Docker Compose를 사용하여 Omnibus GitLab 설치

2020. 10.

InfoGrab

목차

1. 개요

1.1. 문서 개요

1.1.1. 목적

본 문서는 Ubuntu 18.04 LTS에 Docker Compose를 사용하여 Omnibus GitLab을 설치하는 방법을 설명하는데 그 목적이 있습니다.

1.1.2. 범위

본 문서의 범위는 Ubuntu 18.04 LTS에 Docker Engine과 Docker Compose를 설치하고, 이를 이용하여 설치하기 위한 환경 구성과 Omnibus GitLab 설치 및 시작하는 방법에 대한 내용으로 한정되어 있습니다. Docker는 다양한 Linux 배포판(CentOS, Debian, Fedora, Ubuntu 등)에 설치 가능하고, GitLab도 소스로부터 설치, 패키지를 사용하여 설치, Helm chart를 통한 설치 방법이 있으나, 본 문서에서는 제외합니다.

2. Docker Engine 설치

신규 Host Machine에 처음으로 Docker CE을 설치하기 전에, Docker Repository 설정이 필요합니다. 이후에 Repository로부터 Docker Engine를 설치 및 업데이트할 수 있습니다.

2.1. Repository 구성

apt 패키지 인덱스를 업데이트하고 apt가 HTTPS을 통해 repository를 사용할 수 있도록 필요한 패키지들을 설치합니다.

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

Docker의 공식 GPG 키를 추가합니다.

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

Fingerprint(지문)의 마지막 8자를 검색하여, Fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88이 있는 키가 있는지 확인합니다.

```
$ sudo apt-key fingerprint 0EBFCD88
pub      rsa4096 2017-02-22 [SCEA]
         9DC8 5822 9FC7 DD38 854A    E2D8 8D81 803C 0EBF CD88
uid                  [ unknown] Docker Release (CE deb) <docker@docker.com>
sub      rsa4096 2017-02-22 [S]
```

Stable Repository(안정 버전 저장소)를 설정하기 위해 아래 명령어를 사용합니다.

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

공식 Docker 저장소로부터 설치하는 것인지 확인합니다.

```
$ sudo apt-cache policy docker-ce
docker-ce:
  Installed: (none)
  Candidate: 5:19.03.12~3-0~ubuntu-bionic
  Version table:
      5:19.03.12~3-0~ubuntu-bionic 500
          500 https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages
      5:19.03.11~3-0~ubuntu-bionic 500
          500 https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages
                                                  …
```

docker-ce가 설치되지 않았으며 설치 후보는 공식 저장소에서 가져 왔다는 것을 알 수 있습니다.

2.2. Docker Engine 설치

apt 패키지 인덱스를 업데이트하고 Docker Engine과 containerd 최신 버전을 설치합니다.

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

hello-world 이미지를 구동하여 Docker CE가 정상적으로 설치되었는지 확인합니다.

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:7f0a9f93b4aa3022c3a4c147a449bf11e0941a1fd0bf4a8e6c9408b2600777c5
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

2.3. Sudo 없이 docker 명령어 사용하기

Docker daemon은 기본적으로 /var/run/docker.sock에서 생성된 unix domain socket(IPC socket)을 사용하여 통신하는데, root 권한이 있거나 사용자가 docker 그룹의 멤버이어야 합니다. sudo 없이 docker 명령어를 사용하려면, 'docker' 그룹에 사용자를 추가하면 됩니다.

```
$ sudo usermod -aG docker ubuntu
```

Ubuntu에서 로그아웃 후 SSH 재접속합니다.

sudo 없이 docker 명령이 실행되는 것을 확인합니다.

```
$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
1faf80b32fa3   hello-world   "/hello"   5 minutes ago    Exited (0) 5 minutes ago              compassionate_wing
```

3. Docker Compose 설치

Docker Compose는 다중 컨테이너 Docker 애플리케이션을 정의하고 실행하기 위한 도구입니다. Compose에서는 YAML 파일을 사용하여 애플리케이션의 서비스를 구성합니다. 그런 다음, 단일 명령으로 구성(Configuration)에서 모든 서비스를 만들고 시작합니다.

Compose는 프로덕션, 스테이징, 개발, 테스트 및 CI 워크플로우와 같은 모든 환경에서 작동합니다.

Compose 사용은 기본적으로 3 단계 프로세스입니다.

(1) 어디에서나 재현할 수 있도록 Dockerfile를 사용하여 앱의 환경을 정의

(2) 격리된 환경에서 함께 실행할 수 있도록 docker-compose.yml에 앱을 구성하는 서비스 정의

(3) docker-compose up 명령을 실행하면, Compose가 전체 앱을 시작하고 실행

3.1. Compose 설치

GitHub의 Compose repository 릴리즈 페이지에서 Docker Compose 바이너리를 다운로드 할 수 있습니다.

아래 명령을 실행하여 Docker Compose의 현재 안정 버전 릴리스(stable release)를 다운로드 합니다.

```
$ sudo curl \
      -L "https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m)" \
      -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   651  100   651    0     0   2086      0 --:--:-- --:--:-- --:--:--  2079
100 11.6M  100 11.6M    0     0  2829k      0  0:00:04  0:00:04 --:--:-- 3842k
```

바이너리에 실행 권한을 적용합니다.

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

정상 설치되었는지 확인합니다.

```
$ docker-compose --version
docker-compose version 1.26.2, build eefe0d31
```

4. GitLab 설치

4.1. GitLab Docker 이미지

GitLab Docker 이미지는 단일 컨테이너에서 필요한 모든 서비스를 실행하는 GitLab의 모놀리식(Monolithic) 이미지입니다.

GitLab CE와 EE 이미지는 모두 Docker Hub에 있습니다.

- GitLab CE Docker image
- GitLab EE Docker image

4.2. 볼륨 위치 설정

다른 모든 것을 설정하기 전에, 구성(Configuration), 로그 및 데이터 파일이 상주할(reside) 디렉터리를 가리키는 새 환경 변수 $GITLAB_HOME를 구성해야 합니다. 디렉터리가 존재하고 적절한 권한이 부여되었는지 확인합니다.

Vim 에디터로 ~/.bashrc 파일을 열어서 맨 마지막 줄에 아래 내용을 추가하고 저장합니다.

```
export GITLAB_HOME=/srv/gitlab
```

```
$ vim ~/.bashrc
$ source ~/.bashrc
```

GitLab 컨테이너는 호스트 마운트 볼륨을 사용하여 영구 데이터를 저장합니다.

Local location	Container location	Usage
$GITLAB_HOME/data	/var/opt/gitlab	애플리케이션 데이터 저장용
$GITLAB_HOME/logs	/var/log/gitlab	로그 저장용
$GITLAB_HOME/config	/etc/gitlab	GitLab 구성 파일 저장용

4.3. docker-compose.yml 파일 준비

GitLab CE 작업 디렉터리 (Working directory)를 생성합니다.

```
$ cd ~/
$ mkdir gitlab-ce
$ cd gitlab-ce
```

docker-compose.yml 파일을 생성합니다.

```
$ touch docker-compose.yml
$ vim docker-compose.yml
```

아래와 같이 내용을 추가하고 저장합니다.

```
web:
  image: "gitlab/gitlab-ce:12.2.5-ce.0"
```

```
  restart: always
  hostname: "gitlab.example.com"
  environment:
    GITLAB_OMNIBUS_CONFIG: |
      external_url 'https://gitlab.example.com'
      # Add any other gitlab.rb configuration here, each on its own line
  ports:
    - "80:80"
    - "443:443"
    - "22:22"
  volumes:
    - "$GITLAB_HOME/config:/etc/gitlab"
    - "$GITLAB_HOME/logs:/var/log/gitlab"
    - "$GITLAB_HOME/data:/var/opt/gitlab"
```

> image에 gitlab/gitlab-ce:latest 또는 gitlab/gitlab-ee:latest를 지정하면 CE/EE 최신 버전의 GitLab이 설치되고, 특정 태그 버전의 이미지를 사용하면 해당 버전의 GitLab이 설치됩니다.

4.4. Docker Compose 를 사용하여 GitLab 시작

작업 디렉터리에 docker-compose.yml 파일이 있는지 확인합니다.

```
$ ls -al
total 12
drwxrwxr-x 2 ubuntu ubuntu 4096 Sep  1 07:30 .
drwxr-xr-x 7 ubuntu ubuntu 4096 Sep  1 07:30 ..
-rw-rw-r-- 1 ubuntu ubuntu  500 Sep  1 07:29 docker-compose.yml
```

아래 명령을 실행하여 GitLab을 시작합니다.

```
$ docker-compose up -d
Creating gitlab-ce_web_1 ... done
```

아래 명령을 실행하면 GitLab 구동 로그를 확인할 수 있습니다.

```
$ docker-compose logs -f
```

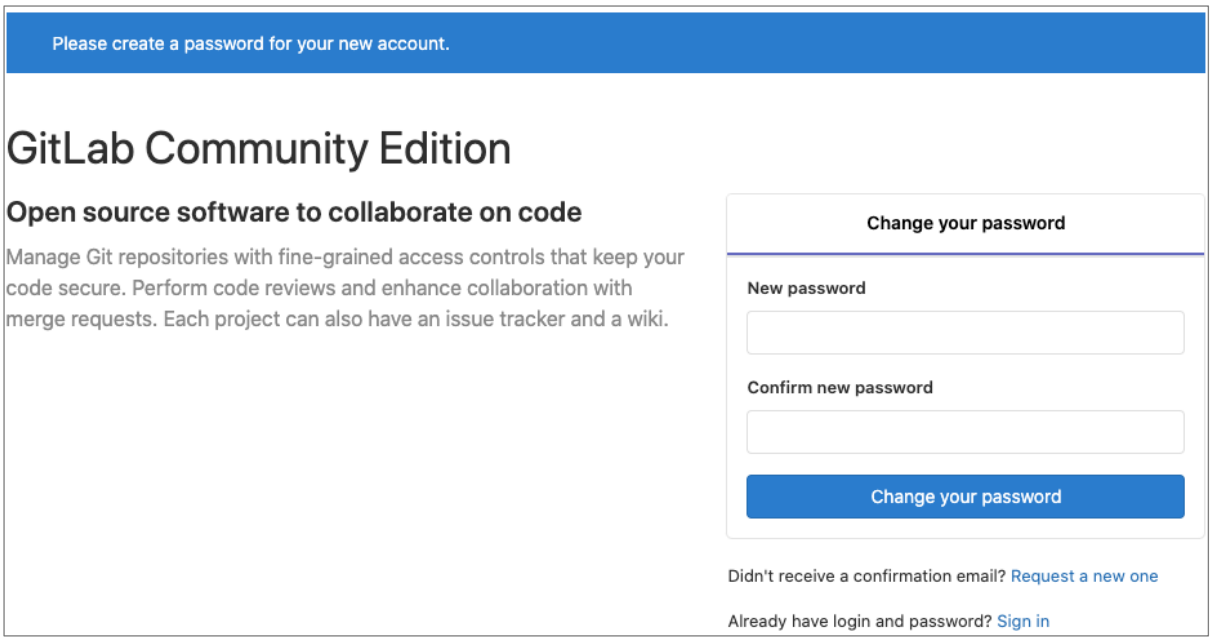
아래 명령을 실행하여 container 목록을 확인할 수 있습니다.

```
$ docker-compose ps -a
      Name                    Command               State                                        Ports
-------------------------------------------------------------------------------------------------------------------
gitlab-ce_web_1   /assets/wrapper   Up (healthy)   0.0.0.0:2222->22/tcp, 0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp
```

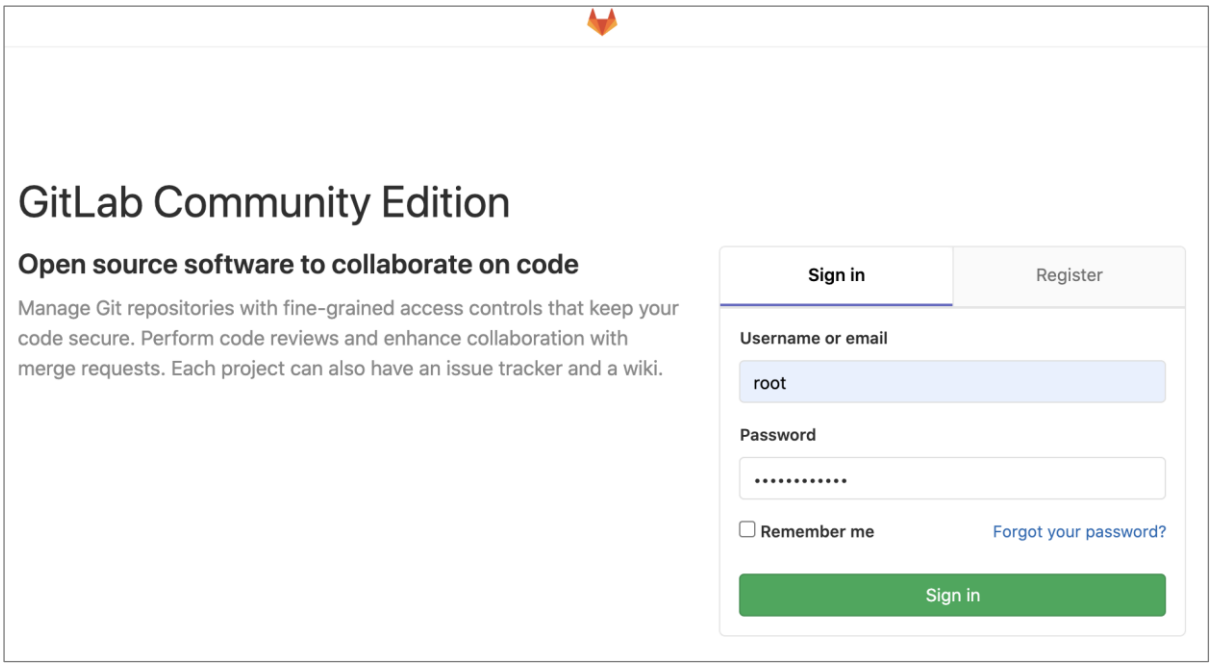
4.5. GitLab 설치 확인

설치한 GitLab CE가 정상 동작하는지 Web으로 접속하여 확인합니다.

external_url로 설정했던 https://gitlab.example.com에 접속합니다. 처음 접속하면 Admin 계정의 비밀번호 변경 페이지로 리다이렉트(Redirect) 됩니다.

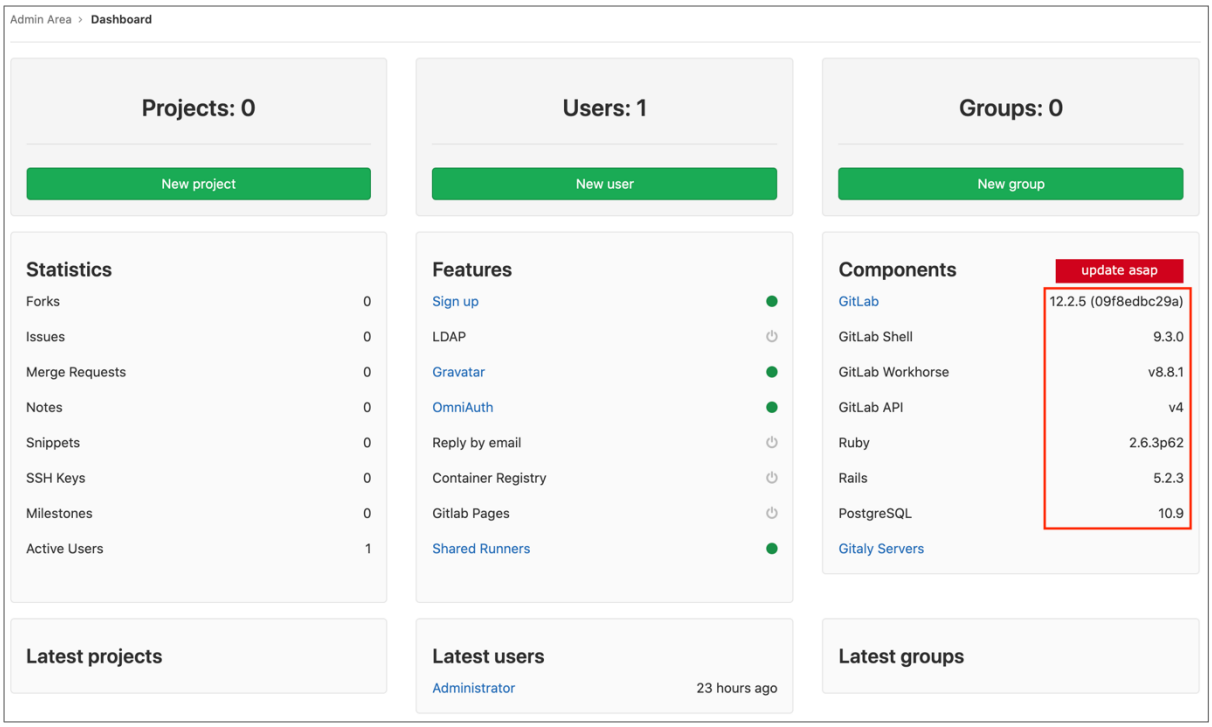


비밀번호와 비밀번호 확인을 입력한 후 **Change your password** 버튼을 클릭합니다. 새로운 비밀번호가 설정되고 Sign in 페이지로 리다이렉트 됩니다.



Username에 'root', Password에 설정한 비밀번호를 입력한 후 **Sign in** 버튼을 클릭하여 로그인합니다.

로그인한 다음, 상단 메뉴바의 렌치(Wrench) 아이콘을 클릭하여 Admin Area 페이지로 이동합니다.



GitLab Components의 버전 정보를 확인할 수 있습니다.