Docker Short notes

FOR DEVOPS ENGINEERS

TRAIN WITH SHUBHAM

Docker Short Notes

Basics of Docker

- Docker was first released in March 2013. It is developed by Solomon Hykes and Sebastien paul.
- Docker is an open-source centralized platform designed to create deploy and run applications.
- Docker uses the container on the host O.S to run applications. It allows applications to use the same Linux Kernel as a system on the host computer, rather than creating a whole virtual O.S.
- We can install Docker on any O.S but the Docker engine runs natively on Linux distribution.
- Docker is written in the 'GO' language.
- Docker is a tool that performs OS-level virtualization, also known as containerization.
- Before Docker, many users face the problem that a particular code runs in the developer's system but not in the user's system.

Advantages of Docker

- No pre-allocation of RAM.
- CI Efficiency → Docker enables you to build a container image and use. that same image across every step of the deployment process.
- Less Cost.
- It is light in weight.
- It can run on physical Hardware, Virtual Hardware, or on Cloud.
- You can re-use the image.
- It took very less time to create the container.

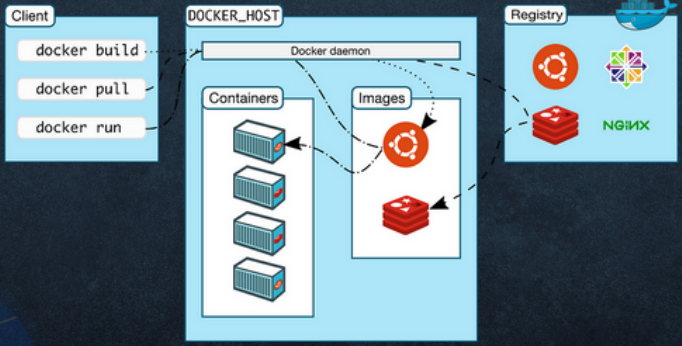
Disadvantages of Docker

- Docker is not a good solution for application that requires a rich GUI.
- Difficult to manage a large number of containers.
- Docker does not provide cross-platform compatibility means if an application is designed to run in a docker container on windows, then it can't run on Linux or vice-versa.
- Docker is suitable when the development O.S and testing O.S are the same, if the O.S is different, we should use Virtual Machine.
- No solution for Data Recovery & Backup.

Note

When Image is running we can say container, When we send a container or not-runnable state we can say Image.

Architecture of Docker



Components of Docker

Docker Daemon

- Docker daemon runs on the Host O.S.
- It is responsible for running containers, and managing docker services.
- Docker daemons can communicate with other daemons.

Docker Client

- Docker users can interact with the docker daemon through a client.
- Docker client uses commands and Rest API to communicate with the docker daemon.
- When a client runs any server command on the docker client terminal, the client terminal sends the docker commands to the docker daemon.
- Docker clients can communicate with more than one daemon.

Docker Host

- Docker Host is used to providing an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.

Docker Hub/Registry

- Docker registry manages and stores the docker images.
- There are two types of registries in the docker.

1) Public Registry → Public registry also called Docker Hub.

2) Private Registry → It is used to share images within the enterprise.

Docker Images

- Docker images are the read-only binary templates used to create docker containers. or, a single file with all dependencies and configurations required to run a program.

→ Ways to create an images

1. Take an image from Docker Hub.

2. Create an image from Docker File.

3. Create an image from existing docker containers.

Docker Container

- The container holds the entire package that is needed to run the application. or,
- In other words, we can say that the image is a template and the container is a copy of that template.
- container is like virtualization when they run on the Docker engine.
- Images become containers when they run on the docker engine.

Docker Installation and important commands

- Create a machine on AWS with Docker installed AMI, and install Docker if not installed. *yum install docker*

</> To see all images present in your local.

```
</>
        docker images
```

</> To find out images in the docker hub.

```
</>
  docker search <image_name>
```

</> To download an image from docker-hub to a local machine.

```
</>
    docker pull <image_name>
```

</> To give a name to the container and run.

```
docker run -it --name <container_name>
    <image_name> /bin/bash [-it →
interactive mode and direct to terminal]
```

</> To check, whether the service is starting or not.

```
service docker status or, service
        docker info
```

</> To start the service.

```
service docker start
```

</> To start/stop the container.

```
docker start <container_name> or,
  docker stop <container_name>
```

</> To go inside the container.

```
docker attach <container_name>
```

</> To see all the containers.

```
docker ps -a
```

</> To see only running containers.

```
docker ps
```

</> To stop the container.

```
docker stop <container_name>
```

</> To delete the container.

```
docker rm <container_name>
```

</> Exiting from the docker container.

```
exit
```

</> To delete images.

```
docker image rm <image_name>
```

Dockerfile Creation

Docker image creation using existing docker file

</> We have to create a container from our image, therefore create one container first—

```
docker run -it --name
<container_name> <image_name>
/bin/bash
```

```
cd tmp/ [Inside directory to create a
file]
```

</> Now create one file inside this tmp directory

```
touch myfile
```

</> Now if you want to see the difference between the base image & changes on it then—

```
docker diff <old_container_name> [D →
Deletion, C → Change, A →
Append/Addition]
```

</> Now create an image of this container

```
docker commit <container_name>
<container_image_name>
```

```
docker images
```

</> Now create a container from this image.

```
docker run -it --name <container_name>
<update_image_name> /bin/bash--> ls--> cd tmp/--
> lso/p --> myfile [You will get all files back]
```

Dockerfile Creation using Dockerfile

Dockerfile

- Dockerfile is a text file it contains some set of instruction
- Automation of docker image creation

FROM

- For the base image. This command must be on top of the docker file.

RUN

- To execute commands, it will create a layer in the image.

MAINTAINER

- Author/Owner/Description

COPY

- Copy files from the local system (dockerVM) we need to provide a source, and destination. (We can't download the file from the internet and any remote repo)

ADD

- Similar to copy, it provides a feature to download files from the internet, also we extract the file from the docker image side.

EXPOSE

- To expose ports such as port 8080 for tomcat, port 80 for Nginx, etc...

WORKDIR

- To set a working directory for a container.

CMD

- Execute commands but during container creation.

ENTRYPOINT

- Similar to CMD, but has higher priority over CMD, the first commands will be executed by ENTRYPOINT only.

ENV

- Environment Variables.

ARG

- To define the name of a parameter and its default value, the difference between ENV and ARG is that after you set on env. using ARG, you will not be able to access that late on when you try to run the Docker container.

Creation of Dockerfile

1. Create a file named Dockerfile
2. Add instructions in Dockerfile
3. Build dockerfile to create an image
4. Run image to create the container

```
[root@ip...]# vi DockerfileFROM ubuntu
RUN echo "Creating our Image" > /tmp/testfile
```

To create an image out of Dockerfile

```
docker build -t <image_name> . [-t → tagname , . ← the current directory for this dot]
```

Check process state

```
docker ps -a
```

</> See images

```
docker images
```

</> To create a container from the above image

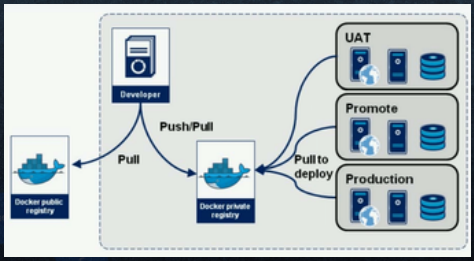
```
docker run -it --name
<container_name> <img_name> /bin/bash
```

</> Ex. Command for next image creation

- No need to create a new Dockerfile we will just update the Dockerfile.

```
[root@ip...]# vi DockerfileFROM ubuntu
WORKDIR /tmp
RUN echo "Hello World" > /tmp/testfile
ENV myname DevOps_Brother
COPY testfile /tmp
ADD test.tar.gz /tmp
```

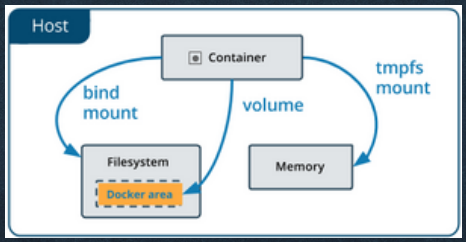
- Open Dockerfile with vi remove all and this code
- We have to create a testfile, test, and test.tar.gz
- Then build the image
- Then create the image



All about Docker Volume

- Volume is simply a directory inside our container.
- Firstly, we have to declare this directory as a volume and then share the volume.
- Even if we stop the container, still we can access volume.
- The volume will be created in one container.
- You can declare a directory as a volume only while creating the container.
- You can't create volume from the existing container.
- You can share one volume across any number of containers.
- The volume will not be included when you update an image.
- We can map volume in two ways—

1. Container ← → Container
2. Host ← → Container



Benefits of Volume

- Decoupling container from storage.
- Share volume among different containers.
- Attach the volume to containers
- On deleting the container, the volume does not delete.

Creating volume from Dockerfile

</> Create a Dockerfile and write

```
FROM ubuntu
VOLUME ["/myvolume1"]
```

</> Then create an image from this Dockerfile.

```
docker build -t <image_name> .
```

</> Now create a container from this image & run

```
docker run -it --name
<container_name> <image_name>
/bin/bash
```

</> Now do ls, you can see *myvolume1*

</> Now share volume with another container [Container ← → Container]

```
docker run -it --name
<container_name> --privileged=true --
volumes-from <container_name>
<os_image_name_ex-ubuntu> /bin/bash
```

- Now after creating container2, myvolume1 is visible, whatever you do in one volume, can be seen from another volume.

Create volume by using the command

</> create volume by using the command

```
docker run -it --name
<container_name> -v /<volume_name>
      <os_name> /bin/bash
```

- Then do ls and change directory to your volume

</> Now create one more container and save volume

```
docker run -it --name <container_name> --
      privileged=true --volumes-from
    <container_name> <os_name> /bin/bash
```

- Now you are inside the container; do ls, and you can see your volume
- Now create one file inside this volume and then check in containers, you can see that file.

</> Volumes [Host ← → Container]

- Verify files in */home/ec2-user*
- Create volumes in host and container and mapped them

```
docker run -it --name <container_name> -v
/home/ec2-user:/<volume_name> --privileged=true
  <os_name> /bin/bash[Check your volume]--> cd
/<volume_name>[Do ls, now you can see all files
     of host machine]--> touch <file_name>
                --> exit
```

- Now check in ec2, you can see the files.

Some other commands

To see all created volumes

```
docker volume ls
```

To create docker volume (normal)

```
docker volume create <volume_name>
```

To delete volume

```
docker volume rm <volume_name>
```

To remove all unused docker volumes

```
docker volume prune
```

To get volume details

```
docker volume inspect <volume_name>
```

To get container details

```
docker container inspect
    <container_name>
```

Some other imp. Terms

The difference between docker *attach* and docker *exec*

- Docker exec creates a new process in the container's environment while *docker attaches* just connect the standard I/O of the main process inside the container to the corresponding standard I/O error of the current terminal.
- docker exec is especially for running new things in an already started container, be it a shell or some other process.

The difference between *expose* and *publish* a docker

We have three options:

1. Neither specifies *expose* nor *-p*
2. Only specify *expose*
3. Specify expose and *-p*

Explain

- If we specify neither expose nor -p the service in the container will only be accessible from inside the container itself.

- If we expose a port, the service in the container is not accessible from outside docker, but from inside other docker containers, so this is good for inter-container communication.

- If we expose and -p a port, the service in the container is accessible from anywhere, even outside docker.

Note

If we do *-p* but do not *expose*, docker does an implicit *expose*. This is because, if a port is open to the public, it is automatically also open to the other docker containers.

Thank You Dosto

TRAIN WITH SHUBHAM