Docker and Docker-Compose Setup on AWS EC2 Instance

JUNE 13, 2018 NIDHI GUPTA LEAVE A COMMENT

This blog will help you to setup a docker and docker-compose on AWS EC2 Instance

Install Docker on AWS

sudo yum update -y

sudo yum install -y docker

sudo service docker start

sudo usermod -a -G docker ec2-user

Docker version 17.09.1-ce, build

Docker installed successfully.

Install Docker-Compose. Get the latest one from
here https://github.com/docker/compose/releases

```
curl -L https://github.com/docker/compose/releases/download/1.20.0/docker-
compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
```

Test Docker installation

Run hello-world image

docker run hello-world

```
[ec2-user@ip-172-31-30-240 ~]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:083de497cff944f969d8499ab94f07134c50bcf5e6b9559b27182d3fa80ce3f7
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://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/

[ec2-user@ip-172-31-30-240 ~]$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              f2a91732366c        3 months ago        1.85kB
```

Build a Image

Create a Dockerfile,requirements.txt,app.py

```
docker build -t friendlyhello .
```

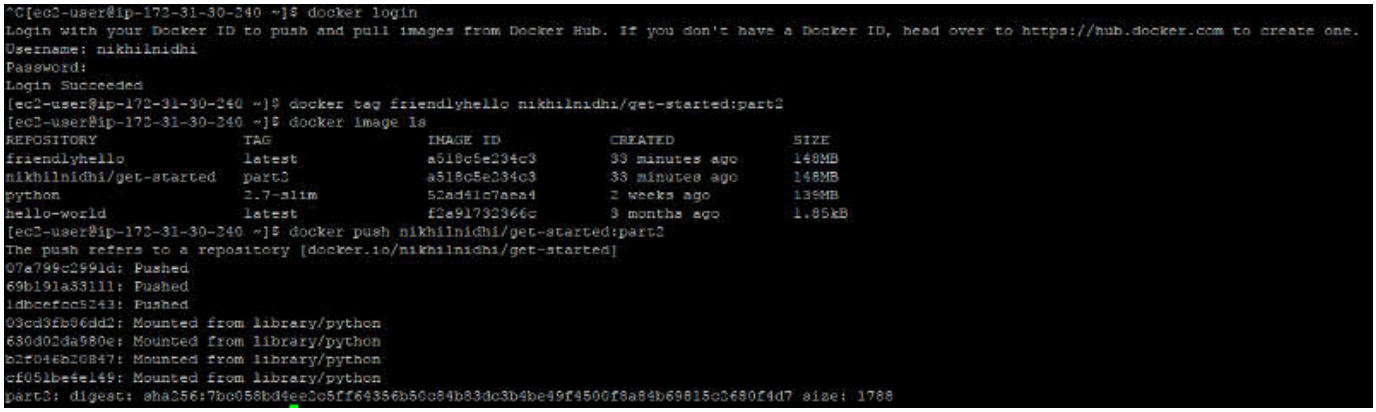
```
docker image ls
docker run -p 4000:80 friendlyhello
```

```
[ec2-user@ip-172-31-30-240 ~]$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
friendlyhello       latest              a518c5e234c3        8 seconds ago       148MB
python              2.7-slim            52ad41c7aea4        2 weeks ago         139MB
hello-world         latest              f2a91732366c        3 months ago        1.85kB
[ec2-user@ip-172-31-30-240 ~]$ docker run -p 4000:80 friendlyhello
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
```

RUn the app in detached mode

```
docker run -d -p 4000:80 friendlyhello
docker ps
docker images
docker stop containerid
```

Push image in Docker Hub



Now you can run your image from anywhere

```
docker run -p 4000:80 username/repository:tag
```

Till now, we have created an image using Dockerfile,push it to DOckerhub so that anyone can use it now.

Some useful commands

Create DockerImage with commit option

1. Run a container from the ubuntu and connect it to its command line:

docker run -i -t ubuntu /bin/bash

2. Install the Git toolkit:

```
root@dee2cb192c6c:/# apt-get update
root@dee2cb192c6c:/# apt-get install -y git
```

1. Check if the Git toolkit is installed:

```
root@dee2cb192c6c:/# which git
        /usr/bin/git
```

1. Exit the container:

```
root@dee2cb192c6c:/# exit
```

1. Check what has changed in the container comparing it to the ubuntu image:

```
$ docker diff dee2cb192c6c
```

The command should print a list of all files changed in the container.

1. Commit the container to the image:

```
$ docker commit dee2cb192c6c ubuntu_with_git
```

Using the exact same method, we can build ubuntu_with_git_and_jdk on top of the ubuntu_with_git image:

```
$ docker run -i -t ubuntu_with_git /bin/bash
root@6ee6401ed8b8:/# apt-get install -y openjdk-8-jdk
root@6ee6401ed8b8:/# exit
$ docker commit 6ee6401ed8b8 ubuntu_with_git_and_jdk
```

Create Image directly using Dockerfile

Create a Docker file with below contents

FROM ubuntu:16.04
MAINTAINER Rafal Leszko
RUN apt-get update && \
apt-get install -y python
COPY hello.py .
ENTRYPOINT [“python”, “hello.py”]

Creata a hello.py

print “Hello World from Python!”

```
docker build -t hello_world_python .
$ docker run hello_world_python
```

Docker Volumes

Let’s start with an example and specify the volume with the -v <host_path>:<container_path> option and connect to the container:

```
$ docker run -i -t -v ~/docker_ubuntu:/host_directory ubuntu:16.04 /bin/bash
```

Now, we can create an empty file in host_directory in the container:

```
root@01bf73826624:/# touch host_directory/file.txt
```

Let’s check if the file was created in the Docker host’s filesystem:

```
root@01bf73826624:/# exit
exit
```

```
$ ls ~/docker_ubuntu/
file.txt
```

We can see that the filesystem was shared and the data was therefore persisted permanently. We can now stop the container and run a new one to see that our file will still be there:

```
$ docker stop 01bf73826624
```

```
$ docker run -i -t -v ~/docker_ubuntu:/host_directory ubuntu:16.04 /bin/bash
root@a9e0df194f1f:/# ls host_directory/
file.txt
```

```
root@a9e0df194f1f:/# exit
```

Instead of specifying the volume with the -v flag, it’s possible to specify the volume as an instruction in the Dockerfile, for example:

```
VOLUME /host_directory
```

Some useful commands

docker ps (to show all running containers)

docker ps -a (to show all containers(stopped and running)

docker images

docker exec -it 4a53d243816e bash (To go inside a container)

Docker setup has completed succesfully with some basic knowledge.

Create a Docker-compose.yml/scale up application

is a YAML file that defines how Docker containers should behave in production.

This `docker-compose.yml` file tells Docker to do the following:

- Pull the image we uploaded in step 2 from the registry.
- Run 5 instances of that image as a service called `web`, limiting each one to use, at most, 10% of the CPU (across all cores), and 50MB of RAM.
- Immediately restart containers if one fails.
- Map port 80 on the host to `web`'s port 80.
- Instruct `web`'s containers to share port 80 via a load-balanced network called `webnet`. (Internally, the containers themselves publish to `web`'s port 80 at an ephemeral port.)
- Define the `webnet` network with the default settings (which is a load-balanced overlay network).

```
docker swarm init
```

```
docker stack deploy -c docker-compose.yml getstartedlab
```

```
[ec2-user@ip-172-31-30-240 ~]$ docker swarm init
Swarm initialized: current node (dbshnyj5m7vx0zqzudi62t5ec) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-1akalltwmj6b4dljap5a6tkhbzjiefuyvvh81d9v0u7avyc8c9-5cnlk7xy3y3k9uzk1qc715so3 172.31.30.240:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[ec2-user@ip-172-31-30-240 ~]$ docker stack deploy -c docker-compose.yml getstartedlab
Creating network getstartedlab_webnet
Creating service getstartedlab_web
[ec2-user@ip-172-31-30-240 ~]$
```

Our single service stack is running 5 container instances of our deployed image on one host.

```
docker service ps getstartedlab_web
```

```
docker container ls -q
```

```
[ec2-user@ip-172-31-30-240 ~]$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                              PORTS
3r9lvxvvfdwj        getstartedlab_web   replicated          5/5                 nikhilnidhi/get-started:part2      *:80->80/tcp
[ec2-user@ip-172-31-30-240 ~]$ docker service ps getstartedlab_web
ID                  NAME                  IMAGE                              NODE                DESIRED STATE       CURRENT STATE            ERROR
s0poao3ilm5d        getstartedlab_web.1   nikhilnidhi/get-started:part2      ip-172-31-30-240    Running             Running 5 minutes ago
yghelks9tj8b        getstartedlab_web.2   nikhilnidhi/get-started:part2      ip-172-31-30-240    Running             Running 5 minutes ago
vrslgz7mcct7        getstartedlab_web.3   nikhilnidhi/get-started:part2      ip-172-31-30-240    Running             Running 5 minutes ago
6tk08i753t0r        getstartedlab_web.4   nikhilnidhi/get-started:part2      ip-172-31-30-240    Running             Running 5 minutes ago
509m64y4tjq2        getstartedlab_web.5   nikhilnidhi/get-started:part2      ip-172-31-30-240    Running             Running 5 minutes ago
[ec2-user@ip-172-31-30-240 ~]$ docker container ls -q
53c8be9c3686
fcd72fd7666d
b607bc6b7ccd
748fce107176
7cl26ed87b83
[ec2-user@ip-172-31-30-240 ~]$
```

You can run `curl -4 http://localhost` several times in a row and you will get different hostnames

```
[ec2-user@ip-172-31-30-240 ~]$ curl -4 http://localhost
<h3>Hello World!</h3><b>Hostname:</b> 53c8be9c3686<br/><b>Visits:</b> <i>cannot connect to Redis, counter disabled</i>[ec2-user@ip-172-31-30-240 ~]$
[ec2-user@ip-172-31-30-240 ~]$ curl -4 http://localhost
<h3>Hello World!</h3><b>Hostname:</b> 7cl26ed87b83<br/><b>Visits:</b> <i>cannot connect to Redis, counter disabled</i>[ec2-user@ip-172-31-30-240 ~]$
```

You can update docker-compose.yml file and re-run the stack command .Docker performs an in-place update, no need to tear the stack down first or kill any containers.

```
docker stack rm getstartedlab
```

```
docker swarm leave --force
```

We have learnt how it should run in production by turning it into a service, scaling it up 5x in the process.

CLuster in Docker

Now we will deploy this application onto a cluster, running it on multiple machines. Multi-container, multi-machine applications are made possible by joining multiple machines into a “Dockerized” cluster called a **swarm**.

Swarm—group of machines that are running Docker and joined into a cluster.

Swarm managers can use several strategies to run containers, such as “emptiest node”—which fills the least utilized machines with containers. Or “global”, which ensures that each machine gets exactly one instance of the specified container. You instruct the swarm manager to use these strategies in the Compose file, just like the one you have already been using.

Swarm managers are the only machines in a swarm that can execute your commands, or authorize other machines to join the swarm as **workers**. Workers are just there to provide capacity and do not have the authority to tell any other machine what it can and cannot do.

`docker swarm init` to enable swarm mode and make your current machine a swarm manager then run `docker swarm join` on other machines to have them join the swarm as workers

Install Docker-machine on AWS EC2

```
curl -L https://github.com/docker/machine/releases/download/v0.14.0/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine && \
sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```