

HourGoods 포팅메뉴얼

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. 프론트엔드 리액트 프로젝트 배포
- 6-1. [jenkins에 publish over SSH 플러그인 설치](#)
 - 6-2. [frontend용 jenkins item 생성 및 배포](#)
 - 6-3. [nginx 적용](#)

[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 (k8팀.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
```

 아래 SSL 발급은 모든 서버 설정을 마친뒤에 하는 것을 추천합니다.

- Cerbot 설치

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

- 인증서 발급 및 https 적용 (싸피에서 발급받은 도메인 주소 입력 k8a팀번호.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

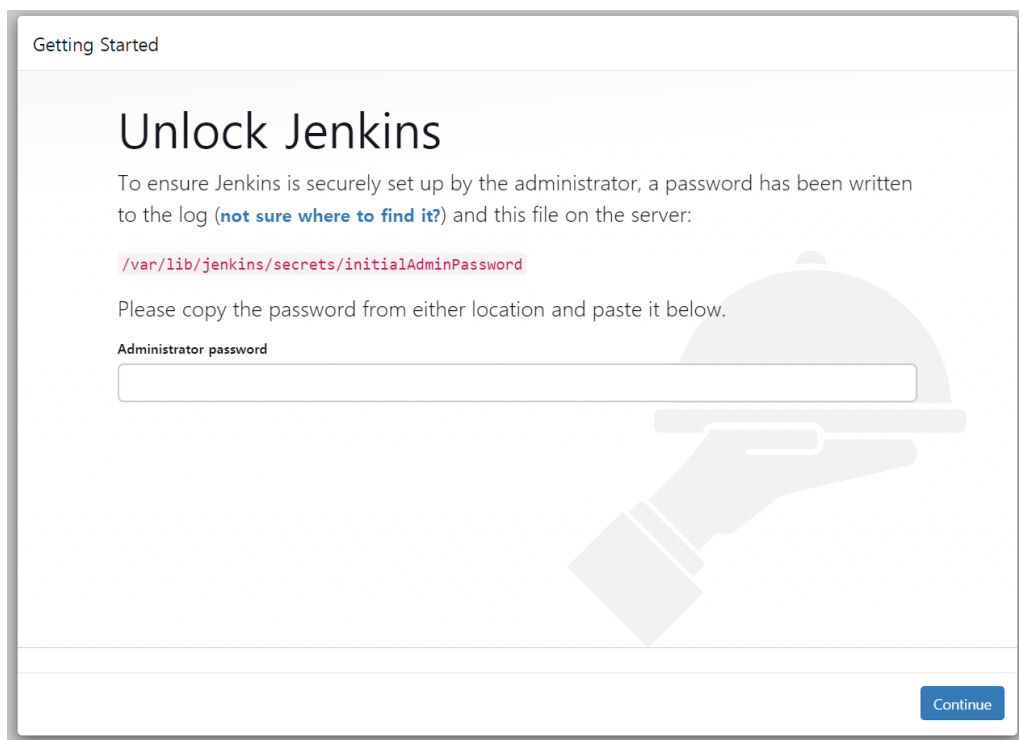
$ sudo apt-get install jenkins

// 확인
$ sudo systemctl status jenkins

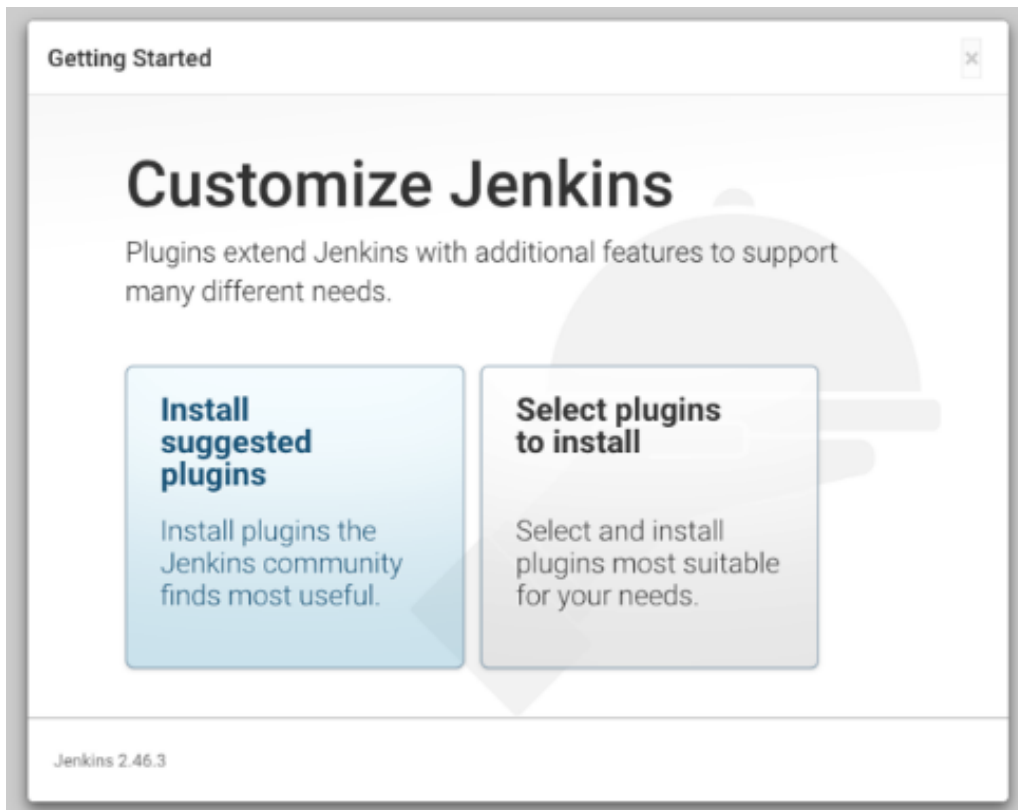
$ sudo systemctl start jenkins

// 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



- [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:

- 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)

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.387.1

Not now
Save and Finish

- 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
```

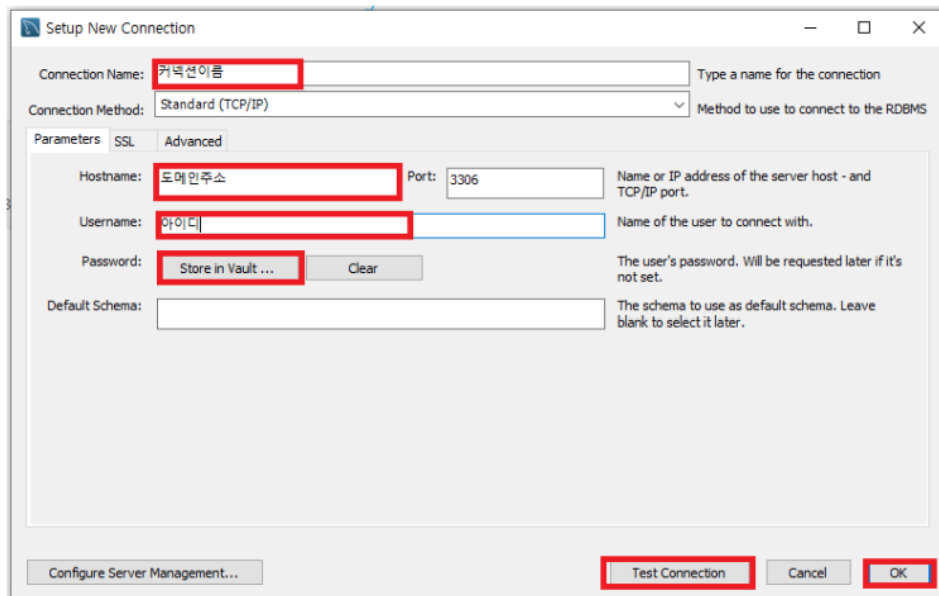
```
...  
// bind-address 를 0.0.0.0으로 변경  
bind-address = 0.0.0.0  
...
```

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
// 위처럼 전부 열어둘 경우, 공격이 들어올 수 있으니, 접근하는 IP만 열어둘 것을 추천

// 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

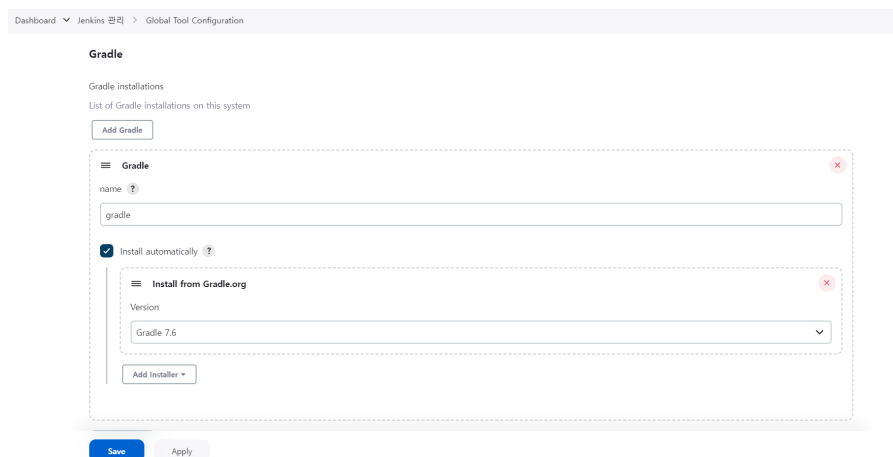
```
$ sudo systemctl status redis-server

$ 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 = "johankong/hourgoods"
        DOCKERHUB_CREDENTIALS = credentials('docker-hub')
    }
    stages {
        stage('Mattermost notification'){
            steps {
                script {
                    try {
                        mattermostSend (
                            color: "#2A42EE",
                            message: "Build STARTED: ${env.JOB_NAME}_Backend #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                        )
                    } catch(e) {
                        currentBuild.result = "FAILURE"
                    }
                }
            }
        }
        stage('Clone') {
            steps {
                git branch: 'backend_release',
                    credentialsId: 'gitlab-clone',
                    url: 'https://lab.ssafty.com/s08-final/S08P31A204'
            }
        }
        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='johankong/*' -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} -f backend/Dockerfile --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 {
                mattermostSend (
                    color: "good",
                    message: "Build SUCCESS: ${env.JOB_NAME}_Backend #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                )
            }
        }
    }
}
```

```

    }
  }
  failure {
    script {
      mattermostSend (
        color: "danger",
        message: "Build FAILED: ${env.JOB_NAME}_Backend #${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/HourGoods/backend/src/main/resources
```

- application-prod.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: update
    show-sql: true
    database-platform: org.hibernate.spatial.dialect.mysql.MySQL56InnoDBSpatialDialect
    properties:
      hibernate:
        format_sql: false
    open-in-view: false
  security:
    user:
      name: ${ADMIN_NAME}
      password: ${ADMIN_PASSWORD}
      roles: ADMIN
  oauth2:
    client:
      registration:
        kakao:
          client-id: ${KAKAO_CLIENT_ID}
          client-secret: ${KAKAO_CLIENT_SECRET}
          redirect-uri: ${KAKAO_REDIRECT_URI}
          authorization-grant-type: authorization_code
          client-authentication-method: POST
          client-name: kakao
          scope:
            - account_email
            - profile_nickname
            - profile_image
      provider:
        kakao:
          authorization-uri: https://kauth.kakao.com/oauth/authorize
          token-uri: https://kauth.kakao.com/oauth/token
          user-info-uri: https://kapi.kakao.com/v2/user/me
          user-name-attribute: id
  servlet:
    multipart:
      max-file-size: 100MB
      max-request-size: 100MB
  redis:
    host: ${REDIS_HOST}
    port: ${REDIS_PORT}
    password: ${REDIS_PASS}
  kopis:
    api-key: ${KOPIS_KEY}
  quartz:
    job-store-type: memory
    threadPool:
      threadCount: 5
  application:
    name: HourGoods

```

```
management:
  info:
    env:
      enabled: true
  endpoints:
    web:
      exposure:
        include: info, health, prometheus
  endpoint:
    health:
      show-details: always
    prometheus:
      enabled: true

notification:
  mattermost:
    enabled: true # mmSender를 사용할 지 여부, false면 알림이 오지 않는다
    webhook-url: ${MM_WEBHOOK_URL} # 위의 Webhook URL을 기입
```

- jenkins 환경 변수

```
# 각 환경 변수에 맞는 값을 적어 .env 파일로 만들어서
# 빌드가 이루어지는 곳에 저장
# /var/lib/jenkins/workspace/HourGoods/backend
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
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/default.d
$ sudo vim port.conf
```

```
set $active_server BLUE;
```

5-4-2. nginx shell script 작성

- jenkins workspace 루트 디렉토리에 switch.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=$(curl -k -L k8a602.p.ssafy.io/port | grep 'BLUE' | wc -l)
echo "현재 구동중인 포트: " "$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 nginxdocker
```

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

```
server {

    index index.html index.htm index.nginx-debian.html;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ /index.html;
    }

    server_name hourgoods.co.kr;
    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/fresh {
        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;
        return 200 '$active_server';
    }

    location ~* ^/(api|swagger-ui|v3/api-docs) {
        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 ~* ^/(ws|topic|pub|app|queue|enter|topic|bidding|hour-bidding) {
        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;
    }

}
```

6. 프론트엔드 리액트 프로젝트 배포

6-1. jenkins에 publish over SSH 플러그인 설치

The screenshot shows the Jenkins 'Plugins' page. On the left, there's a sidebar with 'Updates', 'Available plugins', 'Installed plugins', and 'Advanced settings'. The main area is titled 'Plugins' and has a search bar with 'Publish' entered. Below the search bar, a table lists installed and available plugins. The 'Publish Over SSH' plugin (version 1.24) is marked as installed with a checkmark. Other plugins listed include 'Maven Integration', 'HTML Publisher', 'Pub-Sub "light" Bus', 'Infrastructure plugin for Publish Over X', and 'Slack Notification'. At the bottom, there are buttons for 'Install without restart' and 'Download now and install after restart', along with a note that update information was obtained 19 hours ago.

Install	Name	Release
<input checked="" type="checkbox"/>	Publish Over SSH 1.24 Artifact Uploaders Build Tools Send build artifacts over SSH	1 yr
<input type="checkbox"/>	Maven Integration 3.21 Build Tools This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTS as well as the automated configuration of various Jenkins publishers such as Junix.	1 mo
<input type="checkbox"/>	HTML Publisher 1.31 Build Reports This plugin publishes HTML reports.	7 mo
<input type="checkbox"/>	Pub-Sub "light" Bus 1.17 Library plugins (for use by other plugins) A simple Publish-Subscribe light-weight event bus for Jenkins	8 mo
<input type="checkbox"/>	Infrastructure plugin for Publish Over X 0.22 Send build artifacts somewhere.	5 yr
<input type="checkbox"/>	Slack Notification 664.vc9a_90f8b_c24a_ slack Build Notifiers Integrates Jenkins with Slack, allows publishing build statuses, messages and files to Slack channels.	7 days

6-2. frontend용 jenkins item 생성 및 배포

1. freestyle로 새로운 item 생성

2. 새로운 item > 구성 > 소스 코드 관리

URL - gitlab clone 누르고 나오는 경로를 사용

Credentials - Username : gitlab아이디, password : 발급받은 gitlab access-token

소스 코드 관리

☐ None

☒ Git ?

Repositories ?

Repository URL ?

https://lab.ssfy.com/s08-final/S08P31A204.git

Credentials ?

Add +

Credential used to check out sources.

(from [Git plugin](#))

고급 ▾

Add Repository

3. Dashboard > Jenkins관리 > 시스템 설정 > Publish over SSH

SSH Server

Name ?

자신이 원하는 이름

Hostname ?

서버 ip ex)k8a000.p.ssafy.io

Username ?

서버 username

Remote Directory ?

고급 ^

Edited

☒ Use password authentication, or use a different key ?

Passphrase / Password ?

Path to key ?

Key ?

복여받은 pem key

나머지는 기본 설정 유지

마지막 test configuration > Success 확인

Success

Test Configuration

4. frontend item > 구성 > Build Steps > Execute Shell

```
cd frontend # 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.
```

5. frontend item > 구성 > 빌드 후 조치 > Send build artifacts over SSH

위에서 빌드한 build.tar 파일을 SSH를 통해서 nginx 서버로 옮겨준다.

빌드 후 조치

Send build artifacts over SSH ?

SSH Publishers

SSH Server

Name ?

Remote Server

고급 ▾

Transfers

Transfer Set

Source files ?

frontend/build.tar

Remove prefix ?

frontend

Remote directory ?

Exec command ?

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

Source files : build.tar 가 저장된 위치

Remove prefix : build.tar 를 제외한 앞의 문자

Exec command

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

6-3. nginx 적용

압축해제 후, nginx의 root 위치로 이동시킨 뒤 nginx 재시작

[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
```