

REPORT

젠킨스 튜토리얼 제작



과목명 : 클라우드컴퓨팅 (2분반)
교 수 : 남재현 교수님
소속학과: 소프트웨어학과
학 번 : 32190192
이 름 : 구영민
마 감 : 2024.06.11 (화요일까지)

목 차

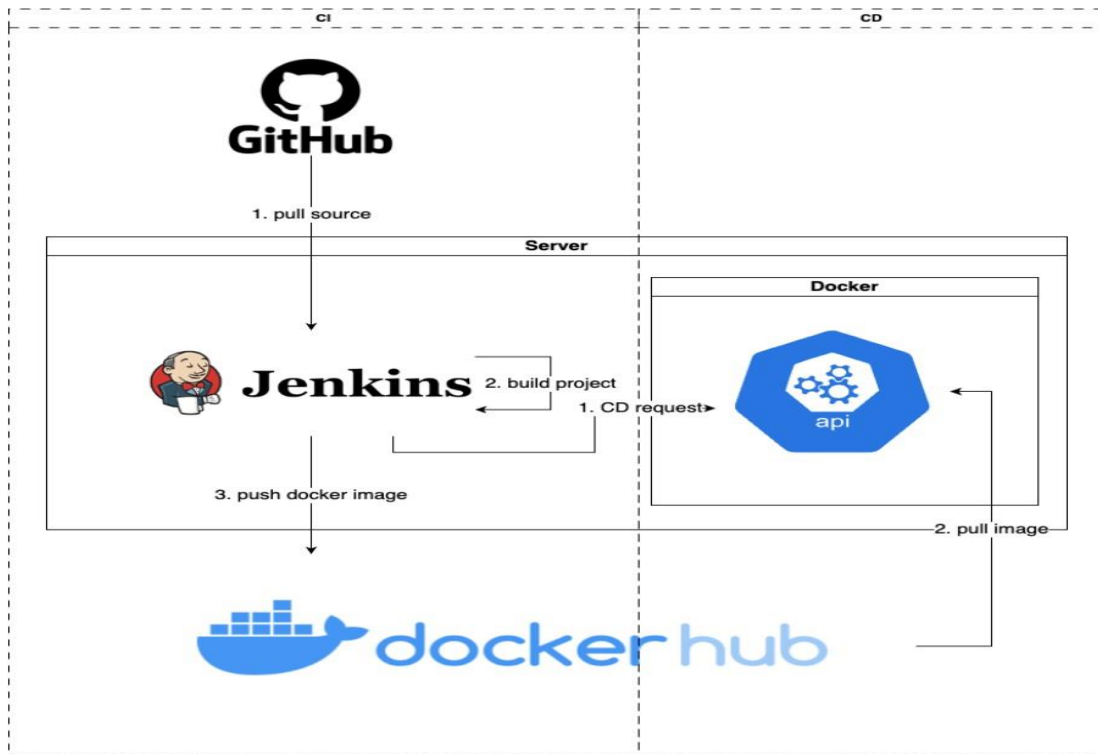
1. 젠킨스.....	5
1) Jenkins란?	5
2) CI (Continuous Integration)	5
3) CD (Continuous Deployment)	5
2. 젠킨스 실습.....	6
1) 젠킨스 이미지 다운로드	6
2) Jenkins에서 Github 연결	8
3) github에서 Jenkins 연동	9
4) Jenkins Pipeline 생성	11
5) Ssh 연동	12
6) Publish over SSH 파이프라인 작성	13
7) 빌드하기	16
8) Docker와 Jenkins 연결	17
9) 젠킨스 적용 확인	22

사진 목차

사진1	5
사진2	6
사진3	6
사진4	7
사진5	8
사진6	8
사진7	9
사진8	10
사진9	11
사진10	12
사진11	12
사진12	12
사진13	14
사진14	15
사진15	16
사진16	16
사진17	17
사진18	18
사진19	18
사진20	19
사진21	20
사진22	22

사진23	22
사진24	23
사진25	23
사진26	24
사진27	24
사진28	25
사진29	25
사진30	26
사진31	26

1. 젠킨스란



[사진 1] CI/CD

1. Jenkins란?

Jenkins는 오픈 소스 자동화 서버로서, 소프트웨어 개발과 배포 프로세스를 자동화하는 데 사용됩니다. CI/CD(Continuous Integration/Continuous Deployment) 파이프라인의 핵심 구성 요소로서, 개발자들이 코드 변경 사항을 지속적으로 통합하고 배포할 수 있도록 도와줍니다. 아래는 Jenkins를 활용하여 소프트웨어 개발 및 배포를 자동화하는 프로세스를 설명합니다.

2. CI (Continuous Integration)

a. 소스 코드 가져오기 (Pull Source)

- Jenkins는 GitHub에서 소스 코드를 가져옵니다. 이를 통해 최신 코드 변경 사항을 지속적으로 반영할 수 있습니다.
- 이는 Jenkins가 GitHub 저장소를 폴링하거나 웹훅을 통해 트리거되도록 설정하여 자동화할 수 있습니다.

b. 프로젝트 빌드 (Build Project)

- Jenkins는 Gradle, Maven, 또는 기타 빌드 도구를 사용하여 프로젝트를 빌드합니다.
- 이 단계에서 코드 품질 검사를 수행하고, 유닛 테스트를 실행하여 코드의 안정성을

확인합니다.

c. Docker 이미지 푸시 (Push Docker Image)

- Jenkins는 빌드된 Docker 이미지를 Docker Hub에 푸시합니다.
- 이렇게 저장된 이미지는 나중에 배포 단계에서 사용됩니다.

3. CD (Continuous Deployment)

a. 배포 요청 (CD Request)

- Jenkins는 배포 요청을 받아, 지정된 타겟 서버 또는 환경에 배포를 시작합니다.

b. 이미지 가져오기 (Pull Image)

- Jenkins는 Docker Hub에서 필요한 Docker 이미지를 가져옵니다.

c. 배포 (Deploy)

- Jenkins는 가져온 Docker 이미지를 사용하여 애플리케이션을 배포합니다. 이 과정은 Kubernetes와 같은 오케스트레이션 도구를 사용하여 자동화할 수 있습니다.

2. 젠킨스 실습

1. 젠킨스 이미지 다운로드

```
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ docker pull jenkins/jenkins
Using default tag: latest
latest: Pulling from jenkins/jenkins
Digest: sha256:647322e93b23879559222bf4aee67c914a98df9c643a89bebab03d7db50c532
Status: Image is up to date for jenkins/jenkins:latest
docker.io/jenkins/jenkins:latest
```

[사진 2] docker pull

이미지를 다운로드합니다.

```
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
jenkins/jenkins     latest         3aaba8413e04   4 days ago     469MB
```

[사진 3] 젠킨스 이미지 확인

다운로드를 확인합니다.

[명령어 설명]

```
docker run -d -p 9090:8080 -v /jenkins:/var/jenkins_home --name jenkins -u root
jenkins/Jenkins
```

-d

이 옵션은 컨테이너를 백그라운드에서 실행하도록 설정합니다. 즉, 터미널을 차지하지 않고 백그라운드에서 계속 실행되게 합니다.

-p

포트 매핑 옵션입니다. 왼쪽의 포트 번호는 호스트(외부) 포트 번호이고, 오른쪽의 포트 번호는 컨테이너 내부의 포트 번호입니다. 예를 들어, -p 9090:8080은 호스트의 9090 포트를 컨테이너 내부의 8080 포트와 연결합니다. 따라서 호스트의 9090 포트에 접속하면 컨테이너의 8080 포트에 트래픽이 전달됩니다.

-v

볼륨 마운트 옵션입니다. 이 옵션은 호스트와 컨테이너 간의 디렉토리 공유를 설정합니다. 예를 들어, -v /jenkins:/var/jenkins_home은 호스트의 /jenkins 디렉토리를 컨테이너 내부의 /var/jenkins_home 디렉토리와 연결합니다. 이렇게 하면 컨테이너 내부의 Jenkins 데이터가 호스트의 /jenkins 디렉토리에 저장되어, 컨테이너가 삭제되더라도 데이터가 유지됩니다. 이 명령어를 통해 Jenkins를 루트 사용자로 실행하며, 호스트의 /jenkins 디렉토리와 컨테이너의 /var/jenkins_home 디렉토리를 연결하고, 호스트의 9090 포트를 컨테이너의 8080 포트와 매핑하여 실행합니다.

```
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
2ce3a59a05f5	jenkins/jenkins	"/usr/bin/tini -- /u..."	6 hours ago	Up 6 hours
		50000/tcp, 0.0.0.0:9090->8080/tcp, :::9090->8080/tcp		jenkins



[사진 4] 젠킨스 이미지 확인

실행중인 것을 확인할 수 있습니다.

2. Jenkins에서 Github 연결

Jenkins 관리 > Manage Credentials

Credentials

T	P	Store ↓	Domain
		System	(global)

[사진 5] Credential 등록 과정1

global 클릭

<input type="checkbox"/>	이름 ↓	Released	설치됨
<input type="checkbox"/>	Publish Over SSH 1.25 Artifact Uploaders Build Tools Send build artifacts over SSH	10 mo ago	1.24

Scope ?


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

Username ?

rn9dud0@naver.com

☐ Treat username as secret ?

Password ?

 Concealed Change Password

ID ?

jenkins-ss-server

Description ?

jenkins-ss-server

Save

[사진 6] Credential 등록 과정2

GitHub 자격 증명 설정

Username

GitHub 사용자 이름을 입력합니다.

Password

GitHub 계정 비밀번호를 입력합니다. 토큰 연동 방식을 사용할 수도 있지만, 여기서는 간단

하게 비밀번호를 사용합니다.

ID

이 값은 Jenkins에서 사용될 식별자입니다. Jenkins에서 사용할 GitHub 변수와 유사하다고 생각하면 됩니다.

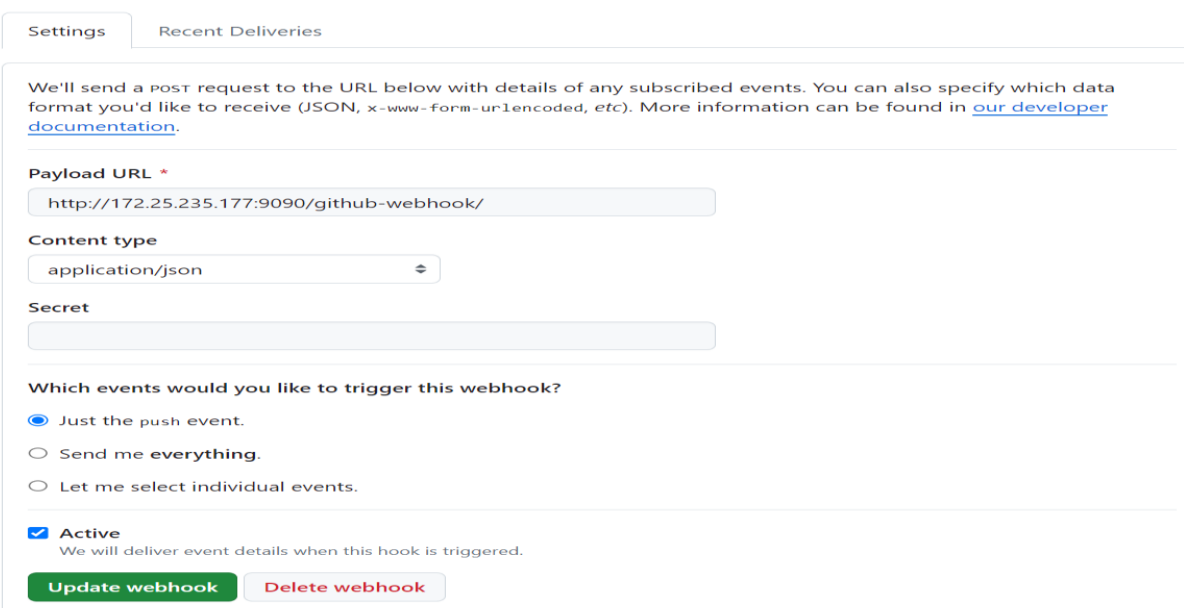
Description

식별 가능한 설명을 입력합니다. 향후 여러 키나 인증 정보가 등록될 경우, 이를 구분하기 쉽게 하기 위해 Description을 작성해두면 관리가 편리합니다.

3. github에서 Jenkins 연동

Repository에서 Settings > Webhooks

Webhooks / Manage webhook



Settings Recent Deliveries

We'll send a post request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *

Content type

Secret

Which events would you like to trigger this webhook?

☒ Just the push event.

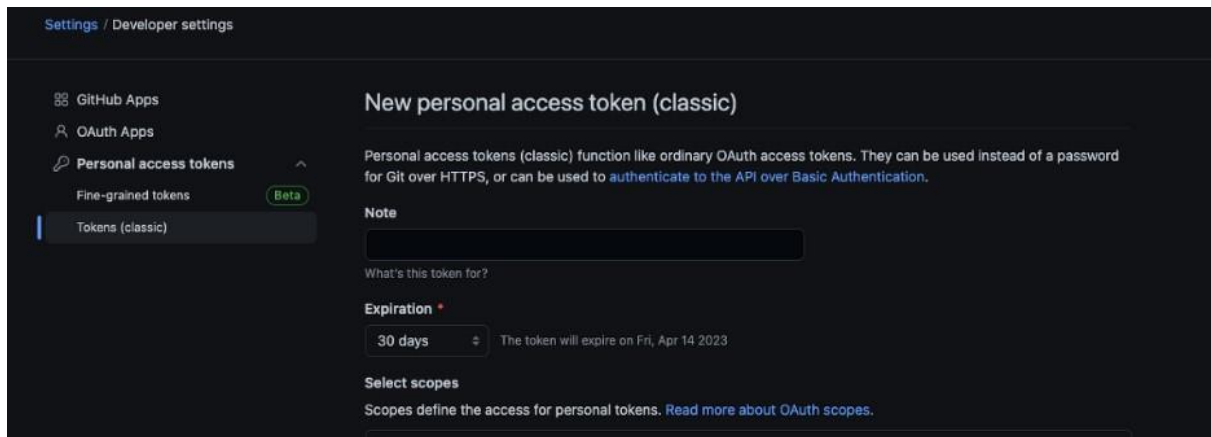
☐ Send me everything.

☐ Let me select individual events.

☒ **Active**
We will deliver event details when this hook is triggered.

[사진 7] github연동 과정1

끝 부분에 github-webhook이라고 적어주기



[사진 8] github연동 과정2

GitHub 개인 액세스 토큰 생성

GitHub 로그인

GitHub 계정에 로그인합니다.

설정 이동

오른쪽 상단의 프로필 아이콘을 클릭한 후, 드롭다운 메뉴에서 Settings를 선택합니다.

개발자 설정

설정 페이지의 왼쪽 사이드바에서 Developer settings를 클릭합니다.

개인 액세스 토큰

Personal access tokens 메뉴를 선택하고 Generate new token 버튼을 클릭합니다.

토큰 생성








토큰에 이름과 필요한 권한을 지정한 후, Generate token 버튼을 클릭하여 새로운 토큰을 생성합니다.

필요한 권한

repo : 리포지토리에 접근하고 조작할 수 있는 권한입니다.

admin:repo_hook : 리포지토리 웹훅을 관리할 수 있는 권한입니다.

4. Jenkins Pipeline 생성

 **Jenkins**      

Dashboard > All > New Item

New Item

Enter an item name

jenkins-ss-server

» A job already exists with the name 'jenkins-ss-server'

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



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, 플레폼 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

[사진 9] Pipeline 생성

Jenkins home -> 새로운 item -> pipeline -> 이름 설정

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Quiet period ?
- ☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

Build Triggers 부분에 Github의 Webhook 기능이 활성화 될 수 있도록 체크

5. Ssh 연동

Publish over SSH

Jenkins SSH Key ?

Passphrase ?

 Concealed

Path to key ?

Key ?

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktbjEAAAAAAAAAGSvbmUAAAAEbm9uZQAAAAAAAAABAAA8lWAAAdzc2gtcn
NhAAAAAwEAAQAAAEAnTCzVmCj8OCC7TBkU7/T/U0lx9O21SlidyXDm4LCcOnJpdmCuHyTq
O1w0BuM81688bwSEHq5R8LMHe6D/Cp6LzDsMj5hPZL0eAqbsXaA4ZkRaAEGF873WcHICIP
JhszhBCT11UQTCHYfqWwJl3AfeM7jyNwTwow6YGUoAt/WO3nnNdVnhxwHqk8zIWVAZwW1e
x22tj5u/hHMCiHFlv1kiXDbAiQitMDiBmx8Gz5bYb18nSatQunCLYDTcl7uCWsbGEW9p6
gbjna/JFY5Sszsl6iHsCMYiRBlqCK+85qtnaodlNlralma8L1pJvl5gE1V1Vxvly/1HD
-----
```

[사진 10] ssh연동 과정1

Jenkins 홈 -> Jenkins 관리 -> system
두가지를 설정한다.

SSH Servers

SSH Server

Name ?

ssh

Hostname ?

172.25.235.177

Username ?

ubuntu

Remote Directory ?

/home/ubuntu

고급 ▼

[사진 11] ssh연동 과정2

Jenkins 설정

Jenkins 홈 -> 관리 -> 시스템 설정 : Jenkins 홈 화면에서 Manage Jenkins를 클릭한 후, Configure System을 선택합니다.

SSH 키 설정

상단에 id_rsa 키 값 입력 : 화면 상단에 복사한 id_rsa 키 값을 입력합니다.

SSH 설정

- Name : 식별할 수 있는 임의의 이름을 입력합니다.
- Hostname : SSH로 배포될 대상 서버의 외부 IP 주소를 입력합니다. SSH 연결을 위해 포트 22번이 포트포워딩을 통해 열려 있어야 합니다.

- Username : 연결할 서버의 사용자 이름을 입력합니다.
- Remote Directory : JAR 파일이 배포될 경로를 입력합니다. 이 폴더는 사전에 생성되어 있어야 합니다.

6. Publish over SSH 파이프라인 작성

Pipeline

Definition

Pipeline script

Script ?

```

1 pipeline {
2   agent any
3
4   stages {
5     stage('github Clone') {
6       steps {
7         git branch: 'main', credentialsId: 'jenkins-ss-server', url: 'https://github.com/rndudals/cloud-computing-bb-server.git'
8       }
9     }
10    stage('build') {
11      steps {
12        sh '''
13          echo 'start bootJar'
14          chmod +x gradlew
15          ./gradlew clean bootJar
16          '''
17      }
18    }
19  }
20 }
```

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

[사진 12] SSH 파이프라인 작성

```

pipeline {
  agent any
  stages {
    stage('github Clone') {
      steps {
        git branch: 'main', credentialsId: 'jenkins-ss-server',
          url: 'https://github.com/rndudals/Cloud-Computing-bb-server.git'
      }
    }
    stage('build') {
      steps {
        sh '''
          echo 'start bootJar'
          chmod +x gradlew
          ./gradlew clean bootJar
          '''
      }
    }
    stage('deploy') {
      steps {
        sshPublisher(
          publishers: [
            sshPublisherDesc(
              configName: 'ssh',
              transfers: [
                sshTransfer(
                  cleanRemote: false,
                  excludes: '',
                  execCommand: 'sh /home/ubuntu/bb-server.sh',
                  execTimeout: 120000,
                  flatten: false,
                  makeEmptyDirs: false,
                  noDefaultExcludes: false,
                  patternSeparator: '[, ]+',
                  remoteDirectory: '/project/server',
                  remoteDirectorySDF: false,
                  removePrefix: 'build/libs',
                  sourceFiles: 'build/libs/*.jar'
                )
              ],
              usePromotionTimestamp: false,
              useWorkspaceInPromotion: false,
              verbose: true
            )
          ]
        )
      }
    }
  }
}

```

[사진 13] 파이프라인 코드

Pipeline

파이프라인 전체 과정을 포함하는 블록을 정의합니다.

agent any

이 파이프라인이 Jenkins의 어떤 에이전트에서든 실행될 수 있음을 명시합니다.
 서버에서 셸스크립트를 생성합니다.

Stages

여러 단계(stage)로 구성된 작업을 정의하는 부분입니다.

stage('github clone')

github clone'이라는 이름의 첫 번째 단계를 시작합니다. 이 단계에서는 GitHub에서 코드를 복제하는 작업이 이루어집니다.

Steps

각 단계에서 수행할 구체적인 명령들을 정의합니다.

Git

Git 명령어를 사용하여 지정된 브랜치('main')에서 코드를 복제합니다. 이때 'jenkins-ss-server'라는 인증 정보를 사용하고, 주어진 URL에서 리포지토리를 복제합니다.

stage('build')

'build'라는 이름의 두 번째 단계로, 빌드 작업을 수행합니다.

Sh

Shell 스크립트를 실행하여 Gradle 래퍼 파일에 실행 권한을 부여하고, 'clean bootJar' 작업을 통해 빌드를 수행합니다.

stage('deploy')

deploy라는 이름의 세 번째 단계로, 배포 작업을 수행합니다.

sshPublisher

SSH를 통해 파일을 원격 서버로 전송하고 원격 명령을 실행합니다. 여기서는 /project/server 디렉토리에 jar 파일을 전송하고, 'bb-server.sh' 스크립트를 실행합니다.

```

ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ nano bb-server.sh
GNU nano 6.2                                bb-server.sh
[> Cloud-Computing-bb-server pid 확인"
CURRENT_PID=$(ps -ef | grep java | grep Cloud-Computing-bb-server | grep -v nohup | awk >
    echo "$CURRENT_PID"
    if [ -z ${CURRENT_PID} ] ;then
    echo "> 현재 구동중인 애플리케이션이 없으므로 종료하지 않습니다."
    else
    echo "> sudo kill -9 $CURRENT_PID"
    sudo kill -9 $CURRENT_PID
    sleep 10
    fi
echo "> Cloud-Computing-bb-server 배포"
JAR_PATH=$(ls -t /deploy/*.jar | head -1)
sudo nohup java -jar -DServer.port=8080 -Dspring.profiles.active=dev ${JAR_PATH} > /depl>
  
```

[사진 14] 스크립트 사진

실행되고 있는 애플리케이션이 있다면 종료하고 다시 배포한다는 내용

- /deploy/ 디렉토리 내에서 가장 최근에 수정된 JAR 파일을 찾아서 그 경로를 JAR_PATH 변수에 저장합니다.
- nohup을 사용하여 터미널 세션이 종료되어도 Java 애플리케이션을 백그라운드에서 계속 실행할 수 있도록 하고, java -jar로 JAR 파일을 실행합니다.
- 실행할 때, 서버 포트(8080)와 스프링 프로파일(dev)을 설정합니다.
- 표준 출력과 표준 에러를 /dev/null로 리다이렉션하여 콘솔에 로그가 출력되지 않게 합니다.

7. 빌드하기

✔ < Build #8

Success 3 hr 29 min ago in 37 sec

✔ github Clone

✔ build

✔ deploy

Stage 'deploy'

🕒 Started 3 hr 28 min ago

⌚ Queued 0 ms

⌚ Took 2.6 sec

ⓘ Success

🔗 [View as plain text](#)

✔ Send build artifacts over SSH

```
0 SSH: Connecting from host [2ce3a59:
1 SSH: Connecting with configuration
2 SSH: Creating session: username [ul
3 SSH: Connecting session ...
4 SSH: Connected
```

[사진 15] 빌드 과정

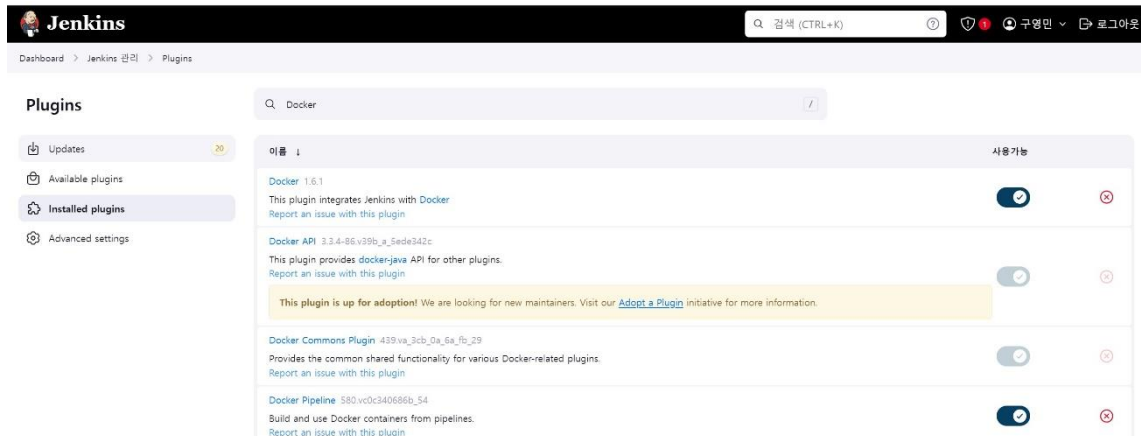
Jenkins 홈 -> Jenkins-ss-server 클릭 -> 지금 빌드 클릭 -> 빌드 성공

```
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ ls
Cloud_Computing      bb-server.sh  home          install-docker.sh  test.sh
bb-server-0.0.1-SNAPSHOT.jar  dockerfiles  index.html    project            volumetest
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ cd project/
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project$ ls
server
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project$ cd server/
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project/server$ ls
bb-server-0.0.1-SNAPSHOT.jar
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project/server$
```


[사진 16] 실행 결과

/home/ubuntu 디렉토리에 project라는 폴더가 생기고 위의 설정한 /project/server 경로에 .jar파일이 생성된 것을 볼 수 있습니다.

8. 도커 젠킨스 연결



[사진 17] 도커 플러그인

이후 젠킨스에서 도커와 관련된 플러그인을 다운로드합니다.

설치할 플러그인

1. Docker Plugin

Jenkins를 Docker와 통합하여 Docker 컨테이너를 관리하고 실행할 수 있도록 해줍니다.

2. Docker API Plugin

다른 플러그인에서 Docker와 상호작용할 수 있도록 Docker Java API를 제공합니다.

3. Docker Commons Plugin

여러 Docker 관련 플러그인에서 공통적으로 사용하는 기능을 제공합니다.

4. Docker Pipeline

Jenkins 파이프라인 스크립트에서 Docker 컨테이너를 빌드하고 사용할 수 있게 해줍니다.



99koo

user Joined January 15, 2024

General

Security

Default Privacy

Notifications

Convert Account

Deactivate Account

Access Tokens

New Access Token

Tokens marked **AUTO-GENERATED** are created on your behalf by Docker Desktop for the CLI to use for authentication. You can have a maximum of 5 auto-generated tokens associated with your account. [Learn more](#)

<input type="checkbox"/>	Description	Source	Scope	Last Used	Created
<input type="checkbox"/>	Jenkins	MANUAL	Read, Write, Delete	May 30, 2024 15:18:00	May 21, 2024 23:27:34

[사진 18] 도커 허브 토큰 발급

도커허브에서 토큰 발급합니다.

Freestyle project

이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(원상관리)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-st job type.

Multi-configuration project

다양한 환경에서의 테스트, 플러그인 특성 빌드, 기타 동등 처리 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.

[사진 19] Multibranch Pipeline 생성

파이브 라인 생성

Branch Sources

Git ✕

Project Repository ?

Credentials ?

+ Add ▼

Behaviours

Discover branches ? ✕

Add ▼

Property strategy

Add property ▼

[사진 20] 깃 연동

Github credential을 누르고, github주소를 입력합니다.

```

pipeline {
  agent any
  options {
    buildDiscarder(logRotator(numToKeepStr: '3'))
  }
  environment {
    DOCKERHUB_CREDENTIALS = credentials('docker_credentials')
    repository = "99koo/bb-server"
    dockerImage = ''
  }
  stages {
    stage('Set Executable Permissions') {
      steps {
        sh 'chmod +x gradlew'
      }
    }
    stage('Gradle Build') {
      steps {
        sh './gradlew clean build'
      }
    }
    stage('Docker Build') {
      steps {
        script {
          dockerImage = docker.build repository + ":%$BUILD_NUMBER"
        }
      }
    }
    stage('Login') {
      steps {
        sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin'
      }
    }
    stage('Push') {
      steps {
        sh 'docker push $repository:$BUILD_NUMBER'
      }
    }
  }
  post {

```

[사진 21] 젠킨스 파일

1. 파이프라인 선언부

pipeline { ... }: Jenkins 파이프라인을 정의하는 블록입니다. 전체 파이프라인 구성은 이 블록 안에 포함됩니다.

2. 에이전트 정의

agent any: 파이프라인이 어떤 Jenkins 에이전트에서든 실행될 수 있도록 지정합니다. 특정 에이전트에서 실행하려면 라벨을 사용할 수 있습니다.

3. 빌드 디스크 정책

options { ... }: 파이프라인 실행 옵션을 정의합니다.

buildDiscarder(logRotator(numToKeepStr: '3')): 빌드 기록을 유지하는 정책을 정의합니다. 최근 3개의 빌드 기록만 보관하고 나머지는 삭제합니다.

4. 환경 변수 설정

environment { ... }: 파이프라인에서 사용할 환경 변수를 정의합니다.

- DOCKERHUB_CREDENTIALS: Jenkins의 자격 증명 관리에서 'docker_credentials'라는 ID로 설정된 Docker Hub 자격 증명을 가져옵니다.
- repository: Docker 이미지를 저장할 리포지토리 이름을 정의합니다.

dockerImage: 빌드된 Docker 이미지의 이름을 저장할 변수입니다.

5. 단계별 작업 정의

5.1. Set Executable Permissions 단계

stage('Set Executable Permissions') { ... }: 'Set Executable Permissions' 단계 정의.

sh 'chmod +x gradlew': gradlew 파일에 실행 권한을 부여합니다.

5.2. Gradle Build 단계

stage('Gradle Build') { ... }: 'Gradle Build' 단계 정의.

sh './gradlew clean build': Gradle을 사용하여 프로젝트를 클린 빌드합니다.

5.3. Docker Build 단계

stage('Docker Build') { ... }: 'Docker Build' 단계 정의.

script { ... }: 여러 줄의 Groovy 스크립트를 실행할 수 있도록 하는 블록입니다.

dockerImage = docker.build repository + ":\$BUILD_NUMBER";
Docker 이미지를 빌드하고, 태그를 repository:빌드 번호 형식으로 지정합니다. 빌드된 Docker 이미지 객체는 dockerImage 변수에 저장됩니다.

5.4. Login 단계

stage('Login') { ... }: 'Login' 단계 정의.

sh 'echo \$DOCKERHUB_CREDENTIALS_PSW | docker login -u
\$DOCKERHUB_CREDENTIALS_USR --password-stdin': Docker Hub에 로그인
합니다. 환경 변수 DOCKERHUB_CREDENTIALS_USR와
DOCKERHUB_CREDENTIALS_PSW를 사용하여 자격 증명을 전달합니다.

5.5. Push 단계

stage('Push') { ... }: 'Push' 단계 정의.

sh 'docker push \$repository:\$BUILD_NUMBER': 빌드된 Docker 이미지를 Docker Hub 리포지토리에 푸시합니다.

9. 젠킨스 적용 확인

```

ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ cd project/
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project$ ls
docker  kubernetes
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project$ cd docker/
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project/docker$ ls
docker-compose.yaml  docker-jenkins.yaml  jenkins  yaml
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project/docker$ cd jenkins/
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project/docker/jenkins$ ls
docker-compose.yaml
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~/project/docker/jenkins$ cat docker-compose.yaml

version: '3.3'

services:
  bb-server:
    image: 99koo/bb-server:0.0.2
    environment:
      SPRING_DATASOURCE_URL: jdbc:mariadb://mariadb:3306/board?useUnicode=true&characterEncoding=UTF
-8
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
    ports:
      - "8090:8080"
    depends_on:
      - mariadb

  mariadb:
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: board
    ports:
      - "3306:3306"
  
```

[사진 22] 도커 컴포즈 파일

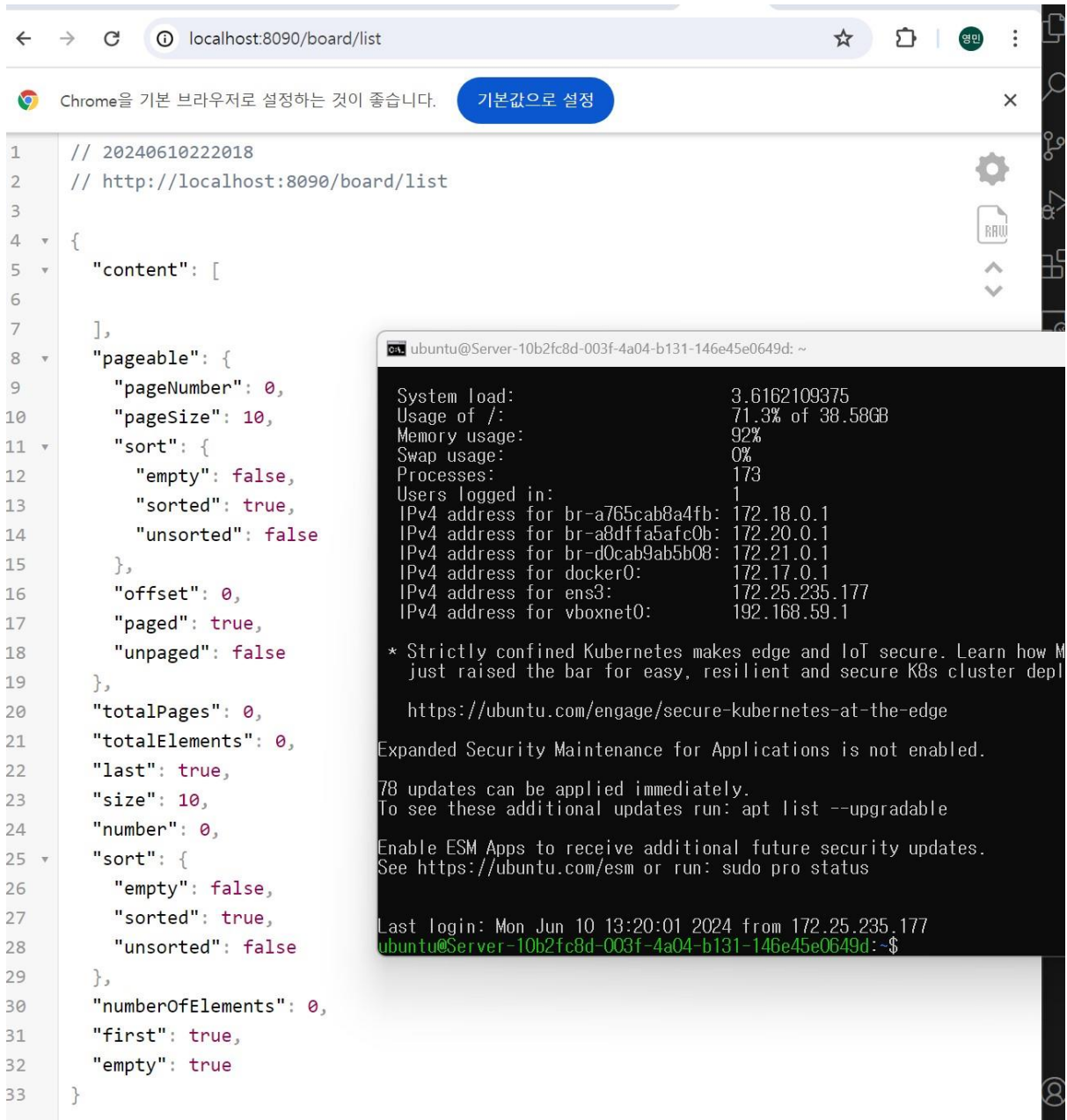
프로젝트에 사용할 bb-server를 docker-compose 파일로 올리고 실행하였습니다.

```

ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
PORTS         NAMES
dc424759a70d   jenkins/jenkins:lts                "/usr/bin/tini -- /u..." 39 minutes ago Up 39 minutes
50000/tcp, 0.0.0.0:9090->8080/tcp, :::9090->8080/tcp   jenkins
d0452263b86e   99koo/bb-server:0.0.2              "java -jar bb-server..." About an hour ago Up About an hour
0.0.0.0:8090->8080/tcp, :::8090->8080/tcp               jenkins_bb-server_1
563f5d931236   mariadb:latest                     "docker-entrypoint.s..." About an hour ago Up About an hour
0.0.0.0:3306->3306/tcp, :::3306->3306/tcp             jenkins mariadb 1
  
```

[사진 23] 도커 컴포즈 실행 확인

젠킨스와 mariadb, bb-server가 실행되고 있는 것을 보실 수가 있습니다.



[사진 24] 정상 접속 확인

1. `ssh -L 8090:localhost:8090 ubuntu@172.25.235.177` 포트포워딩을 시도합니다.
2. bb-server의 `board/list`에 접근합니다.

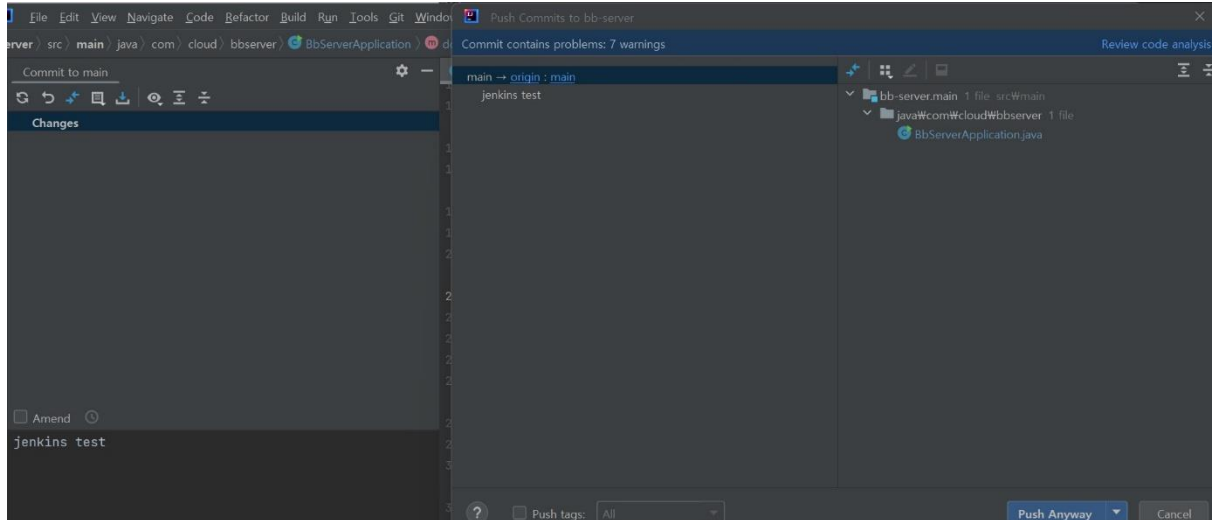
```

@GetMapping(value = "/")
public String doGetHelloWorld() { return "Hello World"; }

```

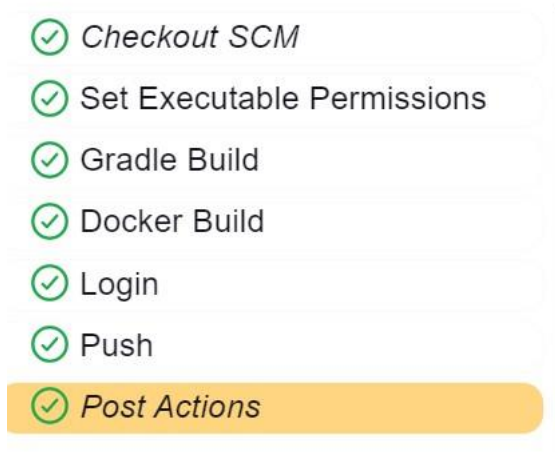
[사진 25] 수정된 bb-server

bb-server를 수정해보겠습니다. HelloWorld를 반환하는 컨트롤러를 추가하였습니다.



[사진 26] 깃허브 푸시

깃허브에 푸시합니다.



[사진 27] 파이프 라인 실행 과정

Jenkins 파이프라인 단계 설명

1. Checkout SCM

소스 코드를 버전 관리 시스템(SCM)으로부터 체크아웃하는 단계입니다. 일반적으로 Git을 사용하여 저장소에서 최신 코드를 가져옵니다.

2. Set Executable Permissions

gradlew 스크립트 파일에 실행 권한을 설정합니다. 이는 Gradle 빌드를 수행하기 위해 필요

합니다.

3. Gradle Build

gradlew clean build 명령어를 실행하여 프로젝트를 클린 빌드합니다.
소스 코드를 컴파일하고, 테스트를 실행하며, 결과물을 생성합니다.

4. Docker Build

Dockerfile을 사용하여 애플리케이션의 Docker 이미지를 빌드합니다.
빌드된 이미지에 Jenkins 빌드 번호를 태그로 추가하여 고유하게 식별할 수 있게 합니다.

5. Login

Docker Hub 자격 증명을 사용하여 Docker Hub에 로그인합니다.
DOCKERHUB_CREDENTIALS_USR 및 DOCKERHUB_CREDENTIALS_PSW 환경 변수를 사용하여
안전하게 로그인합니다.

6. Push

태그된 Docker 이미지를 Docker Hub 리포지토리에 푸시하여 저장합니다.
repository:\$BUILD_NUMBER 형식으로 이미지를 푸시합니다.

```
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ doc
REPOSITORY          TAG          IMAGE ID
mariadb              latest       2b177ae28b69
ubuntu              latest       17c0145030df
99koo/bb-server      0.0.2       497ba975322a
jenkins/jenkins      lts         0bd9f204ffc3
wordpress            latest       cc84570a8e5d
quay.io/metallb/controller v0.13.9     26952499c302
quay.io/metallb/speaker v0.13.9     697605b35935
mariadb              10.6.4-focal 12e05d5da3c5
```

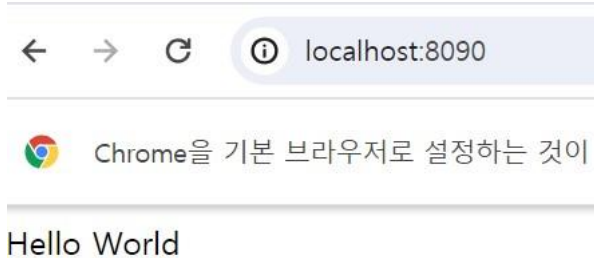
[사진 28] push 전

99koo/bb-server가 하나이지만

```
ubuntu@Server-10b2fc8d-003f-4a04-b131-146e45e0649d:~$ do
REPOSITORY          TAG          IMAGE ID
mariadb              latest       2b177ae28b69
ubuntu              latest       17c0145030df
99koo/bb-server      0.0.2       497ba975322a
99koo/bb-server      18          497ba975322a
jenkins/jenkins      lts         0bd9f204ffc3
wordpress            latest       cc84570a8e5d
quay.io/metallb/controller v0.13.9     26952499c302
quay.io/metallb/speaker v0.13.9     697605b35935
mariadb              10.6.4-focal 12e05d5da3c5
```

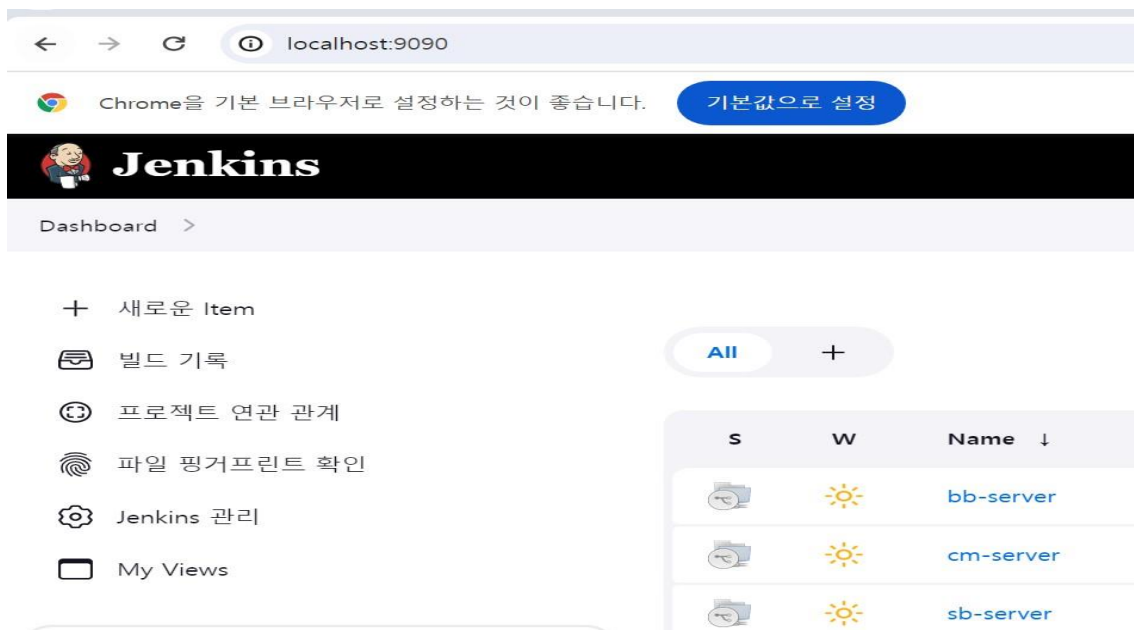
[사진 29] push 후

1. 젠킨스 완료 후에 TAG가 18로 하나가 더 생성되었습니다.
2. 이 후 0.0.2에 Tag 18을 덮어쓰기 한 후 docker compose up -d 명령어를 수행합니다.



[사진 30] 변경 된 내용 적용

변경된 내용이 적용되는 것을 확인할 수 있습니다.



[사진 31] 여러 프로젝트 연동

총 세개의 서버를 연결하여 클라우드컴퓨팅 프로젝트에 적용중입니다.