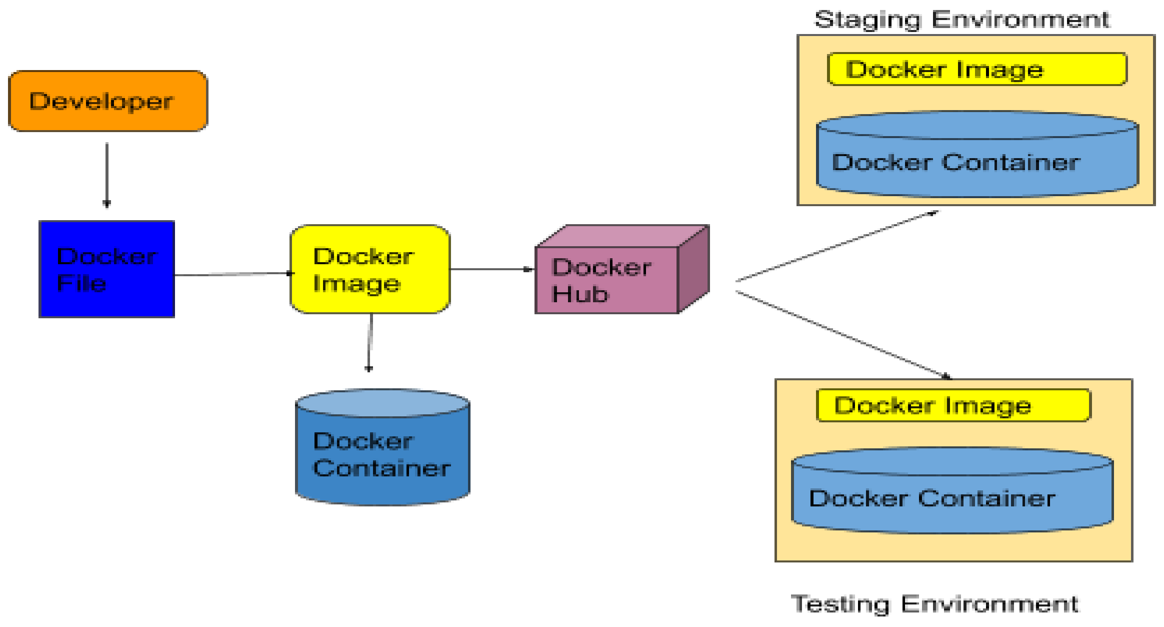
Docker

Docker is a tool designed to make it easier to deploy and run applications by using containers.

Containers allow developers to package up an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Docker is an open source project that automates the deployment of applications inside software containers.

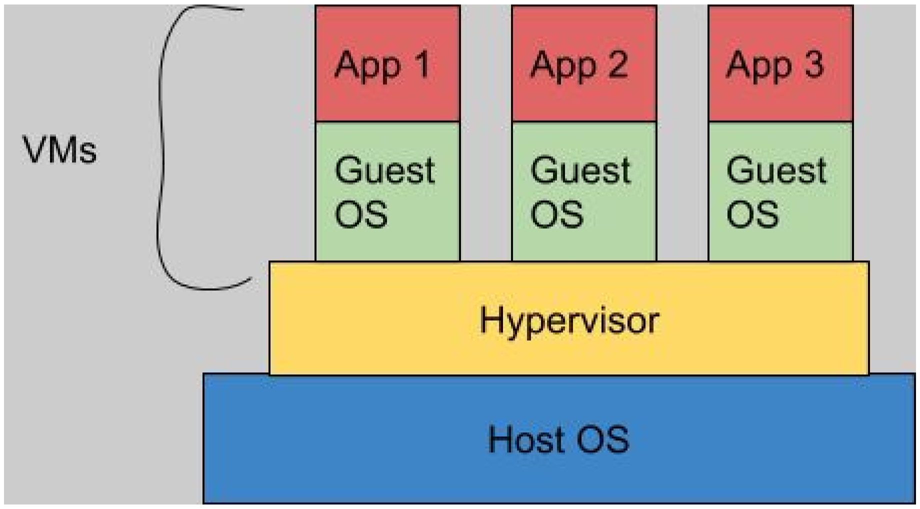
How Docker Works?



Note : Docker is a container Platform.

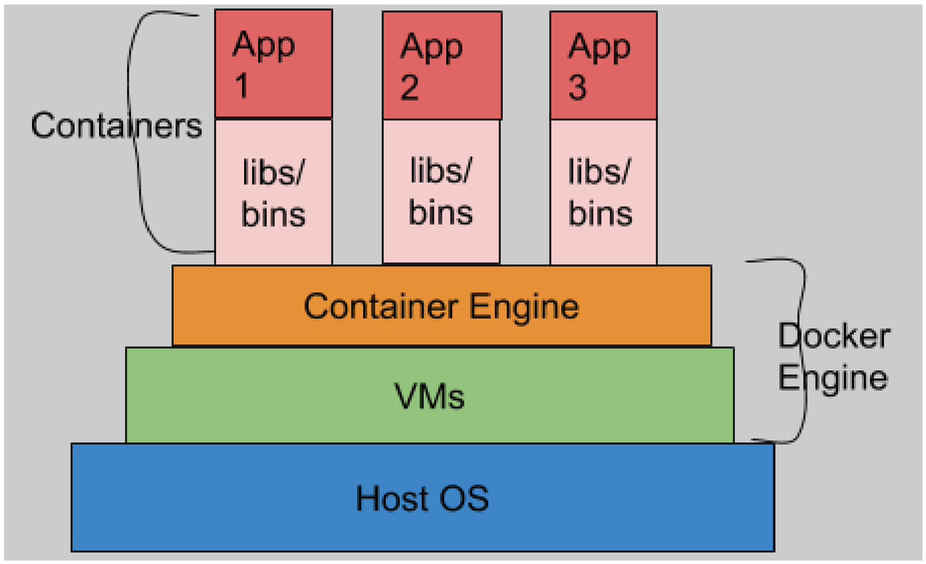
<u>**Virtualization vs Containerization :**</u>

Virtualization

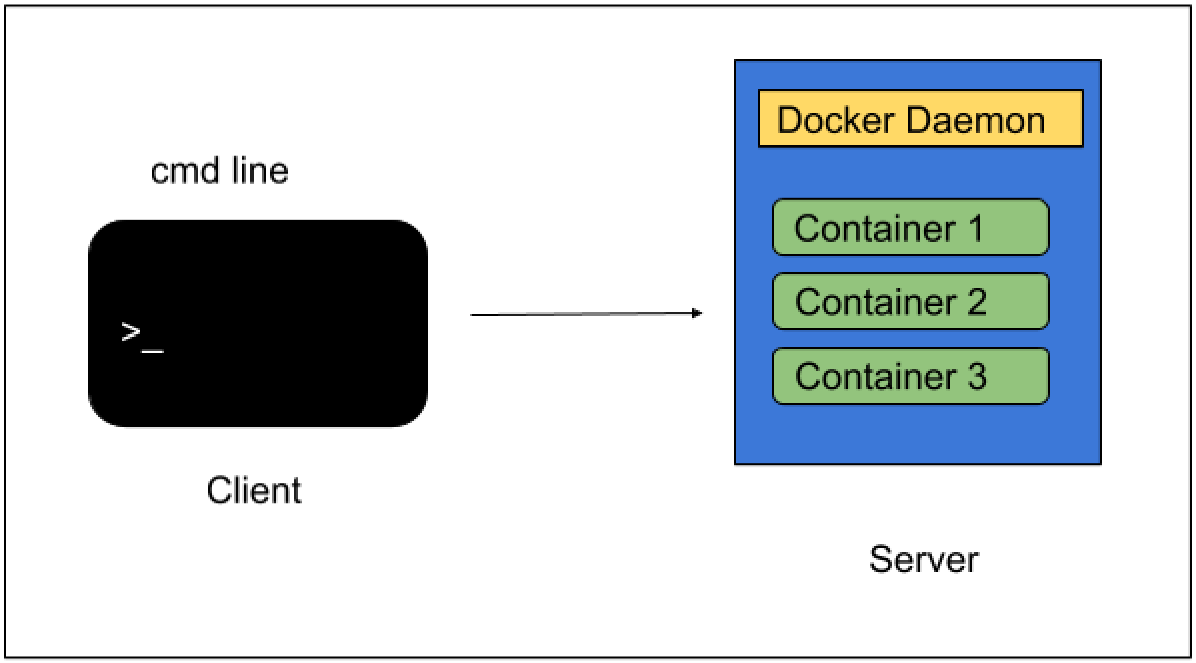


Note: VMs - Resource allocation is fixed and does not change as per application needs.

Containerization



Docket has a client-server architecture.



Docker Engine

- The Daemon (Server) receives the commands from the Docker client through CLI or REST API's.
- Docker client and Daemon can be present on the same host(machine) or different host.

<u>Terminologies we covered:</u>
DockerFile
Docker Images
Docker Containers
Docker Hub/Registry
Docker Client
Docker Server(daemon)
Docker Engine

Benefits of Docker - Why to use Docker?

1. Build an app only once.

   An application inside a container can run on any system that has Docker installed. So there is no need to build and configure app multiple times on different platforms.
2. With Docker you test your application inside a container and ship it inside a container. This means the environment in which you test is identical to one on which the app will run in production.
3. Portability
   - Docker containers can run on any platform. It can run on your local system, Amazon EC2, Google Cloud Platform, Rackspace server, VirtualBox,etc.
4. Version Control
   - Like Git, Docker has in-built version control system.
   - Docker containers work just like Git repositories, allowing you to commit changes to your Docker images and version control them.
5. Isolation
   - With Docker every application works in isolation in its own container and does not interfere with other applications running on the same system.
   - So multiple containers can run on same system without interference.
   - For removal also you can simply delete the container and it will not leave behind any files on traces on the system.
6. Productivity
   - Docker allows faster and more efficient deployments without worrying about running your app on different platforms.
   - It increases productivity many folds.

Docker Installation:

Stable Vs Edge Version Install stable version

- Windows
    - Docker on Windows: Pro and Enterprise
        - https://docs.docker.com/docker-for-windows/install/
    - Docker Toolbox for Win7/8 and Home edition
        - https://docs.docker.com/toolbox/overview/
    - Visual Studio Code:
        - https://code.visualstudio.com/
    - Install cmder from cmder.net for better terminal (Optional)
        - https://cmder.net/
            - Launch commander
                - docker-machine env default
                - Run the command to configure your shell
- Mac
    - https://hub.docker.com/editions/community/docker-ce-desktop-mac
        - Install bash completion (install homebrew if not installed already)
        - https://docs.docker.com/docker-for-mac/#install-shell-completion à follow these commands

Docker Commands

Basic Commands

```
>docker version
>docker -v
>docker --help
```

Images Commands

```
>docker images
>docker run --help
>docker login
>docker pull ubuntu
>docker images
>docker images -q
>docker rmi --help
>docker rmi fce289e99eb9
>docker rmi -f fce289e99eb9
```

Container Commands

```
>docker ps -a
>docker run ubuntu
>docker ps -a
>docker run -it ubuntu
>docker start 228c0a9f3056
>docker stop 228c0a9f3056
```

System Commands:

```
>docker run ubuntu
>docker stats
>docker system df
>docker system prune
>docker system prune -a
```

Docker Images

What are images?

Docker images are templates used to create Docker Containers.
Container is a running instance of image.
Images are stored at Registries(e.g. docker hub)

How to pull image?

```
>docker pull ubuntu
>docker pull ubuntu:18.04
>docker images --help
>docker images
>docker images -q
>docker image -f “dangling=false/true”
>docker image -f “dangling=false/true” -q
```

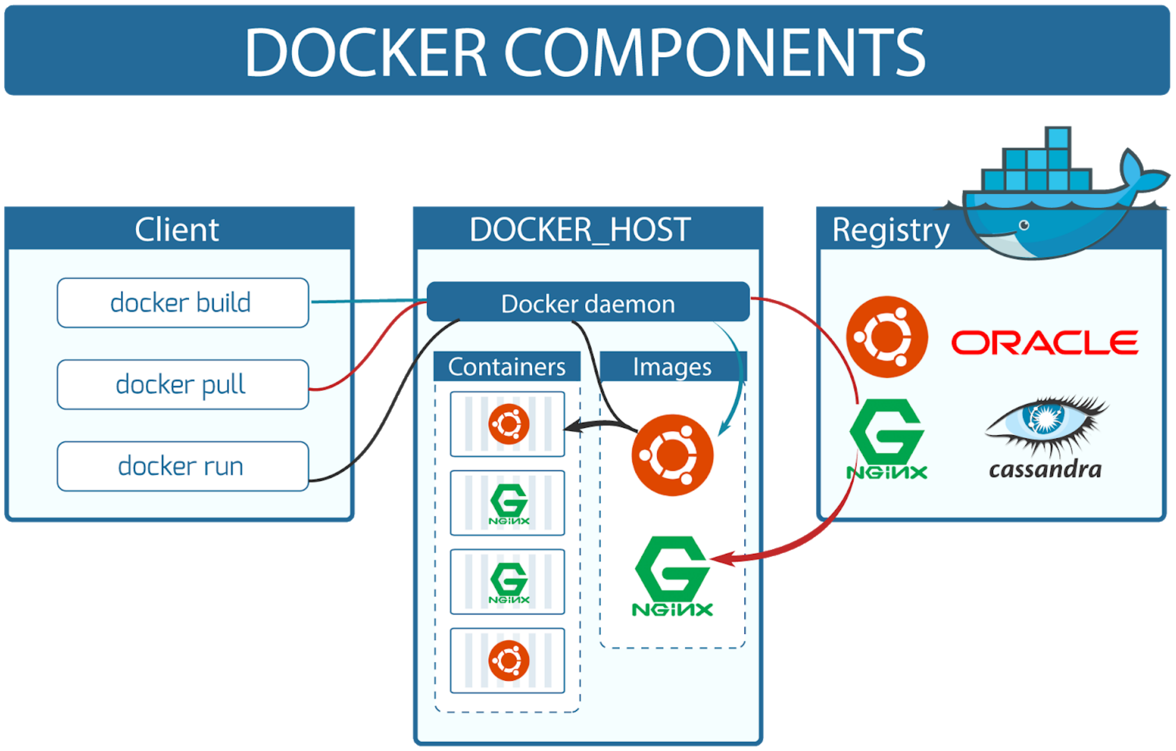
How to run a container using an image?

```
>docker run ubuntu
>docker ps
>docker ps -a
>docker run --name MyUbuntu -it ubuntu bash
>docker ps
>docker inspect ubuntu
>docker images
>docker rmi ubuntu
>docker images
>docker stop MyUbuntu/ubuntu
```

Docker Containers

What are Docker Containers?

Containers are the running instances of the Docker images.



Commands:
>docker ps → give you the container info that are running
>docker run --name MyUbuntu -it ubuntu --->pull and run the ubuntu image and name it MyUbuntu and run the container in interactive mode
>docker start MyUbuntu → will start the container..go into the container
>docker stop MyUbuntu → will stop the container
>docker pause MyUbuntu
>docker unpause MyUbuntu

>docker top MyUbuntu → You can see the top processes of that container
>docker stats MyUbuntu→ You can see the memory usage ,i/o of the container
>docker attach MyUbuntu→ to execute the container and go inside it
>docker kill MyUbuntu→ to kill the container if its running
>docker rm MyUbuntu→ to remove the container
>docker history image-name → to get the history of that image

Docker file

Simple text file with instructions to build image. It is basically an automation of image creation.

Step 1: Create a file named dockerfile
> mkdir docker
> cd docker
> touch dockerfile

#getting base image (if want to create new → use SCRATCH instead of ubuntu)
FROM ubuntu

MAINTAINER Anjali Tandel(anjalitandel52@gmail.com)

RUN apt-get-update

CMD [“echo”,”hello-world”]

Difference between RUN and CMD is: RUN will be executed while building the images.It is basically used for installing dependencies or upgrading the application. However, CMD will be executed while creating and running container from the image.

Step 2: Building image

```
>docker build -t myimage1 Users/Anjali/Desktop/docker/ .
...Step 1:
...Step 2:
...Step 3:
...Successfully built
>docker images
```

Step 3: Run the Image to create a container

```
>docker run imageID
....
...
Hello world
```

<u>Docker Compose</u>

- Tool for defining and running multi-container docker applications.
- We can use yaml files to configure application services (docker-compose.yml).
- We can start all services with a single command : docker compose up.
- We can stop all services with a single command : docker compose down.
- We can scale up selected services when required.

=>To check the docker-compose version

```
$docker-compose -v
```

=>To install docker-compose

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

2. Using PIP

```
pip install -U docker-compose
```

=>Create a docker-compose file at any location on your system(docker-compose.yml)

```
$mkdir compose
$cd /dockercomposefile/
$touch docker-compose.yml
$vim docker-compose.yml
```

```
Version ‘3’

Services:
  Web:
        Image:nginx
        Ports:
    -   8080:80

  Database:
        Image:mysql
```

=> check the validity of file by command,

```
$docker-compose config
```

=> run docker-compose.yml file by command,

```
$docker-compose up -d
$docker ps
```

=> To stop the containers

```
    $docker-compose down
```

=> How to Scale services

```
    $docker-compose up -d --scale database=4
    $docker ps
```

Docker Volumes

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

=>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

```
$docker volume --help
```

=>create a volume

```
$docker volume create myvolume1
```

=>list all the volumes

```
$docker volume ls
```

=>to get the details about volume

```
$docker volume inspect myvolume1
```

=>to remove volume

```
$docker volume rm myvolume1
```

=>remove all the unused volume
$docker volume prune

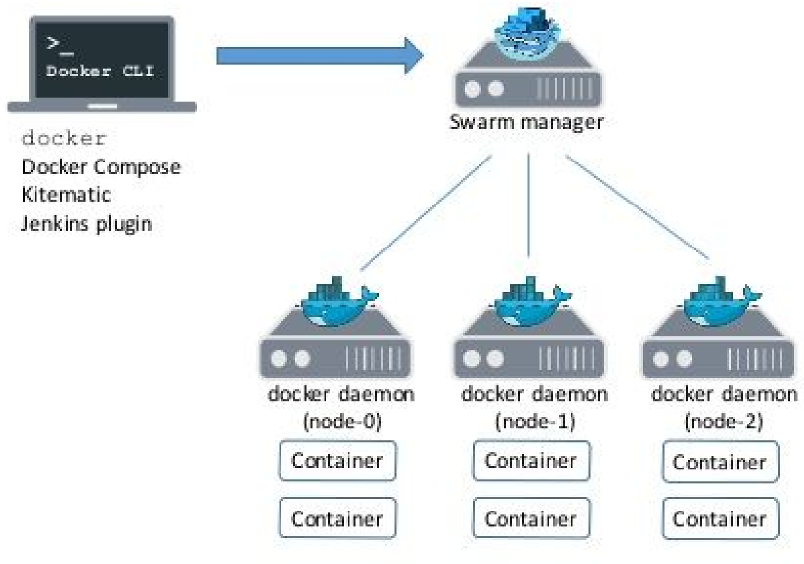
=>the whole process of creating a volume for particular container
$docker pull jenkins
$docker run --name myjenkins1 **-v myvol1:/var/jenkins_home** -p 8080:80 -p 50000:50000 jenkins

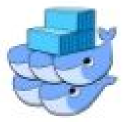
=>to share the volume data between two containers(data is being shared among two containers)
$docker run --name myjenkins2 -v myvol1:/var/jenkins_home -p 9090:80 -p 40000:50000 jenkins
$Check both the ports.

=>To store the data on absolute path of your system
$docker run --name myjenkins3 -v / Users / Anjali / Desktop / Jenkins_home : /var/jenkins_home -p 9191:80 -p 30000:50000 jenkins

Docker Swarm

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





Docker swarm is a tool for container orchestration.

Example : You have 100 containers.
So you need to do,

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

Orchestration: managing and controlling multiple docker containers as a single service tool 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...

=> Check docker machine version
$docker-machine version

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 manager
```

If any error:
docker-machine:Error with pre-create check: "exit status 126"
https://stackoverflow.com/questions/3...
brew cask install virtualbox;

Step 2 : Check machine created successfully

```
docker-machine ls
docker-machine ip manager1(to get the machine ip address)
```

Step 3 : SSH (connect) to docker machine

```
$docker-machine ssh manager1
$docker -machine ssh worker1
```

Step 4 : Initialize Docker Swarm (run this command on manager)

```
$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(192.168.99.101)

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

REFERENCES:

https://docs.docker.com/get-started/p...
https://rominirani.com/docker-swarm-t...