



배포 서버 환경 구성(CI/CD)

1. EC2에서 인스턴스 생성하기
 2. SSH로 서버 접근하기
 3. 서버에 Docker 및 docker-compose 설치
 4. Docker 실행
 5. Jenkins 설치 및 실행
 6. Jenkins 초기 설정
 7. Jenkins 플러그인 추가 설치
 8. Jenkins Credential 등록
 9. Jenkins 세부 설정
 10. Gitlab WebHook 설정
 11. Server에 RServer 생성하기
 12. Jenkins, R server, Database docker-compose로 한번에 생성하기
- 빌드시 유의할 점
네트워크 확인 및 생성 추가
컨테이너 구조?

▼ 1. EC2에서 인스턴스 생성하기

프리티어를 기준으로 작성하겠습니다.

인스턴스를 생성하기 전에 region을 서울로 설정해줍니다. (제일 상단 탭)

AWS에 접속해서 로그인을 하고 나서 EC2로 접근하면, 인스턴스 시작이라는 버튼을 확인할 수 있습니다. 해당 버튼을 클릭하면 인스턴스를 생성하는 페이지가 나옵니다.

인스턴스 시작 정보

Amazon EC2를 사용하면 AWS 클라우드에서 실행되는 가상 머신 또는 인스턴스를 생성할 수 있습니다. 아래의 간단한 단계에 따라 빠르게 시작할 수 있습니다.

이름 및 태그 정보

이름

추가 태그 추가

먼저 이름 및 태그에 프로젝트의 이름을 적어줍니다.

▼ 애플리케이션 및 OS 이미지(Amazon Machine Image) 정보

AMI는 인스턴스를 시작하는 데 필요한 소프트웨어 구성(운영 체제, 애플리케이션 서버 및 애플리케이션)이 포함된 템플릿입니다. 아래에서 찾고 있는 항목이 보이지 않으면 AMI를 검색하거나 찾아보십시오.

최근 사용

Quick Start

Amazon Linux
aws

macOS
Mac

Ubuntu
ubuntu

Windows
Windows

더 많은 AMI 찾아보기

AWS, Marketplace 및 커뮤니티의 AMI 포함

Amazon Machine Image(AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type

프리 티어 사용 가능

ami-0e38c97339cddf4bd (64비트(x86)) / ami-0635b0441fab49c05 (64비트(Arm)) ▼

가상화: hvm ENA 활성화됨: true 루트 디바이스 유형: ebs

설명

Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2023-02-08

아키텍처

AMI ID

64비트(x86) ▼

ami-0e38c97339cddf4bd

확인된 공급 업체

애플리케이션 및 OS이미지에서 해당 서버에서 사용하고 싶은 운영체제를 선택합니다. 저의 경우 Ubuntu를 선택하여 진행하였습니다. 다른 부분은 default로 두고 넘어갑니다.

▼ 인스턴스 유형 정보

인스턴스 유형

t2.micro 프리 티어 사용 가능

패밀리: t2 1 vCPU 1 GiB 메모리
온디맨드 RHEL 요금: 0.0744 USD 시간당
온디맨드 Linux 요금: 0.0144 USD 시간당
온디맨드 SUSE 요금: 0.0144 USD 시간당
온디맨드 Windows 요금: 0.019 USD 시간당

[인스턴스 유형 비교](#)

인스턴스 유형은 프리 티어로 사용 가능한 t2.micro를 선택합니다.

▼ 키 페어(로그인) 정보

키 페어를 사용하여 인스턴스에 안전하게 연결할 수 있습니다. 인스턴스를 시작하기 전에 선택한 키 페어에 대한 액세스 권한이 있는지 확인하세요.

키 페어 이름 - 필수

cicd-project ▼

새 키 페어 생성

다음으로 키 페어를 설정하는 부분입니다. 키 페어의 경우 SSH를 통해서 해당 서버에 접속할 때 필요합니다. 없다면 오른쪽의 새 키 페어 생성이라는 버튼을 클릭해서 키 페어를 생성해주고, 미리 생성해둔게 있다면 해당 키 페어를 선택해줍니다.

네트워크 부분은 default로 설정된 상태로 두고 넘어갑니다.

▼ 스토리지 구성 정보

어드밴스드

1x 30 GiB gp2 루트 볼륨

(암호화되지 않음)

❗

프리 티어를 사용할 수 있는 고객은 최대 30GB의 EBS 범용(SSD)또는 마그네틱 스토리지를 사용할 수 있습니다.

×

새 볼륨 추가

선택한 AMI에 인스턴스가 허용하는 것보다 많은 인스턴스 스토어 볼륨이 포함되어 있습니다. AMI에서 처음 0개의 인스턴스 스토어 볼륨에만 액세스할 수 있습니다.

0x 파일 시스템

편집

다음 스토리지는 프리 티어에서 사용 가능한 최대 용량인 30으로 설정해줍니다.

배포 서버 환경 구성(CI/CD)

3

▼ 요약

인스턴스 개수 [정보](#)

1

소프트웨어 이미지(AMI)

Canonical, Ubuntu, 22.04 LTS, ...[더 보기](#)

ami-0e38c97339cddf4bd

가상 서버 유형(인스턴스 유형)

t2.micro

방화벽(보안 그룹)

새 보안 그룹

스토리지(볼륨)

1개의 볼륨 – 30GiB

❗

프리 티어: 첫 해에는 월별 프리 티어 AMI에 대한 t2.micro(또는 t2.micro를 사용할 수 없는 리전의 t3.micro) 인스턴스 사용량 750시간, EBS 스토리지 30GiB, IO 2백만 개, 스냅샷 1GB, 인터넷 대역폭 100GB가 포함됩니다.

×

취소

인스턴스 시작

이렇게 설정하고 제일 아래에 있는 **인스턴스 시작** 버튼을 눌러서 인스턴스를 생성해줍니다.

이렇게 생성하면 인스턴스 탭에서 생성된 인스턴스를 확인할 수 있습니다.

```
docker run -d --name database -v mariadb-data --env TZ=Asia/Seoul --env MYSQL_ROOT_PASSWORD=root --env MYSQL_DATABASE=test -p 3306
```

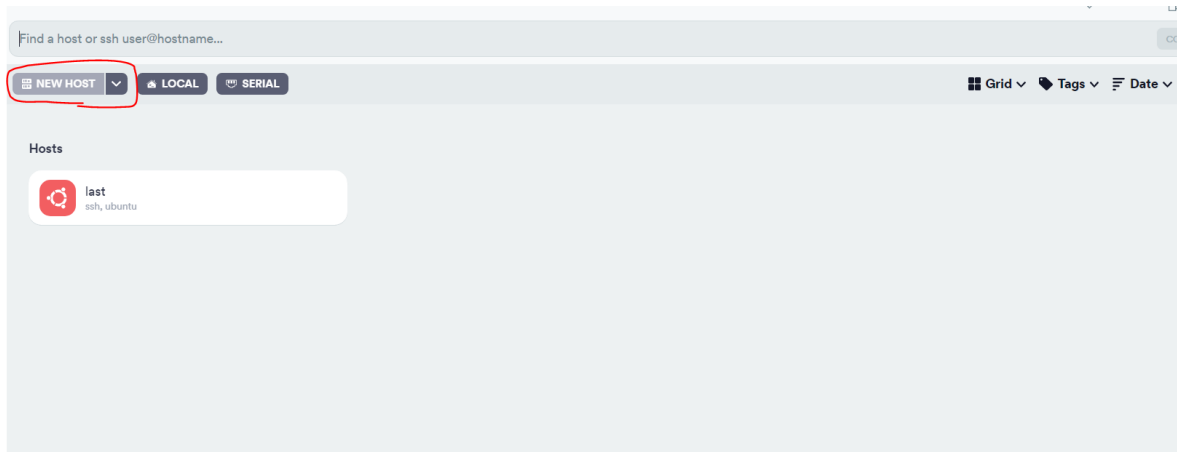
▼ 2. SSH로 서버 접근하기

서버에 접근하기 쉽게 도와주는 tool인 terminus를 설치해줍니다.

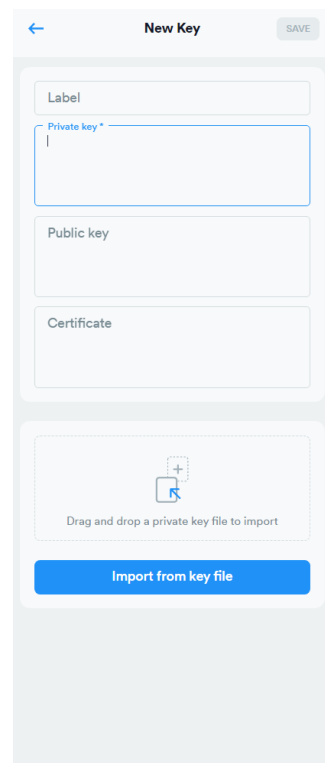
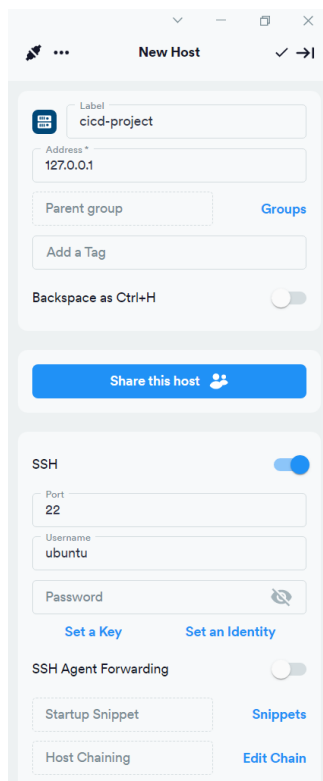
Termius - SSH platform for Mobile and Desktop

Termius helps to organize the work of multiple DevOps and engineering teams. It reduces the admin work for managing users. Enterprise compliance. SOC2 II report.

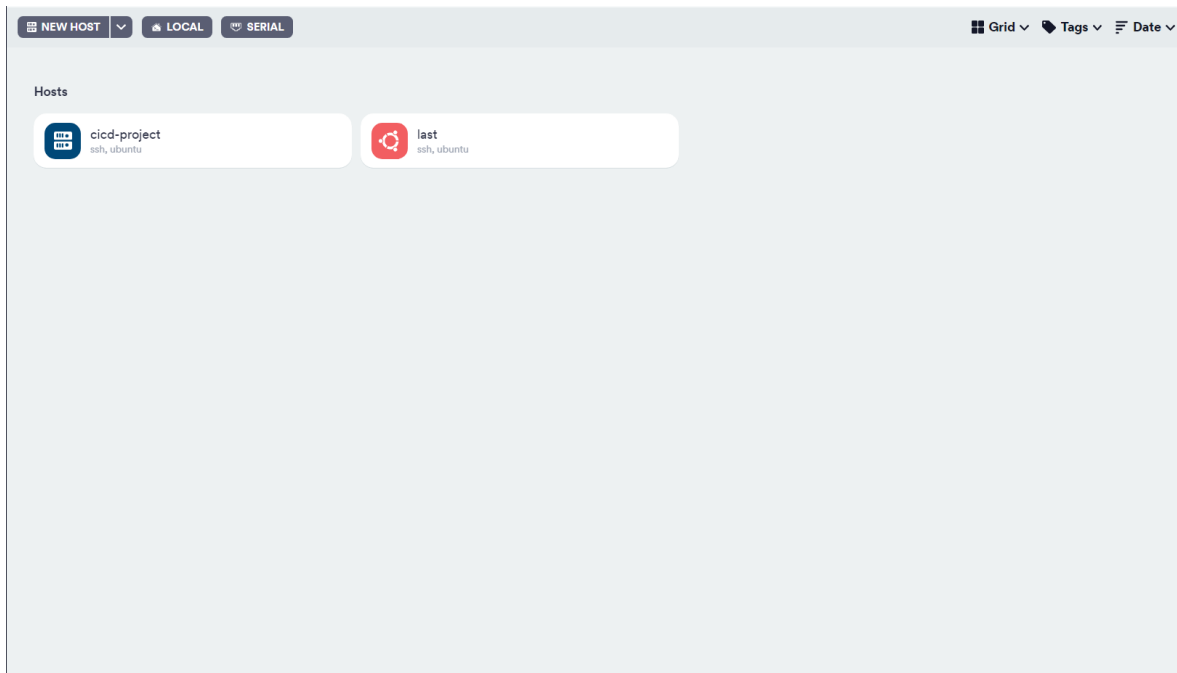
<https://termius.com/>



terminus를 설치하면 위와 같은 화면이 생성되는데, 상단에 있는 new host를 클릭합니다.



그러면 위와 같은 탭이 우측에 생성됩니다. Label에 이름을 적어주고, Address에 서버의 ip주소를 적어줍니다. Port의 경우 기본적으로 SSH는 22번 포트를 사용합니다. 그리고 AWS에서 ubuntu 인스턴스를 생성하면 Username은 ubuntu로 설정되어 있습니다. password는 아까 생성한 pem키를 사용할 것인데, password 아래에 있는 **Set a Key**를 클릭하고 **New Key**를 선택하면 오른쪽에 있는 화면이 나옵니다. 여기서 Label에 해당 키의 별명을 지어주고, pem파일을 제일 하단의 영역에 드래그한 후 저장해줍니다.



이렇게 새로운 서버가 생성된 것을 확인할 수 있고, 더블 클릭을 통해서 원격 서버에 접근할 수 있습니다.

▼ 3. 서버에 Docker 및 docker-compose 설치

1. 패키지 업데이트 진행

```
sudo apt update & apt upgrade
```

2. Docker 설치에 필요한 필수 패키지 설치

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

3. Docker의 GPG key 인증

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

4. Docker Repository 등록

```
sudo add-apt-repository \  
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \  
$(lsb_release -cs) \  
stable"
```

5. Docker 설치

```
sudo apt-get update && sudo apt-get install docker-ce docker-ce-cli containerd.io
```

6. docker-compose 설치

```
sudo curl -L \
"https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" \
-o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose # chmod 를 통해서 실행이 가능하게 세팅

docker-compose -v # docker-compose 명령이 제대로 먹히는 지 확인한다.
```

▼ 4. Docker 실행

```
sudo systemctl enable docker
```

이 명령은 서버가 실행될 때마다 docker가 자동으로 실행되도록 하는 명령입니다.

```
sudo service docker start
```

이 명령은 docker를 실행하는 명령입니다.

▼ 5. Jenkins 설치 및 실행

1. Dockerfile을 만들어서 jenkins를 설치합니다.

```
FROM jenkins/jenkins:lts

USER root

RUN apt-get update \
  && apt-get -y install lsb-release \
  && curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg \
  && echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $" \
  && apt-get update \
  && apt-get -y install docker-ce docker-ce-cli containerd.io

RUN usermod -u 1000 jenkins && \
  groupmod -g 998 docker && \
  usermod -aG docker jenkins
```

단순하게 jenkins이미지만 가지고 jenkins 컨테이너를 생성해도 되지만, 위처럼 진행하는 이유는 jenkins내에서도 docker를 사용하기 위함입니다. (docker.sock을 활용하여 통신)

```
RUN usermod -u 1000 jenkins && groupmod -g 998 docker && usermod -aG docker jenkins
```

에서 1000의 경우 호스트 사용자 아이디를 입력해주는 것으로 `id -u` 명령어를 통해서 확인이 가능합니다.

그리고 998의 경우 도커 그룹 아이디로 `cat /etc/group | grep docker` 를 통해서 확인이 가능합니다.

위에서 해당 부분만 사용자의 정보에 맞게 수정해줍니다.

2. 이미지 생성

```
docker build -t my-jenkins:0.1 .
```

3. volume 생성

```
docker volume create jenkins
```

4. jenkins 컨테이너 생성

```
docker run -d --name jenkins \
-v /var/run/docker.sock:/var/run/docker.sock \
```

```
-v jenkins:/var/jenkins_home \
-p 8080:8080 my-jenkins:0.1
```

▼ 6. Jenkins 초기 설정

1. <http://서버아이피:8080> 으로 접속하기
2. 젠킨스 비밀번호 입력

Getting Started

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

위 사진처럼 초기 관리자 계정의 비밀번호를 입력하라고 하는데, 해당 비밀번호를 입력하는 방법은 2가지가 있다.

첫번째 방법:

```
docker logs jenkins
```

젠킨스 컨테이너의 로그를 보면 관리자 계정의 비밀번호를 확인할 수 있다.

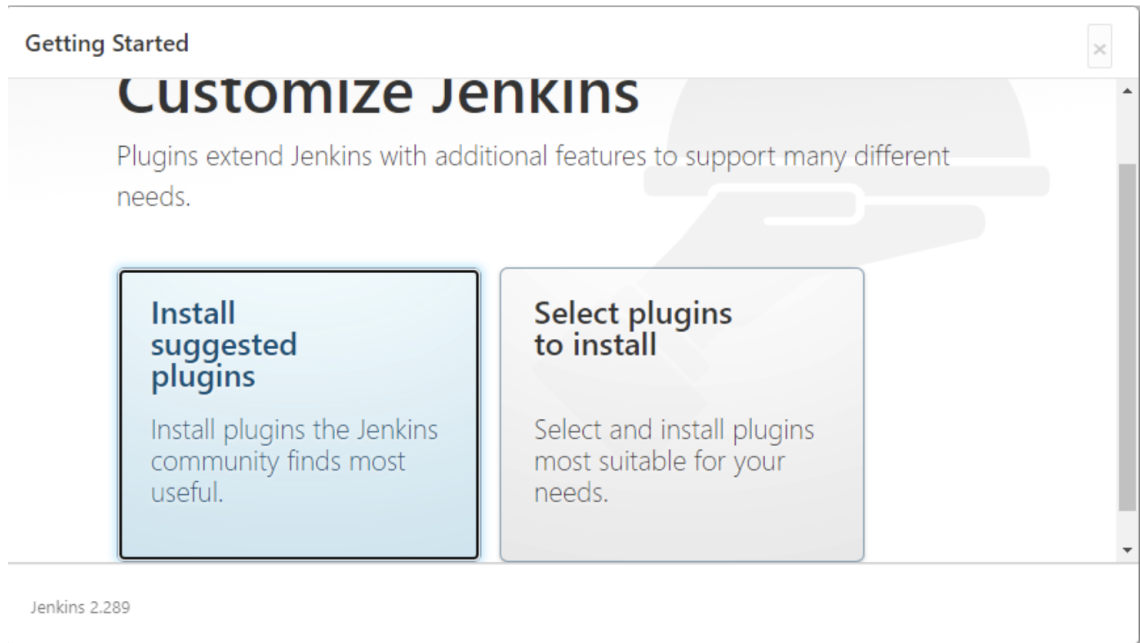
두번째 방법:

```
docker exec -it jenkins /bin/bash
```

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

먼저 jenkins 컨테이너에 접속하고, 아래 경로에서 비밀번호를 확인합니다.

3. 비밀번호를 입력하면, 아래와 같은 화면이 나오는데, Install suggested plugins를 선택합니다.



4. 설치 후 계정 정보를 입력하는 부분이 나오는데, 본인이 설정하고 싶은대로 설정합니다.

▼ 7. Jenkins 플러그인 추가 설치

젠킨스 관리 → plugin Manage로 접근합니다.

우리가 설정해야 할 플러그인은 Gitlab, Gradle이 있습니다.

초기 설정에서 install suggested plugins를 선택했다면, Gradle의 경우 이미 설치되었을 것이기 때문에 Gitlab plugin을 설치 하고 jenkins를 재시작해줍니다.

만약 jenkins페이지에서 재시작이 정상적으로 동작하지 않는다면, 아래 명령어를 통해서 jenkins컨테이너 재시작

```
docker restart {container_id 또는 container name}
```

▼ 8. Jenkins Credential 등록

jenkins관리 → Manage Credential에 들어갑니다.

그리고 System → Global credentials(unrestricted)로 접근합니다.

그리고 상단에 있는 **Add Credentials** 버튼을 클릭합니다.

Credentials를 생성하기 위해서는 gitlab의 accesstoken이 필요하기 때문에 jenkins에 연동할 gitlab 프로젝트에서 accesstoken을 선택합니다.

해당 페이지에서 발급받은 accessToken에서 password로 입력해줍니다.

▼ 9. Jenkins 세부 설정

1. gradle 버전 설정

Jenkins관리 → Global Tool Configuration으로 접근합니다.

해당 페이지에서 Gradle을 아래와 같이 설정해줍니다. (프로젝트 gradle버전과 통일)

Gradle

Gradle installations

List of Gradle installations on this system

Add Gradle

Gradle name ?

Gradle7.5

☒ Install automatically ?

≡ Install from Gradle.org

Version

Gradle 7.5

Add Installer ▾

Add Gradle

2. 프로젝트 생성


a. 새로운 Item 생성 클릭


item 이름을 선택하고 Freestyle project를 선택합니다.

Enter an item name

campinity-develop2

» 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**

b. 소스코드 관리 목록에서 GIT을 선택하고 jenkins에 연결하고 싶은 repository의 경로와 위 과정에서 생성해둔 Credential을 설정해줍니다.

소스 코드 관리

☐ None

☒ Git ?

Repositories ?

Repository URI ?

https://fab.sauy.com/108-ws6mob24-sub2/508P120101.git (깃랩 repository URL)

Please enter Git repository.

Credentials ?

- none - GitLab repository token을 기반으로 만든 Credential목록

+ Add

다음...

c. 연결할 브랜치 설정(commit을 가지고 올 브랜치 선택)

Branches to build ?

Branch Specifier (blank for 'any') ?

*/master

Add Branch

d. 빌드 유발 설정(이번 프로젝트에서는 mr이 발생한 경우에만 jenkins로 빌드를 할 예정이기 때문에, 빌드 유발을 아래와 같이 설정합니다. (빌드 유발 제일 아래에 보면 고급이라는 버튼이 있는데, 이 부분은 나중에 gitlab에서 webhook을 설정할 때 사용합니다.)

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://54.180.133.21:8080/project/last> ?

Enabled GitLab triggers

☐ Push Events

☐ Push Events in case of branch delete

☐ Opened Merge Request Events

☐ Build only if new commits were pushed to Merge Request ?

☒ Accepted Merge Request Events

☐ Closed Merge Request Events

Rebuild open Merge Requests

Never

☐ Approved Merge Requests (EE-only)

☐ Comments

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급...

e. 빌드 스텝(먼저 gradle을 통해서 spring boot 프로젝트를 빌드합니다. 그리고 shell을 통해서 build를 해줍니다.)

Build Steps

≡ Invoke Gradle script ?

● Invoke Gradle ?

Gradle Version
gradle 8.0

☐ Use Gradle Wrapper ?

Tasks ?
build -p /var/jenkins_home/workspace/campinity-develop/Server -x test

고급...

≡ Execute shell ?

Command
See [the list of available environment variables](#)
cd Server
docker-compose up --build -d

저장

Apply

▼ 10. Gitlab WebHook 설정

1. gitlab repository → setting → webhook으로 들어가면 아래와 같은 페이지가 나옵니다.

Webhooks

[Webhooks](#) enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

URL must be percent-encoded if it contains one or more special characters.

Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

Trigger

☒ Push events

Push to the repository.

☐ Tag push events

A new tag is pushed to the repository.

☐ Comments

A comment is added to an issue or merge request.

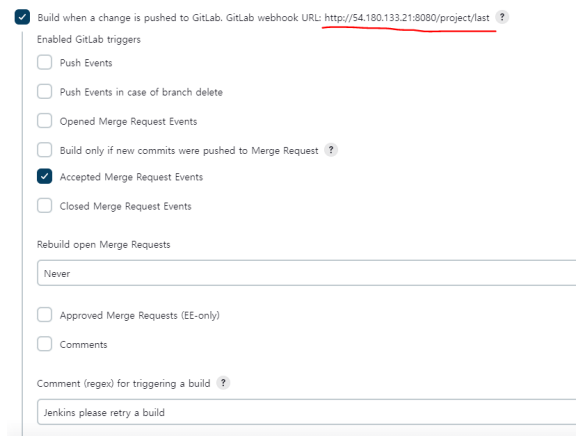
☐ Confidential comments

A comment is added to a confidential issue.

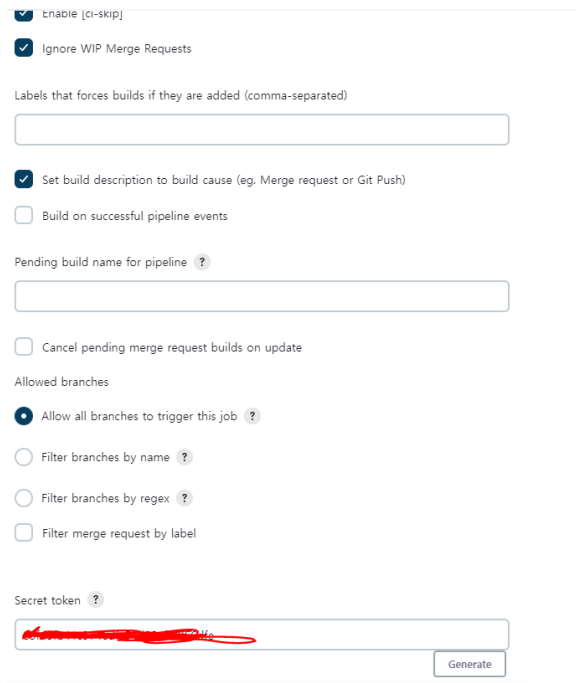
☐ Issues events

An issue is created, updated, closed, or reopened.

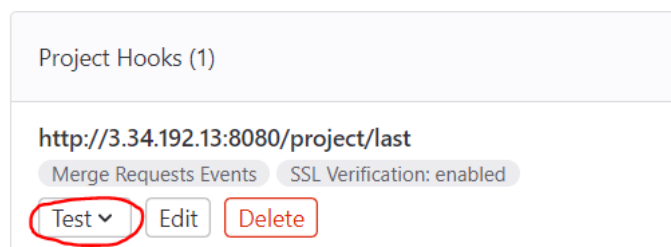
2. webhook의 url은 아래 사진에 보이는 것처럼 jenkins빌드 유발을 보면 url이 적혀 있는데 해당 url을 적어주면 됩니다.



3. Secret key는 빌드 유발의 고급 탭에서 생성할 수 있습니다. (고급을 누르고, 다른 설정은 그대로 둔 후 generate버튼을 클릭하면 됩니다.)



4. Trigger의 경우 이번 프로젝트에서는 mr이 발생했을 때, 다시 빌드하기 때문에 merge request로 설정하면 되고, 각자 상황에 맞게 선택합니다.
5. Test(이렇게 webhook을 설정하고 나면 아래 Test 토글 버튼이 생성되는데, 본인이 설정한 trigger를 선택하면 제대로 연결이 되었는지 Test할 수 있고 응답이 200으로 뜨면 정상적으로 연결된 것입니다.



▼ 11. Server에 RServer 생성하기

1. Rstudio Dockerfile 생성하기

```
# Dockerfile_rstudio -> jenkins의 Dockerfile과 이름이 겹치지 않도록 합니다.

FROM rocker/rstudio:latest

RUN R -e "install.packages('Rserve', repos='http://cran.us.r-project.org')"
CMD ["R", "-e", "Rserve::run.Rserve(remote=TRUE)", "--no-save", "--slave"]
```

`rocker/rstudio` 이미지를 통해서 rstudio를 설치하고, spring boot project에서 통신을 하기 위해서 Rserve를 설치해줍니다.

그리고 Rserve connection을 열어줍니다. Rserve서버와 R 클라이언트가 서로 다른 컴퓨터에서 실행되는 경우에는 `remote=TRUE` 옵션을 주어야 합니다.

2. 이미지 생성 및 컨테이너 생성

```
sudo docker build -f Dockerfile_rstudio -t rstudio .
```

```
sudo docker run -d -it --name rstudio --network last-project-network -e PASSWORD=ssafy1234 -p 8787:8787 -p 6311:6311 rstudio
```

Rocker Project - The Rocker Project
Docker Containers for the R Environment
<https://rocker-project.org/>

3. 6311포트가 열려 있는지 확인

```
sudo docker exec -it rstudio /bin/bash
```

```
apt update
apt upgrade
apt install net-tools
sudo netstat -plnlt
```

4. Working Directory에 R script 생성

기본적으로 WD는 `/home/rstudio` 입니다. 해당 폴더에 우리가 실행시키고자 하는 파일을 생성해둡니다.

5. R client(Spring)에서 접속 및 스크립트 실행

a. 의존성 추가

```
implementation group: 'org.rosuda.REngine', name: 'Rserve', version: '1.8.1'
```

b. connection 연결 및 스크립트 파일 실행

```
package ssafy.demo.service;

import org.rosuda.REngine.REXP;
import org.rosuda.REngine.REXPMismatchException;
import org.rosuda.REngine.Rserve.RConnection;
import org.rosuda.REngine.Rserve.RserveException;
import org.springframework.stereotype.Service;
```

```
@Service
public class Rservice {

    public int[] RTest() {
        RConnection conn = null;
        int[] x = null;
        try {
            conn = new RConnection("rstudio", 6311); // host, port 순서이고, 컨테이너 이름으로 접속하기 위해서는 api 컨테이너와 rstudio
            REXP exp = conn.eval("source('/home/rstudio/r_test_script.R')"); // 서버 파일이 있는 경로(wd)

            x = conn.eval("x").asIntegers();

            System.out.println("-----");
            for (int item: x) {
                System.out.println(item);
            }

            int[] y = conn.eval("y").asIntegers();

            System.out.println("-----");
            for (int item: y) {
                System.out.println(item);
            }
        } catch (RserveException | REXPMismatchException e) {
            throw new RuntimeException(e);
        } finally {
            if (conn != null) {
                conn.close();
            }
        }
        return x;
    }
}
```

6. R 세션이 1시간마다 자동으로 종료되는 문제

`/etc/rstudio/rsession.conf` 파일에 `session-timeout-minutes=0` 으로 설정하면 세션이 자동으로 종료되는 것을 막을 수 있습니다.

RStudio Server: 서버 구성과 관리

RStudio 는 아래의 두개 파일 외에 딱히 설정 할 것이 없다. `/etc/rstudio/rserver.conf`
`/etc/rstudio/rsession.conf` 설정 파일을 수정한 후에는 전체적으로 설정을 확인하기 위해 체크가 필요하다. `$ sudo rstudio-server verify-installation` 네트워크 포트와 주소 기본적으로 RStudio는 웹접속을 위해 8787을
 📄 <https://bhjo0930.tistory.com/entry/RStudio-Server-서버-구성과-관리>



7. R server에서 R명령어를 사용하고 싶으면 R 입력하면 사용 할 수 있고, 종료하고 싶을 땐 q() 을 입력합니다.

▼ 12. Jenkins, R server, Database docker-compose로 한번에 생성하기

위 방법에서는 dockerfile로 일일이 이미지를 생성하고 컨테이너를 생성했습니다. 하지만, 유지 및 관리 측면에서 명령어를 계속 일일이 입력하는 것은 불편하기 때문에 이를 docker-compose를 통해서 생성할 수 있도록 하려고 합니다.

1. 외부 network 생성

```
docker network create --gateway 172.19.0.1 --subnet 172.19.0.0/24 last-project-network
```

2. volume생성(jenkins, mariadb, rstudio)

```
docker volume create jenkins
docker volume create mariadb
docker volume create rstudio
```

3. Jenkins Dockerfile생성

```
FROM jenkins/jenkins:lts

USER root

RUN apt-get update \
  && apt-get -y install lsb-release \
  && curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
  && echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $"
```



```
&& apt-get update \
&& apt-get -y install docker-ce docker-ce-cli containerd.io
RUN usermod -u 1000 jenkins && \
groupmod -g 999 docker && \
usermod -aG docker jenkins
```

4. R server Dockerfile 생성

```
FROM rocker/rstudio:latest

RUN R -e "install.packages('Rserve', repos='http://cran.us.r-project.org')"
```

CMD ["R", "-e", "Rserve::run.Rserve(remote=TRUE)", "--no-save", "--slave"]

5. docker-compose.yml 생성

```
version: '3.2'
services:
  jenkins:
    build: ./
    volumes:
      - jenkins:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - "8080:8080"
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      default
      last-project-network:
        ipv4_address: 172.19.0.2
  database:
    image: mariadb
    container_name: database
    volumes:
      - mariadb:/var/lib/mysql
    restart: always
    environment:
      TZ: Asia/Seoul
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: test
    command: ['--character-set-server=utf8mb4',
      '--collation-server=utf8mb4_unicode_ci']
    ports:
      - "3306:3306"
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      default
      last-project-network:
        ipv4_address: 172.19.0.3
  rstudio:
    build: ./Dockerfile_rstudio
    volumes:
      - rstudio:/home/rstudio
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      default
      last-project-network:
        ipv4_address: 172.19.0.4
    ports:
      - "8787:8787"
      - "6311:6311"
  networks:
    last-project-network:
      external: true
```

6. 컨테이너 생성

```
docker-compose up -d
```

이렇게 진행하는 경우 위에서 1 ~ 4번까지 진행하고, 해당 과정(12)을 진행한 후 5번을 뛰어넘고 11번까지 진행하면 됩니다.

▼ 빌드시 유의할 점

1. AWS 프리티어를 사용하는 경우

AWS 프리티어로 EC2인스턴스를 생성하면, ram이 1기가 밖에 되지 않습니다. gradle 빌드를 하기 위해서는 적어도 2기가가 필요한데 제공되는 저장공간이 부족하기 때문에, 아무것도 없는 텅 빈 프로젝트를 빌드하더라도, gradle demon이 실행된 후 무한로딩이 걸리면서 취소도 제대로 되지 않는 먹통 상태가 되어버립니다.

이 문제를 해결하기 위해서는 용량을 더 늘리면 되겠지만, 그렇게 되면 금전적으로 부담이 될 수 있기 때문에 임시방편으로 해결하자면, swap memory를 사용하면 됩니다.

자세한 방법은 아래 블로그를 참고하면 됩니다.

AWS EC2 프리티어에서 메모리 부족현상 해결방법

AWS free tier를 사용하다보면 2%가 부족할 때가 있다. AWS 프리티어는 가난한 대학생에게는 한줄기 빛과 같은 존재인데, AWS의 프리티어라서 적게 돈이 나가는 것도 좋고, 실제로 이것저것 해볼 수 있다는 측면에서 한줄기의 빛과 같은 존재이다. 하지만, 이러한 프리티어도 한가지의 문제를 가지고 있다. t2.micro의 램이 1GB

☞ <https://sundries-in-myidea.tistory.com/102>



2. Jenkins에서 Docker compose를 사용하는 경우

Jenkins에서 Docker compose를 사용하고자 한다면, docker-compose를 설치해주어야 합니다. 설치 방법은 위에 소개된 방법을 참고하면 됩니다.

▼ 네트워크 확인 및 생성 추가

1. docker network 목록 확인하기

```
docker network ls
```

2. 컨테이너 network 확인하기(제일 아래 쪽에서 확인 가능)

```
docker container inspect [컨테이너명]
```

3. network 생성하기

```
docker network create [네트워크명]
```

4. network gateway설정 및 subnet설정하기

```
docker network create --gateway [ip주소] --subnet [ip주소] [네트워크명]
```

5. run 명령어에서 network 설정하기(-network 옵션 사용)

```
docker run --network [네트워크명] ~~~~~
```

6. 이미 실행중인 컨테이너에 네트워크 추가하기

```
docker network connect [네트워크명] [컨테이너명]
```

- 컨테이너들은 같은 네트워크에 소속되어 있어야 컨테이너명을 통해서 서로 통신할 수 있습니다. ip 주소의 경우 유동적으로 변하기 때문에 ip주소를 사용하기에는 한계가 있습니다.

▼ 컨테이너 구조?

api server의 경우 지속적으로 merge request가 발생할때마다, 계속 재빌드 & 재배포 되어야 하지만, jenkins, database, r server의 경우 재빌드 되거나 재배포 되어야 하는 일이 없다.

그래서 jenkins, database, r server의 경우에는 초기 서버에 컨테이너를 미리 생성해두고 사용하고, api server 혹은 batch server의 경우에는 docker-compose 사용해서 매번 재배포 합니다.

docker-compose를 해서 빌드를 하게 되면 docker run명령어를 통해서 생성한 컨테이너들과 다른 네트워크로 설정이 되는데, 이를 맞춰주기 위해서 docker-compose에서 외부 네트워크를 사용할 수 있도록 설정해주어야 합니다.

```
version: '3.2'
services:
  web-server:
    build: ./
    ports:
      - "8000:8000"
    environment:
      SPRING_DATABASE_URL: jdbc:mariadb://database:3306/test
      SPRING_DATABASE_USERNAME: root
      SPRING_DATABASE_PASSWORD: root
      TZ: Asia/Seoul
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      - default
      - last-project-network
networks:
  last-project-network:
    external: true
```

지금 현재는 미리 생성해두는 jenkins, db, r server의 경우 dockerfile을 통해서 이미지 빌드하고 컨테이너를 생성하는 방식으로 하고 있는데, 그렇게 되면 서버가 다운이 되었을 때나 도커를 재가동할 때, 일일이 명령어를 실행해야 하는 문제점이 있다. 따라서 해당 컨테이너들을 묶어서 생성할 수 있는 docker-compose.yml을 생성하는 것이 더 좋을 것 같음.