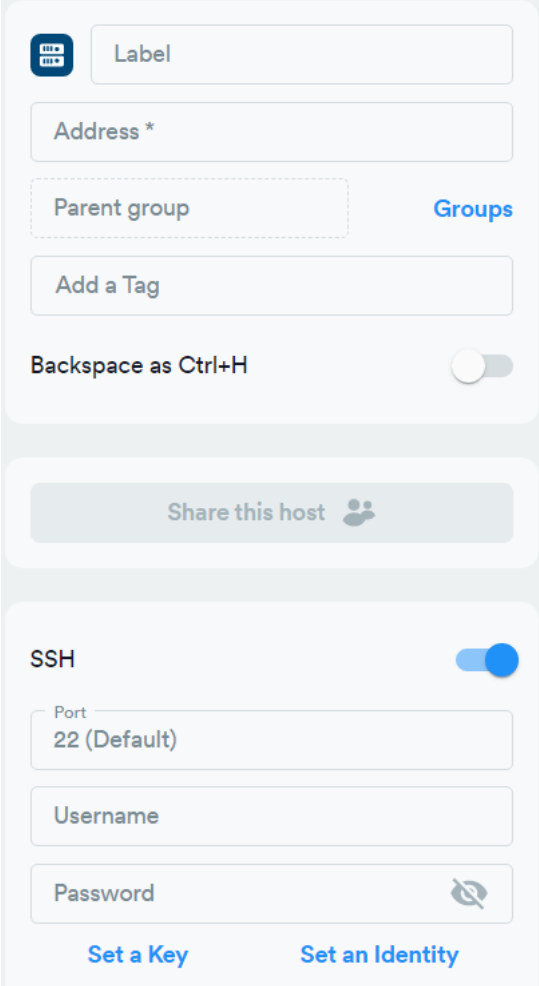
포팅 매뉴얼

1. EC2 초기 서버 설정
    1) MobaXterm 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. Jeknkins 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
7. DB 테이블 생성

1. EC2 초기 서버 설정

1) MobaXterm 이용해 발급받은 EC2 인스턴스 접속

- mobaXterm 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
// 운영체제에서 사용 가능한 패키지들과 그 버전에 대한 정보를 업데이트하는 명령어다.
// 설치되어 있는 패키지를 최신으로 업데이트하는 것이 아닌 설치가능한 리스트를 업데이트하는 것

$ sudo apt-get upgrade
// 운영체제에 apt-get install 명령으로 설치한 패키지들을 최신 버전으로 업그레이드하는 명령어
// apt-get upgrade 명령을 이용하면 apt-get update로 가져온 각 패키지들의 최신 버전에 맞게
// 업그레이드를 한다.
```

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

// lsb-release : 리눅스 배포판 버전확인

$ 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
```

- Cerbot 설치

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

- 인증서 발급 및 https 적용 (싸피에서 발급받은 도메인 주소 입력 `j8a팀번호.p.ssafy.io`

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

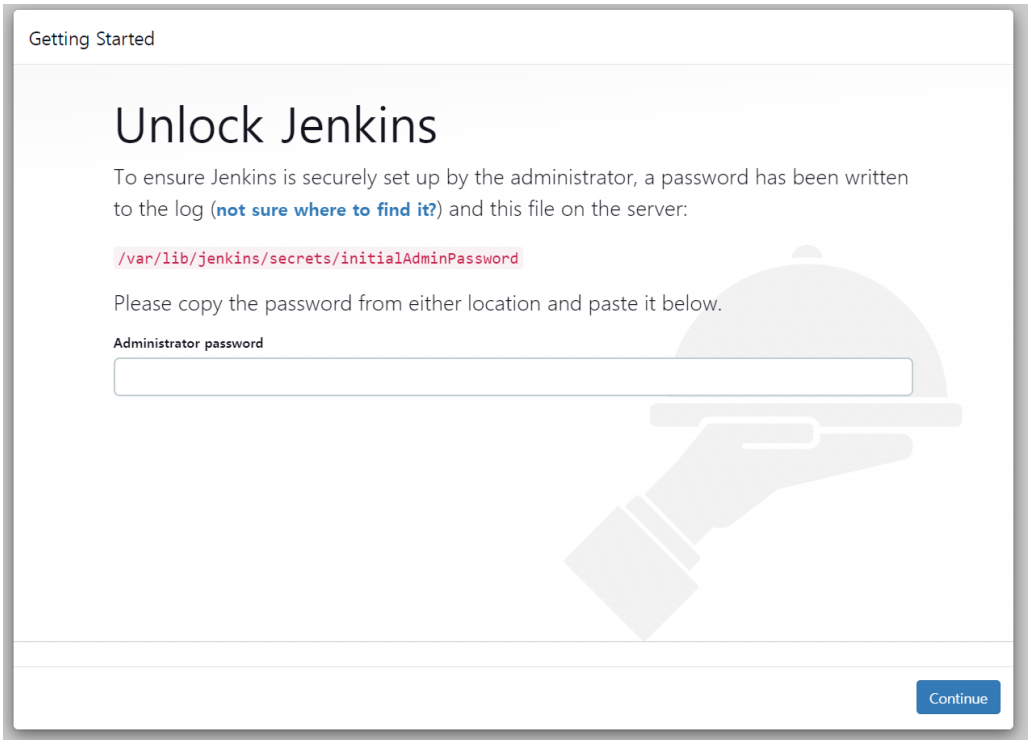
2. Jeknkins CI 환경 구축

1) Jenkins 설치

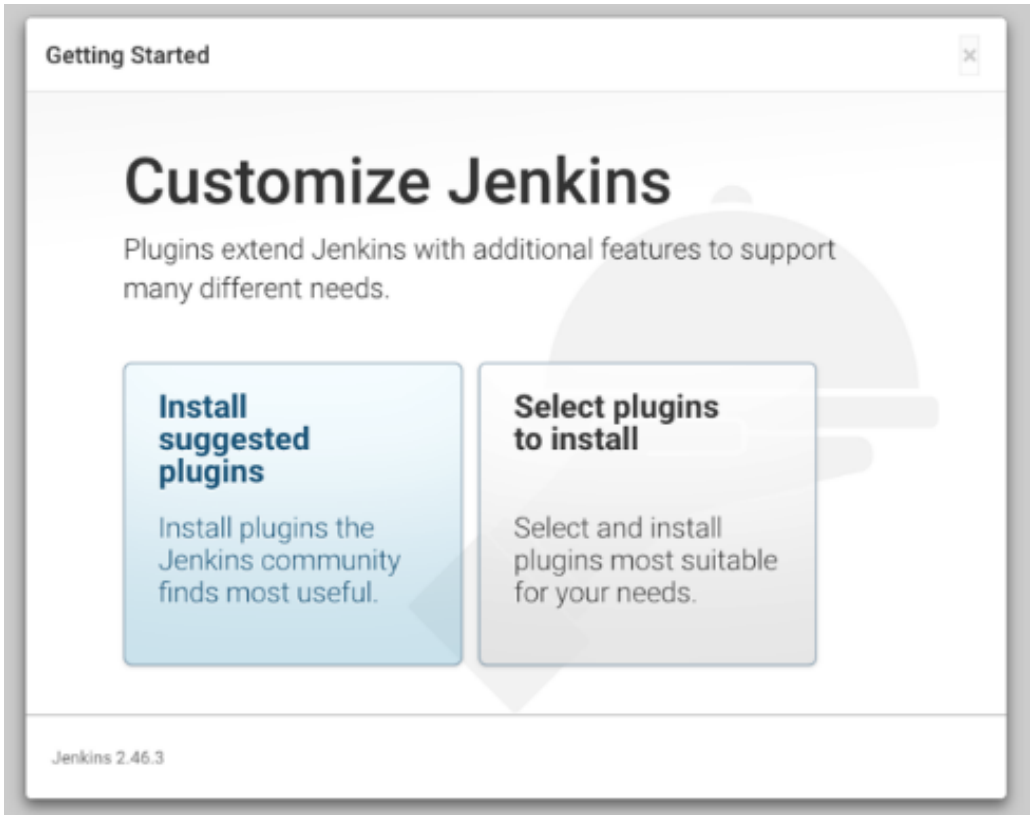
- gitlab backend-develop 브랜치에 변화가 생기면 자동으로 pull 하는 CI 구축
- Jenkins 설치

```
$ 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
```

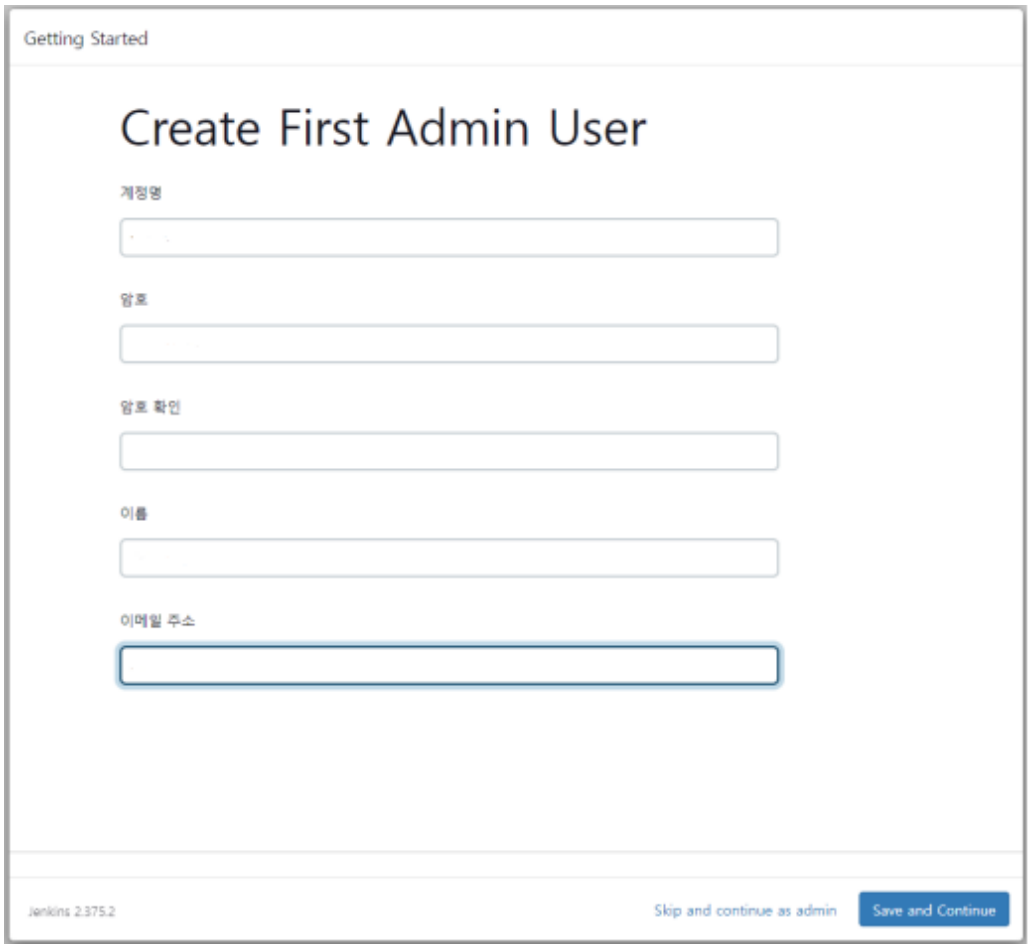
- `http://[도메인]:8080/` 로 접속하여 Jenkins 설정 계속
- 초기 비밀번호 입력하여 접속



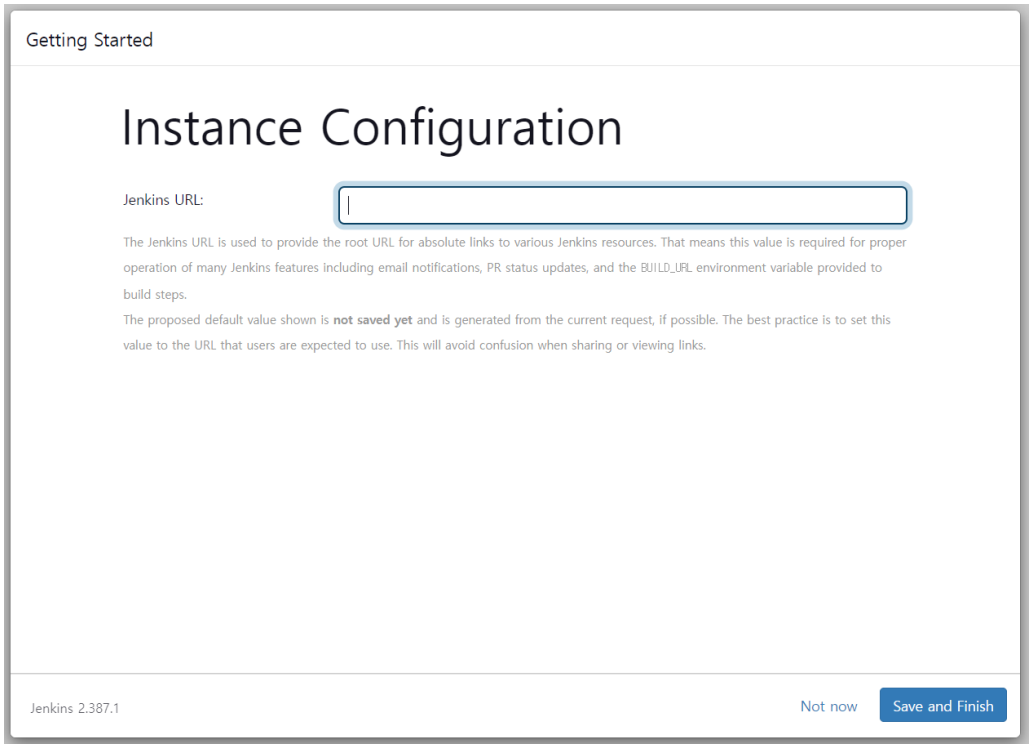
- `Install Suggested plugins` 선택


Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.
Jenkins 2.46.3

- Admin 계정 생성


Getting Started
Create First Admin User
계정명
암호
암호 확인
이름
이메일 주소
Jenkins 2.375.2
Skip and continue as admin
Save and Continue

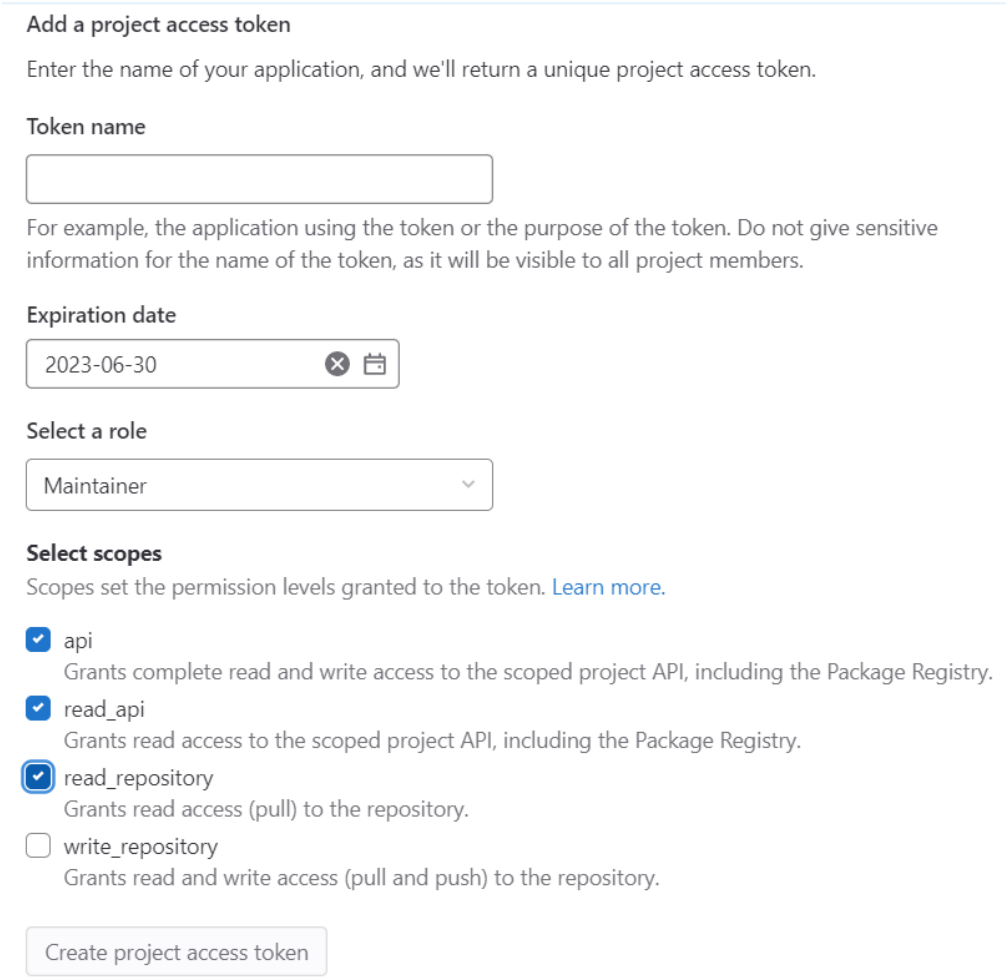
- Jenkins 접속 URL 설정 (`도메인:포트번호`)



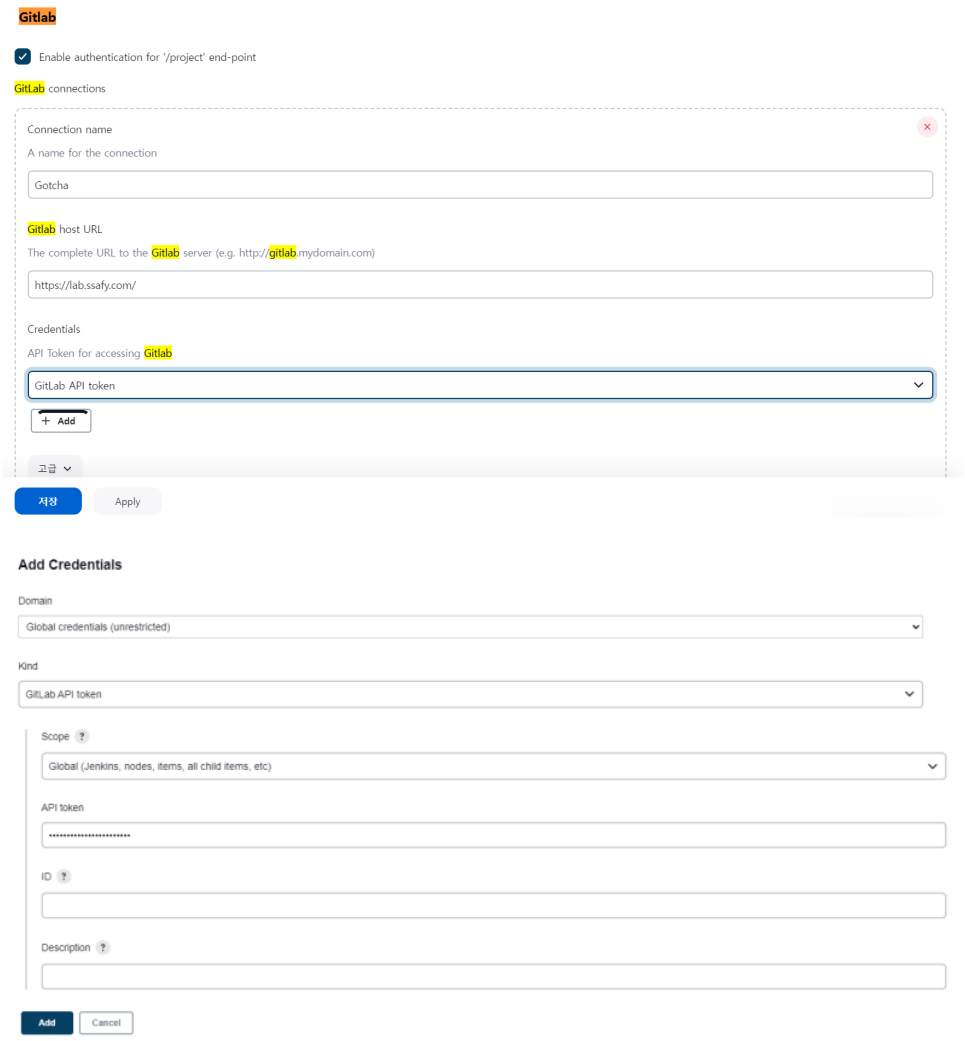
- 추가 플러그인 설치 (DashBoard → Jenkins 관리 → 플러그인 관리 → Available plugins)
- 아래 플러그인 Install without restart 옵션으로 설치
  - Generic Webhook Trigger
  - Gitlab
  - Gitlab API
  - Gitlab Authentication
  - Mattermost Notification
  - Docker Pipeline

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



- Jenkins에 Gitlab 설정
  - Connection name : 원하는 이름 입력
  - Gitlab host URL : 깃랩 메인 주소 입력 (https://lab.ssafy.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



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

- 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 -u root

$ 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
```

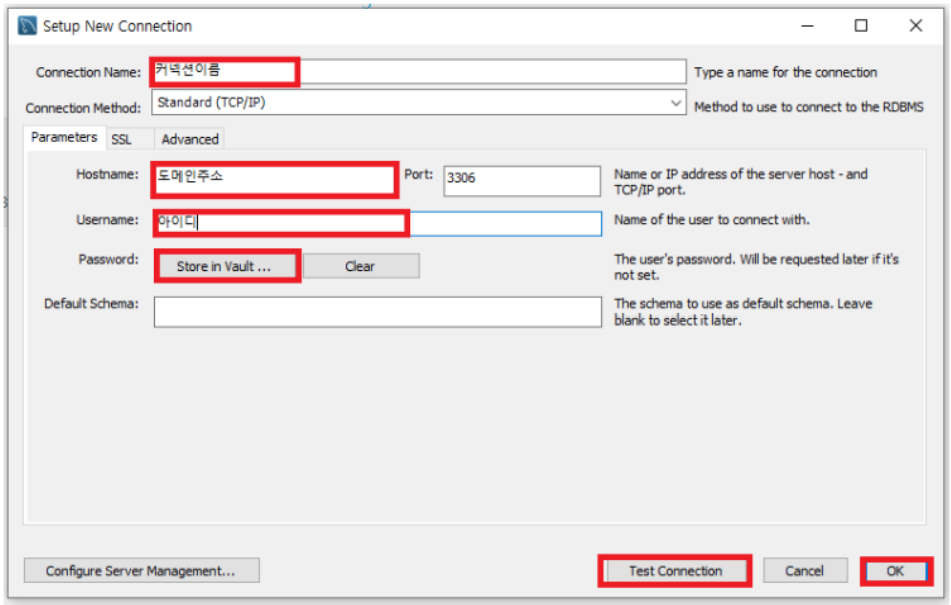
```
...
 // bind-address 를 0.0.0.0으로 변경
 bind-address = 0.0.0.0
...
```

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
```

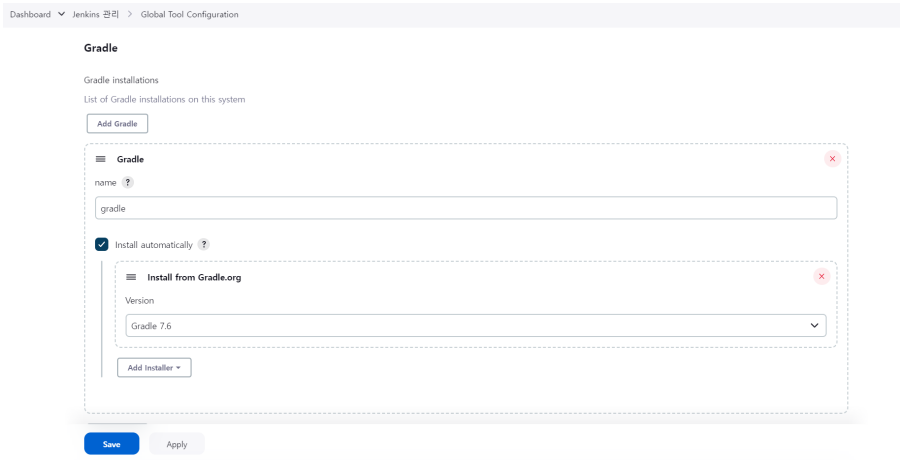
4-4. Redis 설정 확인

```
$ sudo systemctl status redis-server

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

5. Jenkins CD 환경 구축

5-1. Gradle 설치

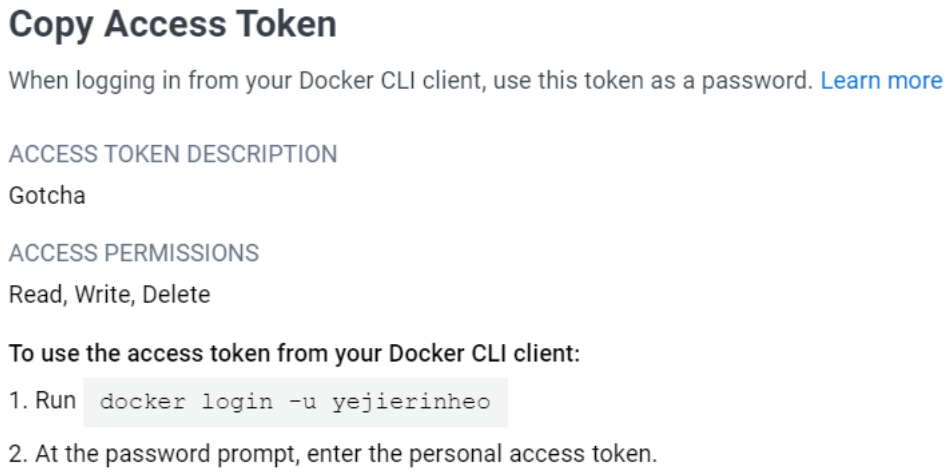
- Jenkins - Jenkins관리 - Global Tool Configuration - Gradle 섹션으로 이동
- Add Gradle에서 해당하는 Gradle 버전을 찾아 세팅 진행



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



5-2-2. Jenkins에서 Credentials 설정

- Jenkins 관리 > Manage Credentials > Domain > 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 passward 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 e아래 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

- 파이프 라인 프로젝트가 있는 폴더 하위에 Dockerfile 생성

```
$ sudo vi /var/lib/jenkins/workspace/[프로젝트명]/backend/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 빌드 해보면 Docker hub에 이미지가 빌드된 것을 확인할 수 있다.

```
pipeline {
    agent any
    environment  {
        DOCKER_REPOSITORY = "yejierinheo/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='yejierinheo/*' -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 Ngninx'){
            steps{
                sh 'bash switch.sh'
            }
        }
    }

// post블록: 특정 stage나 pipeline이 시작되기 이전 또는 이후에 실행 될 confition block을 정의
```

```
    post {
        success {
            script {
                mattermostSend (
                    color: "good",
                    message: "Build SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
                )
            }
        }
        failure {
            script {
                mattermostSend (
                    color: "danger",
                    message: "Build FAILED: ${env.JOB_NAME} #${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-prod.yaml 설정

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            redirect-uri: ${GOOGLE_REDIRECT_URI}
            client-id: ${GOOGLE_CLIENT_ID}
            client-secret: ${GOOGLE_CLIENT_SECRET}
            scope:
              - profile
              - email
          kakao:
            client-id: ${KAKAO_ID}
            client-secret: ${KAKAO_SECRET}
            client-name: ${KAKAO_NAME}
            client-authentication-method: POST
            redirect-uri: ${KAKAO_REDIRECT_URI}
            authorization-grant-type: authorization_code
            scope:
              - profile_nickname
              - account_email
              - 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
  redis:
    host: ${REDIS_HOST}
    port: ${REDIS_PORT}
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: ${DATASOURCE_URL}
    username: ${DATASOURCE_USERNAME}
    password: ${DATASOURCE_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: none
    show-sql: true
    database-platform: org.hibernate.dialect.MySQL8Dialect
    properties:
      hibernate:
        format_sql: true
    open-in-view: false
notification:
  mattermost:
    enabled: true # mmSender를 사용할 지 여부, false면 알림이 오지 않는다
```

```
    webhook-url: ${MM_WEBHOOK_URL} # 위의 Webhook URL을 기입
cloud:
  aws:
    credentials:
      access-key: ${AWS_ACCESS_KEY}
      secret-key: ${AWS_SECRET_KEY}
    region:
      static: ap-northeast-2
    s3:
      bucket: ${BUCKET_NAME}
    stack:
      auto: false
default-image: 'default.jpg'
springdoc:
  swagger-ui:
    disable-swagger-default-url: true
    path: /api/swagger-ui.html
  api-docs:
    path: /api/post-docs
```

- jenkins 환경 변수

```
DATASOURCE_URL=서버URL 값
DATASOURCE_USERNAME=DB유저명
DATASOURCE_PASSWORD=DB패스워드
```

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=80817
    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
$ 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=$(curl -k -L j8a602.p.ssafy.io/port | grep 'BLUE' | wc -l)
echo "현재 구동중인 포트: " "$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/port.conf
echo "> 기존 컨테이너 삭제"
sudo docker kill $(docker ps -qf publish=$CURRENT_PORT)
echo "> nginx 재시작"
sudo systemctl reload nginx
```

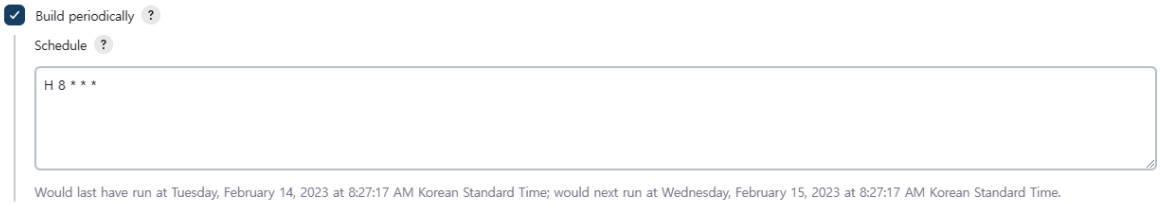
- /etc/nginx/sites-enabled/deafult에서 수정

```
server {
        server_name  colortherock.com;
        root         /usr/share/nginx/index.html;

        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. 프론트엔드 리액트 프로젝트 배포

새로운 프로젝트 freestyle로 설정해서 새로 생성

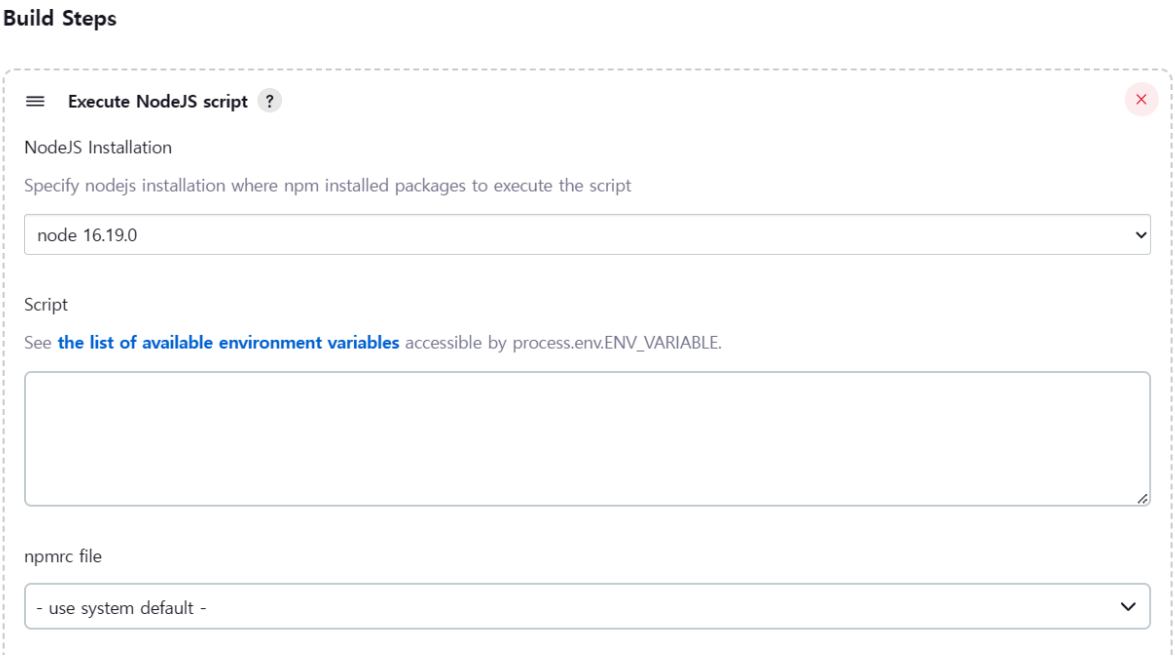
git 프로젝트 설정



매일 아침 8시 27분에 자동 빌드

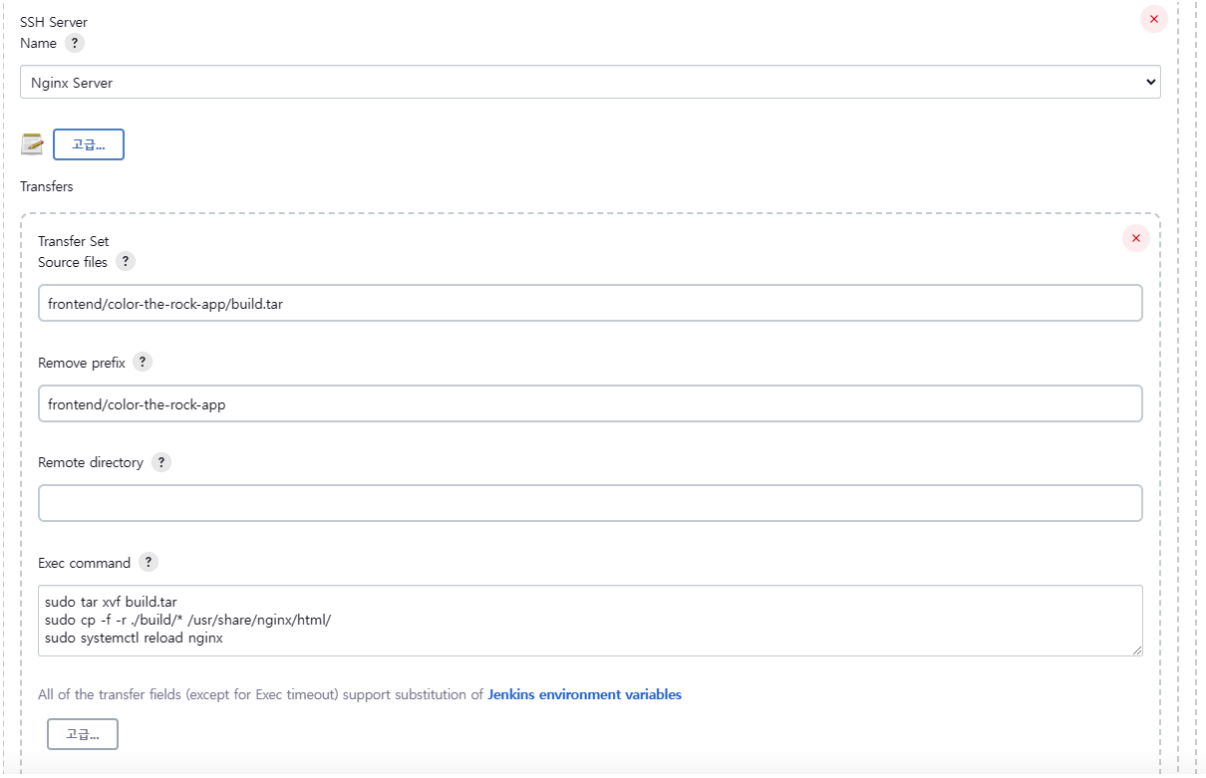
```
# Use the official Node.js image as the base image
FROM node:18.13-alpine

# 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



빌드 파일을 nginx에 전송해야 한다.

buid.tar 파일을 nginx 서버에 이동시키고

압축을 푼다음 nginx를 재시작 한다.

7. DB 테이블 생성

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema deployz
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema deployz
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `deployz` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci ;
USE `deployz` ;

-- -----------------------------------------------------
-- Table `deployz`.`member`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`member` (
  `idx` BIGINT NOT NULL AUTO_INCREMENT,
  `account` VARCHAR(30) NULL DEFAULT NULL,
  `deleted_flag` BIT(1) NULL DEFAULT b'0',
  `password` VARCHAR(100) NULL DEFAULT NULL,
  `personal_access_token` VARCHAR(100) NULL DEFAULT NULL,
  `profile_image` VARCHAR(100) NULL DEFAULT NULL,
  PRIMARY KEY (`idx`))
ENGINE = InnoDB
AUTO_INCREMENT = 2
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`project`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`project` (
  `idx` BIGINT NOT NULL AUTO_INCREMENT,
  `deleted_flag` BIT(1) NOT NULL DEFAULT b'0',
  `description` VARCHAR(100) NULL DEFAULT NULL,
  `image_path` VARCHAR(100) NULL DEFAULT NULL,
  `project_name` VARCHAR(50) NULL DEFAULT NULL,
  `member_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FKnxihqwa34t025tkklw5h9oeii` (`member_idx` ASC) VISIBLE,
  CONSTRAINT `FKnxihqwa34t025tkklw5h9oeii`
    FOREIGN KEY (`member_idx`)
    REFERENCES `deployz`.`member` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`item`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`item` (
  `idx` BIGINT NOT NULL AUTO_INCREMENT,
  `branch_name` VARCHAR(50) NULL DEFAULT NULL,
  `build_version` VARCHAR(50) NULL DEFAULT NULL,
  `deleted_flag` BIT(1) NULL DEFAULT b'0',
  `framework_type` VARCHAR(100) NULL DEFAULT NULL,
  `java_version` VARCHAR(10) NULL DEFAULT NULL,
  `name` VARCHAR(20) NULL DEFAULT NULL,
  `port_number` BIGINT NULL DEFAULT NULL,
  `target_folder_path` VARCHAR(100) NULL DEFAULT NULL,
  `project_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FKmo1vswy80lmt96dhuputpmarj` (`project_idx` ASC) VISIBLE,
  CONSTRAINT `FKmo1vswy80lmt96dhuputpmarj`
    FOREIGN KEY (`project_idx`)
    REFERENCES `deployz`.`project` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`build_history`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`build_history` (
  `idx` BIGINT NOT NULL AUTO_INCREMENT,
  `deleted_flag` BIT(1) NOT NULL DEFAULT b'0',
  `message` LONGTEXT NULL DEFAULT NULL,
  `register_time` DATETIME(6) NULL DEFAULT NULL,
  `status` VARCHAR(20) NULL DEFAULT NULL,
  `item_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FKfsrgkqib6eygh24fg9l8wx593` (`item_idx` ASC) VISIBLE,
  CONSTRAINT `FKfsrgkqib6eygh24fg9l8wx593`
    FOREIGN KEY (`item_idx`)
    REFERENCES `deployz`.`item` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`deploy`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`deploy` (
  `idx` BIGINT NOT NULL,
  `deleted_flag` BIT(1) NOT NULL DEFAULT b'0',
  `last_modified_date` DATETIME(6) NULL DEFAULT NULL,
  `register_time` DATETIME(6) NULL DEFAULT NULL,
  `status` VARCHAR(255) NULL DEFAULT NULL,
  `item_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FK8iunx3bylv6mjk7fk63bvy9hx` (`item_idx` ASC) VISIBLE,
  CONSTRAINT `FK8iunx3bylv6mjk7fk63bvy9hx`
    FOREIGN KEY (`item_idx`)
    REFERENCES `deployz`.`item` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `deployz`.`git_config`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`git_config` (
  `idx` BIGINT NOT NULL AUTO_INCREMENT,
  `deleted_flag` BIT(1) NULL DEFAULT b'0',
  `git_access_token` VARCHAR(200) NULL DEFAULT NULL,
  `host_url` VARCHAR(100) NULL DEFAULT NULL,
  `project_id` INT NOT NULL,
  `repository_url` VARCHAR(200) NULL DEFAULT NULL,
  `project_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FKap3g9hvn5whby8vgffw0pemkb` (`project_idx` ASC) VISIBLE,
  CONSTRAINT `FKap3g9hvn5whby8vgffw0pemkb`
    FOREIGN KEY (`project_idx`)
    REFERENCES `deployz`.`project` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`git_history`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`git_history` (
  `idx` BIGINT NOT NULL,
  `branch_name` VARCHAR(50) NULL DEFAULT NULL,
  `deleted_flag` BIT(1) NOT NULL DEFAULT b'0',
  `event_date` DATETIME(6) NULL DEFAULT NULL,
  `message` VARCHAR(100) NULL DEFAULT NULL,
  `git_config_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FKpjioyt6f0t2hapqglvj9htgpq` (`git_config_idx` ASC) VISIBLE,
  CONSTRAINT `FKpjioyt6f0t2hapqglvj9htgpq`
    FOREIGN KEY (`git_config_idx`)
    REFERENCES `deployz`.`git_config` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`git_token`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`git_token` (
  `idx` BIGINT NOT NULL AUTO_INCREMENT,
  `branch_name` VARCHAR(50) NULL DEFAULT NULL,
  `deleted_flag` BIT(1) NOT NULL DEFAULT b'0',
  `secret_token` VARCHAR(200) NULL DEFAULT NULL,
  `git_config_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FK85dfnmtojtadsqy5u50syrly1` (`git_config_idx` ASC) VISIBLE,
  CONSTRAINT `FK85dfnmtojtadsqy5u50syrly1`
    FOREIGN KEY (`git_config_idx`)
    REFERENCES `deployz`.`git_config` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`hibernate_sequence`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`hibernate_sequence` (
  `next_val` BIGINT NULL DEFAULT NULL)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`nginx_config`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`nginx_config` (
  `idx` BIGINT NOT NULL AUTO_INCREMENT,
  `deleted_flag` BIT(1) NOT NULL DEFAULT b'0',
  `domain` VARCHAR(50) NULL DEFAULT NULL,
  `ssl_certificate` VARCHAR(100) NULL DEFAULT NULL,
  `ssl_certificate_key` VARCHAR(100) NULL DEFAULT NULL,
  `project_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FKstisl98gcceca0k8ohhy0dw8b` (`project_idx` ASC) VISIBLE,
  CONSTRAINT `FKstisl98gcceca0k8ohhy0dw8b`
    FOREIGN KEY (`project_idx`)
    REFERENCES `deployz`.`project` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`project_state`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`project_state` (
  `idx` BIGINT NOT NULL,
  `deleted_flag` BIT(1) NOT NULL DEFAULT b'0',
  `last_failure_date` DATETIME(6) NULL DEFAULT NULL,
  `register_time` DATETIME(6) NULL DEFAULT NULL,
  `status` VARCHAR(255) NULL DEFAULT NULL,
  `step` VARCHAR(20) NULL DEFAULT NULL,
  `project_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FK5e6l8mbi70p4332huwfgjqsv9` (`project_idx` ASC) VISIBLE,
  CONSTRAINT `FK5e6l8mbi70p4332huwfgjqsv9`
    FOREIGN KEY (`project_idx`)
    REFERENCES `deployz`.`project` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `deployz`.`proxy_config`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `deployz`.`proxy_config` (
  `idx` BIGINT NOT NULL AUTO_INCREMENT,
  `deleted_flag` BIT(1) NOT NULL DEFAULT b'0',
  `path_name` VARCHAR(50) NULL DEFAULT NULL,
  `path_url` VARCHAR(50) NULL DEFAULT NULL,
  `nginx_config_idx` BIGINT NULL DEFAULT NULL,
  PRIMARY KEY (`idx`),
  INDEX `FKcc3wdjuj3e1oy344u3au89sl5` (`nginx_config_idx` ASC) VISIBLE,
  CONSTRAINT `FKcc3wdjuj3e1oy344u3au89sl5`
    FOREIGN KEY (`nginx_config_idx`)
    REFERENCES `deployz`.`nginx_config` (`idx`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```