----------Docker----------

Docker is a container management service. The keywords of Docker are develop, ship and run anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

The initial release of Docker was in March 2013 and since then, it has become the buzzword for modern world development, especially in the face of Agile-based projects.

Features of Docker

Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.

With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.

You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.

Since Docker containers are pretty lightweight, they are very easily scalable.

Components of Docker

Docker has the following components

Docker for Mac − It allows one to run Docker containers on the Mac OS.

Docker for Linux − It allows one to run Docker containers on the Linux OS.

Docker for Windows − It allows one to run Docker containers on the Windows OS.

Docker Engine − It is used for building Docker images and creating Docker containers.

Docker Hub − This is the registry which is used to host various Docker images.

Docker Compose − This is used to define applications using multiple Docker containers.

We will discuss all these components in detail in the subsequent chapters.

For installation of docker please go to docker official sites.

To start the docker :

service docker start ---> start the docker daemon

chkconfig docker on ---> to start docker daemon on boot. (in redhat)

systemctl enable docker ---> to start docker daemon on boot (in ubuntu)

service docker restart ---> restart the docker daemon

-> to give the access to particular user to use docker

usermod -a -G docker <username>

Commands in docker:

$ docker info

$ docker --version

LOGIN:

login to docker hub :

$ docker login

$ docker logout

Images:

$ docker images (to see the images available in the host)

$ docker images -q (in quite mode)

$ docker images -q --filter "dangling=true" ---> only dangling images

$ docker tag <imagename>:<tag> <imageid>

$ docker pull <imagename>:<tag> ---> (from dockerhub)

$ docker push <imagename>:<tag> ---> (to push images to dockerhub)

$ docker rmi <imagename> or <image id> -----> to remove particular images

$ docker rmi $(docker images -q) to delete all images

$ docker rmi -f forcely

$ docker rmi $(docker images -q --filter "dangling=true") remove dangling images

---

Containers process:

$ docker ps (to see the running containers list)

$ docker ps -a (to see the all containers list)

$ docker ps -aq (-q is in quite mode means only containers id)

Dealing with docker containers

$ docker run <image> -----> (to start a container with a image)

$ docker run -it --name <cont. name> -p xxxx:xxxx <imagename>

-p port forward

-it interactive mode

--name container name

$ docker run -d <imagename>

-d detached mode

docker rm <container id or container name> (to delete container)

docker rm -f forcely

docker rm $(docker ps -aq) -------> all container delete

docker rm $(docker ps -q) ----------> remove all running containers

---

docker start <container id or container name> ---> to start container

docker stop <container id> ----> to stop container

docker exec -it <container id or name> <shell> -----> to connect that container

----------DOCKER VOLUMES----------\

1. What are Volumes
2. How to create / list / delete volumes
3. How to attach volume to a container
4. How to share volume among containers
5. What are bind mounts

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

: docker volume //get information

: docker volume create

: docker volume ls

: docker volume inspect

: docker volume rm

: docker volume prune

Use of Volumes

==========

Decoupling container from storage

Share volume (storage/data) among different containers

Attach volume to container

On deleting container volume does not delete

Commands

docker run --name MyJenkins1 -v myvol1:/var/jenkins_home -p 8080:8080 -p 50000:50000 jenkins

docker run --name MyJenkins2 -v myvol1:/var/jenkins_home -p 9090:8080 -p 60000:50000 jenkins

docker run --name MyJenkins3 -v /Users/raghav/Desktop/Jenkins_Home:/var/jenkins_home -p 9191:8080 -p 40000:50000 jenkins

References

https://hub.docker.com/_/jenkins/

https://docs.docker.com/storage/volumes/

NOTES

By default all files created inside a container are stored on a writable container layer

The data doesn't persist when that container is no longer running

A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.

Docker has two options for containers to store files in the host machine

so that the files are persisted even after the container stops

VOLUMES and BIND MOUNTS

Volumes are stored in a part of the host filesystem which is managed by Docker

Non-Docker processes should not modify this part of the filesystem

Bind mounts may be stored anywhere on the host system

Non-Docker processes on the Docker host or a Docker container can modify them at any time

In Bind Mounts, the file or directory is referenced by its full path on the host machine.

Volumes are the best way to persist data in Docker

volumes are managed by Docker and are isolated from the core functionality of the host machine

A given volume can be mounted into multiple containers simultaneously.

When no running container is using a volume, the volume is still available to Docker and is not removed automatically. You can remove unused volumes using docker volume prune.

When you mount a volume, it may be named or anonymous.

Anonymous volumes are not given an explicit name when they are first mounted into a container

Volumes also support the use of volume drivers, which allow you to store your data on remote hosts or cloud providers, among other possibilities.

----------DOCKER_NETWORKS----------=

Docker networks creation commands

docker network ls ---> to list

docker network create --subnet 10.10.0.0/16 --gateway 10.10.0.1 --driver=bridge <network name>

docker network rm <network name>

docker network prune ---> to delete unused networks

docker network inspect <net. name>

to use the particvular network for container while running execute the following command:

docker run -itd --name web1 --net <network name> -p 80:80 nginx

---

Own image creation:

To create a image from an existing container

docker commit -t <imagename>:<tag> <container id>

you can push this image to your repo in dockerhub by typing following commands.

-> Change the name of the container to this format XXXXXXXXXXXXXXX/<imagename>:<tag>

docker push XXXXXXXXXXXXXX/imagename:tag

To create a own image by Dockerfile

1. create a dockerfile

RUN

Used to execute a command during the build process of the docker image.

ADD

Copy a file from the host machine to the new docker image. There is an option to use an URL for the file, docker will then download that file to the destination directory.

ENV

Define an environment variable.

CMD

Used for executing commands when we build a new container from the docker image.

ENTRYPOINT

Define the default command that will be executed when the container is running.

WORKDIR

This is directive for CMD command to be executed.

USER

Set the user or UID for the container created with the image.

VOLUME

Enable access/linked directory between the container and the host machine.

Now let's start to create our first dockerfile.

check the link below

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#create-ephemeral-containers

1. What is Dockerfile
2. How to create Dockerfile
3. How to build image from Dockerfile
4. Basic Commands

TIPS & TRICKS

Dockerfile :

A text file with instructions to build image

Automation of Docker Image Creation

FROM

RUN

CMD

Step 1 : Create a file named Dockerfile

Step 2 : Add instructions in Dockerfile

Step 3 : Build dockerfile to create image

Step 4 : Run image to create container

COMMANDS

: docker build

: docker build -t ImageName:Tag directoryOfDocekrfile

: docker run image

.....

..... DOCKER COMPOSE .....

https://docs.docker.com/compose/overview/

check the link for docker compose documents

(BASIC DOCKER-COMPOSE.yml is in github link)

1. What | Why - Docker Compose
2. How to install
3. How to create docker compose file
4. How to use docker compose file to create services
5. Basic Commands

TIPS

Docker compose

: tool for defining & running multi-container docker applications

: use yaml files to configure application services (docker-compose.yml)

: can start all services with a single command : docker compose up

: can stop all services with a single command : docker compose down

: can scale up selected services when required

Step 1 : install docker compose

(already installed on windows and mac with docker)

docker-compose -v

2 Ways

1. https://github.com/docker/compose/rel...

2. Using PIP

pip install -U docker-compose

Step 2 : Create docker compose file at any location on your system

docker-compose.yml

Step 3 : Check the validity of file by command

docker-compose config

Step 4 : Run docker-compose.yml file by command

docker-compose up -d

Steps 5 : Bring down application by command

docker-compose down

TIPS

How to scale services

—scale

docker-compose up -d --scale database=4

.....

.....Docker Swarm.....

Today we will learn :

1. What is Docker Swarm
2. Why to use it
3. How to create and manage Docker Swarm
4. Create service on docker swarm
5. Scaling services up and down
6. Features/Helpful Tips

A swarm is a group of machines that are running Docker and joined into a cluster

Docker Swarm is a tool for Container Orchestration

Let’s take an example

You have 100 containers

You need to do

- Health check on every container
- Ensure all containers are up on every system
- Scaling the containers up or down depending on the load
- Adding updates/changes to all the containers

Orchestration - managing and controlling multiple docker containers as a single service

Tools available - Docker Swarm, Kubernetes, Apache Mesos

Pre-requisites

1. Docker 1.13 or higher
2. Docker Machine (pre installed for Docker for Windows and Docker for Mac)https://docs.docker.com/machine/insta...

https://docs.docker.com/get-started/p...

Step 1 : Create Docker machines (to act as nodes for Docker Swarm) Create one machine as manager and others as workers

```
# docker-machine create --driver hyperv manager1
# docker-machine create --driver virtualbox manager1
```

docker-machine:Error with pre-create check: “exit status 126”

https://stackoverflow.com/questions/3...

FOR MAC # brew cask install virtualbox;

Create one manager machine

and other worker machines

Step 2 : Check machine created successfully

docker-machine ls

docker-machine ip manager1

Step 3 : SSH (connect) to docker machine

docker-machine ssh manager1

Step 4 : Initialize Docker Swarm

docker swarm init --advertise-addr MANAGER_IP

docker node ls

(this command will work only in swarm manager and not in worker)

Step 5 : Join workers in the swarm

Get command for joining as worker

In manager node run command

docker swarm join-token worker

This will give command to join swarm as worker

docker swarm join-token manager

This will give command to join swarm as manager

-> SSH into worker node (machine) and run command to join swarm as worker

In Manager Run command - docker node ls to verify worker is registered and is ready

Do this for all worker machines

Step 6 : On manager run standard docker commands

docker info

check the swarm section

no of manager, nodes etc

Now check docker swarm command options

#docker swarm

Step 7 : Run containers on Docker Swarm

docker service create --replicas 3 -p 80:80 --name serviceName nginx

Check the status:

docker service ls

docker service ps serviceName

Check the service running on all nodes

Check on the browser by giving ip for all nodes

Step 8 : Scale service up and down

On manager node

docker service scale serviceName=2

Inspecting Nodes (this command can run only on manager node)

docker node inspect nodename

docker node inspect self

docker node inspect worker1

Step 9 : Shutdown node

```
# docker node update --availability drain worker1
```

Step 10 : Update service

```
# docker service update --image imagename:version web
# docker service update --image nginx:1.14.0 serviceName
```

Step 11 : Remove service

```
# docker service rm serviceName
```

docker swarm leave : to leave the swarm

---

to stop the docker machine

```
# docker-machine stop machineName : to stop the machine
# docker-machine rm machineName : to remove the machine
```

.....