

# Gotcha 포팅매뉴얼

1. EC2 초기 서버 설정
    - 1) [Terminus 이용해 발급받은 EC2 인스턴스 접속](#)
    - 2) [패키지 관리자 업데이트](#)
    - 3) [JDK 다운로드](#)
    - 4) [Docker 설치](#)
    - 5) [Docker-compose 설치](#)
    - 6) [Nginx 설치와 SSL 발급](#)
  2. Jenkins CI 환경 구축
    - 1) [Jenkins 설치](#)
    - 2) [Jenkins - Gitlab 연동](#)
    - 3) [Web-hook 설정](#)
  3. MySQL 설정
    - 1) [DB 설치](#)
    - 2) [유저 생성 후 권한 설정](#)
    - 3) [MySQL 외부 접속 허용](#)
    - 4) [MySQL 재실행](#)
    - 5) [Workbench 접속 확인](#)
  4. Redis 설치 및 환경 세팅
    - 4-1. [Redis 서버 설치](#)
    - 4-2. [Redis 버전 확인](#)
    - 4-3. [Redis 설정 파일 수정](#)
    - 4-4. [Redis 설정 확인](#)
  5. Jenkins CD 환경 구축
    - 5-1. [Gradle 설치](#)
    - 5-2. [Docker hub image build&push 설정](#)
      - 5-2-1. [Docker hub Credentials 등록](#)
      - 5-2-2. [Jenkins에서 Credentials 설정](#)
      - 5-2-3. [Jenkins pipeline용 Credentials 설정](#)
      - 5-2-4. [Jenkins sudo 권한 설정](#)
      - 5-2-5. [이미지를 빌드할 수 있는 Dockerfile 작성](#)
      - 5-2-6. [Jenkins pipeline script 작성](#)
    - 5-3. [Docker hub image pull & deploy](#)
      - 5-3-1. [application.yaml 수정](#)
      - 5-3-2. [스프링서버 배포 Shell script 작성](#)
    - 5-4. [Nginx 설정](#)
      - 5-4-1. [기본 포트 설정](#)
      - 5-4-2. [nginx shell script 작성](#)
  6. 오토 스케일링 설정
    - 6-1 [AMI 만들기](#)
    - 6-2 [시작 템플릿 만들기](#)
    - 6-2 [로드 밸런서 생성](#)
    - 6-3 [오토 스케일링 설정](#)
  7. [프론트엔드 리액트 프로젝트 배포](#)
- [부록]
1. [Jenkins 포트 수정하기](#)

## 1. EC2 초기 서버 설정

### 1) Terminus 이용해 발급받은 EC2 인스턴스 접속

- Terminus 설치
- Hosts에서 New Host 생성

- Address : 제공받은 도메인 입력 (j8팀.p.ssafy.io)
- Set a Key에 제공받은 .pem 파일 추가
- Host 세팅 완료 후, 더블 클릭하면 해당 인스턴스로 접속 할 수 있음

## 2) 패키지 관리자 업데이트

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

## 3) JDK 다운로드

```
$ sudo apt-get install openjdk-11-jdk
$ java -version
```

## 4) Docker 설치

- 오래된 버전이 있다면 아래 명령어를 입력해 삭제

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

- Repository 설정

```
$ sudo apt-get update

$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

$ sudo mkdir -m 0755 -p /etc/apt/keyrings

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

$ echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- Docker Engine 설치

```
$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

$ docker --version
```

## 5) Docker-compose 설치

```
sudo apt-get update

sudo apt-get install docker-compose-plugin

docker compose version
```

## 6) Nginx 설치와 SSL 발급

- Nginx 설치

```
$ sudo apt update

$ sudo apt install nginx
```

- Nginx 실행 확인

```
$ sudo systemctl start nginx

$ sudo systemctl status nginx

$ sudo systemctl stop nginx
```

- Certbot 설치

```
$ sudo apt install certbot python3-certbot-nginx
```

- 인증서 발급 및 https 적용 (싸피에서 발급받은 도메인 주소 입력 [j8a팀번호.p.ssafy.io](https://j8a팀번호.p.ssafy.io))

```
$ sudo certbot --nginx -d [도메인 주소]
```

## 2. Jeknkins CI 환경 구축

### 1) Jenkins 설치

- gitlab backend-develop 브랜치에 변화가 생기면 자동으로 pull 하는 CI 구축

- Jenkins 설치

```
$ curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null

$ 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

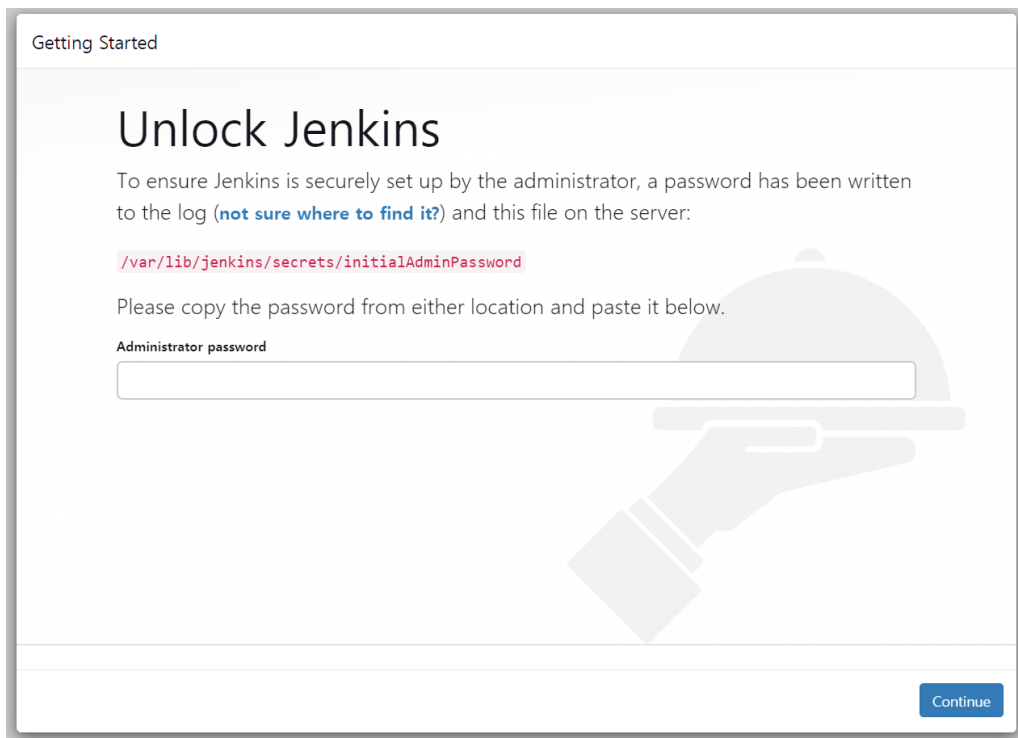
$ sudo apt-get install jenkins

// 확인
$ sudo systemctl status jenkins

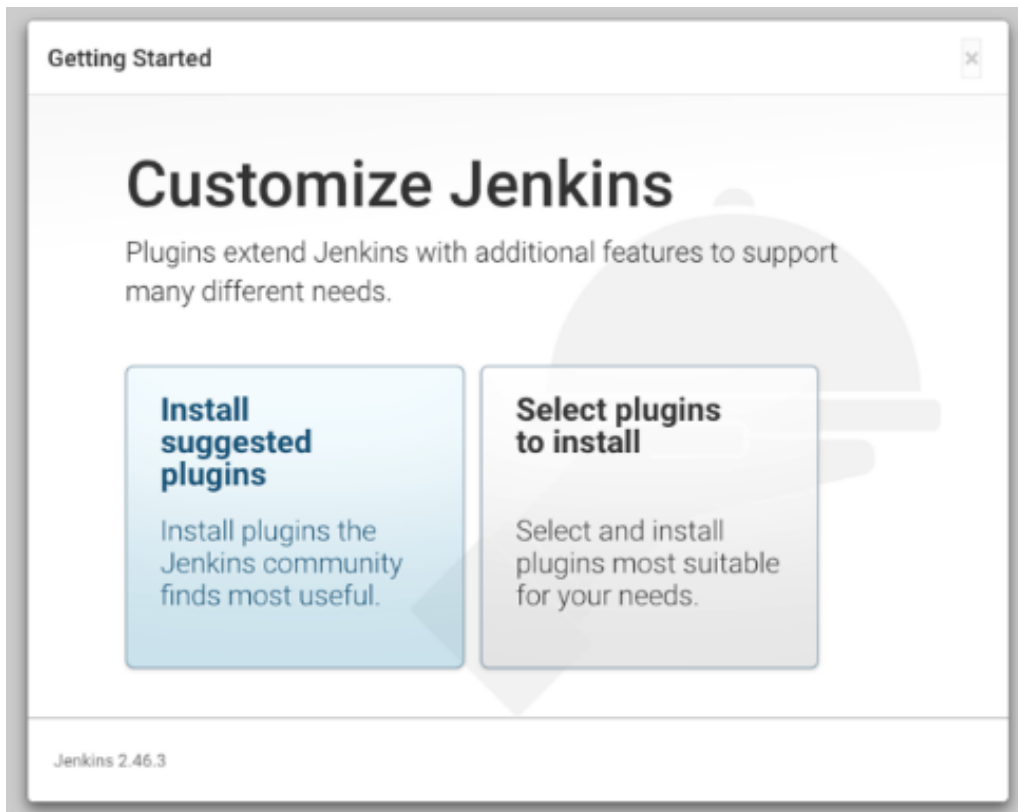
$ sudo systemctl start jenkins

// 초기 비밀번호 확인
$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

- [http://\[도메인\]:8080/](http://[도메인]:8080/) 로 접속하여 Jenkins 설정 계속
- 초기 비밀번호 입력하여 접속

The image shows the 'Getting Started' screen of the Jenkins web interface. The title is 'Unlock Jenkins'. Below the title, it says: 'To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server: /var/lib/jenkins/secrets/initialAdminPassword'. It then asks the user to 'Please copy the password from either location and paste it below.' There is a text input field labeled 'Administrator password'. In the background, there is a faint illustration of a hand holding a tray with a dome-shaped object. At the bottom right, there is a blue button labeled 'Continue'.

- [Install Suggested plugins](#) 선택



- Admin 계정 생성

The image shows the 'Getting Started' window in Jenkins 2.375.2. The main heading is 'Create First Admin User'. The form contains the following fields and labels:

- 계정명** (Username): A text input field.
- 암호** (Password): A text input field.
- 암호 확인** (Confirm Password): A text input field.
- 이름** (Name): A text input field.
- 이메일 주소** (Email Address): A text input field.

At the bottom left, the version 'Jenkins 2.375.2' is displayed. At the bottom right, there are two buttons: 'Skip and continue as admin' and 'Save and Continue'.

- Jenkins 접속 URL 설정 (도메인:포트번호)

- 추가 플러그인 설치 (DashBoard → Jenkins 관리 → 플러그인 관리 → Available plugins)
- 아래 플러그인 Install without restart 옵션으로 설치
  - Generic Webhook Trigger
  - Gitlab
  - Gitlab API
  - Gitlab Authentication
  - Mattermost Notification
  - Docker Pipeline

## 2) Jenkins - Gitlab 연동

- Gitlab Access Token 발급 (팀장이 권한 풀어주면 settings 접근 가능)
- Gitlab project → Settings → Access Tokens
- 아래와 같이 설정 후에 토큰 생성

### Add a project access token

Enter the name of your application, and we'll return a unique project access token.

#### Token name

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

#### Expiration date

#### Select a role

#### Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

- ☒ api  
Grants complete read and write access to the scoped project API, including the Package Registry.
- ☒ read\_api  
Grants read access to the scoped project API, including the Package Registry.
- ☒ read\_repository  
Grants read access (pull) to the repository.
- ☐ write\_repository  
Grants read and write access (pull and push) to the repository.

Create project access token

- Jenkins에 Gitlab 설정
  - Connection name : 원하는 이름 입력
  - Gitlab host URL : 깃랩 메인 주소 입력 (<https://lab.ssfy.com>)
  - Credentials : [add](#) 를 통해 새로운 credentials 추가
  - 추가시 Domain : Global credentials 선택
  - Kind : GitLab API token
  - Scope : Global (Jenkins~ )
  - API token : 위에서 생성한 토큰 입력

**GitLab**

☒ Enable authentication for '/project' end-point

**GitLab connections**

Connection name

A name for the connection

**GitLab** host URL

The complete URL to the **GitLab** server (e.g. http://**gitlab**.mydomain.com)

Credentials

API Token for accessing **GitLab**

[+ Add](#)

고급 ▾

[저장](#) [Apply](#)

**Add Credentials**

Domain

Global credentials (unrestricted) ▾

Kind

GitLab API token ▾

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▾

API token

\*\*\*\*\*

ID ?

Description ?

[Add](#) [Cancel](#)

### 3) Web-hook 설정

- Jenkins 프로젝트 생성
  - 새로운 item 클릭하여 Pipeline 으로 프로젝트 생성
- Build Trigger 설정
  - Build Triggers 항목의 Build when a change is push to Gitlab~ 항목 체크
    - 뒤에 있는 URL 저장해 놓기
  - 고급 버튼 누르고 Secret token 에서 Generate 버튼 클릭

Secret token ?

[Generate](#)

[Clear](#)

- GitLab 설정
  - 설정의 webhook 탭으로 이동하여 URL과 secret token 입력
  - trigger의 push events에서 트리거를 발생시킬 브랜치 이름 입력
  - add webhook 버튼을 눌러 완료

## 3. MySQL 설정



## 1) DB 설치

```
$ sudo apt-get install mysql-server
```

## 2) 유저 생성 후 권한 설정

```
$ sudo mysql -uroot  
  
$ create user '아이디'@'%'identified by '비밀번호'  
  
$ grant all privileges on *.*to '아이디'@'%';
```

## 3) MySQL 외부 접속 허용

```
$ cd /etc/mysql/mysql.conf.d  
  
$ sudo vi mysqld.cnf
```

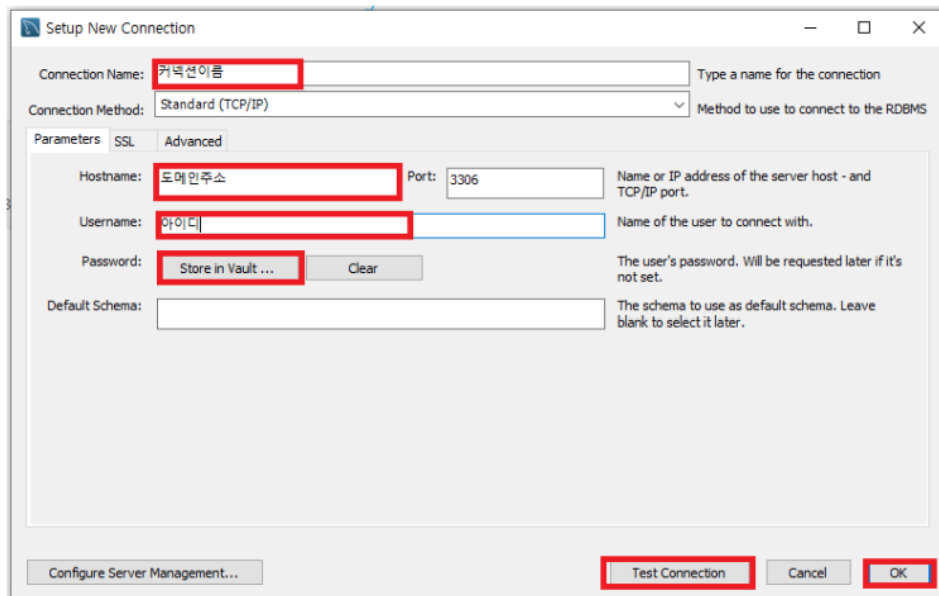
```
...  
// bind-address 를 0.0.0.0으로 변경  
bind-address = 0.0.0.0  
...
```

## 4) MySQL 재실행

```
$ sudo service mysql restart
```

## 5) Workbench 접속 확인

- 새로운 커넥션 만들기



## 4. Redis 설치 및 환경 세팅

### 4-1. Redis 서버 설치

```
$ sudo apt-get install redis-server
```

## 4-2. Redis 버전 확인

```
$ redis-server --version
```

## 4-3. Redis 설정 파일 수정

```
$ sudo vi /etc/redis.conf
```

```
// 원격 액세스 가 가능하도록 서버를 열어줌
bind 0.0.0.0 ::1

// 메모리 최대 사용 용량 및 메모리 초과시 오래된 데이터를 지워 메모리 확보하도록 정책 설정
maxmemory 2g
maxmemory-policy allkeys-lru
```

```
$ sudo systemctl restart redis-server
```

## 4-4. Redis 설정 확인

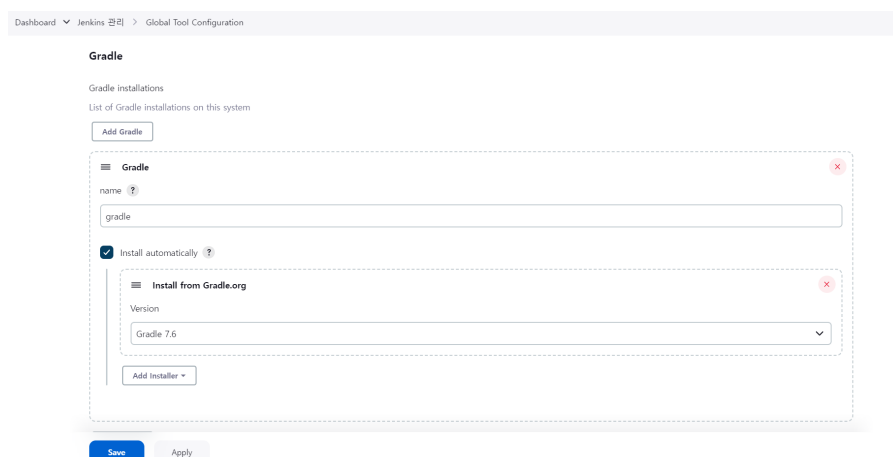
```
$ sudo systemctl status redis-server

$ redis-cli ping # PONG을 반환하면 설정 완료!
```

# 5. Jenkins CD 환경 구축

## 5-1. Gradle 설치

- Jenkins - Jenkins관리 - Global Tool Configuration - Gradle 섹션으로 이동
- Add Gradle에서 해당하는 Gradle 버전을 찾아 세팅 진행



## 5-2. Docker hub image build&push 설정

### 5-2-1. Docker hub Credentials 등록

- Dockerhub token 등록하여 Jenkins 통해 docker push 할 수 있도록 함
- Docker Hub > Account Settings > Security > New Access Token > Access Token description > Generate > token 복사

## Copy Access Token

When logging in from your Docker CLI client, use this token as a password. [Learn more](#)

### ACCESS TOKEN DESCRIPTION

Gotcha

### ACCESS PERMISSIONS

Read, Write, Delete

To use the access token from your Docker CLI client:

1. Run `docker login -u yejierinheo`
2. At the password prompt, enter the personal access token.

### 5-2-2. Jenkins에서 Credentials 설정

- Jenkins 관리 > Manage Credentials > Add Credentials
- Username : Dockerhub id
- password : Dockerhub token
- id: 원하는 credential 이름 지정

### 5-2-3. Jenkins pipeline용 Credentials 설정

- Gitlab API Token으로는 pipeline에서 git clone 불가
- Dashboard > jenkins 관리 > Credentials > domain(global)로 들어가 add credentials
- username with password 종류로 선택해 username에는 gitlab email 주소 password는 API Token 입력하여 생성
- 해당 credential로 파이프라인에서 깃 클론 가능

### 5-2-4. Jenkins sudo 권한 설정

- pipeline에 sudo 명령어 실행시 pwd입력하라는 창 뜸

```
$ sudo vi /etc/sudoers
```

- %sudo ALL = (ALL:ALL) ALL의 아랫줄에 아래 코드 추가 후 `:wq!` 입력 후 빠져나옴

```
jenkins ALL=(ALL) NOPASSWD: ALL
```

```
$ sudo service jenkins restart
```

### 5-2-5. 이미지를 빌드할 수 있는 Dockerfile 작성

- 파이프라인 프로젝트가 있는 폴더 하위에 Dockerfile 생성

```
$ sudo vi /var/lib/jenkins/workspace/[프로젝트명]/Dockerfile
```

- 아래와 같이 작성

```
FROM openjdk:11
ARG JAR_FILE=backend/build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENV USE_PROFILE local
ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=${USE_PROFILE}", "/app.jar"]
```

## 5-2-6. Jenkins pipeline script 작성

- 여기까지 작성 후 빌드해보면 Docker hub에 이미지가 빌드된 것을 확인할 수 있다.

```
pipeline {
    agent any
    environment {
        DOCKER_REPOSITORY = "yejierinho/gotcha"
        DOCKERHUB_CREDENTIALS = credentials('Gotcha_docker')
    }

    stages {
        stage('Mattermost notification'){
            steps {
                script {
                    try {
                        mattermostSend (
                            color: "#2A42EE",
                            message: "Build STARTED: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                        )
                    } catch(e) {
                        currentBuild.result = "FAILURE"
                    }
                }
            }
        }

        stage('Clone') {
            steps {
                git branch: 'backend_develop',
                credentialsId: 'Gotcha-gitlab',
                url: 'https://lab.ssafy.com/s08-ai-image-sub2/S08P22A602'
            }
        }

        stage('Clean build') {
            steps {
                sh '''
                    cd backend
                    ls -al
                    chmod +x ./gradlew
                    sudo ./gradlew clean bootjar
                '''
            }
        }

        stage('Docker build'){
            steps{
                echo 'docker build'
                sh """ #!/bin/bash
                PREV_IMAGE=`sudo docker images --filter=reference='yejierinho/*' -q`
                echo "prev IMAGE : \${PREV_IMAGE}"
                if [[ -n \${PREV_IMAGE} ]]; then
                    echo "prev image delete"
                    sudo docker rmi \${PREV_IMAGE}
                fi
                echo \${DOCKERHUB_CREDENTIALS_PSW} | sudo docker login -u \${DOCKERHUB_CREDENTIALS_USR} --password-stdin
                sudo docker build . -t \${DOCKER_REPOSITORY} --no-cache
                sudo docker push \${DOCKER_REPOSITORY}
                """
            }
        }

        stage('Deploy Spring Server'){
            steps{
                sh 'bash deploy.sh'
            }
        }

        stage('Switch Nginx'){
            steps{
                sh 'bash switch.sh'
            }
        }
    }

    post {

```

```

        success {
            script {
                if(currentBuild.result != "SUCCESS") {
                    mattermostSend (
                        color: "danger",
                        message: "Build FAILED: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                    )
                } else {
                    mattermostSend (
                        color: "good",
                        message: "Build SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                    )
                }
            }
        }
    }
}

```

## 5-3. Docker hub image pull & deploy

### 5-3-1. application.yaml 수정

- jenkins 워크스페이스의 application.yaml 위치로 이동

```
$ cd /var/lib/jenkins/workspace/Gotcha/backend/src/main/resources
```

- application.yaml 설정

```

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: ${DATASOURCE_URL}
    username: ${DATASOURCE_USERNAME}
    password: ${DATASOURCE_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: create
      show-sql: true
      database-platform: org.hibernate.dialect.MySQL8Dialect
    properties:
      hibernate:
        format_sql: true
      open-in-view: false

```

- jenkins 환경 변수

```

DATASOURCE_URL
DATASOURCE_USERNAME
DATASOURCE_PASSWORD

```

### 5-3-2. 스프링서버 배포 Shell script 작성

- jenkins workspace 루트 디렉토리에 deploy.sh로 생성

```

#!/usr/bin/env bash

echo "> $DOCKER_REPOSITORY"
sudo true > RESULT
sudo chmod 666 /var/run/docker.sock
# 현재 사용하고 있는 포트와 유휴 상태인 포트를 체크한다.
RESPONSE=$(curl -s localhost:8080/actuator/health)
echo "> RESPONSE : "$RESPONSE

IS_ACTIVE=$(echo ${RESPONSE} | grep 'UP' | wc -l)
echo "> IS_ACTIVE "$IS_ACTIVE
if [ $IS_ACTIVE -eq 1 ];
then
    IDLE_PORT=8081
    IDLE_PROFILE=prod-green
    CURRENT_PORT=8080
    CURRENT_PROFILE=prod-blue
else

```

```

IDLE_PORT=8080
IDLE_PROFILE=prod-blue
CURRENT_PORT=8081
CURRENT_PROFILE=prod-green
fi

echo "> 다음 사용할 포트" $IDLE_PORT
echo "> 다음 사용할 프로필 " $IDLE_PROFILE

# 도커 허브에서 PULL을 한다.
docker pull $DOCKER_REPOSITORY
docker rm $(docker ps --filter status=exited -q)
docker rmi -f $(docker images -f "dangling=true" -q)

# 도커를 통해 컨테이너를 실행시킨다.

echo "> sudo nohup docker run -p $IDLE_PORT:8080 -e "USE_PROFILE=$IDLE_PROFILE" --env-file .env $DOCKER_REPOSITORY > nohup.out 2>&1 &"
sudo nohup docker run -p $IDLE_PORT:8080 --env-file .env -e "USE_PROFILE=prod" $DOCKER_REPOSITORY > nohup.out 2>&1 &

echo "> 60초동안 5초마다 Health Check"

for RETRY in {1..12}
do
    for i in {1..5} ;
    do
        echo "> Health Check까지 " $(( 6 - i ))초 남음

        sleep 1
        done

        RESPONSE=$(curl -s localhost:${IDLE_PORT}/actuator/health)
        IS_ACTIVE=$(echo ${RESPONSE} | grep 'UP' | wc -l)

        if [ $IS_ACTIVE -ge 1 ]; then
            echo "> Health Check Success"
            echo "IDLE_PORT" $IDLE_PORT
            echo "IDLE_PORT" > RESULT
            exit 0
        else
            echo "> Health Check Failed"
            echo "> Health Check RESPONSE : " $RESPONSE
        fi
    fi
    if [ $RETRY -eq 10 ]; then
        echo "> Health Check Failed"
        echo "FAIL" > RESULT
    fi
done

exit 1

```

## 5-4. Nginx 설정

### 5-4-1. 기본 포트 설정

- 외부 설정파일에 기본 포트 설정

```

$ sudo cd /etc/nginx/conf.d
$ sudo vim port.conf

```

```

set $active_server BLUE;

```

### 5-4-2. nginx shell script 작성

- jenkins workspace 루트 디렉토리에 deploy.sh로 생성

```

#!/usr/bin/env bash
# 현재 사용중인 포트를 확인한다.
RESPONSE=$(curl -s -k -L j8a602.p.ssafy.io/actuator/health)
echo "> RESPONSE : "$RESPONSE

IS_ACTIVE=$(echo ${RESPONSE} | grep 'UP' | wc -l)
echo "> IS_ACTIVE" "$IS_ACTIVE"
CURRENT_PORT=$(curl -k -L j8a602.p.ssafy.io/port | grep 'BLUE' | wc -l)
echo "현재 구동중인 포트: " "$CURRENT_PORT"
if [ "$IS_ACTIVE" -eq 1 ];

```

```

then
    if [ "$CURRENT_PORT" -eq 1 ];
    then
        IDLE_PORT=8081
        IDLE_PROFILE=GREEN
        CURRENT_PORT=8080
        CURRENT_PROFILE=BLUE
    else
        IDLE_PORT=8080
        IDLE_PROFILE=BLUE
        CURRENT_PORT=8081
        CURRENT_PROFILE=GREEN
    fi
else
    IDLE_PORT=8080
    IDLE_PROFILE=BLUE
    CURRENT_PORT=8081
    CURRENT_PROFILE=GREEN
fi
echo "전환할 포트: " "$IDLE_PORT"

echo "> 포트 세팅 변경"
echo "set \${active_server} $IDLE_PROFILE;" | sudo tee /etc/nginx/default.d/port.conf
echo "> 기존 컨테이너 삭제"
sudo docker kill $(docker ps -qf publish=$CURRENT_PORT)
echo "> nginx 재시작"
sudo systemctl reload nginx

```

- /etc/nginx/sites-enabled/default에서 수정

```

server {
    server_name  colorthero.com;
    root         /usr/share/nginx/html;

    # Load configuration files for the default server block.

    include /etc/nginx/default.d/*.conf;

    location ~* ^/(oauth2|login){
        proxy_pass http://$active_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location = /actuator/health {
        proxy_pass http://$active_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location = /api/refresh {
        proxy_pass http://$active_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location /nginx_status{
        stub_status on;
        access_log off;
        allow 127.0.0.1;
        deny all;
    }
    location = /port {
        add_header Content-Type text/plain;
        if ($remote_addr = "nginx ip 주소"){
            return 200 '$active_server';
        }
        deny all;
    }
}

```

## 6. 오토 스케일링 설정

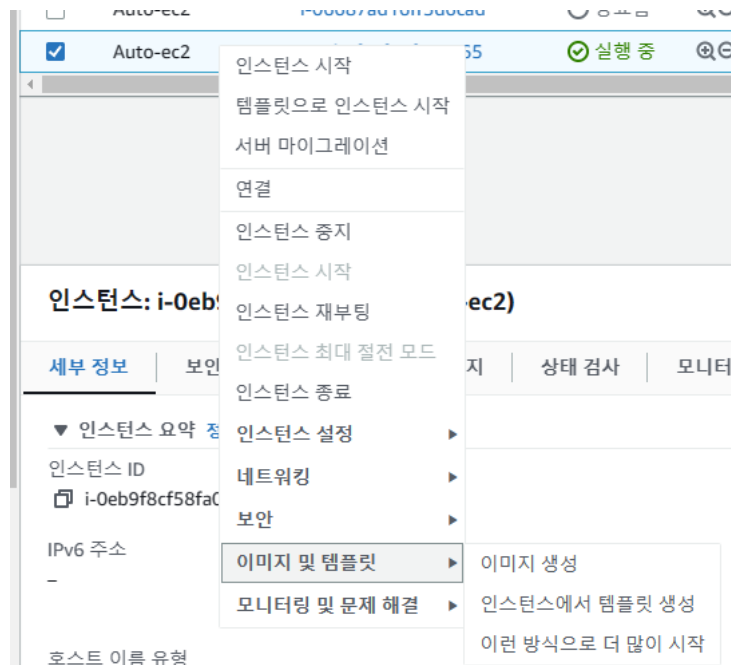
## 6-1 AMI 만들기

- AWS에서 AMI를 만들기 위해 인스턴스를 생성한다.

이미지를 만들기 전에 docker랑 swap 메모리를 늘려야 한다.

```
sudo yum install -y docker
sudo dd if=/dev/zero of=/swapfile bs=128M count=32
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
sudo swapon -s
sudo vi /etc/fstab
# 파일 맨끝에 추가한다.
/swapfile swap swap defaults 0 0
```

- 생성한 인스턴스에서 이미지를 만들자.



## 6-2 시작 템플릿 만들기

AWS EC2 → 인스턴스 → 시작 템플릿 → 시작 템플릿 생성 이동



### 시작 템플릿 이름 및 설명

시작 템플릿 이름 - 필수

이 계정에 대해 고유해야 합니다. 최대 128자입니다. '&', '\*', '@' 등의 특수 문자나 공백은 사용할 수 없습니다.

템플릿 버전 설명

최대 255자

**Auto Scaling 지침 정보**  
EC2 Auto Scaling에 이 템플릿을 사용하려면 이 항목을 선택합니다.

☐ EC2 Auto Scaling에 사용할 수 있는 템플릿을 설정하는 데 도움이 되는 지침 제공

- ▶ 템플릿 태그
- ▶ 원본 템플릿

- 템플릿 이름 설정

### ▼ 애플리케이션 및 OS 이미지(Amazon Machine Image) 정보

AMI는 인스턴스를 시작하는 데 필요한 소프트웨어 구성(운영 체제, 애플리케이션 서버 및 애플리케이션)이 포함된 템플릿입니다. 아래에서 찾고 있는 항목이 보이지 않으면 AMI를 검색하거나 찾아보십시오.

최근 사용 | **내 AMI** | Quick Start

☐ 시작 템플릿에 포함하지 않음
 ☒ 내 소유
 ☐ 나와 공유됨


 더 많은 AMI 찾아보기  
 AWS, Marketplace 및 커뮤니티의 AMI 포함

Amazon Machine Image(AMI)

|                          |   |
|--------------------------|---|
| python-server-image      |   |
| ami-0966bc8c7c2d86fcd    |   |
| 2023-03-27T07:53:01.601Z | 가상화: hvm ENA 활성화됨: true 루트 디바이스 유형: ebs |
| 부트 모드: uefi-preferred    |   |

설명

-

| 아키텍처   | AMI ID                |
|--------|-----------------------|
| x86_64 | ami-0966bc8c7c2d86fcd |

- 만들었던 AMI를 고른다.

▼ 인스턴스 유형 정보

어드밴스드

인스턴스 유형

시작 템플릿에 포함하지 않음 ▼

인스턴스 유형 비교

▶ 키 페어(로그인) 정보

키 페어를 사용하여 인스턴스에 안전하게 연결할 수 있습니다. 인스턴스를 시작하기 전에 선택한 키 페어에 대한 액세스 권한이 있는지 확인하세요.

▼ 네트워크 설정 정보

서브넷 정보

시작 템플릿에 포함하지 않음 ▼

새 서브넷 생성

서브넷을 지정하면 네트워크 인터페이스가 템플릿에 자동으로 추가됩니다.

방화벽(보안 그룹) 정보

보안 그룹은 인스턴스에 대한 트래픽을 제어하는 방화벽 규칙 세트입니다. 특정 트래픽이 인스턴스에 도달하도록 허용하는 규칙을 추가합니다.

☒ 기존 보안 그룹 선택

☐ 보안 그룹 생성

보안 그룹 정보

보안 그룹 선택 ▼

보안 그룹 규칙 비교

▶ 고급 네트워크 구성

- 인스턴스 유형 , 키 페어 , VPC랑 서브넷을 설정한다.

## 6-2 로드 밸런서 생성

EC2 → 로드 밸런서 → 로드 밸런서 생성으로 페이지 이동

Network mapping Info

The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.

VPC Info

Select the virtual private cloud (VPC) for your targets. Only VPCs with an internet gateway are enabled for selection. The selected VPC cannot be changed after the load balancer is created. To confirm the VPC for your targets, view your [target groups](#).

-

vpc-0bc15f886a1b4acc5

IPv4: 172.31.0.0/16

▼

↻

Mappings Info

Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.

☐ ap-northeast-2a (apne2-az1)

☐ ap-northeast-2b (apne2-az2)

☐ ap-northeast-2c (apne2-az3)

☐ ap-northeast-2d (apne2-az4)

- 가용영역과 VPC를 설정한다.

### Security groups [Info](#)

A security group is a set of firewall rules that control the traffic to your load balancer.

Security groups

▼
↻

[Create new security group](#)

default sg-045e8cf4e576fd68b ✕

VPC: vpc-0bc15f886a1b4acc5

### Listeners and routing [Info](#)

A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.

▼ Listener HTTP:80

Remove

Protocol

Port

Default action [Info](#)

HTTP ▼

:

80

1-65535

Forward to

Select a target group ▼

↻

[Create target group](#)

#### Listener tags - *optional*

Consider adding tags to your listener. Tags enable you to categorize your AWS resources so you can more easily manage them.

Add listener tag

You can add up to 50 more tags.

Add listener

- 보안그룹과 포워딩할 포트를 설정 8000번으로 설정해서 포트를 라우팅 한다.
- 라우팅을 할 타겟 그룹을 만들어야 한다.
- create target group으로 들어가 그룹을 만들어야한다.

Choose a target type

☒ Instances

- Supports load balancing to instances within a specific VPC.
- Facilitates the use of [Amazon EC2 Auto Scaling](#) to manage and scale your EC2 capacity.

☐ IP addresses

- Supports load balancing to VPC and on-premises resources.
- Facilitates routing to multiple IP addresses and network interfaces on the same instance.
- Offers flexibility with microservice based architectures, simplifying inter-application communication.
- Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.

☐ Lambda function

- Facilitates routing to a single Lambda function.
- Accessible to Application Load Balancers only.

☐ Application Load Balancer

- Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
- Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.

Target group name

alb-test

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Protocol

TCP

:

Port

80

VPC

Select the VPC with the instances that you want to include in the target group.

-

vpc-0bc15f886a1b4acc5

IPv4: 172.31.0.0/16

- 타겟 그룹을 만들도록 하자

### Health checks

The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.

Health check protocol

HTTP ▼

Health check path

Use the default path of "/" to ping the root, or specify a custom path if preferred.

/

Up to 1024 characters allowed.

► Advanced health check settings

### Attributes

ⓘ Certain default attributes will be applied to your target group. You can view and edit them after creating the target group.

► **Tags - optional**

Consider adding tags to your target group. Tags enable you to categorize your AWS resources so you can more easily manage them.

헬스체크 경로 url을 설정해야 한다.

타겟 그룹을 만들고 로드 밸런서 생성을 완료하자.

## 6-3 오토 스케일링 설정

EC2 → 오토스케일링 → 오토 스케일링 생성 페이지로 이동

## 시작 템플릿 또는 구성 선택 Info

이 Auto Scaling 그룹에서 시작하는 모든 EC2 인스턴스에 공통된 설정이 포함된 시작 템플릿을 지정합니다. 현재 시작 구성을 사용하는 경우 시작 템플릿으로 마이그레이션할 수 있습니다.

### 이름

#### Auto Scaling 그룹 이름

그룹을 식별할 이름을 입력합니다.

현재 리전에서 이 계정에 대해 고유해야 하며 255자를 넘지 않아야 합니다.

### 시작 템플릿 Info

[시작 구성으로 전환](#)

#### 시작 템플릿

Amazon Machine Image(AMI), 인스턴스 유형, 키 페어 및 보안 그룹과 같은 인스턴스 수준 설정이 포함된 시작 템플릿을 선택합니다.

시작 템플릿 선택 ▼



[시작 템플릿 생성](#)

[취소](#)

[다음](#)

- 이름과 시작 템플릿 설정하기

## 네트워크 [Info](#)

대부분의 애플리케이션에서는 여러 가용 영역을 사용할 수 있으며 EC2 Auto Scaling이 여러 영역 간에 인스턴스를 균일하게 분산할 수 있습니다. 기본 VPC와 기본 서브넷은 빠르게 시작하는 데 적합합니다.

### VPC

Auto Scaling 그룹의 가상 네트워크를 정의하는 VPC를 선택합니다.

vpc-0bc15f886a1b4acc5  
172.31.0.0/16 Default



[VPC 생성](#)

### 가용 영역 및 서브넷

선택한 VPC에서 Auto Scaling 그룹이 사용할 수 있는 가용 영역과 서브넷을 정의합니다.

가용 영역 및 서브넷 선택



ap-northeast-2a | subnet-08cbcd0f17228a8f2 X  
172.31.0.0/20 Default

[서브넷 생성](#)

## 인스턴스 유형 요구 사항 [Info](#)

[시작 템플릿 재정의](#)

시작 템플릿과 동일한 인스턴스 속성 또는 인스턴스 유형을 유지하거나, 다른 인스턴스 속성을 지정하거나 인스턴스 유형을 수동으로 추가하여 시작 템플릿을 재정의하도록 선택할 수 있습니다.

시작 템플릿

[auto-scaling](#)

lt-052a0e565aa4d1272

인스턴스 유형

t2.micro

버전

Default

설명

-

- az, VPC 설정

## 로드 밸런싱 [Info](#)

아래 옵션을 사용하여 Auto Scaling 그룹을 기존 로드 밸런서 또는 사용자가 정의한 새 로드 밸런서에 연결합니다.

☐ 로드 밸런서 없음

Auto Scaling 그룹에 대한 트래픽은 로드 밸런서가 앞에 있지 않습니다.

☒ 기존 로드 밸런서에 연결

기존 로드 밸런서 중에서 선택합니다.

☐ 새 로드 밸런서에 연결

Auto Scaling 그룹에 연결할 기본 로드 밸런서를 빠르게 생성합니다.

### 기존 로드 밸런서에 연결

Auto Scaling 그룹에 연결할 로드 밸런서를 선택합니다.

☒ 로드 밸런서 대상 그룹에서 선택

이 옵션을 사용하면 Application Load Balancer, Network Load Balancer 또는 Gateway Load Balancer를 연결할 수 있습니다.

☐ Classic Load Balancer에서 선택

#### 기존 로드 밸런서 대상 그룹

Auto Scaling 그룹과 동일한 VPC에 속하는 인스턴스 대상 그룹만 선택할 수 있습니다.

대상 그룹 선택



### 상태 확인

#### 상태 확인 유형 [Info](#)

EC2 Auto Scaling은 상태 확인에 실패한 인스턴스를 자동으로 교체합니다. 로드 밸런싱을 활성화한 경우 항상 활성화된 EC2 상태 확인 외에도 ELB 상태 확인을 활성화할 수 있습니다.

☒ EC2 ☐ ELB

#### 상태 확인 유예 기간

EC2 Auto Scaling이 새 인스턴스가 서비스 상태가 된 후 새 인스턴스에 대한 첫 번째 상태 확인을 수행할 때까지의 시간입니다.

300

초

- 기존 만들어진 로드 밸런서에 연결한다.



### 그룹 크기 - 선택 사항 Info

원하는 용량을 변경하여 Auto Scaling 그룹의 크기를 지정합니다. 최소 및 최대 용량 한도를 지정할 수도 있습니다. 원하는 용량은 한도 범위 내에 있어야 합니다.

원하는 용량

최소 용량

최대 용량

### 크기 조정 정책 - 선택 사항

크기 조정 정책을 사용하여 수요의 변화를 충족하도록 Auto Scaling 그룹의 크기를 동적으로 조정할지 여부를 선택합니다. Info

☐

대상 추적 크기 조정 정책

원하는 결과를 선택하고 크기 조정 정책에 따라 필요한 경우 용량을 추가 및 제거하여 해당 결과를 달성합니다.

☒

없음

### 인스턴스 축소 보호 - 선택 사항

인스턴스 축소 보호

- 원하는 인스턴스 크기, 최소 용량, 최대 용량을 조절 해서 설정한다.
- 대상 추적 크기 조정 정책을 수정한다.

## 7. 프론트엔드 리액트 프로젝트 배포

새로운 프로젝트 freestyle로 설정해서 새로 생성

Git ?

Repositories ?

Repository URL ?

https://lab.ssafy.com/s08-webmobile1-sub2/S08P12A407

Credentials ?

suker800@gmail.com/\*\*\*\*\*

+ Add

고급...

Add Repository

git 프로젝트 설정

☒ Build periodically ?

Schedule ?

H 8 \* \* \*

Would last have run at Tuesday, February 14, 2023 at 8:27:17 AM Korean Standard Time; would next run at Wednesday, February 15, 2023 at 8:27:17 AM Korean Standard Time.

## 매일 아침 8시 27분에 자동 빌드

```
# Use the official Node.js image as the base image
FROM node:18.13-alpine

# Set the working directory
WORKDIR /app

# Copy the package.json and package-lock.json files
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application files
COPY . .

ENV GENERATE_SOURCEMAP=false
ENV NODE_OPTIONS="--max-old-space-size=4096"
# Build the React project
RUN npm run build

# Specify the command to run when the container starts
CMD [ "npm", "run", "serve" ]
```

## 리액트 Dockerfile 설정

```
cd frontend/color-the-rock-app # 프론트 디렉토리로 이동
rm -rf build* # 기존 빌드파일 삭제
sudo chmod 666 /var/run/docker.sock
QUIT_CONTAINER=$(docker ps --filter status=exited -q)
if [ -n "$QUIT_CONTAINER" ]
then
  docker rm $QUIT_CONTAINER
fi
DANGLING_IMAGE=$(docker images -f "dangling=true" -q)
if [ -n "$DANGLING_IMAGE" ]
then
  docker rmi -f $DANGLING_IMAGE # none 태그 이미지 삭제
fi

docker build . -t react-alpine # react 이미지 빌드
CONTAINER=$(docker run -d -p 3000:3000 react-alpine npm run start) # 컨테이너 실행
docker cp $CONTAINER:/app/build ./build # 컨테이너 안에 있는 빌드 파일로 로컬로 복사
docker kill $(docker ps -f ancestor=react-alpine -q) # 도커 컨테이너 종료
tar cvf build.tar ./build # 배포하기 쉽게 tar로 묶는다.
```

## 프론트 배포 스크립트

SSH Server

Name ?

Nginx Server

고급...

Transfers

Transfer Set

Source files ?

frontend/color-the-rock-app/build.tar

Remove prefix ?

frontend/color-the-rock-app

Remote directory ?

Exec command ?

sudo tar xvf build.tar  
sudo cp -f -r ./build/\* /usr/share/nginx/html/  
sudo systemctl reload nginx

All of the transfer fields (except for Exec timeout) support substitution of [Jenkins environment variables](#)

고급...

빌드 파일을 nginx에 전송해야 한다.

build.tar 파일을 nginx 서버에 이동시키고

압축을 푼다음 nginx를 재시작 한다.

## [부록]

### 1. Jenkins 포트 수정하기

- 8080, 8081은 스프링서버의 포트로 사용하기 위해 Jenkins 서버는 8082로 옮겨줌
- 아래 과정을 거쳐 다시 세팅해주면 된다

```
$ sudo chmod 777 /usr/lib/systemd/system/jenkins.service
$ sudo vim /usr/lib/systemd/system/jenkins.service
$ sudo chmod 444 /usr/lib/systemd/system/jenkins.service
$ sudo systemctl daemon-reload
$ sudo service jenkins restart
$ sudo lsof -i -P -n | grep jenkins
```