



포팅 매뉴얼

메이븐으로 backend 서버 빌드 배포하기

부여받은 EC2 주소로 접속을 위해 CLI 도구를 받습니다.

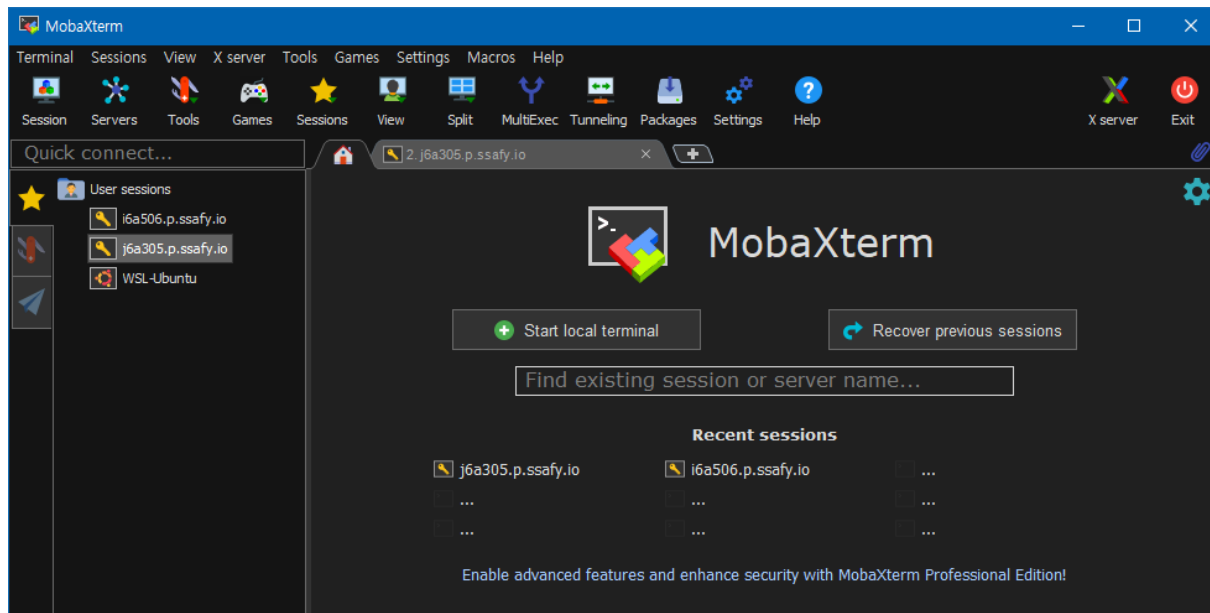
저희의 경우 mobaXterm을 사용하였고,

배포 과정에서도 이를 사용한다는 가정하에 진행하겠습니다.

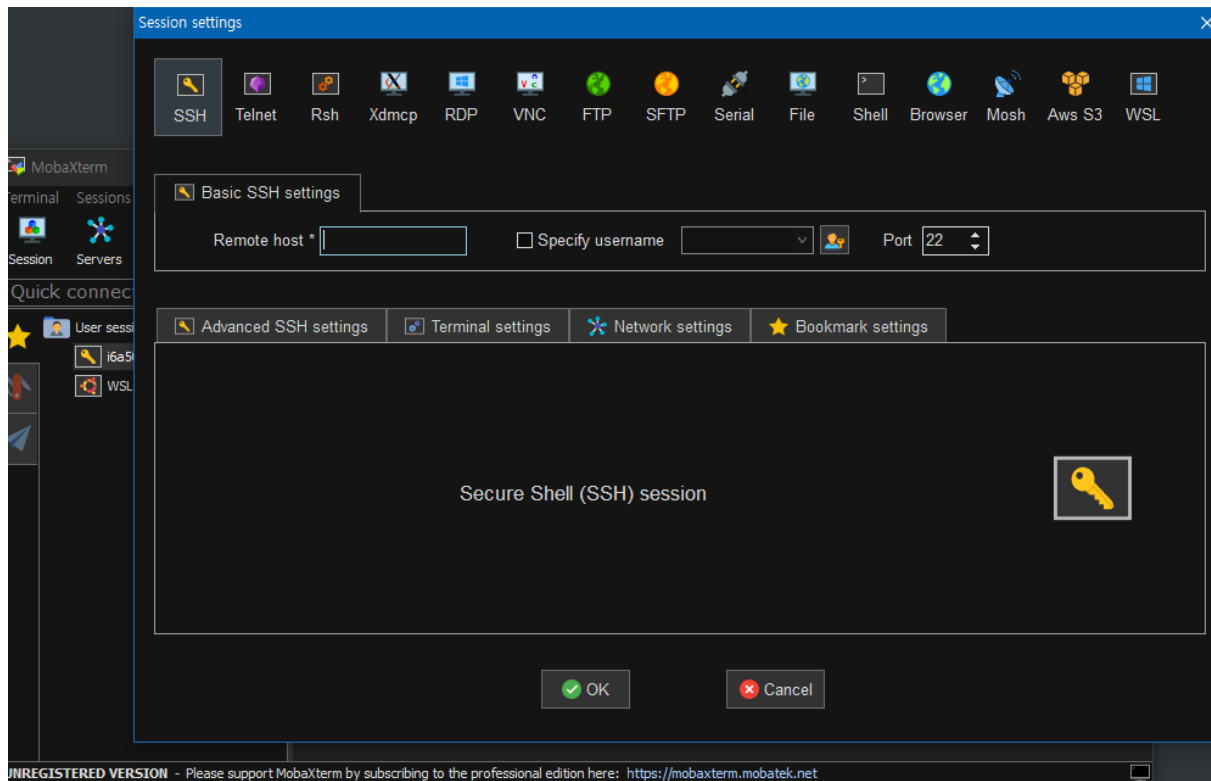
<https://mobaxterm.mobatek.net/download.html>

접속 후 home edition으로 다운로드

압축을 풀어주면 mobaxterm 실행파일 생성



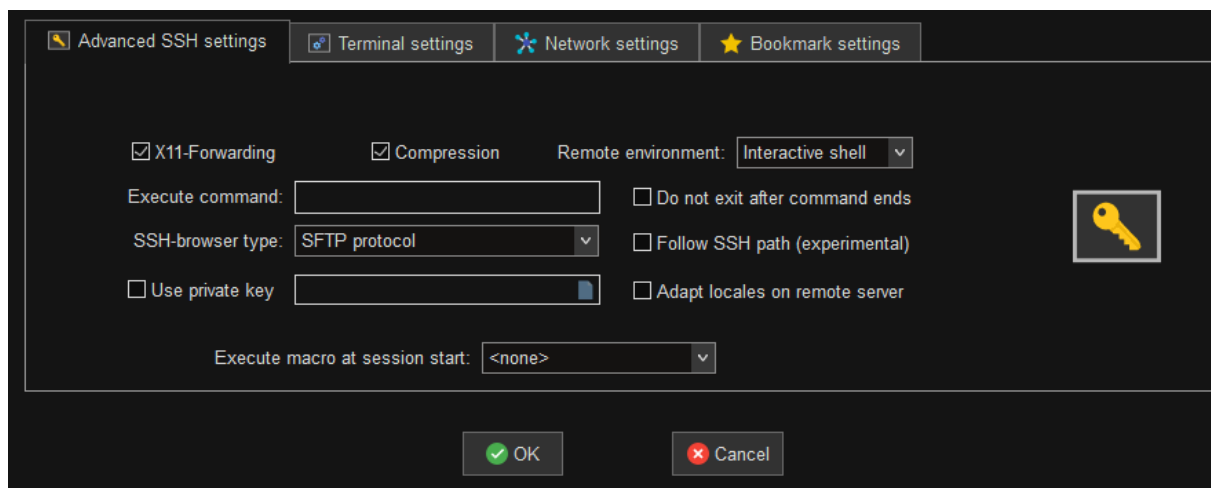
접속화면



세션 클릭 후 SSH 클릭

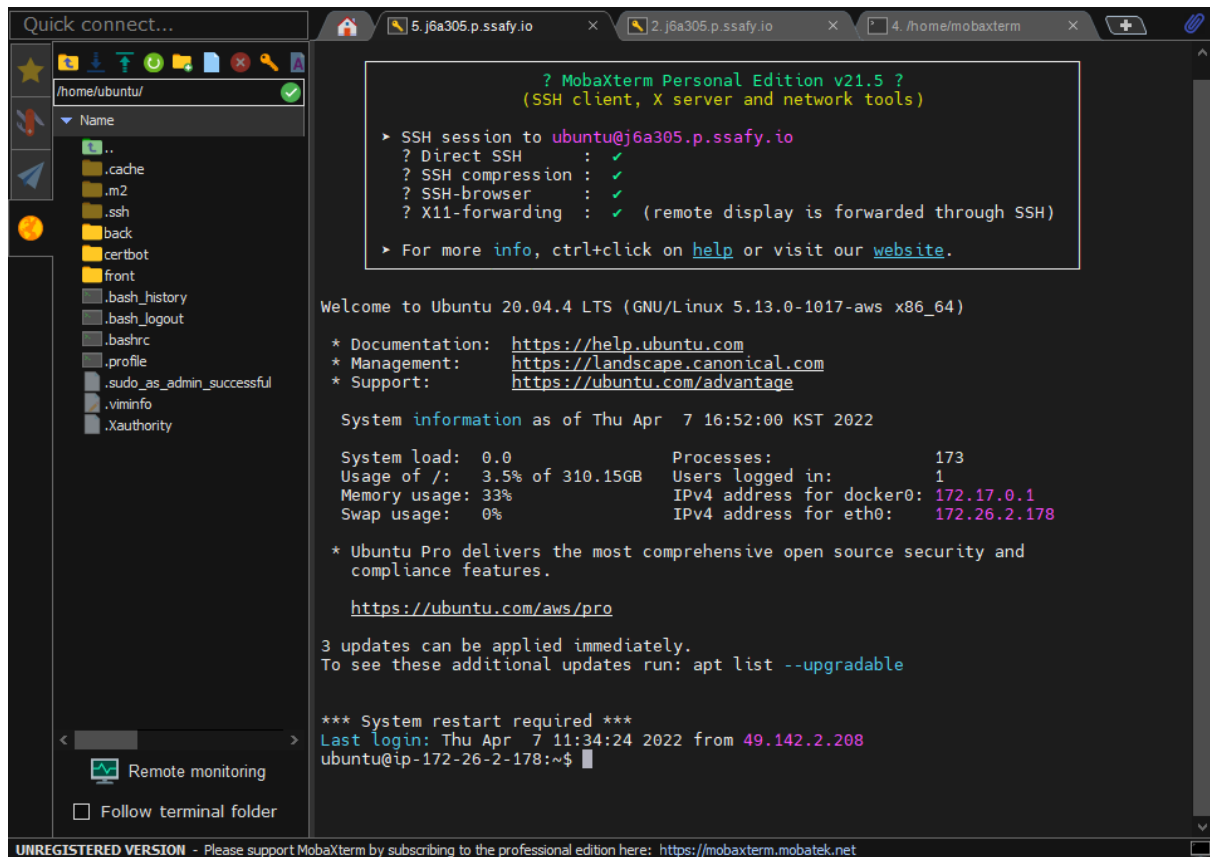
host에 부여받은 주소 (j6a305.p.ssafy.io) 입력

username에는 ubuntu 입력



Advanced SSH settings 에서 sshkey.pem 파일을 매칭 시켜서

OK 클릭하면 SSH 접속이 가능합니다.



EC2 접속 후

```

~$ sudo apt-get update
~$ sudo apt-get upgrade

```

기본적으로 사용되는 라이브러리들을 업데이트 해줍니다.

Git 연결

git clone 으로 받은 연결에 추가적으로 인증없이 사용하고 싶다면 다음 명령어를 순차적으로 실행

```

git config --global user.name (이름)
git config --global user.email (이메일)

# .git 디렉토리가 존재하는 위치에서
git config credential.helper store

```

명령어를 순차적으로 실행한다면 다음부터는 추가 인증 없이 사용 가능하다

스프링 설치

메이븐 설치

```
~$ sudo apt install maven
```

버전 확인

```
~$ mvn -v
Apache Maven 3.6.3
Maven home: /usr/share/maven
Java version: 1.8.0_312, vendor: Private Build, runtime: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "5.4.0-1018-aws", arch: "amd64", family: "unix"
```

깃 클론

```
git clone https://lab.ssfy.com/s06-blockchain-sub2/S06P22A305.git
```

명령어 입력후 계정과 비밀번호 입력

`pom.xml` 이 있는 디렉토리에서

```
$ mvn package
```

target 디렉터리 생성 뒤 [persona-0.0.1-SNAPSHOT.jar] 파일 생성

Dockerfile

```
FROM openjdk:8-jdk-alpine
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

스프링 도커로 빌드

```
docker build -t spring .
```

스프링 컨테이너 시작

```
docker run -d --rm -p 9999:9999 -v files:/files --name spring spring
```

스프링 접속이 가능해집니다.

AWS에 도커 설치

```
~$ sudo apt install apt-transport-https
~$ sudo apt install ca-certificates
~$ sudo apt install curl
```

컬은 특정한 웹사이트에서 데이터를 다운받을때 사용.

Docker 공식 GPG 키 등록

```
~$ sudo apt install software-properties-common
~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

저장소 설정

```
~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
~$ sudo apt update
```

도커 설치

```
~$ apt-cache policy docker-ce
~$ sudo apt install docker-ce
```

도커 명령어

도커 실행 로그 실시간 확인

```
docker logs -t -f [docker name]
```

도커 리스트 확인

```
docker ps (-a)
```

-a 옵션을 추가할 경우 실행중인 컨테이너만 확인 가능

도커 중지/시작

```
docker stop/start [docker name]
```

도커 컨테이너 삭제

```
docker rm [docker name]
```

도커 이미지 파일 확인

```
docker images
```

도커 이미지 삭제

```
docker rmi [image name]
```

도커 볼륨 리스트 확인

```
docker volume ls
```

도커 사용하지 않는 볼륨 삭제

```
docker volume prune (y/n)
```

docker에 mysql 실행

mysql의 경우 docker 에 이미지가 존재하기 때문에 실행 가능

```
docker run -d -p 3306:3306 - --name my sql -e MYSQL_ROOT_PASSWORD=ssafytest1234 -d mysql:8.0
```

mysql docker 접속 하여 확인

```
~$docker exec -it mysql /bin/bash  
root@a1286eac2ea0:/# mysql -u root -p
```

```
ubuntu@ip-172-26-7-250:~/S06P12A506/backend/target$ docker exec -it mysql /bin/bash  
root@a1286eac2ea0:/# mysql -u root -p  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 2095  
Server version: 8.0.28 MySQL Community Server - GPL  
  
Copyright (c) 2000, 2022, Oracle and/or its affiliates.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql> █
```

mysql접속 성공!

사용자 생성

```
create user 'testuser'@'%' identified by 'password';
```

외부에서 접속 할 수 있도록 권한 부여

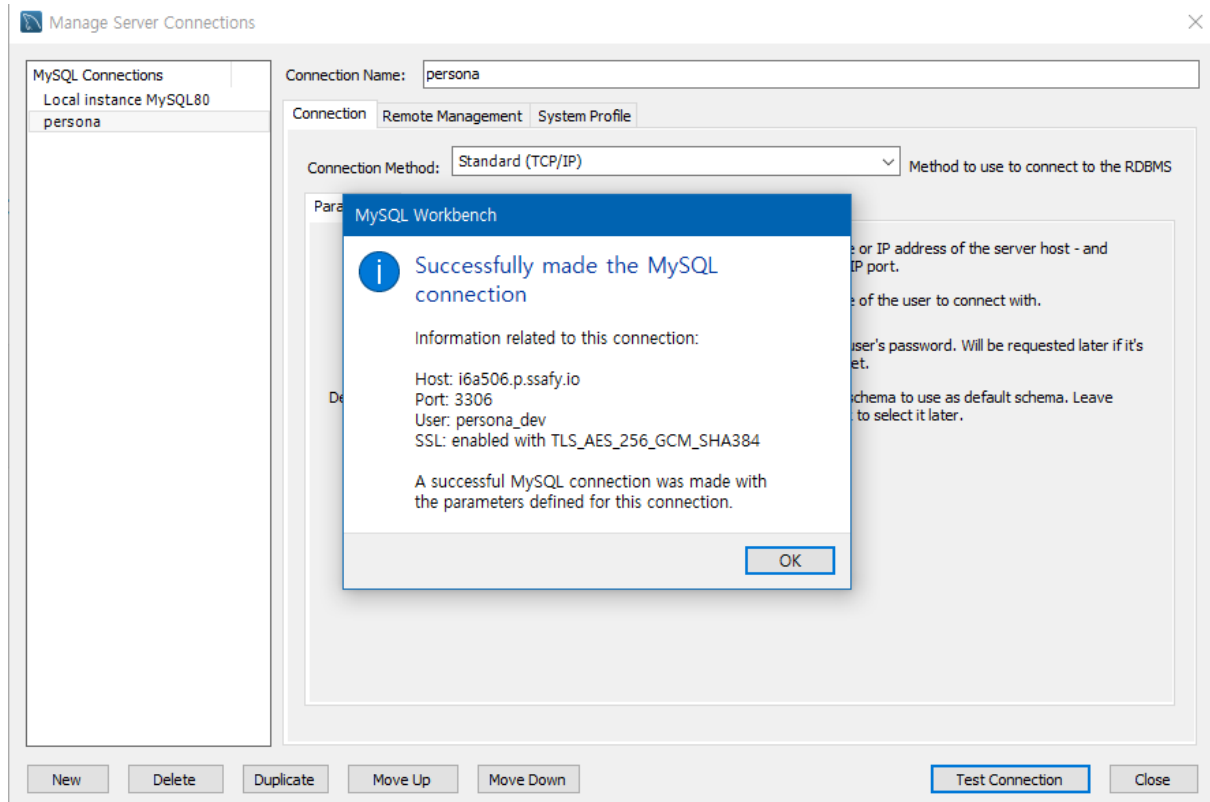
```
grant all privileges on . to 'testuser'@'%';
```

변경된 권한을 적용

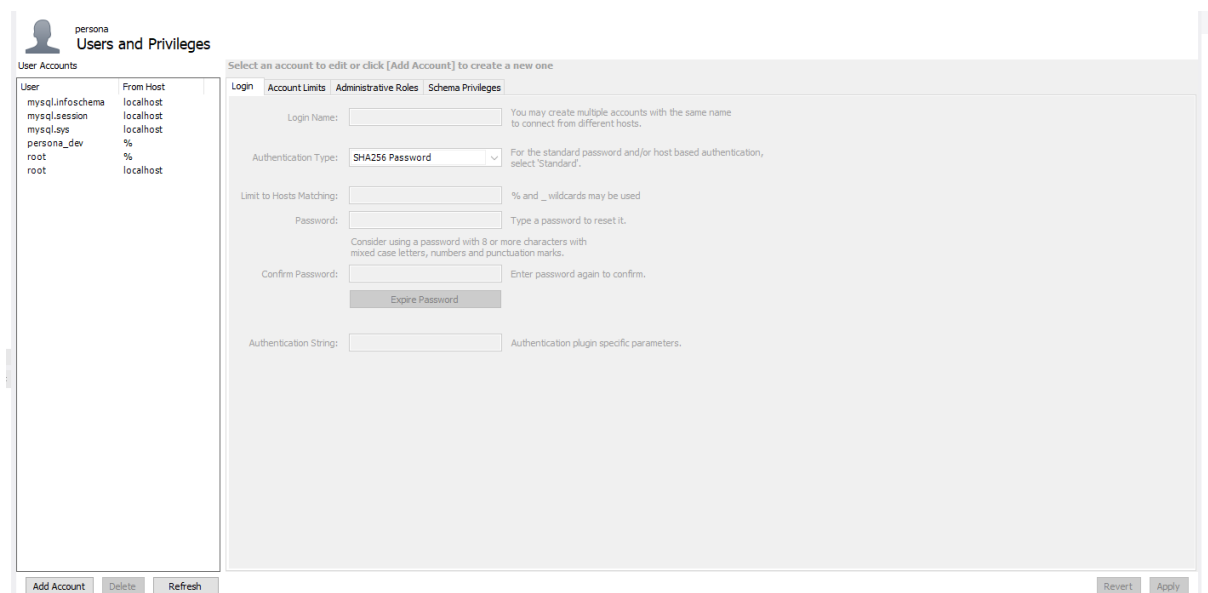
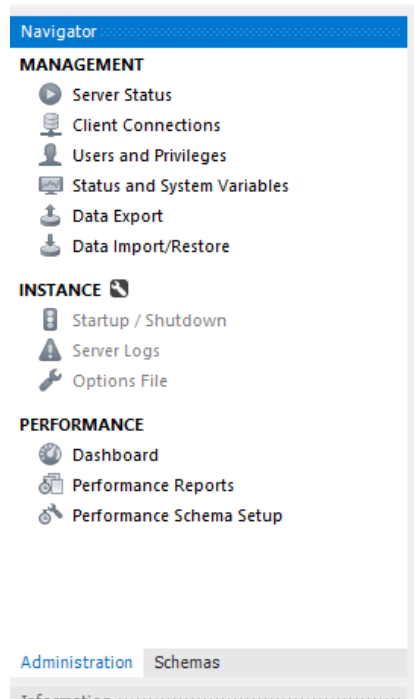
```
flush privileges;
```

<https://downloads.mysql.com/archives/installer/>

접속 하여 window에 mysql 다운로드



접속하여 좌측 Administration 클릭 후 Users and Privileges 클릭



적절한 권한을 지닌 계정을 생성하여 불필요한 root 계정 사용을 막을 수 있습니다.

NginX 설치

`sudo apt-get install nginx` 명령어로 nginx 설치

`sudo apt-get remove nginx` 명령어로 nginx 삭제

`sudo apt-get purge nginx nginx-common` 모든것을 깔끔하게 정리하기위함

설치한 뒤 확인하니 `/etc/nginx` 폴더가 생성되어있음을 확인

`cd /etc/nginx/conf.d` 로 conf.d 디렉토리로 이동

나만의 .conf 파일을 하나 만들어준다

```
# myssl.conf

server{
    server_name j6a305.p.ssafy.io www.j6a305.p.ssafy.io;
    listen 80;
    return 301 https://$host$request_uri;
}

server{

    listen 443 ssl;
    server_name www.j6a305.p.ssafy.io j6a305.p.ssafy.io;

    access_log off;

    proxy_connect_timeout 1d;
    proxy_send_timeout 1d;
    proxy_read_timeout 1d;

    ssl_certificate /etc/letsencrypt/live/j6a305.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j6a305.p.ssafy.io/privkey.pem;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_redirect off;
    }
    location /api {
        proxy_pass http://127.0.0.1:9999;
        proxy_redirect off;
    }
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```

80 포트로 들어온 api들은 `/api` 로 분기하여 443 포트에서 3000번 포트와 9999포트로 전송

또한, spring에서 `application.properties` 에 `server.servlet.context-path=/api` 를 넣어줌으로써 spring api들은 항상 `/api` 으로 시작할수있도록 설정

`systemctl start nginx` 로 nginx 시작

`systemctl restart nginx` 로 nginx 재부팅

젠킨스 설치

작업의 편리를 위한 도커의 권한부여

```
sudo chmod 006 /var/run/docker.sock
```

이미지 다운로드

```
docker pull jenkins/jenkins:jdk11
```

젠킨스 컨테이너 실행

```
docker run -d --restart always -p 8080:8080 -v /jenkins:/var/jenkins_home --name jenkins -u root jenkins/jenkins:jdk11
```

- **d** 는 데몬, 백그라운드에서 계속 실행
- **-restart** 는 호스트 재부팅 시, 자동으로 컨테이너 재시작
- **p** 는 포트바인딩, EC2의 8080포트를 도커 컨테이너의 8080포트와 연결한다는 의미. 8080포트는 젠킨스의 디폴트 port
- **v** 는 볼륨 마운팅,
 - 볼륨은, 호스트(내 PC)의 디스크공간(볼륨)을 도커 컨테이너와 함께 사용, 그래서 /jenkins라는 디렉토리를 컨테이너 내의 /var/jenkins_home과 연동하게 되는데, 컨테이너에서도 해당 디렉토리 내부에 read+write가 가능하고, 호스트에서도 가능. 볼륨을 연결하므로써 컨테이너가 삭제되더라도 지워져선 안 되는 데이터를 보존 또는 쉽게 호스트와 컨테이너와의 파일을 공유가능
- **-name** 은 컨테이너의 이름. 여기서는 jenkins로 설정
- **u** 는 유저 이름. 여기서는 root로 설정

웹 콘솔로 접속

<http://j6a305.p.ssafy.io:8080/>

로 접속하면 젠킨스 기본 설정 페이지로 이동

Unlock Jenkins

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/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

초기 비밀번호 입력 요구가 뜨는데 해당 부분에서 확인 가능
(또는 젠킨스를 실행할 때 로그를 확인 하거나 볼륨으로 연결되었으니
`cat /jenkins/secrets/initialAdminPassword` 를 통해 받아올 수 도 있습니다.)

비밀번호를 입력하면 해당창이 뜹니다.

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.

추천 플러그인을 설치한 후

Getting Started

Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

Jenkins 2.295

[Skip and continue as admin](#)

[Save and Continue](#)

젠킨스 계정을 생성합니다. (해당 계정으로 젠킨스에 로그인할 수 있습니다.)

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 2.295

Not now

Save and Finish

마지막 URL 입력창에는 EC2의 퍼블릭 IP를 적으면 됩니다.

The screenshot shows the Jenkins Dashboard with a sidebar on the left containing links to '새로운 Item', '시작', '빌드 기록', 'Jenkins 관리', 'My Views', 'Lockable Resources', and '새로운 뷰'. The main area displays a table of jobs:

S	W	Name	최근 성공	최근 실패	최근 소요 시간
✓	🔍	back	2 hr 2 min #73	2 hr 31 min #68	19 sec
✓	🔍	front	3 min 27 sec #26	2 hr 46 min #18	1 min 40 sec

Below the table, there are links for '아이콘: S M L', 'Icon legend', 'Atom feed 모두', 'Atom feed 실패', and 'Atom feed 최근 빌드'.

접속 성공 시 화면

도커파일로 백엔드 컨테이너 제작

```
~$ cd home/ubuntu/back/back/
```

아래의 명령어로 Dockerfile 파일 작성

```
~/back/back$ vim Dockerfile
```

```
FROM openjdk:8-jdk-alpine
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

vi는 리눅스의 텍스트 문서 작성 프로그램으로 vi 또는 vim으로 생성/수정할 수 있습니다.

작성 시에는 insert에 해당하는 i 로 들어가서 작성하고 작성 후에는 esc 키를 눌러 저장 모드로 나올 수 있습니다.

저장

→ :wq

강제 종료

→ :q!

```
Step 1/4 : FROM openjdk:8-jdk-alpine
---> a3562aa0b991
Step 2/4 : ARG JAR_FILE=target/*.jar
---> Running in d2ef76c1e658
Removing intermediate container d2ef76c1e658
---> 39a0cdb7bd17
Step 3/4 : COPY ${JAR_FILE} app.jar
---> 5748a92f1136
Step 4/4 : ENTRYPOINT ["java","-jar","/app.jar"]
---> Running in e254df6d592b
```

컨테이너 생성 시 각 문장은 step 으로 나뉘어 동작 하게 됩니다.

```
FROM openjdk:8-jdk-alpine
```

어떤 도커 이미지를 사용하여 빌드할 지 정합니다.

```
ARG JAR_FILE=target/*.jar
```

target 디렉터리 아래에 만들어진 jar 파일을 JAR_FILE 이라는 변수에 저장합니다.

```
COPY ${JAR_FILE} app.jar
```

JAR_FILE 에 저장한 파일을 현재 폴더에 app.jar로 복제 생성합니다.

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

java -jar 명령어를 통해 app.jar를 실행합니다.

docker 컨테이너 생성

```
docker build -t spring .
```

back이라는 도커 이미지가 생성된것을 확인

```
docker images
```









```
ubuntu@ip-172-26-2-178:~/back/back$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
front                latest             cff58d3a0556       7 minutes ago      1.68GB
spring               latest             bf49d975920f       2 hours ago        154MB
server               latest             a4fcc98d3ff1       8 days ago         142MB
jenkins/jenkins     jdk11              9c941210afa1       8 days ago         460MB
```

docker 실행

```
sudo docker run -d --name back -v files:/files -p 9999:9999 spring
```


파일 정보를 저장하기 위해 files 디렉터리를 마운트 하여 실행


젠킨스 설정


-  새로운 Item
-  사람
-  빌드 기록
-  프로젝트 연관 관계
-  파일 핑거프린트 확인
-  Jenkins 관리
-  My Views
-  Lockable Resources
-  New View


젠킨스 관리에 들어가서


System Configuration

 시스템 설정
환경 변수 및 경로 정보등을 설정합니다.

 Global Tool Configuration
Configure tools, their locations and automatic installers.

 플러그인 관리
Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.
▲ 업데이트 가능함

 노드 관리
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.






 Managed files
e.g. settings.xml for maven, central managed scripts, custom files, ...

플러그인 관리 클릭

업데이트된 플러그인 목록			설치 가능		설치된 플러그인 목록		고급	
이름	버전	설명	사용 가능	설치	이름	버전	설명	사용 가능
JSch dependency plugin	0.1.55.2	Jenkins plugin that brings the JSch library as a plugin dependency, and provides an SSHAuthenticatorFactory for using JSch with the ssh-credentials plugin. Report an issue with this plugin	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
SSH Build Agents plugin	1.33.0	Allows to launch agents over SSH, using a Java implementation of the SSH protocol. Report an issue with this plugin	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
SSH Credentials Plugin	1.19	Allows storage of SSH credentials in Jenkins Report an issue with this plugin	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
SSH plugin	2.6.1	This plugin executes shell commands remotely using SSH protocol. Report an issue with this plugin	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
SSH server	3.1.0	Adds SSH server functionality to Jenkins, exposing CLI commands through it	<input checked="" type="checkbox"/>	<input type="checkbox"/>				

SSH 라고 검색하여 위의 해당하는 플러그인을 설치 후 재시작 해줍니다.

Security 항목에서 Mange Credentials 클릭

Security			
	Configure Global Security Secure Jenkins: define who is allowed to access/use the system.		Manage Credentials Configure credentials
	In-process Script Approval Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.		Configure Credential Providers Configure the credential providers and types
			Manage Users Create/delete/modify users that can log in to this Jenkins

New credentials

Kind
SSH Username with private key

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

ID ?

Description ?
AWS EC2

Username
ubuntu

☐ Treat username as secret ?

Private Key
☒ Enter directly

Key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA...
-----END RSA PRIVATE KEY-----
```

Passphrase

Create

해당 항목으로 인증키를 생성 해 줍니다.

Scope

Global 로 설정해 줍니다.

ID

구분 하기 위한 값입니다. 비워주고 생성하면 자동으로 값이 들어갑니다.

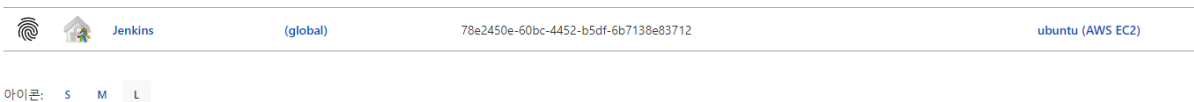
Description

설명을 위한 부분입니다. 필수값은 아니니 적당히 써도 됩니다.

Username

꼭 필요한 부분입니다. 어떤 계정으로 들어갈 지 정해주는 부분이며 우리가 사용하는 AWS기준으로는 “ubuntu”라고 정확히 입력해줘야만 했습니다.

추가로 **Enter directly**를 눌러 key입력창을 활성화 한 다음 pem 키 내부에 있는 RSA private key를 입력해서 저장해 줍니다.



목록에 생성된 것을 확인 가능 합니다.

다시 시스템 설정으로 돌아가서

SSH remote hosts

SSH sites

Hostname ?

172.26.7.250

Port ?

22

Credentials

ubuntu (AWS EC2) ▾

➕Add ▾

Pty ?

serverAliveInterval ?

0

timeout ?

0

Check connection

삭제


SSH sites 로 연결후 Credentials 에서 방금 추가한 키를 선택


Successful connection


Check connection


Check connection을 누르면 성공하는 것을 확인할 수 있습니다.


이후 새로운 아이템을 클릭


 새로운 Item


 사람


 빌드 기록

 프로젝트 연관 관계

 파일 핑거프린트 확인

 Jenkins 관리


 My Views


 Lockable Resources


 New View


Enter an item name


» Required field


 **Freestyle project**
이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

 **Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.


 **Multi-configuration project**
다양한 환경에서의 테스트, 플러그인 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.

 **Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

 **Multibranch Pipeline**
Creates a set of Pipeline projects according to detected branches in one SCM repository.

 **Organization Folder**
Creates a set of multibranch project subfolders by scanning for repositories.






If you want to create a new item from other existing, you can use this option:

 Copy from

OK

이름을 지정하고 Freestyle project 로 생성해줍니다.

Dashboard ▶ **gitlab_connect** ▶

-  대시보드로 돌아가기
-  상태
-  변경사항
-  작업공간
-  Build Now
-  구성
-  Project 삭제
-  Rename

Project gitlab_connect

 **작업 공간**

 **최근 변경사항**

고정링크

- Last build, (#51), 2 hr 46 min 전
- Last stable build, (#51), 2 hr 46 min 전

해당 프로젝트 내부로 들어와서 구성을 선택하고

☒ Execute shell script on remote host using ssh ?

SSH site

ubuntu@3.38.98.84:22

Pre build script ?

```
cd /home/ubuntu/back/back
docker stop spring
docker rmi spring
sudo rm -rf target
sudo git pull origin develop
sudo mvn package
docker build -t spring .
docker run -d --rm -p 9999:9999 -v files/files --name spring spring
```

















Post build script ?

☐ Hide command from console output

☐ Inspect build log for published Gradle build scans

☐ With Ant ?

빌드 환경에서 ssh와 연결지어서 빌드를 눌렀을 때 ec2에서 실행될 명령어들을 작성해줍니다.

	#73	2022. 4. 7. 오후 2:18
	#72	2022. 4. 7. 오후 1:57
	#71	2022. 4. 7. 오후 1:56
	#70	2022. 4. 7. 오후 1:55
	#69	2022. 4. 7. 오후 1:54
	#68	2022. 4. 7. 오후 1:49
	#67	2022. 4. 7. 오후 1:38
	#66	2022. 4. 7. 오후 1:33
	#65	2022. 4. 7. 오후 1:28
	#64	2022. 4. 7. 오후 1:25
	#63	2022. 4. 7. 오후 1:22
	#62	2022. 4. 7. 오후 1:20
	#61	2022. 4. 7. 오전 10:48
	#60	2022. 4. 7. 오전 10:48
	#59	2022. 4. 7. 오전 2:02
	#58	2022. 4. 7. 오전 1:28

이후 빌드 히스토리와 콘솔내역을 통해 빌드정보를 알 수 있습니다.

FrontEnd

npm 설치

```
apt install npm
```

최신버전이 설치가 안된다면 다음 명령어를 순차적으로 실행

```
# 1
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
# Zsh 사용자
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | zsh

# 2
source ~/.bashrc
# Zsh 사용자
source ~/.zshrc

# 3
nvm -v

# 4
nvm install node # 최신 버전 설치
nvm install --lts # 최신 LTS 버전 설치
nvm install 16.14.0 # 특정 버전 설치
nvm install 16 # 특정 버전 16의 최신 릴리즈 설치
```

front root에 Dockerfile를 생성한다

Dockerfile은 다음과 같다

```
FROM node
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm i --force -y
COPY . .
EXPOSE 3000
RUN npm run build
CMD [ "npm", "run", "start"]
```

Dockerfile이 존재하는 위치로 이동

도커 빌드

```
docker build -t front .
```

도커 컨테이너 실행

```
docker run -d --rm -p 3000:3000 -v files:/usr/src/app/public/files --name front front
```