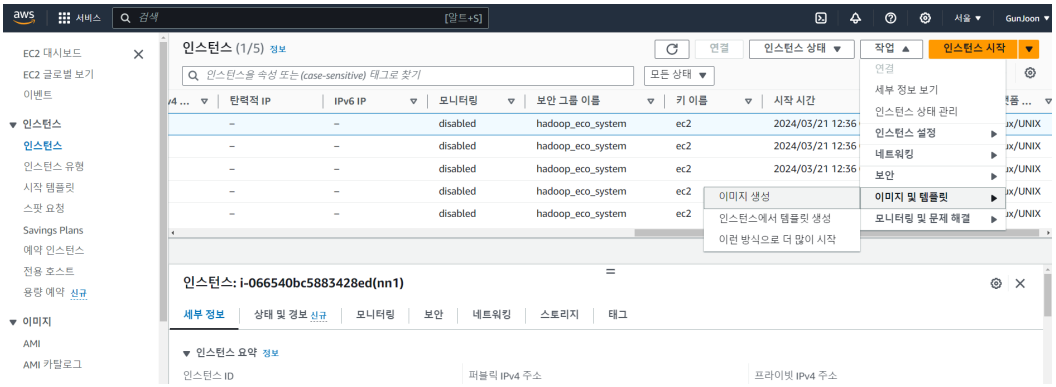
```
# Zookeeper 데이터 디렉토리 생성
sudo mkdir -p /usr/local/zookeeper/data
sudo mkdir -p /usr/local/zookeeper/logs

# Zookeeper 디렉토리 사용자 그룹 변경
sudo chown -R $USER:$USER /usr/local/zookeeper

# myid 파일 편집
sudo vim /usr/local/zookeeper/data/myid

# 아래 내용 수정 후 저장
1
```

- ssh key 생성

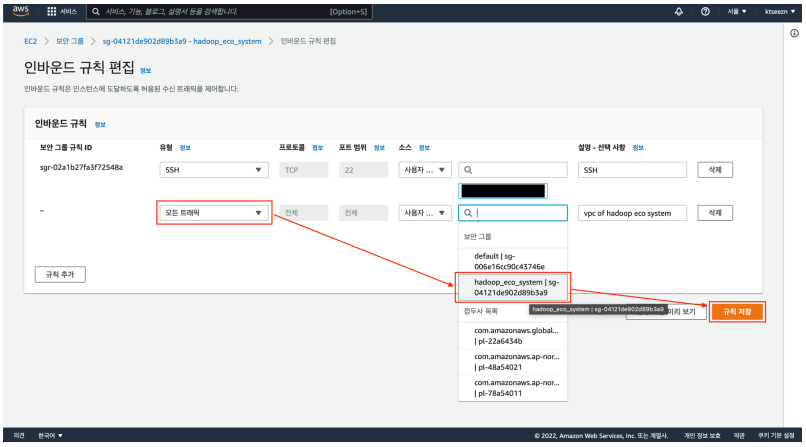
```
# ssh key 생성
ssh-keygen -t rsa # 이후 Enter만 세 번 입력 탁! 탁! 탁!

# authorized_keys 생성
cat >> ~/.ssh/authorized_keys < ~/.ssh/id_rsa.pub

# localhost 접속 테스트
ssh localhost
# Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

5.6 AMI 생성, 복제

- 만든 이미지로 4개의 인스턴스 생성
- 추가된 4개의 인스턴스에 각각 이름을 dn1, dn2, dn3, nn2라고 지정



- AMI로 복제한 인스턴스까지 총 5대의 인스턴스끼리 같은 VPC에서 모든 트래픽 통신이 가능하도록 설정
- hadoop_eco_system 보안 그룹을 선택하고 인바운드 규칙 편집을 선택해 모든 트래픽 허용



5.7 로컬 PC SSH 설정

- 배포된 5개의 인스턴스끼리 SSH 통신이 가능하도록 설정하고 각 서버의 호스트 이름 설정
- SSH Config 설정

```
# My Mac terminal

# config 편집
vim ~/.ssh/config

# 아래 내용으로 수정 후 저장
Host nn1
        HostName 3.37.62.0(ex)
        User ubuntu
        IdentityFile ~/identity/hadoop_eco_system.pem

Host nn2
        HostName 3.39.9.193(ex)
        User ubuntu
        IdentityFile ~/identity/hadoop_eco_system.pem

Host dn1
        HostName 3.34.194.251(ex)
        User ubuntu
        IdentityFile ~/identity/hadoop_eco_system.pem

Host dn2
        HostName 15.164.210.108(ex)
        User ubuntu
        IdentityFile ~/identity/hadoop_eco_system.pem
```

```
Host dn3
        HostName 54.180.153.190(ex)
        User ubuntu
        IdentityFile ~/identity/hadoop_eco_system.pem
```

- 서버 접속 테스트
- SSH Config 편집 후 로컬에서 nn1, nn2, dn1, dn2, dn3으로 접속 가능한지 테스트
- 모든 서버 호스트 이름 설정

```
sudo hostnamectl set-hostname nn1

# ssh 접속
ssh nn2
sudo hostnamectl set-hostname nn2
exit
ssh dn1
sudo hostnamectl set-hostname dn1
exit
ssh dn2
sudo hostnamectl set-hostname dn2
exit
ssh dn3
sudo hostnamectl set-hostname dn3
```

- 모든 인스턴스에 Hosts 파일 복제
  - nn1에서만 진행

```
# 복제
cat /etc/hosts | ssh nn2 "sudo sh -c 'cat >/etc/hosts'"
cat /etc/hosts | ssh dn1 "sudo sh -c 'cat >/etc/hosts'"
cat /etc/hosts | ssh dn2 "sudo sh -c 'cat >/etc/hosts'"
cat /etc/hosts | ssh dn3 "sudo sh -c 'cat >/etc/hosts'"
```

- hdfs-site.xml 파일 복제
  - nn1에서만 진행

```
# EC2 Ubuntu terminal(nn1)

# 복제
cat $HADOOP_HOME/etc/hadoop/hdfs-site.xml | ssh nn2 "sudo sh -c 'cat >$HADOOP_HOME/etc/hadoop/hdfs-s
```