**Install Docker Engine on Ubuntu**

**Uninstall old versions:-**

Older versions of Docker were called docker, docker.io, or docker-engine. If these are installed, uninstall them:

sudo apt-get remove docker docker-engine docker.io containerd runc

It’s OK if apt-get reports that none of these packages are installed.

The contents of /var/lib/docker/, including images, containers, volumes, and networks, are preserved. If you do not need to save your existing data, and want to start with a clean installation.

### Install using the repository:-

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

**Set up the repository:-**

1. Update the apt package index and install packages to allow apt to use a repository over HTTPS:

sudo apt-get update

sudo apt-get install ca-certificates curl gnupg lsb-release

1. Add Docker’s official GPG key:

sudo mkdir -p /etc/apt/keyrings

curl –fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

1. Use the following command to set up the repository:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

**Install Docker Engine:-**

Update the apt package index, and install the latest version of Docker Engine, containerd, and Docker Compose, or go to the next step to install a specific version:

sudo apt-get update

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

sudo service docker start

For test pull the hello world image.

sudo docker run hello-world

**Installing and Configuring the Docker Registry:-**

Docker Registry is itself an application with multiple components, so you will use Docker Compose to manage your configuration. To start an instance of the registry, you’ll set up a docker-compose.yml file to define the location where your registry will be storing its data.

1. On the server you have created to host your private Docker Registry, you can create a docker-registry directory, move into it, and then create a data subfolder with the following commands:

Before starting Open port 5000 for this and port 8080 for docker gui.

mkdir ~/docker-registry && cd $\_

mkdir data

Use your text editor to create the docker-compose.yml configuration file:

vim docker-compose.yml

Add the following content to the file, which describes the basic configuration for a Docker Registry:

docker-compose.yml

version: '3'

services:

registry:

image: registry:2

ports:

- "5000:5000"

environment:

REGISTRY\_STORAGE\_FILESYSTEM\_ROOTDIRECTORY: /data

volumes:

- ./data:/data

The environment section sets an environment variable in the Docker Registry container with the path /data. The Docker Registry application checks this environment variable when it starts up, and as a result begins to save its data to the /data folder.

However, as you have included the volumes: - ./data:/data line, Docker will start to map the /data directory in that container to /data on your registry server. The end result is that the Docker Registry’s data all gets stored in ~/docker-registry/data on the registry server.

The ports section, with configuration 5000:5000, tells Docker to map port 5000 on the server to port 5000 in the running container. This allows you to send a request to port 5000 on the server, and have the request forwarded to the registry application.

You can now start Docker Compose to check the setup:

apt install docker-compose

docker-compose up

You will see download bars in your output that show Docker downloading the Docker Registry image from Docker’s own registry. Within a minute or two, you’ll see output that looks similar to the following (versions might vary):

Output of docker-compose up

Starting docker-registry\_registry\_1 ... done

Attaching to docker-registry\_registry\_1

registry\_1 | time="2018-11-06T18:43:09Z" level=warning msg="No HTTP secret provided - generated random secret. This may cause problems with uploads if multiple registries are behind a load-balancer. To provide a shared secret, fill in http.secret in the configuration file or set the REGISTRY\_HTTP\_SECRET environment variable." go.version=go1.7.6 instance.id=c63483ee-7ad5-4205-9e28-3e809c843d42 version=v2.6.2

registry\_1 | time="2018-11-06T18:43:09Z" level=info msg="redis not configured" go.version=go1.7.6 instance.id=c63483ee-7ad5-4205-9e28-3e809c843d42 version=v2.6.2

registry\_1 | time="2018-11-06T18:43:09Z" level=info msg="Starting upload purge in 20m0s" go.version=go1.7.6 instance.id=c63483ee-7ad5-4205-9e28-3e809c843d42 version=v2.6.2

registry\_1 | time="2018-11-06T18:43:09Z" level=info msg="using inmemory blob descriptor cache" go.version=go1.7.6 instance.id=c63483ee-7ad5-4205-9e28-3e809c843d42 version=v2.6.2

registry\_1 | time="2018-11-06T18:43:09Z" level=info msg="listening on [::]:5000" go.version=go1.7.6 instance.id=c63483ee-7ad5-4205-9e28-3e809c843d42 version=v2.6.2

You’ll address the No HTTP secret provided warning message later in this tutorial. The output shows that the container is starting. The last line of the output shows it has successfully started listening on port 5000.

By default, Docker Compose will remain waiting for your input, so hit CTRL+C to shut down your Docker Registry container.

You have set up a full Docker Registry listening on port 5000. At this point the registry won’t start unless you bring it up manually. Also, Docker Registry doesn’t come with any built-in authentication mechanism, so it is currently insecure and completely open to the public. In the following steps, you will address these security concerns.

**Setting Up Authentication:-**

you can now secure your registry with HTTP authentication to manage who has access to your Docker Registry. To achieve this, you’ll create an authentication file with htpasswd and add users to it. HTTP authentication is quick to set up and secure over a HTTPS connection, which is what the registry will use.

You can install the htpasswd package by running the following:

sudo apt install apache2-utils

Now you’ll create the directory where you’ll store our authentication credentials, and change into that directory. $\_ expands to the last argument of the previous command, in this case ~/docker-registry/auth:

mkdir ~/docker-registry/auth && cd $\_

Next, you will create the first user as follows, replacing username with the username you want to use. The -B flag specifies bcrypt encryption, which is more secure than the default encryption. Enter the password when prompted:

htpasswd -Bc registry.password username

Next, you’ll edit the docker-compose.yml file to tell Docker to use the file you created to authenticate users.

cd ~/docker-registry

vim docker-compose.yml

You can add environment variables and a volume for the auth/ directory that you created, by editing the docker-compose.yml file to tell Docker how you want to authenticate users. Add the following highlighted content to the file:

docker-compose.yml

version: '3'

services:

registry:

image: registry:2

ports:

- "5000:5000"

environment:

REGISTRY\_AUTH: htpasswd

REGISTRY\_AUTH\_HTPASSWD\_REALM: Registry

REGISTRY\_AUTH\_HTPASSWD\_PATH: /auth/registry.password

REGISTRY\_STORAGE\_FILESYSTEM\_ROOTDIRECTORY: /data

volumes:

- ./auth:/auth

- ./data:/data

For REGISTRY\_AUTH, you have specified htpasswd, which is the authentication scheme you are using, and set REGISTRY\_AUTH\_HTPASSWD\_PATH to the path of the authentication file. Finally, REGISTRY\_AUTH\_HTPASSWD\_REALM signifies the name of htpasswd realm.

You can now verify that your authentication works correctly, by running the registry and checking that it prompts users for a username and password.

docker-compose up

**Starting Docker Registry as a Service:-**

You want to ensure that your registry will start whenever the system boots up. If there are any unforeseen system crashes, you want to make sure the registry restarts when the server reboots. Open up docker-compose.yml:

vim docker-compose.yml Add the following line of content under registry::

docker-compose.yml

...

registry:

restart: always

...

You can start your registry as a background process, which will allow you to exit the ssh session and persist the process:

docker-compose up –d

**Publishing to Your Private Docker Registry:-**

Go to /etc/docker and create file daemon.json

vim /etc/docker/daemon.json

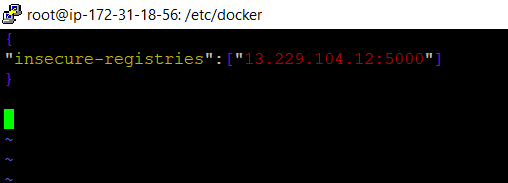
and paste the following script with public ip address of server with port 5000 like

below.

{

"insecure-registries":["13.229.104.12:5000"]

}



You are now ready to publish an image to your private Docker Registry, but first you have to create an image.

docker pull nginx

docker login 13.229.104.12:5000 (e.g public\_ip\_address:5000)

Enter the username and corresponding password from earlier. Next, you will tag the image with the private registry’s location in order to push to it:

docker tag nginx 13.229.104.12:5000/nginx

docker push 13.229.104.12:5000/nginx

Your image will be pushed to local repository.

## Pulling From Your Private Docker Registry:-

Return to your registry server so that you can test pulling the image from your **client** server. It is also possible to test this from a third server.

Log in with the username and password you set up previously:

docker login 13.229.104.12:5000

You’re now ready to pull the image. Use your domain name and image name, which you tagged in the previous step:

docker pull 13.229.104.12:5000/nginx

docker images

Now you will see, you have successfully pulled image from local repository.

Let’s launch a Docker image for the user interface:

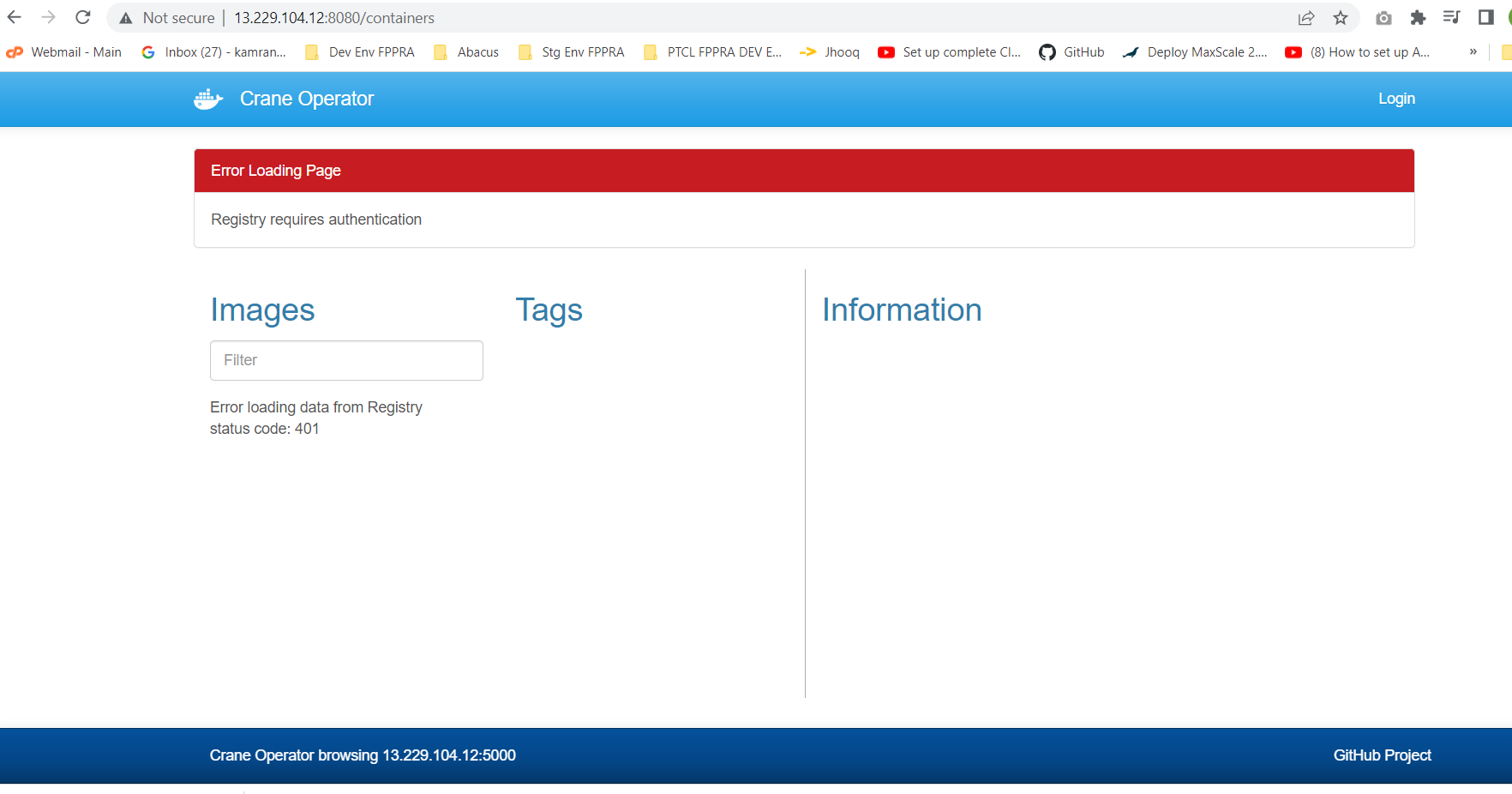
docker run -d -p 8080:80 --name=docker\_registry\_ui -e REGISTRY\_HOST=13.229.104.12 -e REGISTRY\_PORT=5000 -e REGISTRY\_PROTOCOL=http -e SSL\_VERIFY=false -e ALLOW\_REGISTRY\_LOGIN=true -e REGISTRY\_ALLOW\_DELETE=true \ parabuzzle/craneoperator:latest

REGISTRY\_HOST=13.229.104.12 means server ip address.

GO to browser and enter public ip address along with port 8080.

13.229.104.12:8080

you will see login page, now enter username and password you kept for local repository.



Now you will see the images here like this.

