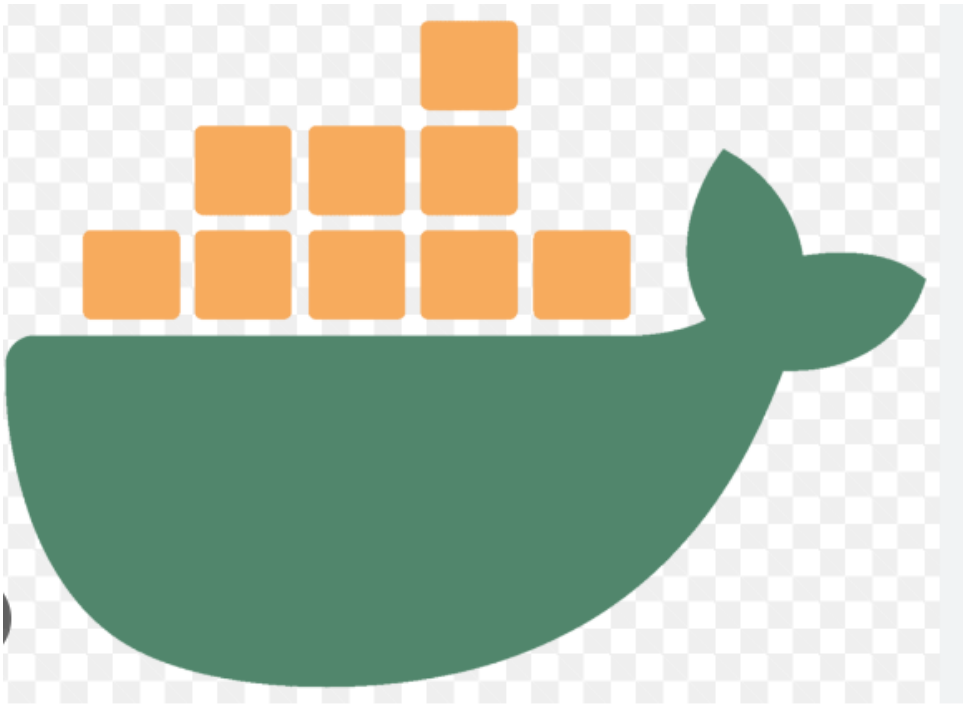
Docker for DevOps Engineers.

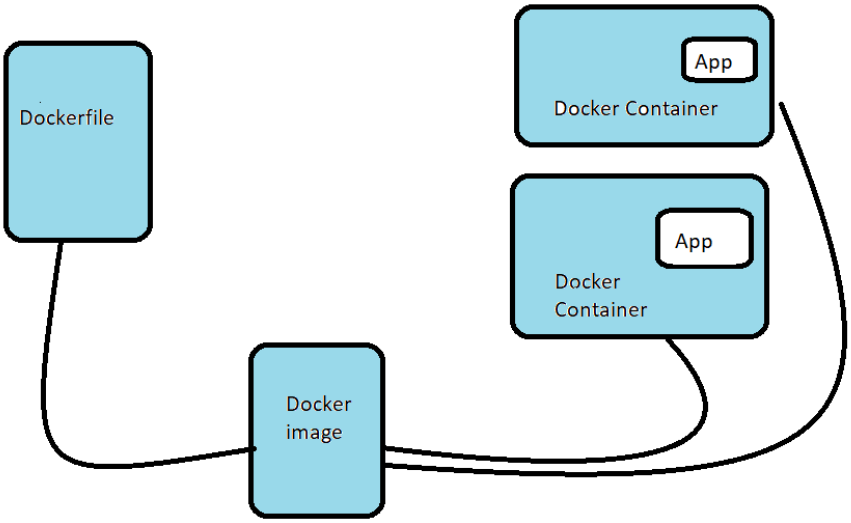
Docker

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.



Docker is a tool that performs OS-level virtualization, called as Containerization. Using this container Docker run applications. It allows applications to use the same Linux.

Basic Components of Docker:



1. DockerFile :

Dockerfile is a text file, but without .txt extension. This Dockerfile uses the Docker Platform to Create the image and run the containers. A Dockerfile is a set of instructions for using this instructions Docker create a container. Using this Dockerfile you can automate the process to deploy a application smoothly.

Here are some Docker Commands:

- **FROM: -**

  This Command is used to specify the base image and version. For ex:

  FROM python: 3.9

  Here we use python image and the version of this python is 3.9

- **RUN:**

Execute a command in the image. This command is run during the building the image process.

RUN pip install –r requirments.txt

Here we go to requirments.txt file and run all dependencies.

- **COPY:**

This COPY command is used to copy the files from host machine to image.

Using this command all your data are copied to image.

COPY. . --- copy data from current location to image current location

Limitation: - But there are some limitations i.e.

1. You Cannot copy data from URL to Image
2. You cannot copy data from tar file to image

- **ENV:**

This ENV command are used to set the environment variable in the image.

ENV MYSQL_ROOT_PASSWORD =root@123

These are also persistent with the container and can be referred to at any moment.

- **CMD:**

CMD execute the command same as shell command and they are not capable of run or execute an image

You can use a cmd in both the way

1. CMD ls –l
2. CMD [“ls”,”-l”]

CMD is the Default argument passed to ENTRYPOINT.

- **ENTRYPOINT:**

ENTRYPOINT execute the command same as CMD. This is latest version of CMD Command.

The Advantage of using ETRYPOINT is, they cannot be ignored or overridden arguments unlike CMD.

- **EXPOSE:**

The EXPOSE Command used to expose the container port to anywhere.

- **ADD:**

The ADD Command is used to copy the files from host machine to image Same as COPY Command.

The only difference is that, Using ADD command You can copy the data from URL or any TAR AARCHIVE files.

Using COPY command this is not possible to copy the data from URL or any other TAR ARCHIVE files.

- **MAINTAINER: -**

  This Command is used to Specify the Author/Owner name, who is managing this Dockerfiile.

2. Docker Image:

A Docker Image is a Read only file used to execute code in Docker Container. A Docker Images containes Application code,libraries,Dependencies and other files needed to make an application run. When user runs an image, it can become one or many instances of a container.

some Useful Commands:

1] docker images: - to list out all images

2] docker rmi <image_name> :- to Remove image

3]docker pull <image_name> :- to download image from Docker Hub

3. Docker Container:

A DockerContainer is a lightweight runnable instance of Dockerimage.Container Includes everything, that need to run an application. You can create,stop,delete container using Docker CLI or Docker API.

some Useful Commands:

1] docker PS: To see all running Container

2] docker run –d –p <host_port_no>:<container_port_no> --name <container name> <image_name> :- To create a Container

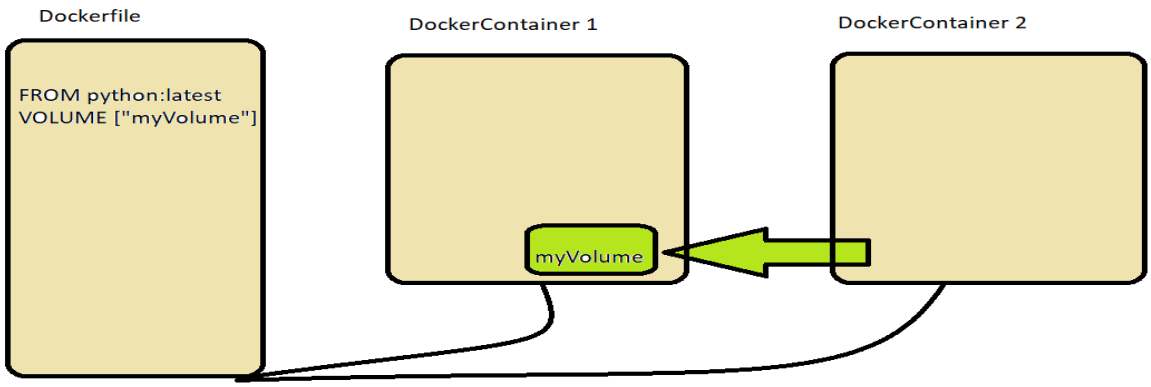
3] docker exec –it <container_ID> /bin/bash: - Go inside the Container

4] docker stop <Container_ID> :- Stop the Docker Container

5] docker rm <Container_ID> :- Remove the Docker Container

4. Docker Volume:

Docker Volumes are the preferred mechanism for persisting data generated by and used by Docker Containers.



You can declare a directory as a volume only while creating the container.Also you can't create volume from the existing container.

You can share one volume across any no.of containers.

We can Map volume in Two Ways:

1] Host to Container:

a] first you create a volume in your host system

docker volume create –name "flask-volume" --opt device=<local volume path> --opt type=none –opt o=bind

b] Then create a docker-compose.yaml file and m ap volumes name and container path

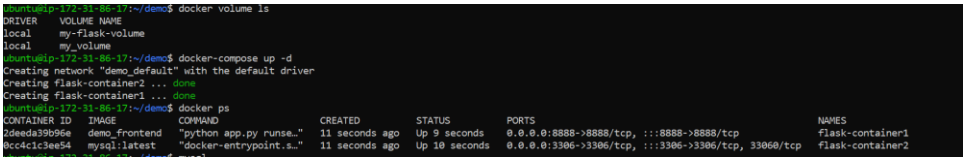
```
version : '3.3'
services :
  frontend :
      container_name : "flask-container1"
      build : .
      ports :
        - '8888:8888'
      volumes :
        - my-flask-volume:/app

  bakend :
    container_name : "flask-container2"
    image : mysql:latest
    ports :
       - '3306:3306'
    environment:
                  MYSQL_ROOT_PASSWORD: PASSWORD

volumes :
     my-flask-volume :
       external : true
```

c] Run the docker-compose file and check the container is running or not?

docker-compose up –d

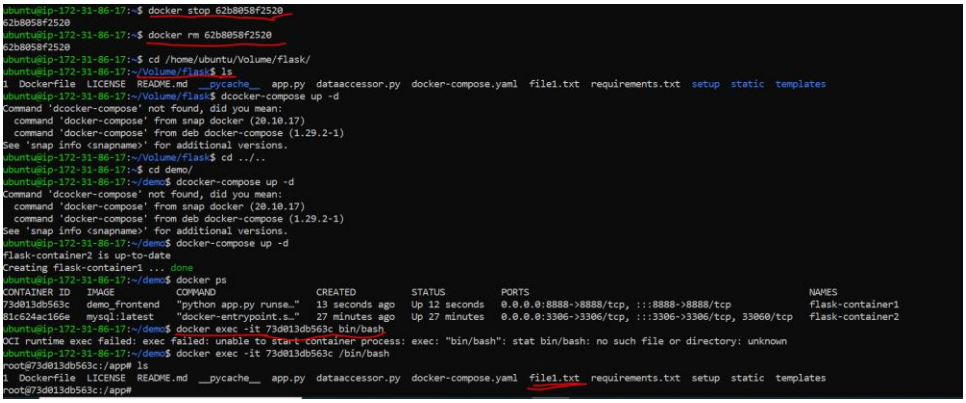


d] Go to host volume path and check the all-container data are available or not?

If data is available, then for testing purpose you can add/create a file and check this file is available on container or not?

e] then next step is removing container and check the data is persist in host volume folder/directory.

Again, create a container and check all data is available same as volume



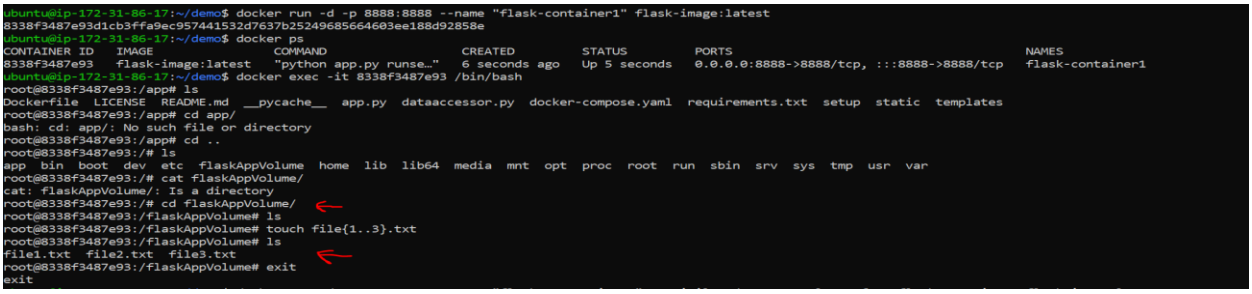
2] Container to Container: -

Steps: -

1] first create a dockerfile and add VOLUMES inside Dockerfile. See above fig.

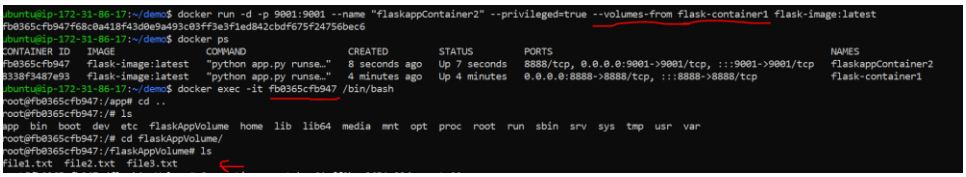
2] then create a container, and see the volume is created inside container.

inside “flask-volume”, we add 3 files named as **file1, file2, file3.**



3] Create a new Container and add some arguments.

docker run –d –p 9001:9001 –name “flaskContainer2” --privileged=true --volumes-from <first_containername > <image name>



After that, we see volumes are shared.

some Useful Commands:

1] docker volume ls: -check all available volume list

2] docker volume prune: - remove all unused volumes

3] docker volume inspect <volume name>: - Detailed information of Docker Volume

4] docker volume rm <volumed>: - To Remove Volume

5. Docker Compose:

Docker Compose is a tool for running multi-container applications on Docker defined using the yaml file format. A Compose file is used to define how one or more containers that make up your application are configured. Once you have a Compose file, you can create and start your application with a single command:

Docker-compose up

some Useful Commands:

1] vi docker-compose.yaml :- To create a docker-compose file

```
version : '3.3'
services :
  frontend :
      container_name : "flask-container1"
      build : .
      ports :
        - '8888:8888'
      volumes :
        - my-flask-volume:/app

  bakend :
    container_name : "flask-container2"
    image : mysql:latest
    ports :
       - '3306:3306'
    environment:
                  MYSQL_ROOT_PASSWORD: PASSWORD

volumes :
     my-flask-volume :
       external : true
```

2] docker-compose up –d :- Run docker-compose file(Create multi containers at a time) in detach (Background) mode.

3] docker-compose down: - stop the containers automatically

4] docker ps :- To check all currently running containers

6. Docker Swarm:

A Docker Swarm is a container orchestration tool running the Docker Application. A Docker Swarm is a group of either physical or virtual machines that running the Docker application and that have been configured to join together in a cluster.

Components: -

1]service: -Defines task that needs to be executed on the manager and worker node

2]Task: - They are the docker containers that execute commands.

3] Manager Node: - Takes the commands to create services, allocate IP’s and assign tasks to manager node.

4] Worker Node: - Performs the task of running a container.

First step is,you need to set the server as **Leader/Manager.**

Run the below command to initialize a swarm as a master/Manager node

docker swarm init

```
ubuntu@ip-172-31-92-62:~$ sudo docker swarm init
Swarm initialized: current node (jmsii5x0dnmnhs4nardny694d) is now a manager.

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

    docker swarm join --token SWMTKN-1-480zc030k0709lgvs6fkozejtcx82rtk48yly0f4xobwe738ha-1rpkdc784kbs823uxjh1rpl9g 172.31.92.62:2377

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

You can run below command to see the information of a swarm manager. Using this command, we can easily identify who is a manager?

docker info

```
ubuntu@ip-172-31-92-62:~$ docker info
Client:
 Context:    default
 Debug Mode: false

Server:
 Containers: 1
  Running: 1
  Paused: 0
  Stopped: 0
 Images: 1
 Server Version: 20.10.12
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: systemd
 Cgroup Version: 2
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: active
  NodeID: jmsii5x0dnmnhs4nardny694d
  Is Manager: true
  ClusterID: che9l0fm2csgwm7u5k14twfl1
  Managers: 1
  Nodes: 2
  Default Address Pool: 10.0.0.0/8
  SubnetSize: 24
  Data Path Port: 4789
  Orchestration:
   Task History Retention Limit: 5
  Raft:
   Snapshot Interval: 10000
   Number of Old Snapshots to Retain: 0
```

You need to join another node to swarm manager, you need a token key.

Use below token key to join with Manager Node.

```
ubuntu@ip-172-31-91-93:~$ docker swarm join --token SWMTKN-1-480zc030k0709lgvs6fkozejtcx82rtk48yly0f4xobwe738ha-1rpkdc784kbs823uxjh1rpl9g 172.31.92.62:2377
This node joined a swarm as a worker.
```

For creating a service, below command

Docker service create –name <service name> --replicas <no> --publish <port no> <dockerhub imagename>

Manager Continually monitor the worker node health. If any worker container is failed, the manager creates new container.

```
ubuntu@ip-172-31-92-62:~$ docker service create --name new-flask-app --replicas 2 --publish 9001:9001 rushis750/new-flask-image:latest
pa991iyhuxf2dartjdgij85yk
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
```

To check current running service

docker service ls

```
ubuntu@ip-172-31-92-62:~$ docker service ls
ID             NAME          MODE         REPLICAS   IMAGE                          PORTS
pa991iyhuxf2   new-flask-app   replicated   2/2        rushis750/new-flask-image:latest   *:9001->9001/tcp
ubuntu@ip-172-31-92-62:~$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS     NAMES
35018b18a58e   rushis750/new-flask-image:latest   "/bin/sh -c 'FLASK_A…"   2 minutes ago   Up 2 minutes             new-flask-app.2.1gybzl3pbyjphx3zdxhe1ry0i
ubuntu@ip-172-31-92-62:~$
```

```
ubuntu@ip-172-31-91-93:~$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS     NAMES
043adee062db   rushis750/new-flask-image:latest   "/bin/sh -c 'FLASK_A…"   2 minutes ago   Up 2 minutes             new-flask-app.1.gd04z5daqgueqiki9f200nkdp
ubuntu@ip-172-31-91-93:~$
```

Using Deployment, we can run multiple containers on a different server at a time.

For creating Deployment, we need to create a .yaml file. Inside yaml file we define a container like docker-compose.

```
ubuntu@ip-172-31-86-17:~/demo$ vi node-cluster.yaml
ubuntu@ip-172-31-86-17:~/demo$ docker stack deploy -c node-cluster.yaml node-stack
Creating network node-stack_default
Creating service node-stack_web
Creating service node-stack_backend
Creating service node-stack_web-server
ubuntu@ip-172-31-86-17:~/demo$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
d61ce4721d03   nginx:latest   "/docker-entrypoint.…"   4 seconds ago   Up 2 seconds   80/tcp    node-stack_web-server.1.qqkwid8mac9zdksj6q5ymi0zq
```

```
ubuntu@ip-172-31-82-204:~$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                 NAMES
9aaa2aa7f00a   mysql:latest   "docker-entrypoint.s…"   28 seconds ago   Up 27 seconds   3306/tcp, 33060/tcp   node-stack_backend.1.sbmjb2rinmri0hmzmf3hdsa11
```

```
ubuntu@ip-172-31-80-230:~$ sudo docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED             STATUS             PORTS      NAMES
329fd6ee0180   rushis750/new-todo-image:latest   "python manage.py ru…"   About a minute ago   Up About a minute   9001/tcp   node-stack_web.1.0ghmlxgwyty1xleuq275u3pxi
```

some Useful Commands:

1] docker swarm init :- initialize docker swarm manager

2] docker swarm join-token worker: - to create a token, use this token to connect with Swarm Manager

3] docker service create: - create a service

4] docker service ls: - Running service list

5] docker service rm <service name>:- Remove service

6] docker nodes ls:- Check Connected nodes list

7] docker swarm leave:- disconnect with swarm manager. Used only with worker server.

7. Docker Network:

Docker allows you to create virtual spaces called networks, where you can connect multiple containers (small packages that hold all the necessary files for a specific application to run) together. This way, the containers can communicate with each other and with the host machine (the computer on which the Docker is installed). When we run a container, it has its own storage space that is only accessible by that specific container. If we want to share that storage space with other containers, we can't do that.

reference