포팅 메뉴얼 (BOOKIZ)

BOOKIZ

SSAFY 서울캠퍼스 7기 특화 A103

하미르(팀장), 김민지, 김수환, 이도경, 이종은, 최병성

2022.08.22 ~ 2022.10.07

목차

1. 프로젝트 기술 스택

1. React

- Visual Studio Code 1.71.2
- HTML5, CSS3, Javascript(ES6)

package.json

```
{
  "name": "bookiz_front",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^0.27.2",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-icons": "^4.4.0",
    "react-router-dom": "^6.4.0",
    "react-scripts": "5.0.1",
    "react-speech-recognition": "^3.9.1",
    "styled-components": "^5.3.5",
    "swiper": "^8.4.2",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
```

```
    }
}
```

2. Spring (gradle)

- IntelliJ 2022.2.1 Ultimate Edition
- java8

build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.7.3'
    id 'io.spring.dependency-management' version '1.0.13.RELEASE'
    id 'java'
}

group = 'com.ssafy'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation group: 'org.modelmapper', name: 'modelmapper', version: '2.3.8'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation group: 'org.json', name: 'json', version: '20090211'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'mysql:mysql-connector-java'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

tasks.named('test') {
    useJUnitPlatform()
}
```

3. Django

- Visual Studio Code 1.71.2
- python 3.7

requirements.txt

```
asgiref==3.5.2
certifi==2022.9.24
charset-normalizer==2.1.1
click==8.1.3
colorama==0.4.5
coreapi==2.3.3
coreschema==0.0.4
Django==3.2.15
django-cors-headers==3.13.0
django-rest-swagger==2.2.0
djangorestframework==3.13.1
gTTS==2.2.4
idna==3.4
importlib-metadata==4.12.0
itypes==1.2.0
Jinja2==3.1.2
MarkupSafe==2.1.1
openapi-codec==1.3.2
playsound==1.2.2
pytz==2022.2.1
requests==2.28.1
simplejson==3.17.6
six==1.16.0
SpeechRecognition==3.8.1
sqlparse==0.4.2
typing_extensions==4.3.0
uritemplate==4.1.1
urllib3==1.26.12
wincertstore==0.2
zipp==3.8.1
```

4. 데이터베이스

- mysql 8.0.30

5. 인프라/서버

- AWS EC2
- nginx 1.18.0

2. 빌드 & 실행

1. React

```
// 빌드
$ npm install
$ npm run build


// 실행
$ npm install
$ npm run server
```

2. Spring

```
// 빌드
$ ./gradlew build
```

3. Django

```
// 빌드
$ pip freeze > requirements.txt

// 실행
$ python -m venv venv // 가상환경이름
$ source venv/Scripts/activate
$ pip install -r requirements.txt
$ python manage.py runserver
```

3. 배포가이드

1. 개요

Pem key 하나가 주어진 상황에서 Spring boot, React, Django로 진행하는 프로젝트의 CI/CD 과정을 기록합니다.

AWS EC2, MobaXterm, Docker, Nginx, Jenkins, Gitlab을 사용합니다.

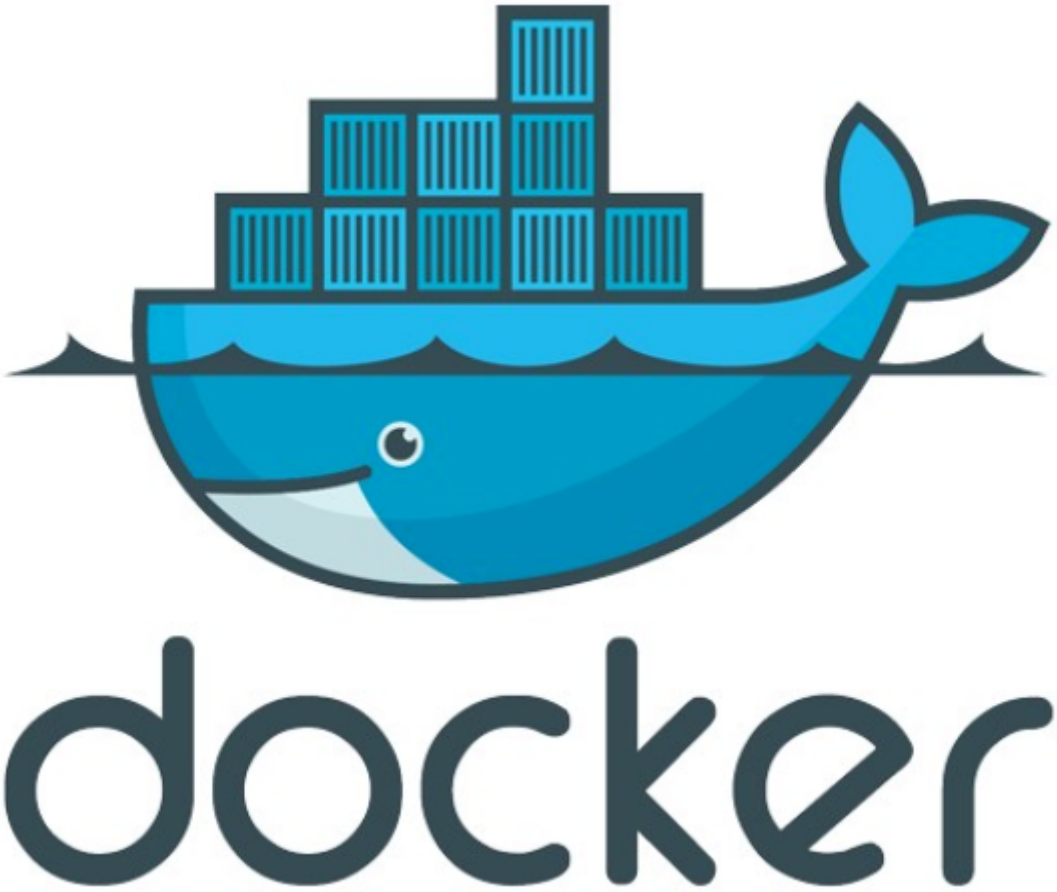
1. AWS EC2

AWS는 아마존 웹 서비스이고, 클라우드 컴퓨팅 서비스를 제공합니다. 그 중 EC2란 제품을 사용합니다.

2. MobaXterm

MobaXterm은 EC2 환경 터미널을 제공하는 툴입니다.

3. Docker



도커는 가상화 기술입니다. 비슷한 비교 대상으로 Anaconda(가상환경), Virtual Machine(가상머신)이 있습니다.

아나콘다는 로컬 PC에 종속되기 때문에 옆자리 PC에선 사용이 불가합니다. 반면, 도커의 컨테이너는 로컬 환경에 종속되지 않습니다.

Virtual Machine은 Hypervisor라는 기반 위에 Guest OS가 실행되는 개념. Guest OS는 Host Operating System의 메모리와 물리적 장치를 직접 접근하여 사용하므로 매우 느립니다.

도커는 컨테이너 내부에서 실행하기 때문에 Host Operating System에 접근할 필요가 없어서 효율적입니다.

요약 : 도커는 필요한 소프트웨어를 컨테이너에 담아 독립적으로 실행할 수 있습니다.

4. Nginx

웹서버 소프트웨어. 가볍고 성능이 좋습니다. 아파치의 C10K문제(하나의 웹서버에 1만개 이상의 클라이언트 접속을 처리하지 못하는 문제)를 해결하기 위해 러시아의 Igor Sysoev라는 개발자에 의해 2002년에 개발됐습니다.

Event Driven(비동기처리방식) 요청이 들어오면 어떤 동작을 해야하는지만 알려주고 다른 요청을 처리하는 방식.

CPU와 관계없이 모든 입출력을 전부 Event Listener로 전달하기 때문에 흐름이 끊기지 않고 응답이 빠르게 진행되어 1개의 프로세스로 보다 더 빠른 작업이 가능합니다.

새로운 요청이 들어와도 새로운 프로세스, 쓰레드를 생성하지 않기에 생성 비용이 없고, 적은 자원으로 효율적인 운영이 가능합니다.

구조 : 하나의 Master Process와 다수의 Worker Process로 구성되어 실행된다.

Master Process : 설정 파일을 읽고, 유효성 검사 및 Worker Process 관리 Worker Process : 모든 요청 처리

요약 : Nginx는 들어오는 요청을 적절한 목적지로 이동시켜주고, 로드밸런싱을 위해 사용합니다.

5. Jenkins

젠킨스는 Git과 연동해서 자동으로 빌드와 배포를 도와줍니다.

6. Gitlab

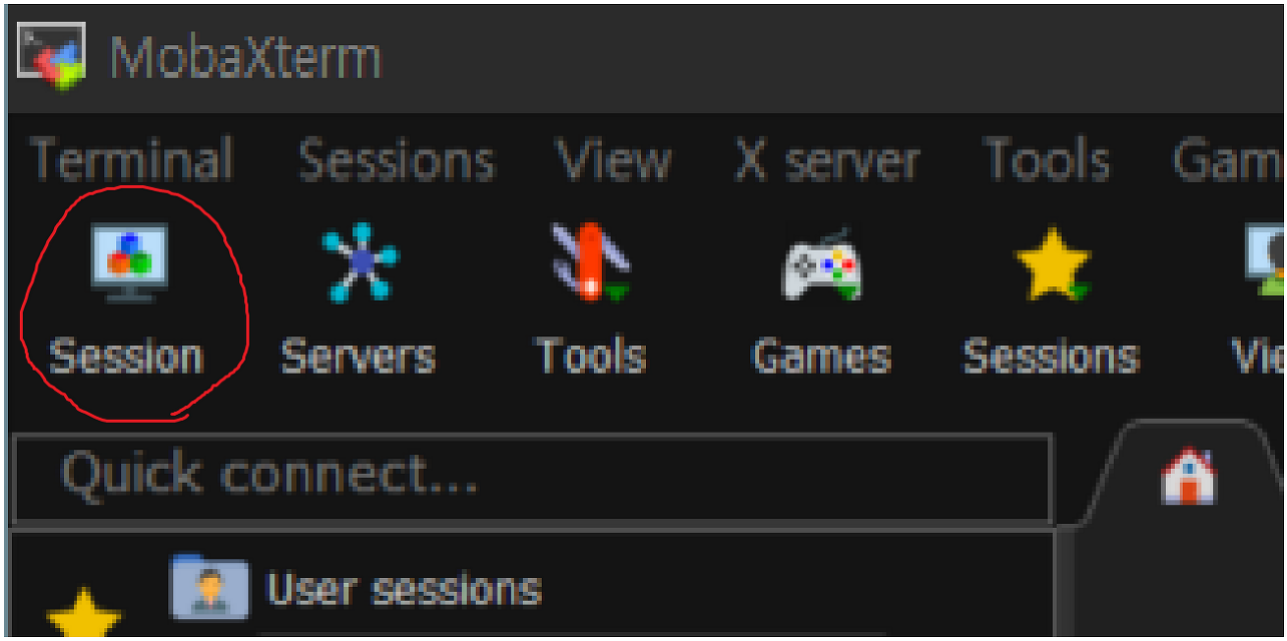
깃 저장소 및 CI/CD, 이슈 추적, 보안성 테스트 등의 기능을 갖춘 웹 기반의 데브옵스 플랫폼입니다.

(출처: https://ko.wikipedia.org/wiki/%EA%B9%83%EB%9E%A9)

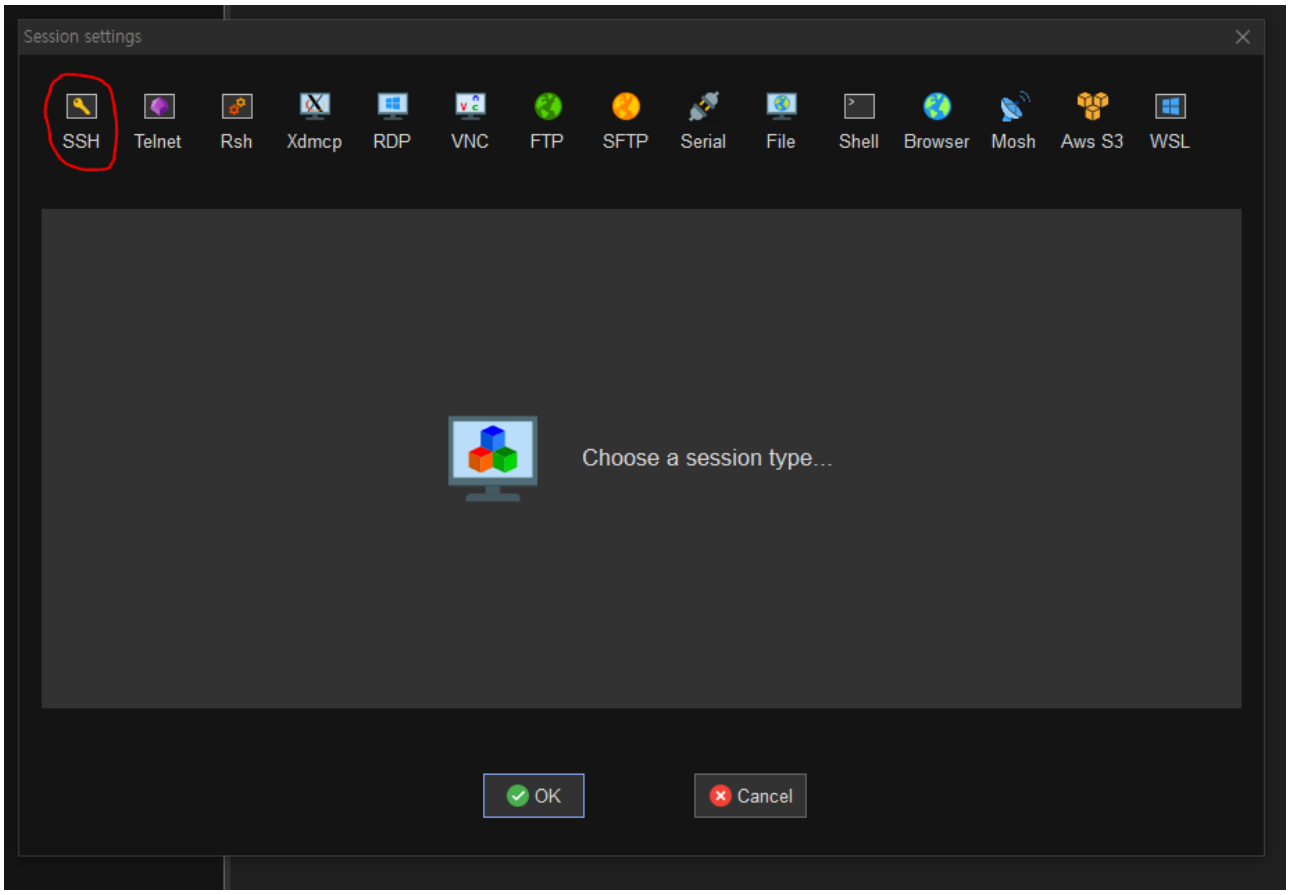
2. MobaXterm 설치 & EC2 접속

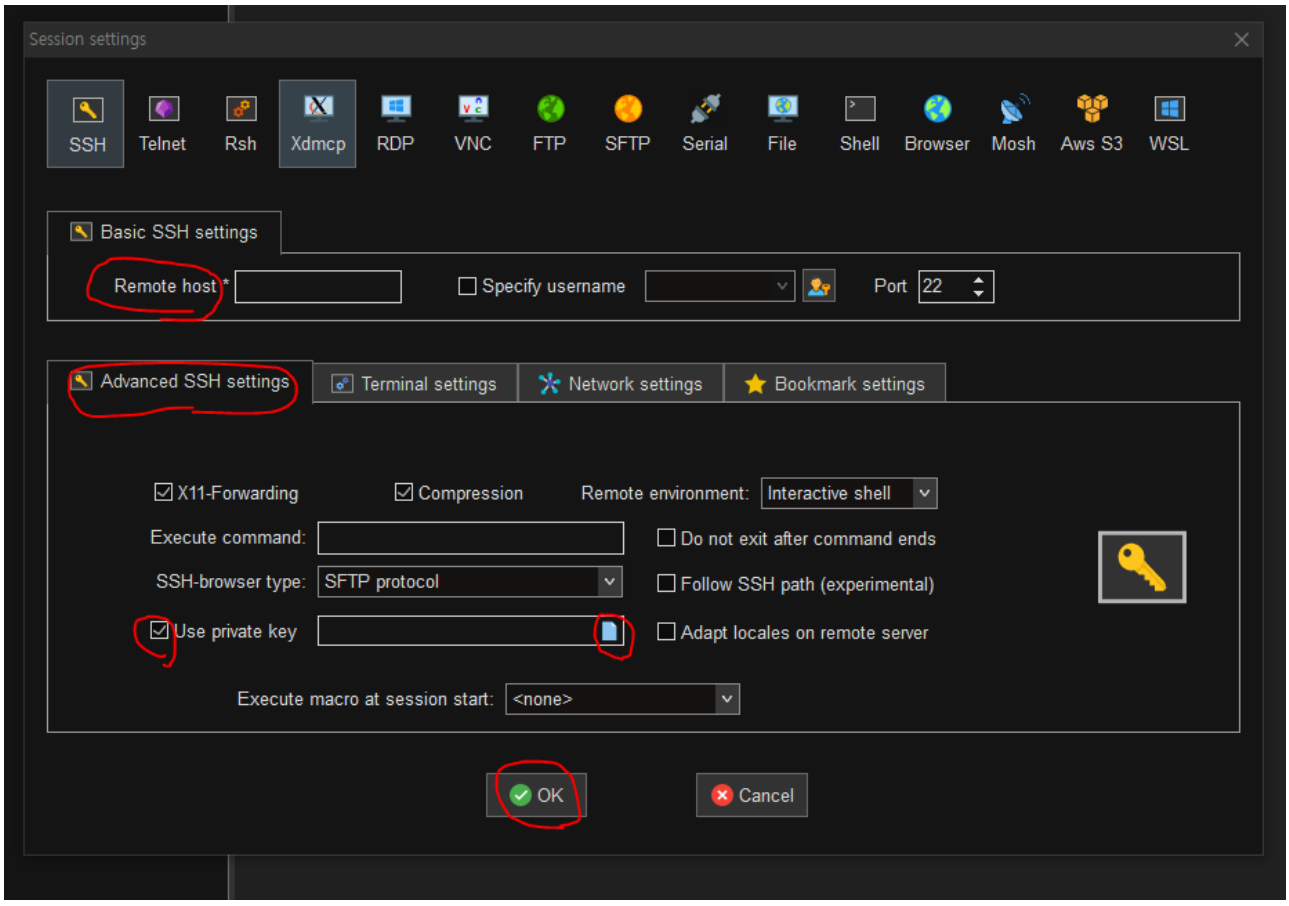
1. MobaXterm 설치 (https://mobaxterm.mobatek.net/download.html)

2. Session 클릭



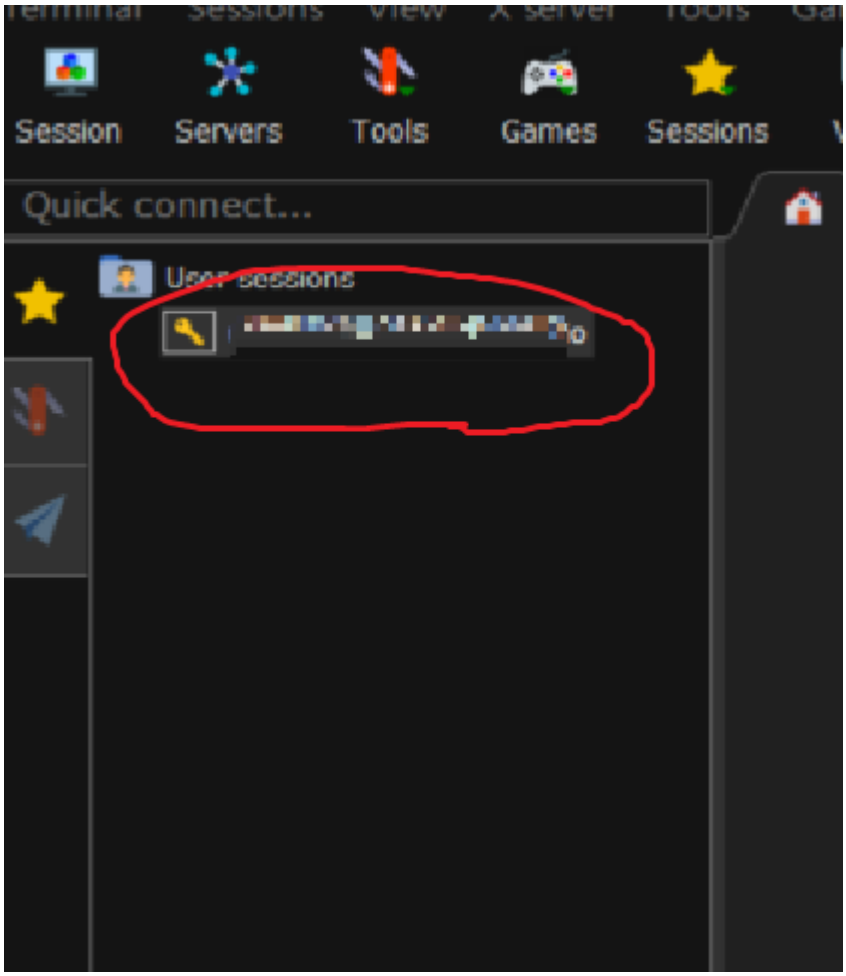
3. Session settings

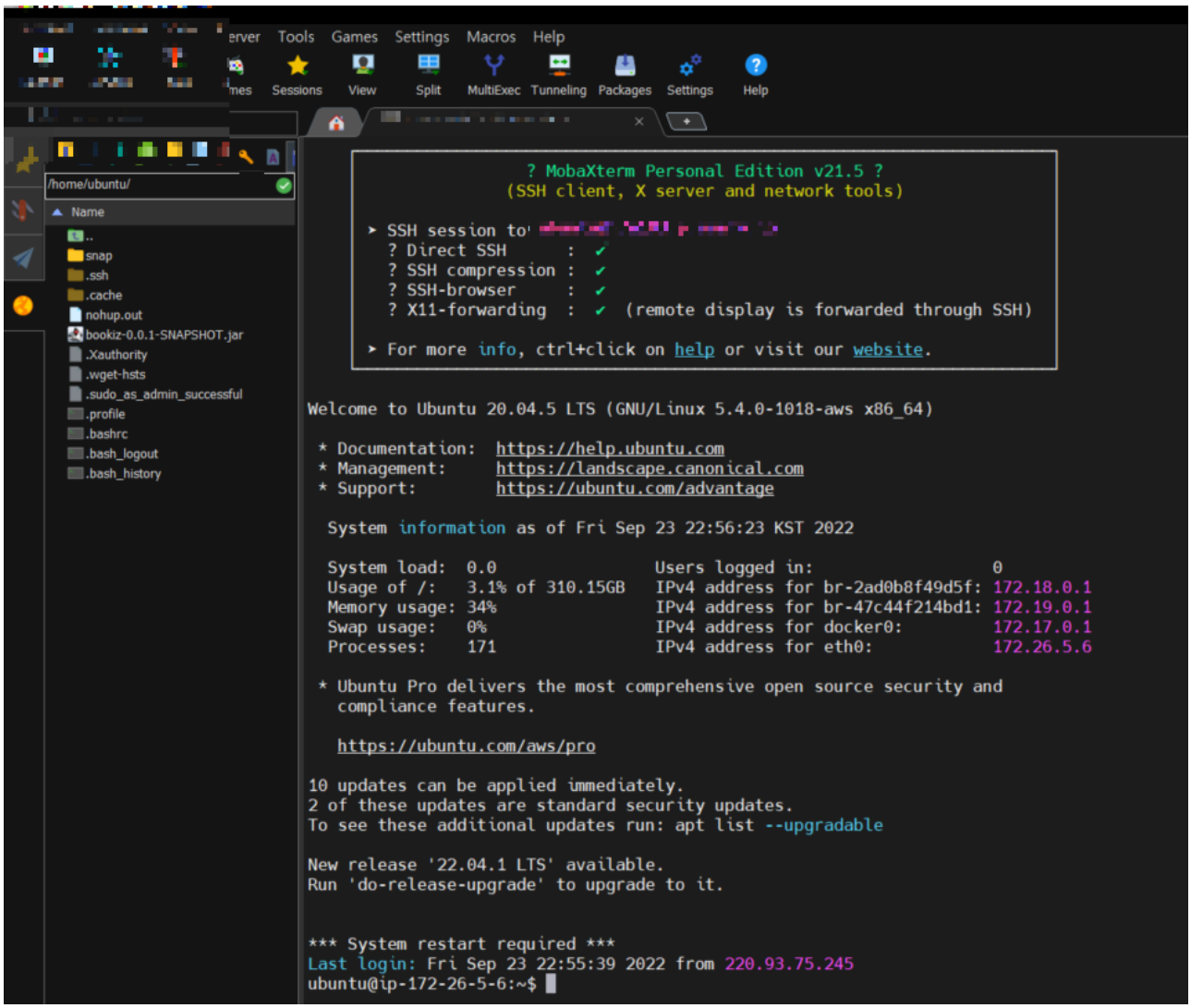




Remote host : ubuntu@<domain ip주소> ex) ubuntu@l9z123.q.happy.ko

Use private key : .pem 파일이 저장된 경로로 이동해서 .pem 파일 선택


Session
Servers
Tools
Games
Sessions
Quick connect...
User sessions


Tools Games Settings Macros Help
Sessions View Split MultiExec Tunneling Packages Settings Help
/home/ubuntu/
Name
snap
.ssh
.cache
nohup.out
bookiz-0.0.1-SNAPSHOT.jar
.Xauthority
.wget-hsts
.sudo_as_admin_successful
.profile
.bashrc
.bash_logout
.bash_history
? MobaXterm Personal Edition v21.5 ?
(SSH client, X server and network tools)
? Direct SSH : ✓
? SSH compression : ✓
? SSH-browser : ✓
? X11-forwarding : ✓ (remote display is forwarded through SSH)
For more info, ctrl+click on help or visit our website.
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-1018-aws x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
System information as of Fri Sep 23 22:56:23 KST 2022
System load: 0.0
Usage of /: 3.1% of 310.15GB
Memory usage: 34%
Swap usage: 0%
Processes: 171
Users logged in: 0
IPv4 address for br-2ad0b8f49d5f: 172.18.0.1
IPv4 address for br-47c44f214bd1: 172.19.0.1
IPv4 address for docker0: 172.17.0.1
IPv4 address for eth0: 172.26.5.6
* Ubuntu Pro delivers the most comprehensive open source security and compliance features.
https://ubuntu.com/aws/pro
10 updates can be applied immediately.
2 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable
New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
*** System restart required ***
Last login: Fri Sep 23 22:55:39 2022 from 220.93.75.245
ubuntu@ip-172-26-5-6:~$

OK !

EC2 환경 접속 성공!

3. EC2에 Docker & Nginx 설치

MobaXterm으로 EC2 환경에 접속하고 진행

1. EC2 서버 시간 설정

```
$ sudo timedatectl set-timezone ‘Asia/Seoul’
```

2. 우분투 패키지 정보 업데이트

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

3. Docker 설치

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

아래와 같은 Error 발생한 경우

```
E: Package 'docker-ce' has no installation candidate
E: Unable to locate package docker-ce-cli`
E: Unable to locate package containerd.io`
E: Couldn't find any package by glob 'containerd.io'
E: Couldn't find any package by regex 'containerd.io'
E: Unable to locate package docker-compose-plugin
```

아래 명령 순서대로 실행

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
-
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-
plugin
```

4. Nginx 설치

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install nginx
```

5. 업로드한 이미지를 EC2에 저장하기 위한 설정 (선택)

```
$ sudo docker volume create images
```

추후 spring boot 컨테이너를 생성할 때, images 볼륨과 마운트하기 위해 미리 만듭니다.

만약, 업로드한 이미지를 EC2에 저장할 일이 없다면 생략해도 됩니다.

4. MySQL 컨테이너 생성 & Workbench 연동

1. mysql 설치

```
// 프로젝트에서 사용중인 mysql 버전
$ sudo docker pull mysql:8.0.30
```

```
// 설치 확인
$ sudo docker images
```

```
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
mysql        8.0.30    43fcfca0776d   7 days ago   449MB
```

2. mysql 컨테이너 띄우기

```
$ sudo docker run --name bookiz_mysql -e
MYSQL_ROOT_PASSWORD=yourrootpassword -d -p 3306:3306 mysql:8.0.30
```

- --name <컨테이너 이름>: 생성하는 컨테이너의 이름 정해주는 옵션
- -e MYSQL_ROOT_PASSWORD=<mysql root계정 비밀번호 설정> : 환경변수 설정 옵션
- -d : 컨테이너를 백그라운드에서 실행한다는 옵션
- -p <호스트 포트번호>:<컨테이너 포트번호> : 호스트와 컨테이너의 포트번호 설정
- mysql:8.0.30 : 지금 생성할 컨테이너는 mysql 이미지를 사용할 것이고, 버젼은 8.0.30
- 추가적인 docker run의 옵션은 `sudo docker --help` 로 확인 가능

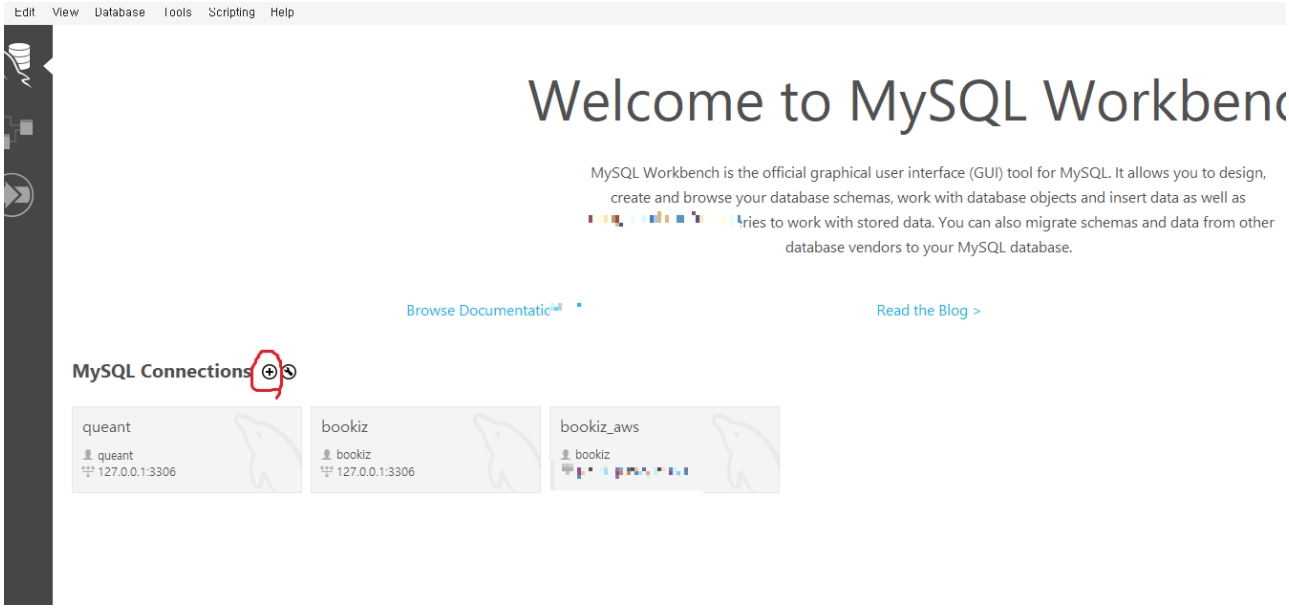
3. mysql 컨테이너 접속

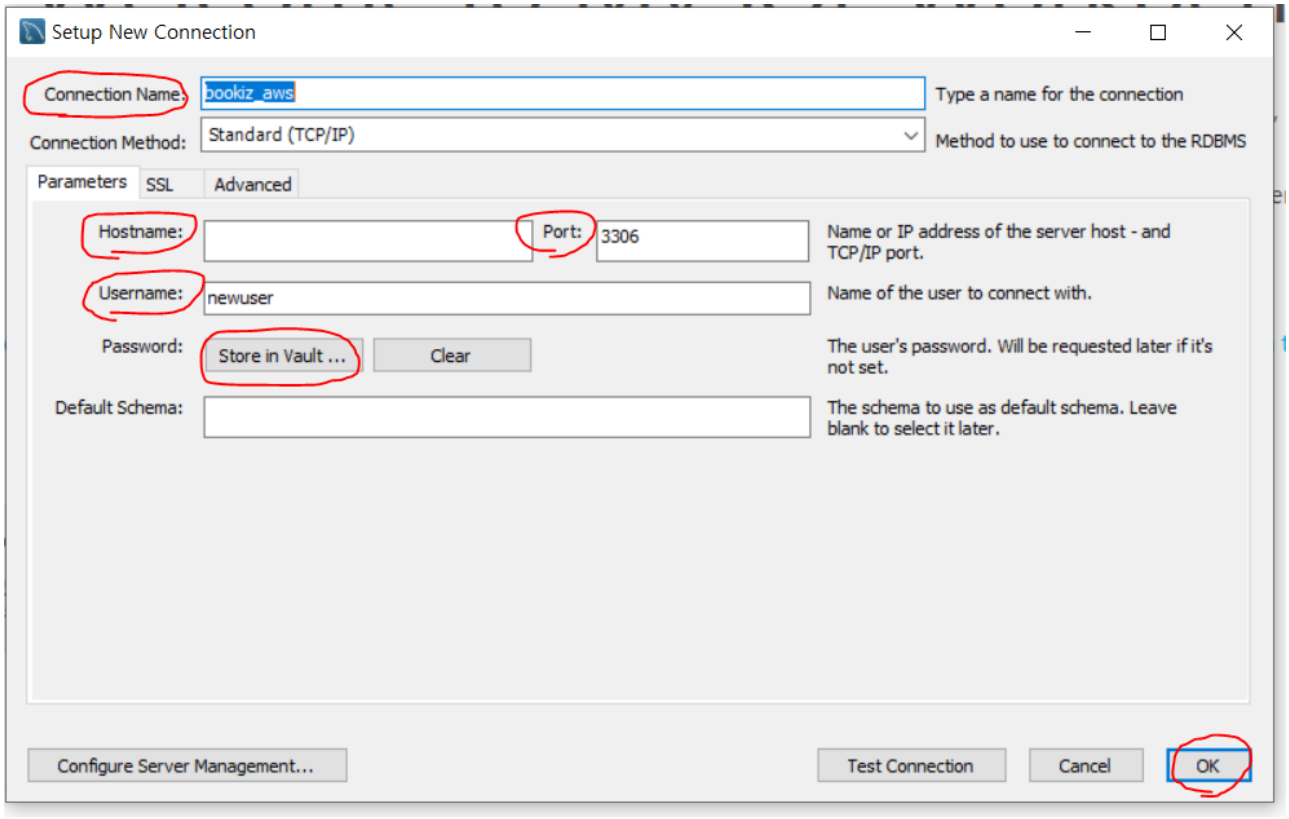
```
// bookiz_mysql 컨테이너를 bash 커맨드를 이용해서 접속
$ sudo docker exec -it bookiz_mysql bash
```

4. mysql 로그인 & 프로젝트계정 생성 및 권한부여

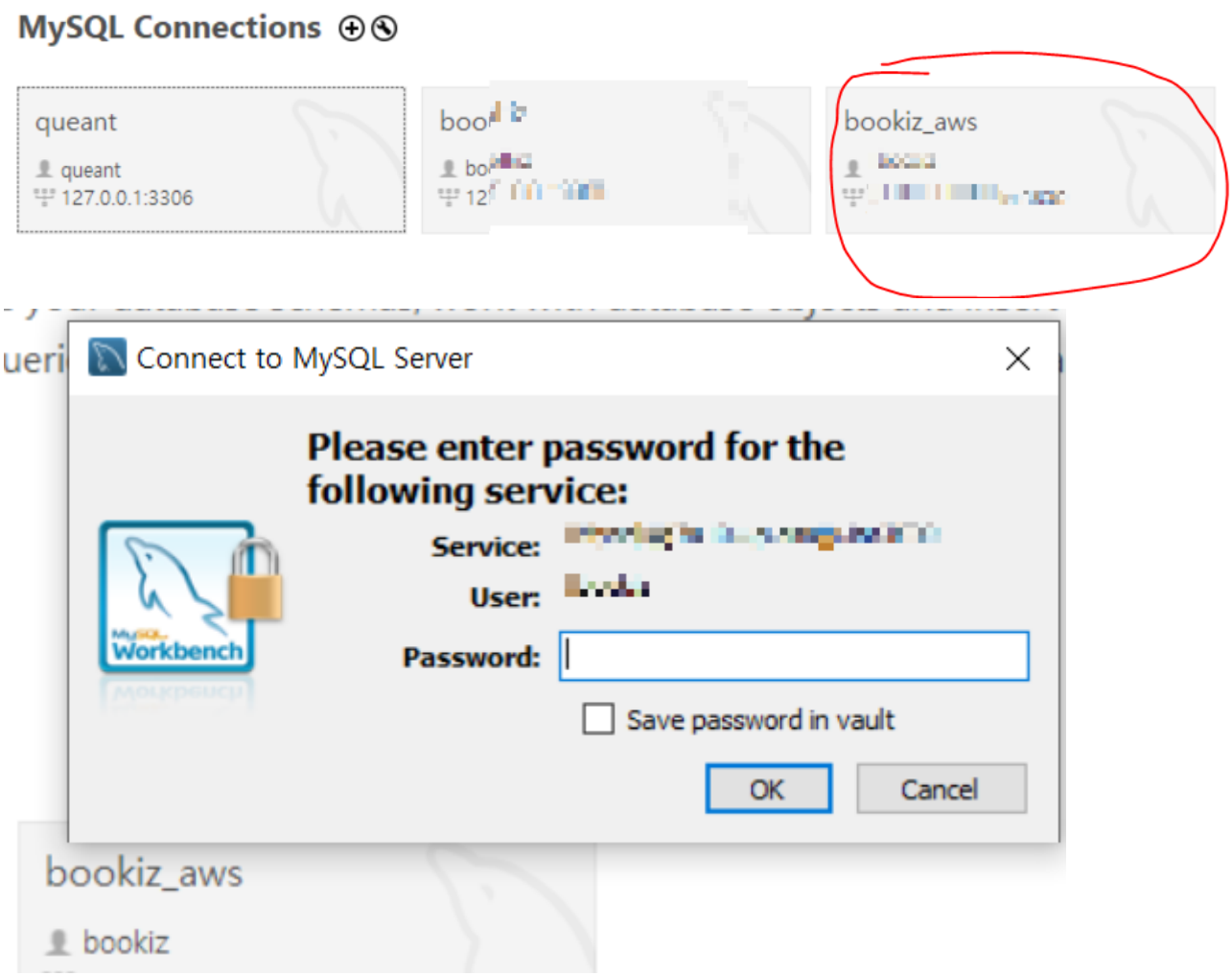
```
mysql -u root -p

Enter password: <root패스워드 입력>

create user 'newuser'@'%' identified by 'newpassword';

grant all privileges on *.* to 'newuser'@'%';

flush privileges;

exit
```

5. Workbench 연동

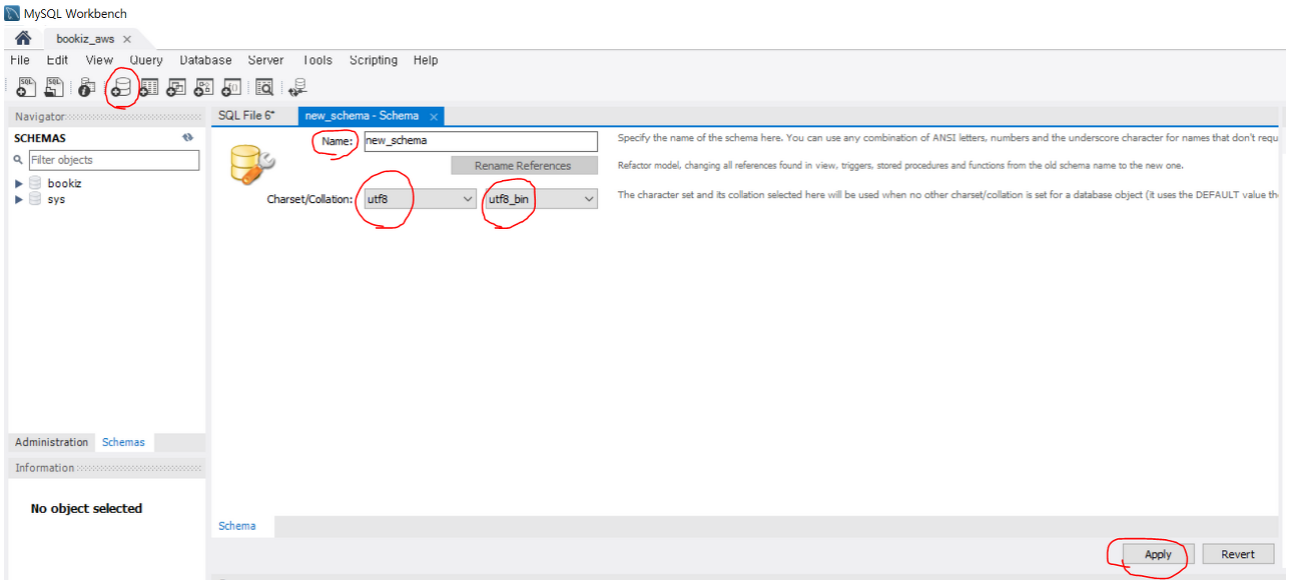




- Connection Name : 커넥션 이름 임의로 정하면 됩니다.
- Hostname : <domain ip>
- Port : '2. mysql 컨테이너 띄우기'에서 설정한 호스트 포트번호 <3306>
- Username : '4. mysql 로그인 & 프로젝트계정 생성 및 권한부여'에서 만든 <newuser>
- Password > Store in Vault > '4. mysql 로그인 & 프로젝트계정 생성 및 권한부여'에서 만든 <newpassword>
- OK!



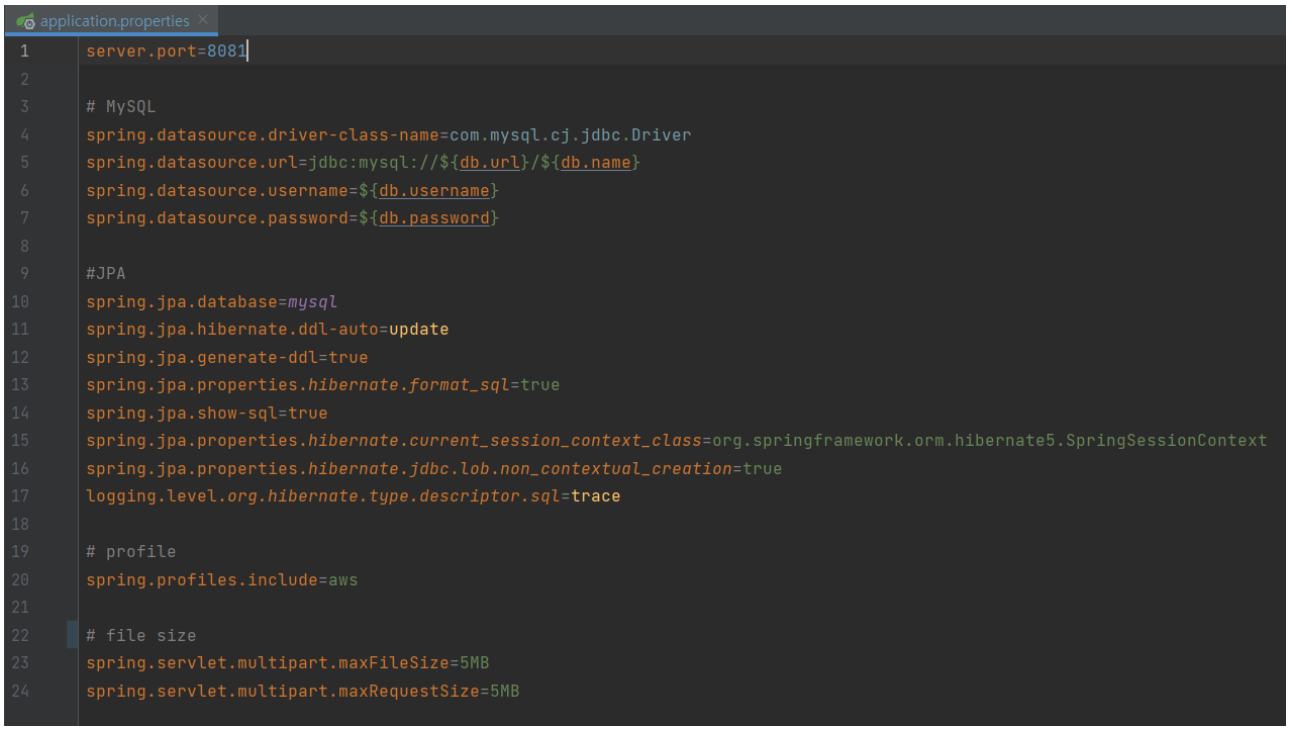
<newpassword> 입력



- new schema 생성
- Name : 임의로 설정
- Charset/Collation : utf8 / utf8_bin
- Apply !

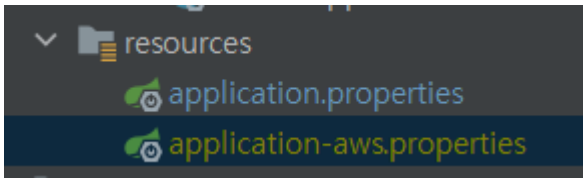
6. Spring boot 설정

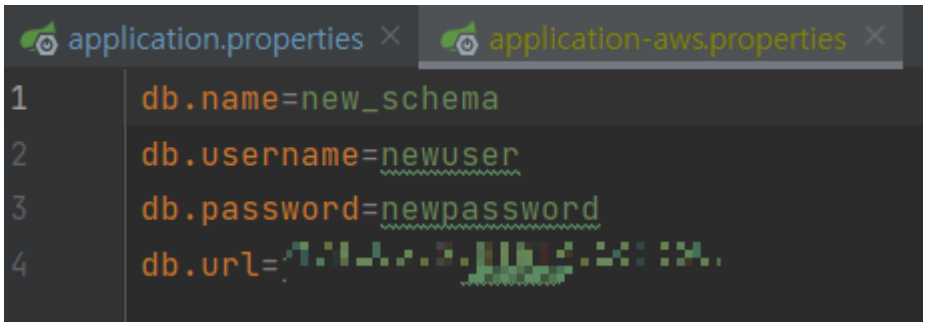
- application.properties




```
server.port=8081

# MySQL
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://${db.url}/${db.name}
spring.datasource.username=${db.username}
spring.datasource.password=${db.password}

#JPA
spring.jpa.database=mysql
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.current_session_context_class=org.springfram
ework.orm.hibernate5.SpringSessionContext
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
logging.level.org.hibernate.type.descriptor.sql=trace

# profile
spring.profiles.include=aws

# file size
spring.servlet.multipart.maxFileSize=5MB
spring.servlet.multipart.maxRequestSize=5MB
```

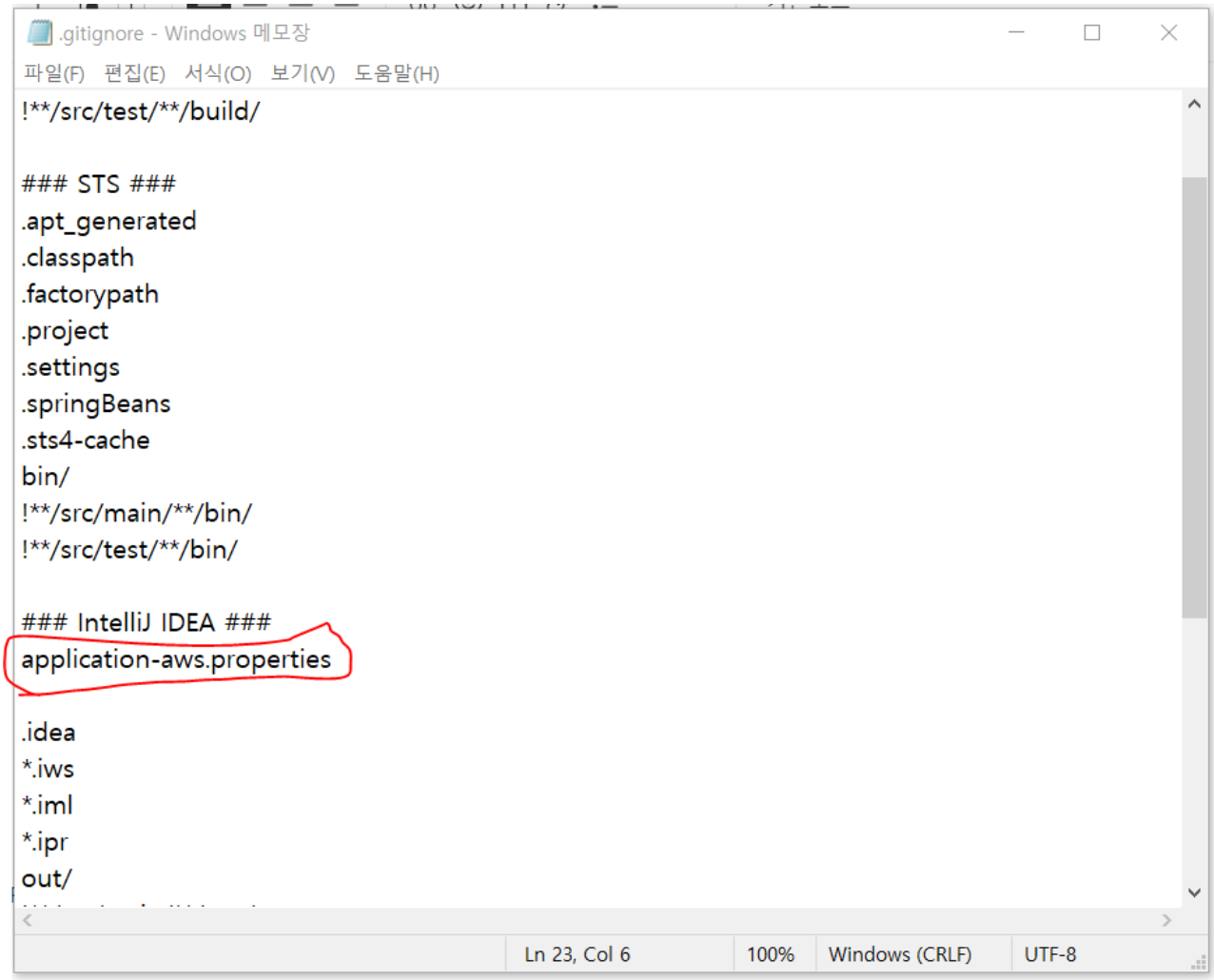
- application.properties와 같은 폴더에 application-aws.properties 파일 생성






```
db.name : 새로 만든 스키마 이름

db.username : 새로 만든 유저 이름

db.password : 새로 만든 유저의 비밀번호

db.url : 도메인 ip주소:호스트 포트번호
```

- gitignore에 application-aws.properties 등록



5. HTTPS 설정

https 설정을 해야 보안 및 이미지 등 전송도 가능합니다.

MobaXterm EC2 환경에서 입력

1. 인증키 발급

```
$ sudo apt update

$ sudo apt install snapd

$ sudo snap install --classic certbot

// 설치 확인
$ certbot --version

$ sudo certbot certonly --nginx -d <도메인 ip>

// 이메일 입력
// 서버 등록 승낙(Y)
// 이메일을 재단에 공유해서 관련 정보 받으실? (Y/N) 선택
// Successfully received certificate. (인증키 발급 성공)
```

인증키 발급에 성공하면

/etc/letsencrypt/live/<도메인 ip>/ 경로에 fullchain.pem, privkey.pem이 저장됩니다.

2. nginx 설정

```
$ sudo vi /etc/nginx/sites-available/default
```

위에 명령어를 입력하면 vim 편집기 화면이 나옵니다.

'i'를 누르면 <-- INSERT -->로 바뀌며 입력이 가능한 상태로 바뀝니다.

모자이크 부분에 도메인 ip 입력

```
server {
      listen 80;
      server_name <도메인 ip> www.<도메인 ip>;
      return 308 https://<도메인 ip>$request_uri;
}


server {
      listen 443 ssl;
      server_name <도메인 ip> www.<도메인 ip>;

      ssl_certificate      /etc/letsencrypt/live/<도메인 ip>/fullchain.pem;
      ssl_certificate_key  /etc/letsencrypt/live/<도메인 ip>/privkey.pem;

      client_max_body_size 5M;
}
```

간단히 설명하면

80은 http

443은 https를 의미합니다.

return 308 https://<도메인 ip>$request_uri;

=> 80으로 들어온 server_name 요청을 443으로 반환하겠다!

80(http)으로 들어온 도메인 ip 요청

=> 상단 listen 80이 쓰여진 server {} 구문 실행

=> return 443(https)

443(https)으로 들어온 도메인 ip 요청

=> 인증키 검증(ssl_cetificate, ssl_certificate_key)

=> path값에 해당하는 location 구문 실행

client_max_body_size는 nginx에서 파일 사이즈 용량 설정

인증키 경로

/etc/letsencrypt/live/<도메인 ip>/fullchain.pem /etc/letsencrypt/live/<도메인 ip>/privkey.pem

```
$ sudo service nginx restart
```

도메인 ip로 들어갔을 때 자물쇠 확인 !

6. Jenkins 컨테이너 생성

1. 도커에 젠킨스 이미지 받아오기

```
$ sudo docker pull jenkins/jenkins:lts
```

2. 젠킨스 컨테이너 생성

```
$ sudo docker run -v /var/run/docker.sock:/var/run/docker.sock -v
/home/ubuntu:/secret \
-v /jenkins:/var/jenkins_home -it -d -p 9090:8080 -u root --privileged=true
\
--name bookiz_jenkins jenkins/jenkins:lts
```

- -v <EC2경로>:<지금 생성될 컨테이너의 경로> : 컨테이너를 생성하면서 EC2의 경로와 마운트하는 옵션
- -d : 백그라운드 실행
- -p : 포트 옵션
- -u : 유저 옵션 (root계정으로 컨테이너 접근)
- --privileged=true : 권한 설정
- --name : 컨테이너 이름
- -v /home/ubuntu:/secret => 지금 생성될 컨테이너의 /secret 디렉토리에 들어가면 EC2의 /home/ubuntu 경로의 파일에 접근할 수 있습니다.

3. 젠킨스 컨테이너에 들어가서 비밀번호 확인

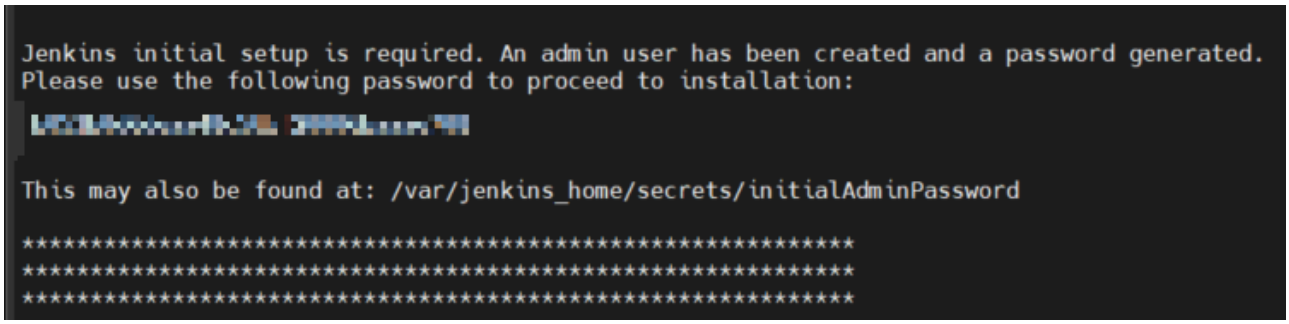
```
// EC2
$ sudo docker exec -it bookiz_jenkins bash
```

```
// 젠킨스 컨테이너
$ cat /var/jenkins_home/secrets/initialAdminPassword
```

위 명령어를 입력해서 나오는 비밀번호를 저장합니다.

또는 logs로도 확인 가능합니다.

```
// EC2
$ sudo docker logs bookiz_jenkins
```



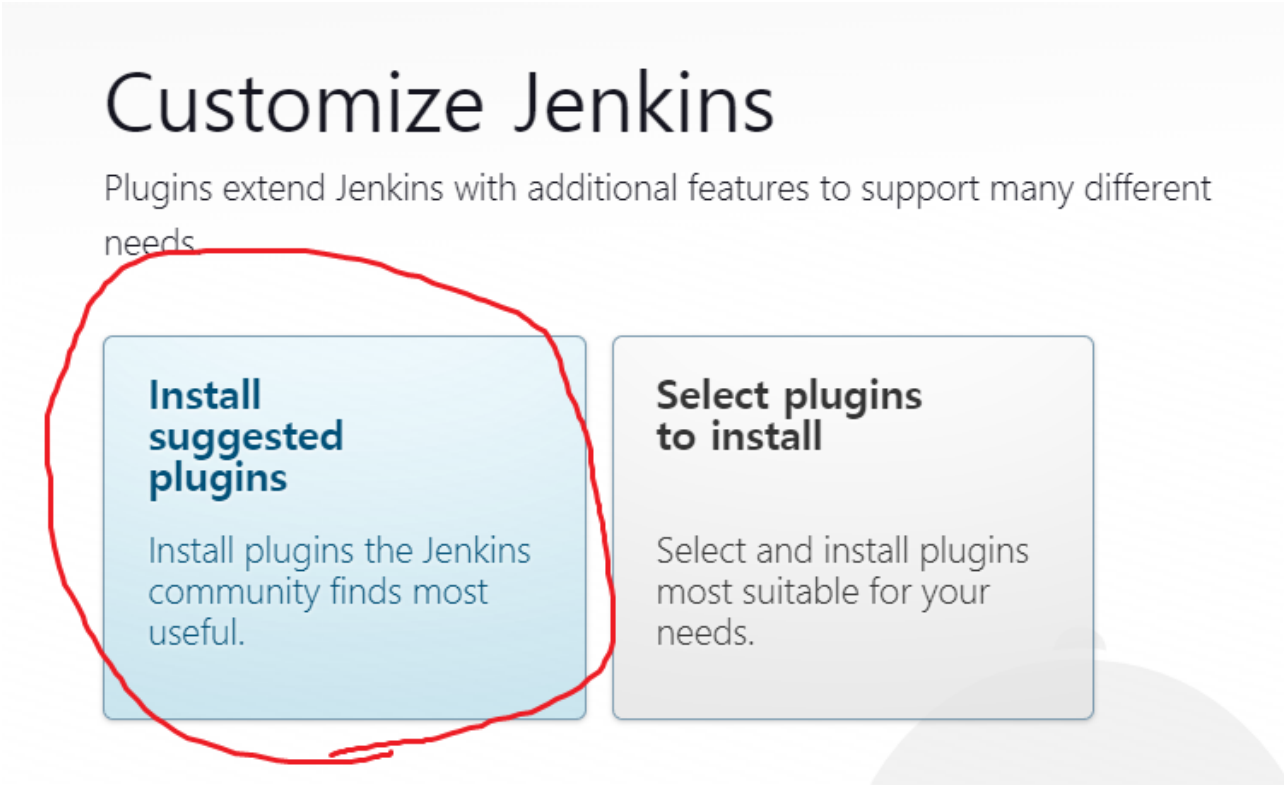
4. 젠킨스 컨테이너에 도커 설치

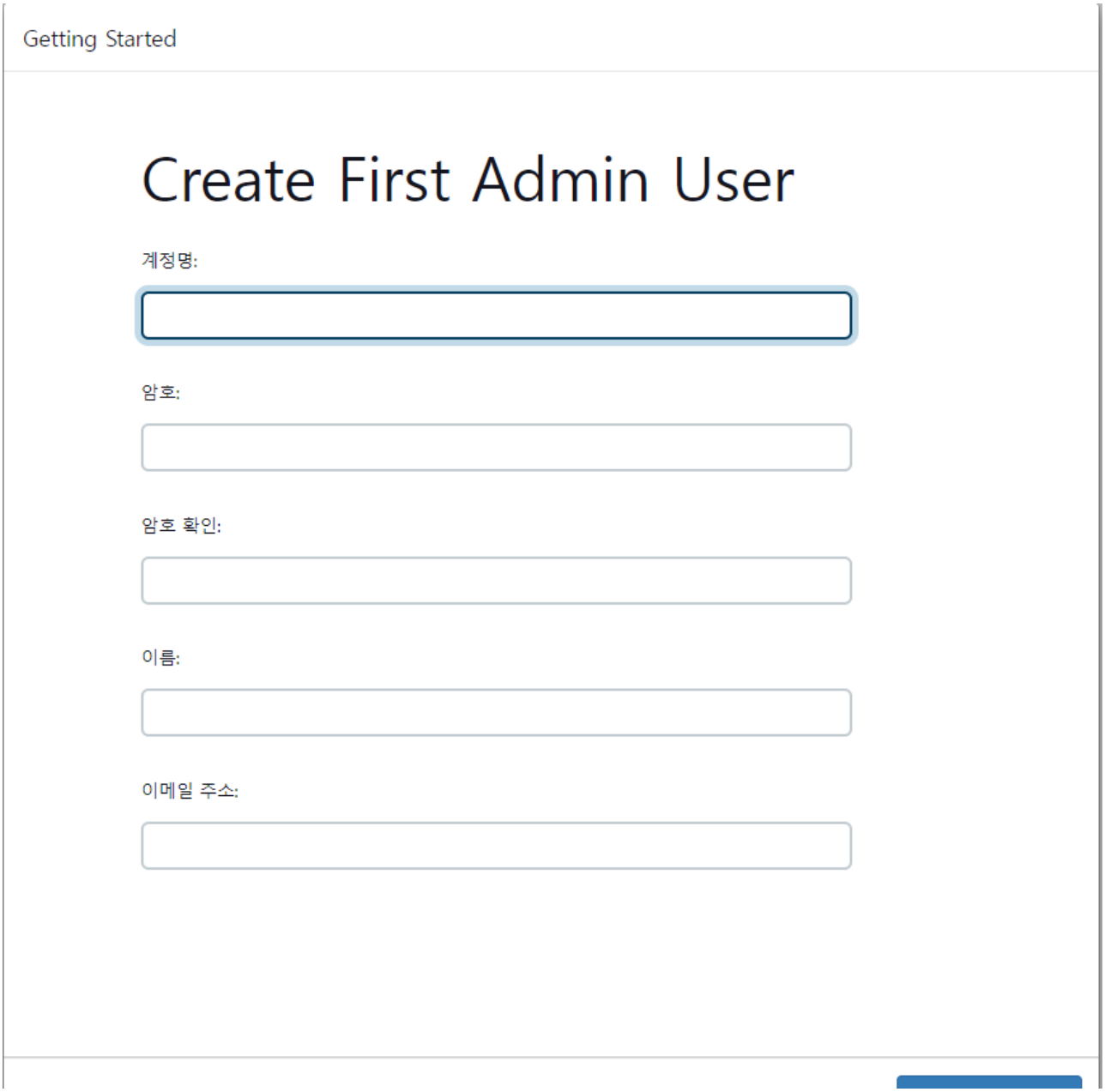
```
// 젠킨스 컨테이너
$ apt update

$ apt-get install -y ca-certificates
$ apt-get install curl
$ apt-get install software-properties-common
$ apt-get install apt-transport-https
$ apt-get install gnupg
$ apt-get install lsb-release

$ mkdir -p /etc/apt/keyrings

$ curl -fsSL https://download.docker.com/linux/debian/gpg | 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/debian \
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list >
```

```
/dev/null

$ apt-get update

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

5. 브라우져 열고 주소창에 <도메인 IP>:9090 접속

위에서 저장한 비밀번호 입력


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.


Getting Started
Create First Admin User
계정명:
암호:
암호 확인:
이름:
이메일 주소:

계정명 : 사용할 Jenkins ID

암호 : Jenkins 비밀번호

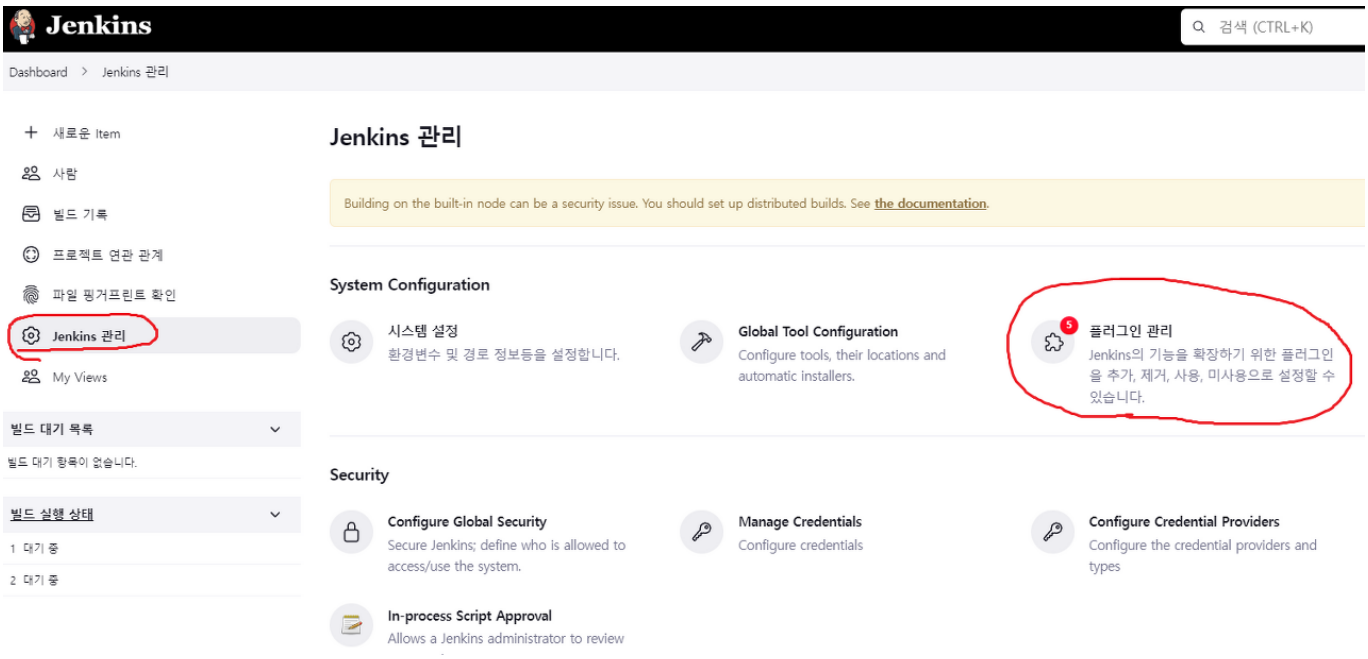
이름 : 이름

이메일주소 : 이메일주소

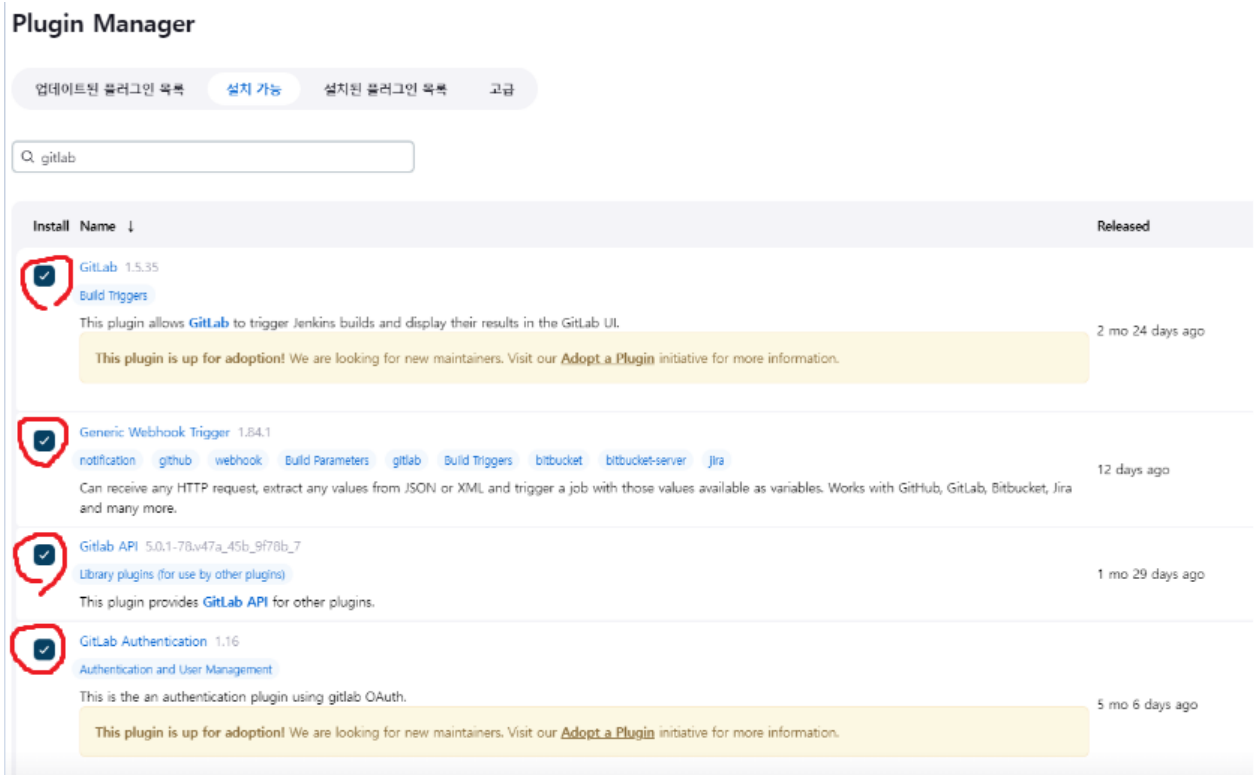
Save and Continue !

7. Jenkins 설정

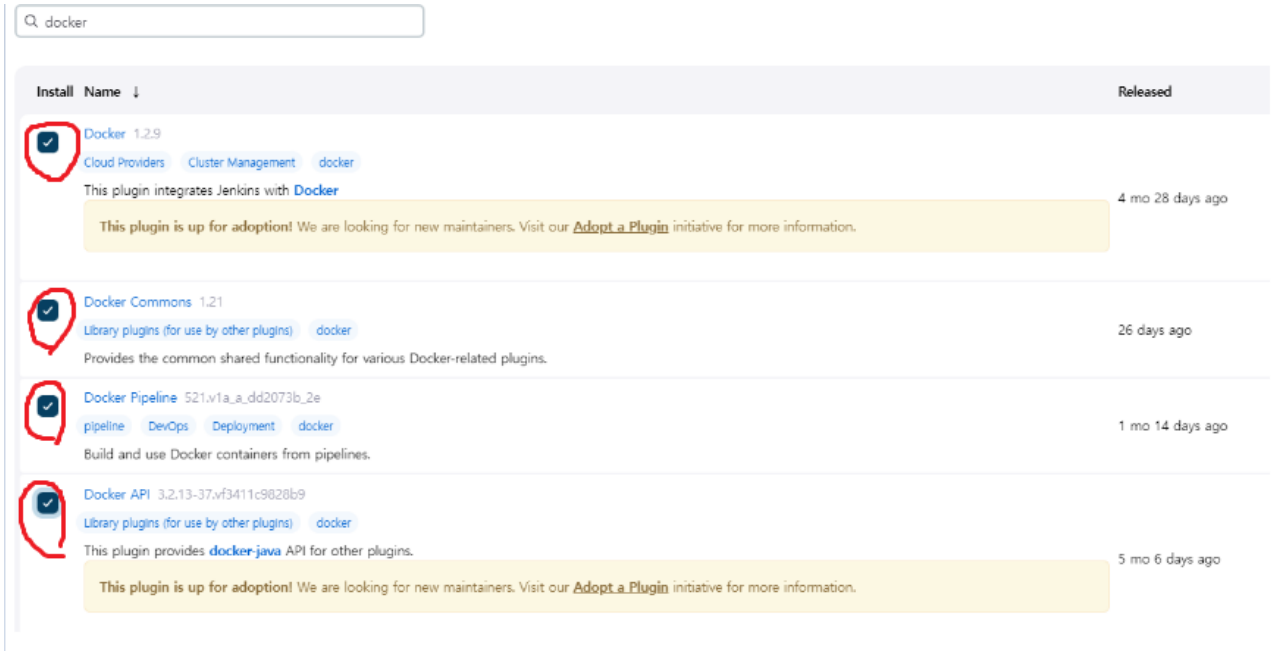
http://<도메인ip>:9090/ 에 접속!



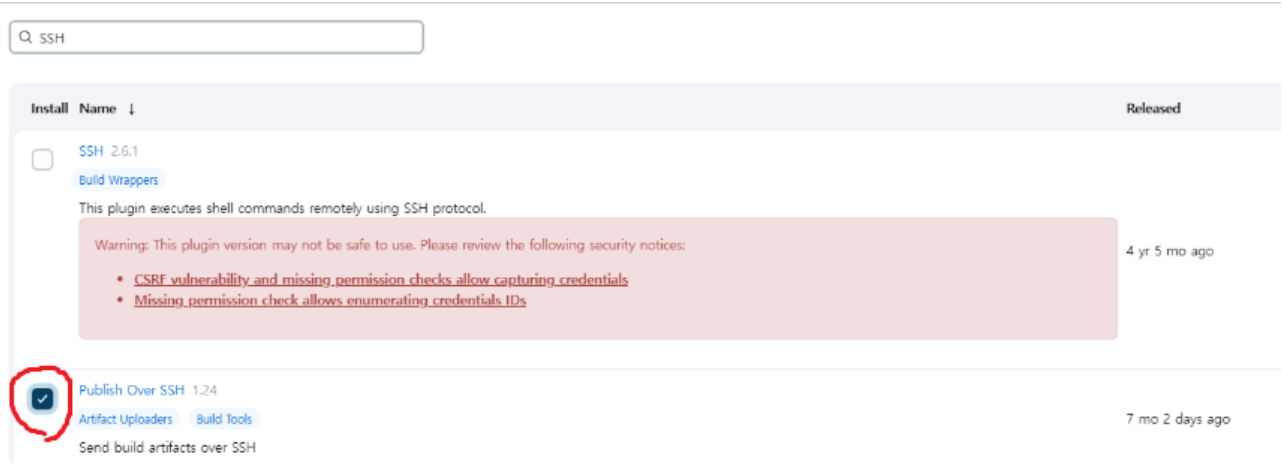
1. gitlab 검색 => Install without restart



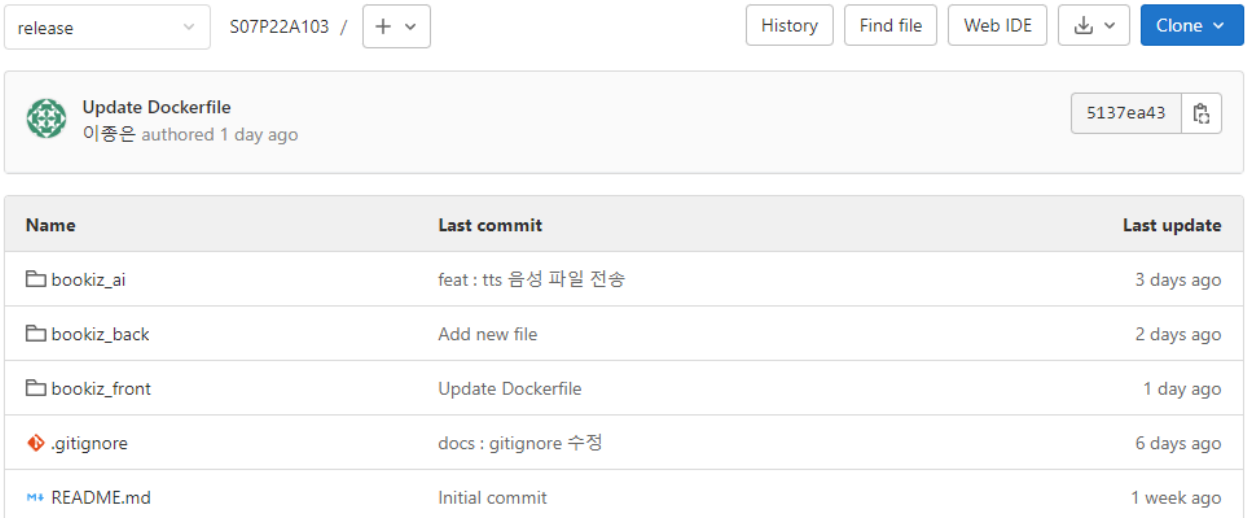
2. docker 검색 => Install without restart



3. SSH 검색 => Install without restart



4. Git repository 구조 확인

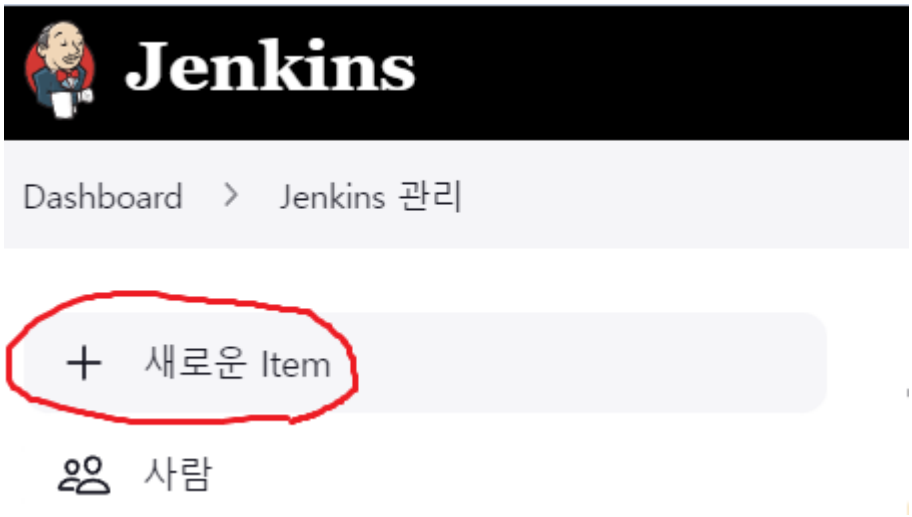


bookiz_ai : Django

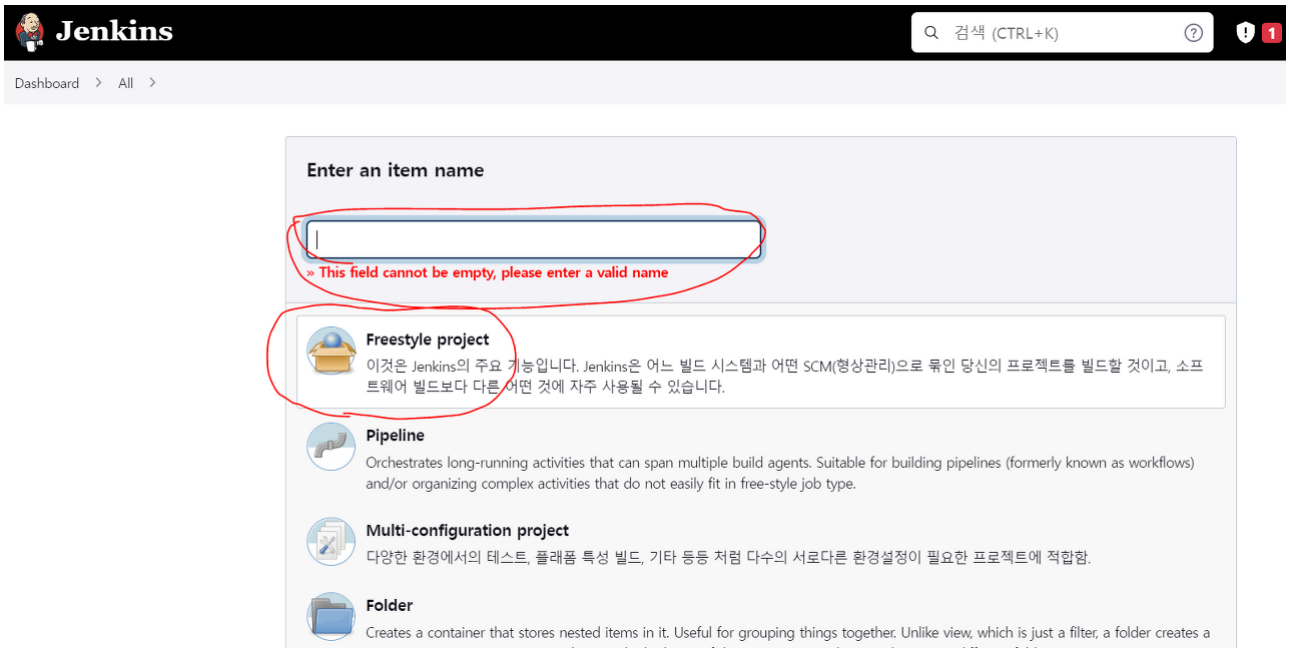
bookiz_back : Spring boot

bookiz_front : React

5. Jenkins 새로운 Item

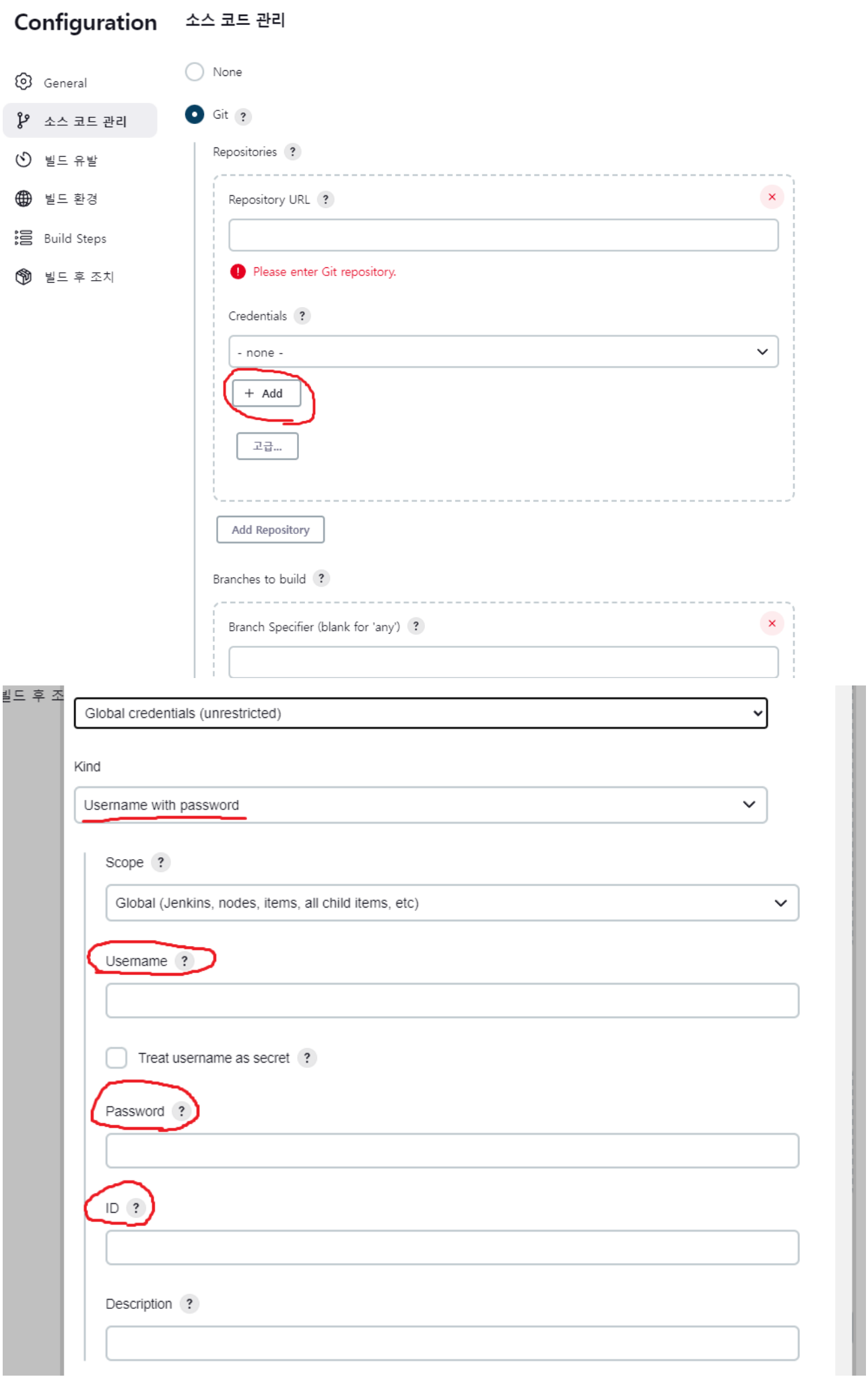


6. Jenkins 프로젝트 이름 입력, Freestyle project 선택



7. 소스 코드 관리

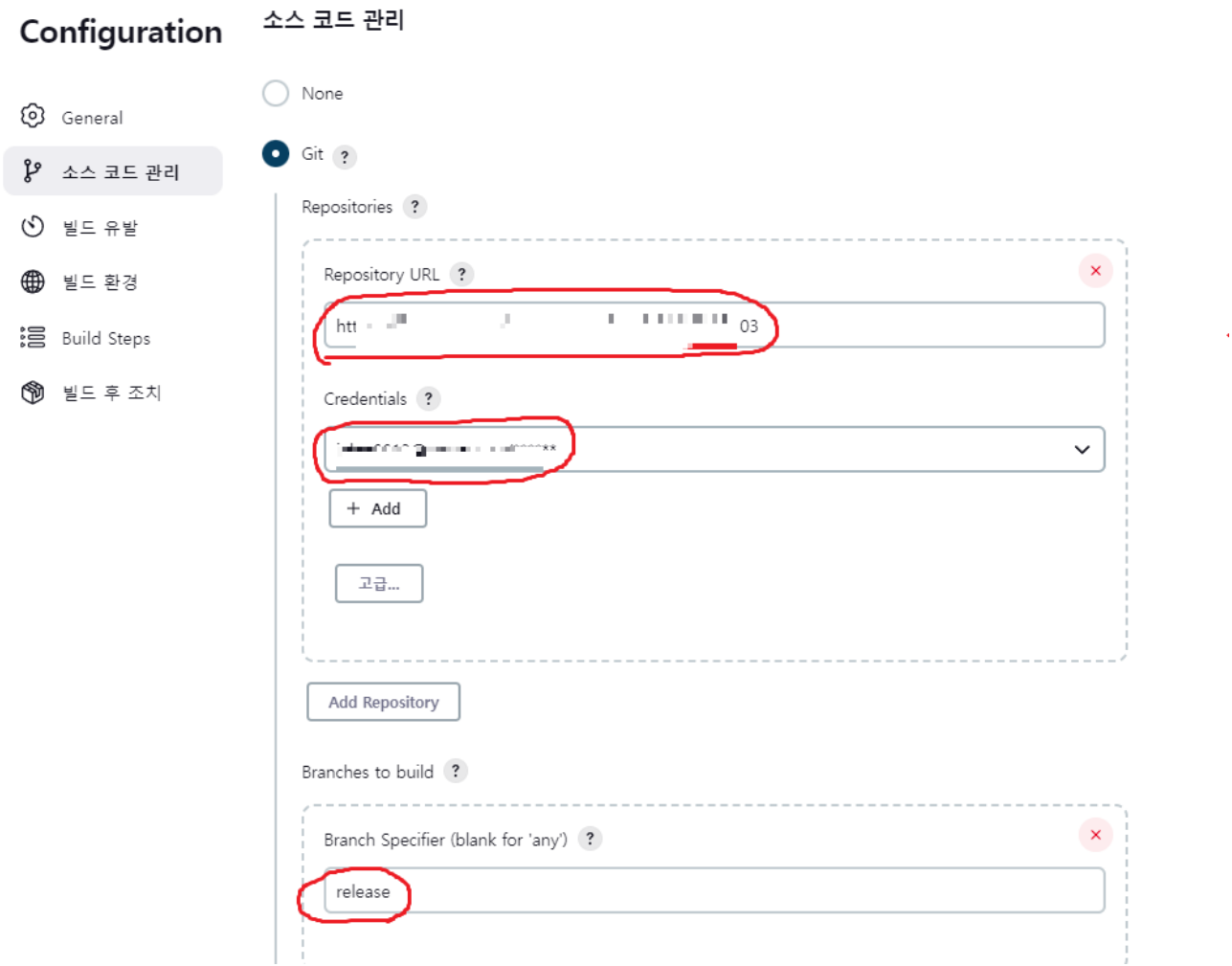
Git > Credentials > Add


Configuration
소스 코드 관리
General
소스 코드 관리
빌드 유발
빌드 환경
Build Steps
빌드 후 조치
None
Git
Repositories
Repository URL
Please enter Git repository.
Credentials
- none -
+ Add
고급...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
Global credentials (unrestricted)
Kind
Username with password
Scope
Global (Jenkins, nodes, items, all child items, etc)
Username
Treat username as secret
Password
ID
Description

Username : Git ID

Password : Git 비밀번호

ID : 젠킨스에서 구분할 ID

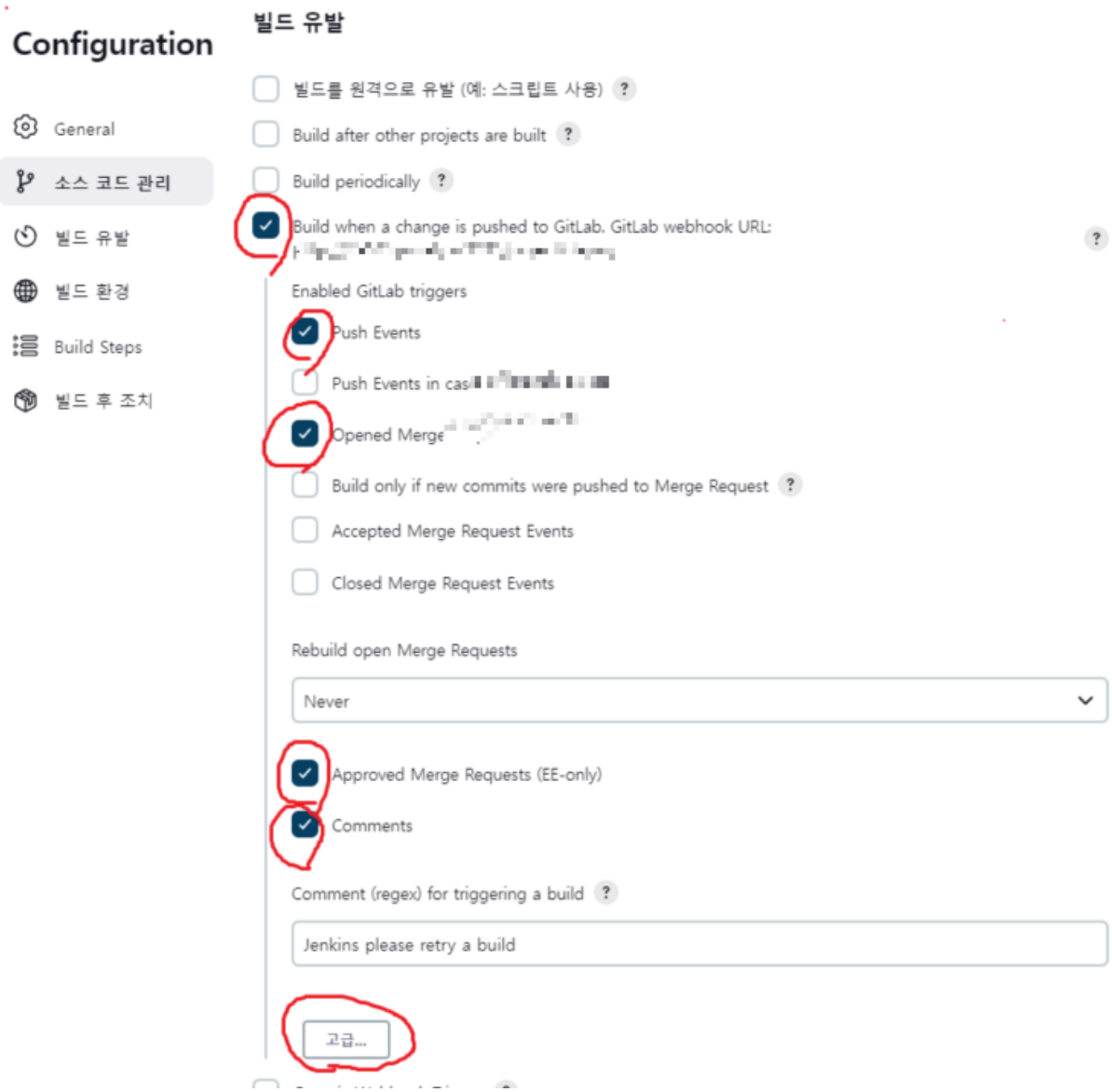


Repository URL : Git Repo 주소

Credentials : 위에서 생성한 ID

Branches to build > Branch Specifier : 자동배포 할 브랜치 이름

8. 빌드 유발

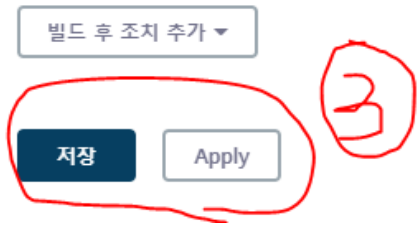


Generate 누르면 생기는 Secret token 따로 저장 해둡니다.

9. Build Steps

Build Steps

빌드 후 조치



1번의 Add build step 클릭

Execute Shell 선택

Execute shell에 아래 내용 작성

```
cd ${WORKSPACE}
docker ps -q --filter name=bookiz_spring | grep -q . && docker stop
bookiz_spring && docker rm bookiz_spring
docker ps -a -q --filter name=bookiz_spring | grep -q . && docker rm
bookiz_spring
docker cp ./application-aws.properties
```

```
bookiz_jenkins:/var/jenkins_home/workspace/deploy/bookiz_back/src/main/resou
rces
docker build -t bookiz_spring_image -f ./Dockerfile_spring ./bookiz_back
docker run -d -p 8081:8081 -v images:/bookiz_back/tale --name bookiz_spring
bookiz_spring_image

cd ${WORKSPACE}
docker ps -q --filter name=bookiz_react | grep -q . && docker stop
bookiz_react && docker rm bookiz_react
docker ps -a -q --filter name=bookiz_react | grep -q . && docker rm
bookiz_react
docker build -t bookiz_react_image -f ./Dockerfile_react ./bookiz_front
docker run -d -p 3000:3000 --name bookiz_react bookiz_react_image

cd ${WORKSPACE}
docker ps -q --filter name=bookiz_django | grep -q . && docker stop
bookiz_django && docker rm bookiz_django
docker ps -a -q --filter name=bookiz_django | grep -q . && docker rm
bookiz_django
docker build -t bookiz_django_image -f ./Dockerfile_django ./bookiz_ai
docker run -d -p 8082:8082 --name bookiz_django bookiz_django_image
```

위에서부터 스프링, 리액트, 쟝고 컨테이너를 생성하는 구문입니다.

이미 컨테이너가 있다면 삭제 후 다시 생성하고, Dockerfile을 읽어서 이미지를 생성합니다.

생성한 이미지로 컨테이너를 실행하는데,

스프링은 8081포트, 리액트는 3000포트, 쟝고는 8082포트로 실행합니다.

다만, 스프링은 이미지 업로드 기능을 위해 -v 옵션으로 images 볼륨과 마운트 했습니다.

(선택) 이미지 업로드 연동

```
$ docker run -d -p 8081:8081 -v images:/bookiz_back/tale --name
bookiz_spring bookiz_spring_image
```

Spring boot 프로젝트 폴더에 tale이란 폴더를 만들어놓고, images란 이름의 도커 볼륨과 마운트합니다.

백엔드 로직으로 tale 폴더에 이미지나 파일이 업로드 될때마다 마운트된 images 볼륨에도 저장이 됩니다.

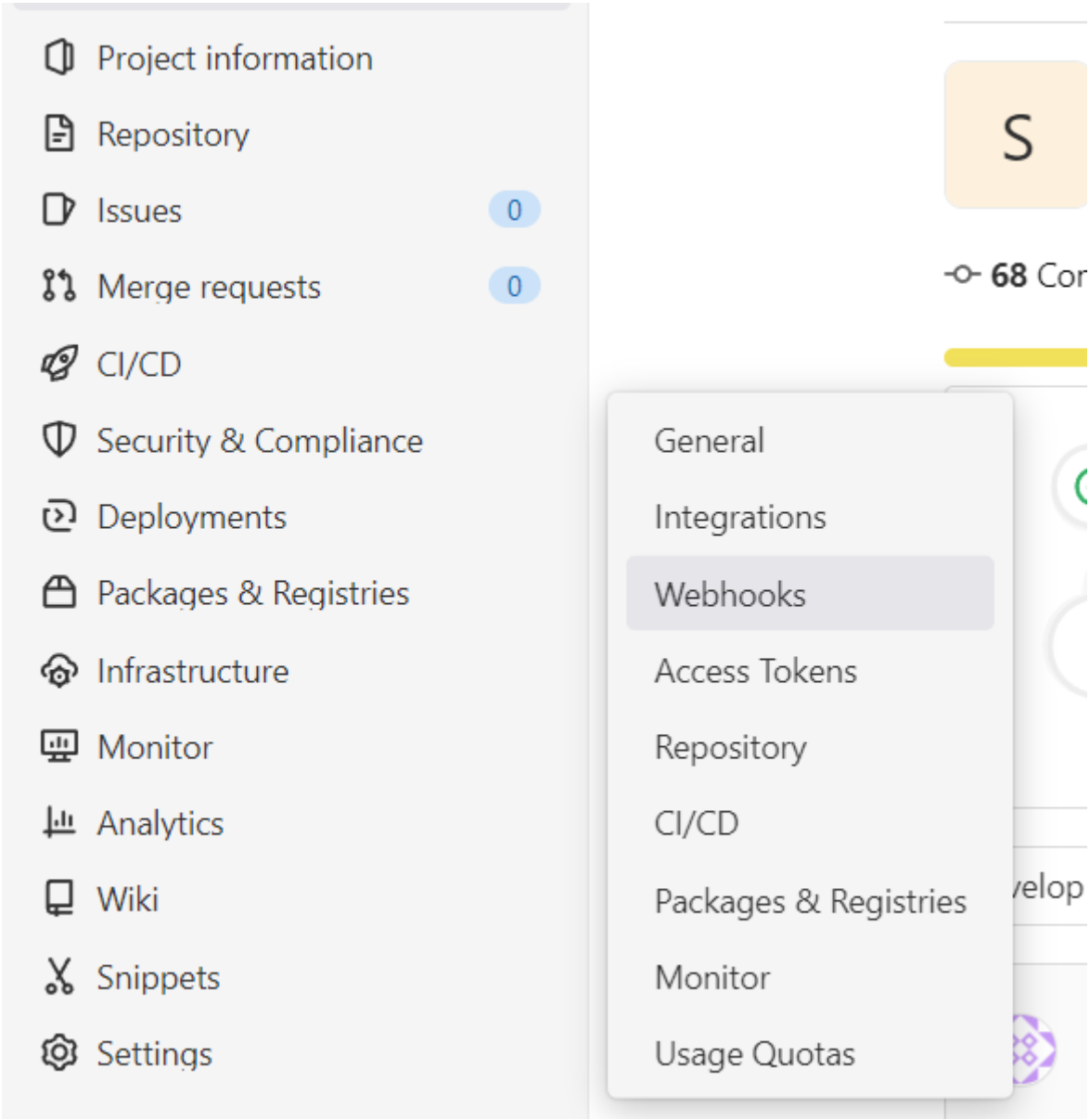
그럼 스프링 컨테이너가 빌드될 때마다 삭제, 생성되어도 volume에 저장된 이미지나 파일이 남아있어서 접근 가능합니다.

만약, 이 기능이 필요없다면 -v 옵션은 생략해도 됩니다.

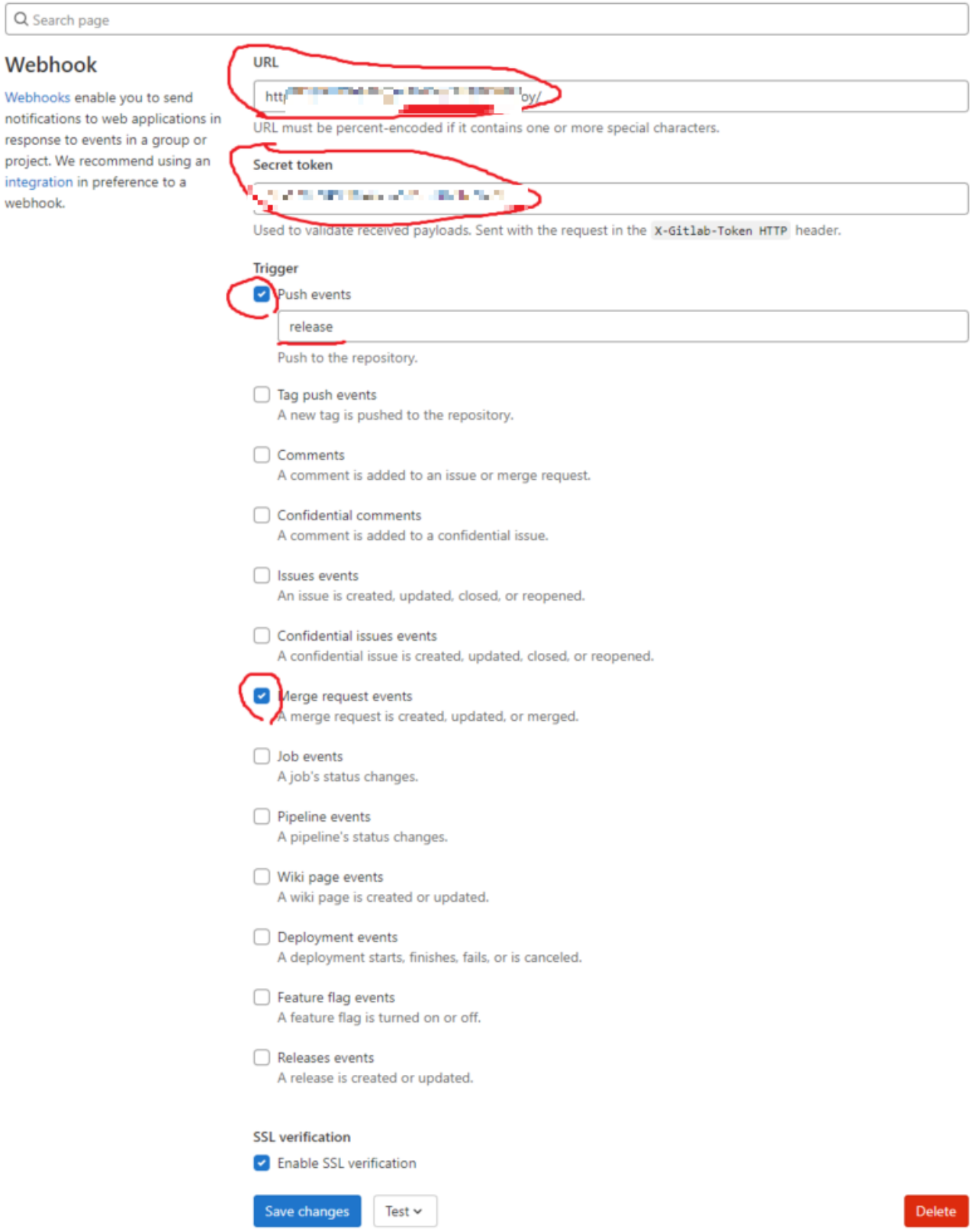
Apply > 저장 !

8. Gitlab Webhook

1. Gitlab > Settings > Wehbooks



2. Webhook



URL : http://<도메인 ip>:9090/project/<Jenkins에서 새로운 Item했을 때 만든 이름>/

Secret token : Jenkins 설정에서 8. 빌드 유발 단계의 Secret token 값을 입력합니다.

Push events : 체크

아래에는 자동 배포를 적용할 브랜치 이름을 적습니다.

Merge request events : 체크

저장!

9. Dockerfile & CI/CD 완성

이제 젠킨스의 Execute Shell에서 작성했던 이미지를 만들기 위한 Dockerfile을 작성합니다.

bash나 커맨드창을 켜서 아래 명령어로 도커 파일 3개를 생성합니다.

```
$ touch Dockerfile_spring
$ touch Dockerfile_react
$ touch Dockerfile_django
```

1. Dockerfile_spring

```
FROM openjdk:8
RUN mkdir -p /bookiz_back
COPY ./ /bookiz_back
WORKDIR /bookiz_back
RUN chmod +x ./gradlew
RUN ./gradlew build
ENTRYPOINT ["java", "-jar", "./build/libs/bookiz-0.0.1-SNAPSHOT.jar"]
EXPOSE 8081
```

2. Dockerfile_react

```
FROM node:16
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
ENV PATH /usr/src/app/node_modules/.bin:$PATH
COPY package*.json ./
RUN npm install
COPY ./ ./
EXPOSE 3000
CMD ["npm", "start"]
```

3. Dockerfile_django

```
FROM python:3.7
ENV PYTHONUNBUFFERED 1
RUN mkdir /bookiz_ai
WORKDIR /bookiz_ai
ADD ./requirements.txt /bookiz_ai/
RUN pip install -r requirements.txt
ADD ./ /bookiz_ai/
EXPOSE 8082
CMD ["python", "./manage.py", "runserver", "0.0.0.0:8082"]
```

Dockerfile 명령어를 간단히 설명하면

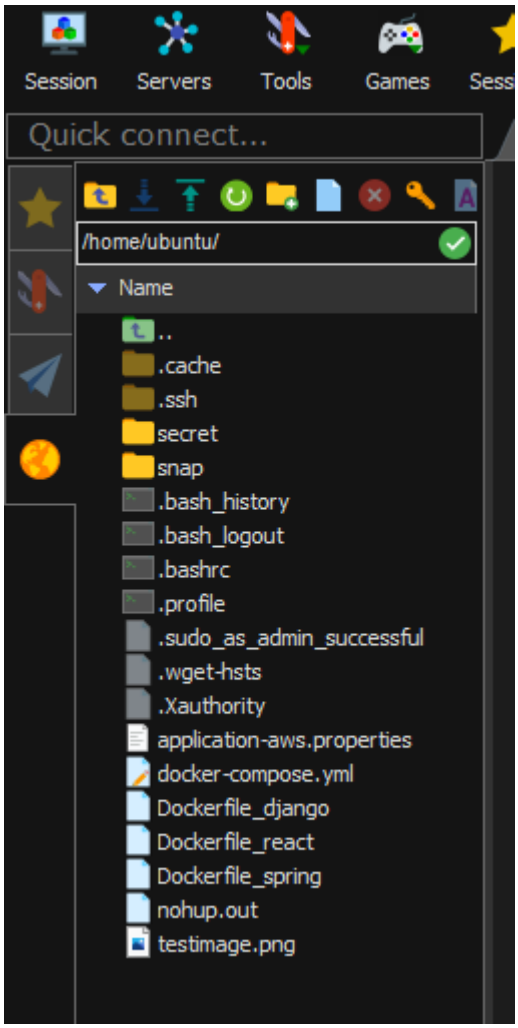
- FROM 에 적은 것을 기반으로 컨테이너를 생성
- RUN 명령어 실행
- COPY 호스트 경로에서 컨테이너 경로로 복사
- WORKDIR 명령어가 실행되는 경로 설정
- ENTRYPOINT, CMD 도커파일에서 한 번 실행
- EXPOSE 해당 포트로 서버 on

위 3개의 도커파일은 Execute Shell의 docker build 구문에서 실행됩니다.

도커파일에 작성된 내용을 기반으로 도커 이미지를 뙇!! 만듭니다.

docker run -d -p --name <컨테이너 이름> <이미지 이름> 명령어로 뙇 만든 이미지를 기반으로 컨테이너를 생성 & 실행합니다.

4. 마우스로 드래그해서 Dockerfile, application-aws.properties 바탕화면 => EC2로 옮기기



그냥 바탕화면에 만든 도커파일들 드래그로 끌어다 옮기면 복사됩니다.

젠킨스 컨테이너 만들 때 아래 명령문으로 만들었습니다.

```
$ sudo docker run -v /var/run/docker.sock:/var/run/docker.sock -v
/home/ubuntu:/secret \
-v /jenkins:/var/jenkins_home -it -d -p 9090:8080 -u root --privileged=true
\
--name bookiz_jenkins jenkins/jenkins:lts
```

-v /home/ubuntu:/secret

=> EC2의 /home/ubuntu에 있는 파일은 젠킨스 컨테이너의 /secret 경로에서도 접근할 수 있습니다.

드래그로 옮긴 Dockerfile과 application-aws.properties는 젠킨스 컨테이너의 /secret 경로에도 있습니다.

이 파일들을 젠킨스 컨테이너의 적절한 경로로 복사합니다.

먼저, 젠킨스 컨테이너에 들어갑니다.

```
// EC2
$ sudo docker exec -it bookiz_jenkins bash
```

Dockerfile과 application-aws.properties를 필요한 경로에 복사합니다.

```
// 젠킨스 컨테이너
cp /secret/Dockerfile_spring /var/jenkins_home/workspace/deploy/
cp /secret/Dockerfile_react /var/jenkins_home/workspace/deploy/
cp /secret/Dockerfile_django /var/jenkins_home/workspace/deploy/

cp /secret/application-aws.properties
/var/jenkins_home/workspace/deploy/bookiz_back/src/main/resources/
```

도커파일을 저 경로에 복사하는 이유를 알기 위해 Jenkins Build step에 작성한 내용을 다시 확인합니다.

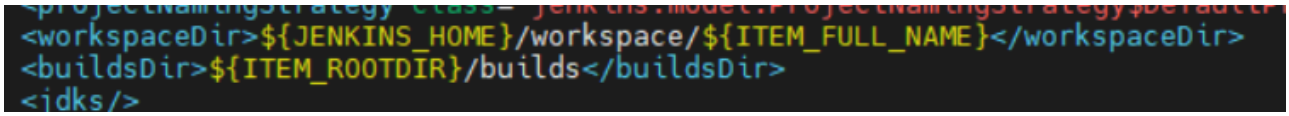
```
cd ${WORKSPACE}
docker ps -q --filter name=bookiz_spring | grep -q . && docker stop
bookiz_spring && docker rm bookiz_spring
docker ps -a -q --filter name=bookiz_spring | grep -q . && docker rm
bookiz_spring
docker cp ./application-aws.properties
bookiz_jenkins:/var/jenkins_home/workspace/deploy/bookiz_back/src/main/resou
rces
docker build -t bookiz_spring_image -f ./Dockerfile_spring ./bookiz_back
docker run -d -p 8081:8081 -v images:/bookiz_back/tale --name bookiz_spring
bookiz_spring_image

cd ${WORKSPACE}
```

```
docker ps -q --filter name=bookiz_react | grep -q . && docker stop
bookiz_react && docker rm bookiz_react
docker ps -a -q --filter name=bookiz_react | grep -q . && docker rm
bookiz_react
docker build -t bookiz_react_image -f ./Dockerfile_react ./bookiz_front
docker run -d -p 3000:3000 --name bookiz_react bookiz_react_image

cd ${WORKSPACE}
docker ps -q --filter name=bookiz_django | grep -q . && docker stop
bookiz_django && docker rm bookiz_django
docker ps -a -q --filter name=bookiz_django | grep -q . && docker rm
bookiz_django
docker build -t bookiz_django_image -f ./Dockerfile_django ./bookiz_ai
docker run -d -p 8082:8082 --name bookiz_django bookiz_django_image
```

cd${WORKSPACE}를 실행하면서 젠킨스 컨테이너의 {WORKSPACE}경로로 이동합니다.

${WORKSPACE}의 위치는 아래 명령어로 확인할 수 있습니다. (변경도 가능)

```
$ sudo vi jenkins/config.xml
```

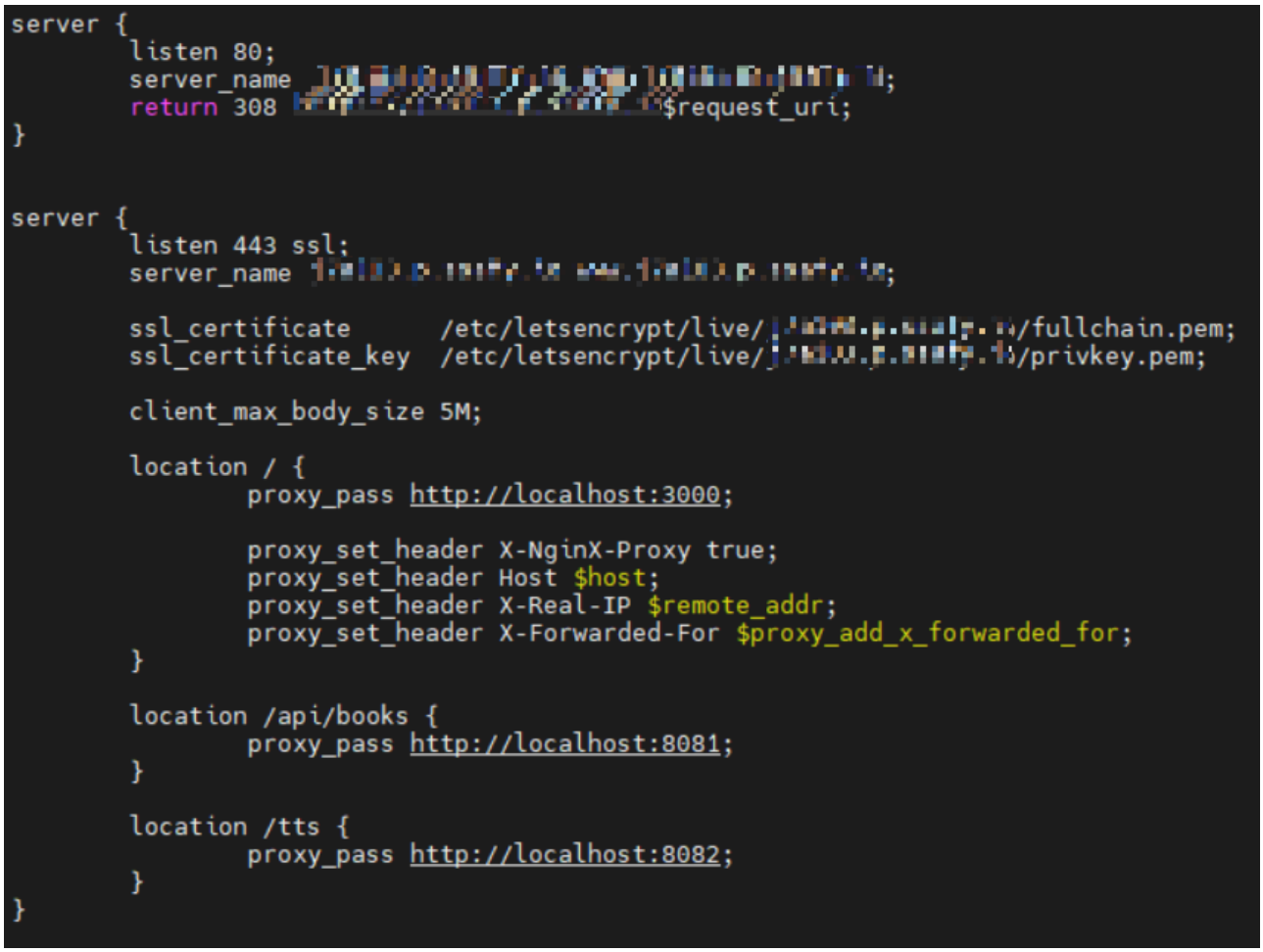


docker build -t bookiz_spring_image -f ./Dockerfile_spring ./bookiz_back

=> ./bookiz_back 경로를 호스트 경로로 ./Dockerfile_spring 도커파일을 실행해서 bookiz_spring_image 라는 이름의 이미지를 만들겠다. (-f는 도커파일 이름을 커스텀했기 때문에 사용하는 옵션)

5. nginx 설정

```
$ sudo vi /etc/nginx/sites-available/default
```



```
server {
        listen 80;
        server_name [redacted];
        return 308 [redacted]$request_uri;
}


server {
        listen 443 ssl;
        server_name [redacted];

        ssl_certificate      /etc/letsencrypt/live/[redacted]/fullchain.pem;
        ssl_certificate_key  /etc/letsencrypt/live/[redacted]/privkey.pem;

        client_max_body_size 5M;

        location / {
                proxy_pass http://localhost:3000;

                proxy_set_header X-NginX-Proxy true;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }

        location /api/books {
                proxy_pass http://localhost:8081;
        }

        location /tts {
                proxy_pass http://localhost:8082;
        }
}
```

```
server {
        listen 80;
        server_name <도메인 ip> www.<도메인 ip>;
        return 308 https://<도메인 ip>$request_uri;
}


server {
        listen 443 ssl;
        server_name <도메인 ip> www.<도메인 ip>;

        ssl_certificate      /etc/letsencrypt/live/<도메인 ip>/fullchain.pem;
        ssl_certificate_key  /etc/letsencrypt/live/<도메인 ip>/privkey.pem;

        client_max_body_size 5M;

        location / {
                proxy_pass http://localhost:3000;

                proxy_set_header X-NginX-Proxy true;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }

        location /api/books {
```

```
                proxy_pass http://localhost:8081;
        }

        location /tts {
                proxy_pass http://localhost:8082;
        }
}
```

location / {} 는 react

location /api/books {} 는 Spring boot에서 만든 api 경로

location /tts {} 는 Django에서 만든 api 경로이며,

각각의 포트 번호로 서버를 on 해놓은 이미지를 컨테이너로 생성했으니 localhost:<해당하는 포트번호>를 적으면 됩니다.

```
$ sudo service nginx restart
```

도메인 ip에 접속 시 프로젝트 화면이 보이면 성공 !