

Dockers and Containers

Virtualization:

Virtualization is the ability to run multiple operating systems on a single physical system and share the underlying hardware resources.

It is the process by which one server hosts the appearance of many computers.

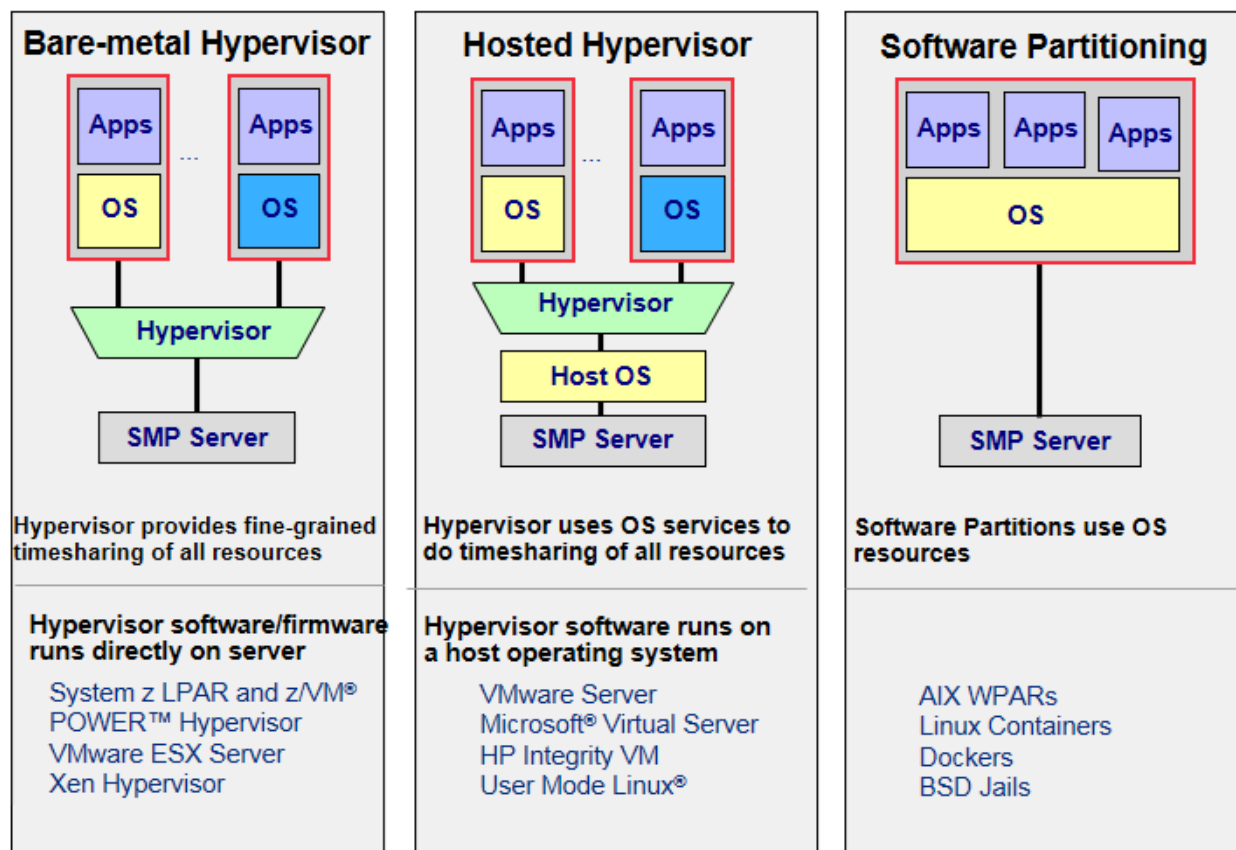
Virtualization is used to improve IT throughput and costs by using physical resources as a pool from which virtual resources can be allocated.

Different types of virtualizations:

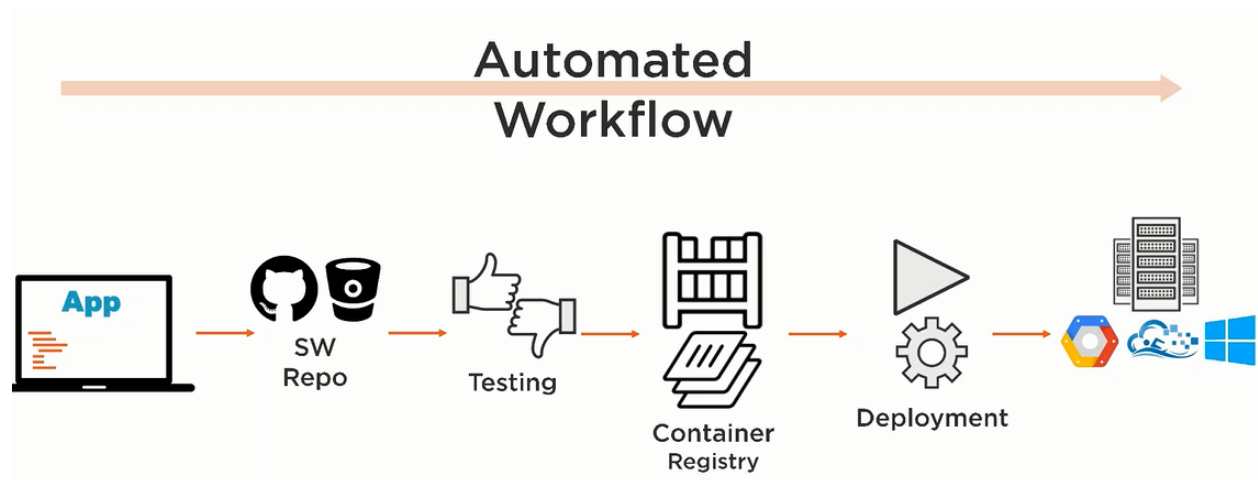
Bare metal hypervisor ex: Vmware ESX, IBM PowerVM, Xen

Hosted Hypervisor ex: KVM

Software partitioning: Dockers, WPARs



SMP – Symmetric multi processing



LXC (Linux Containers) is an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel

Uses kernel features calls cgroups (Linux Kernel control groups)– that allows limitation and prioritization of resources like CPU/memory, /i/o, network

Namespace isolation – that allows complete isolation of an application view of Operating system environment including process trees. networking ,user ids and mounted file system

LXC combines kernel's cgroups and support for isolated namespaces to provide an isolated environment for applications

Note: LXC is a Linux technology. It does not work on Windows or Mac.

Dockers are useful in:

To avoid having too much software on the machine

To try deploying a web app

To try provisioning scripts on fake machine first then try on actual machine

Dockers leverage LXC – Lightweight alternative to full virtualization such as provided by traditional hypervisors like KVM VMware Xen or ESXi

Installing docker:

Steps for RHEL:

1. Execute below command on the node

```
sudo yum-config-manager --add-repo  
https://docs.docker.com/engine/installation/linux/repo\_files/centos/docker.repo
```

```
sudo yum -y install docker-engine
```

```
sudo systemctl start docker
```

```
service docker status
```

Ensure the below file is created in /etc/yum.repos.d

```
cat /etc/yum.repos.d/docker.repo
```

```
root@reviewb ~]# cd /etc/yum.repos.d/
```

```
[root@reviewb yum.repos.d]# ls
```

```
iso.repo
```

```
[root@reviewb yum.repos.d]# cat /docker
```

```
cat: /docker: Is a directory
```

```
[root@reviewb yum.repos.d]# cat /docker.repo
```

```
[docker-main]
```

```
name=Docker Repository
```

```
baseurl=https://yum.dockerproject.org/repo/main/centos/7/
```

```
enabled=1
```

```
gpgcheck=1
```

```
gpgkey=https://yum.dockerproject.org/gpg
```

```
[docker-testing]
```

```
name=Docker Repository
```

```
baseurl=https://yum.dockerproject.org/repo/testing/centos/$releasever/
```

```
enabled=0
```

```
gpgcheck=1
```

```
gpgkey=https://yum.dockerproject.org/gpg
```

```
[docker-beta]
```

```
name=Docker Repository
```

```
baseurl=https://yum.dockerproject.org/repo/beta/centos/7/
```

```
enabled=0
```

```
gpgcheck=1
```

```
gpgkey=https://yum.dockerproject.org/gpg
```

[docker-nightly]

name=Docker Repository

baseurl=https://yum.dockerproject.org/repo/nightly/centos/7/

enabled=0

gpgcheck=1

gpgkey=https://yum.dockerproject.org/gpg

docker --help

--config=~/.docker Location of client config files

-D, --debug Enable debug mode

-H, --host=[] Daemon socket(s) to connect to

-h, --help Print usage

-l, --log-level=info Set the logging level

--tls Use TLS; implied by --tlsverify

--tlscacert=~/.docker/ca.pem Trust certs signed only by this CA

--tlscert=~/.docker/cert.pem Path to TLS certificate file

--tlskey=~/.docker/key.pem Path to TLS key file

--tlsverify Use TLS and verify the remote

-v, --version Print version information and quit

Commands:

attach Attach to a running container

build Build an image from a Dockerfile

commit Create a new image from a container's changes

cp Copy files/folders between a container and the local filesystem

create Create a new container

diff Inspect changes on a container's filesystem

events Get real time events from the server

exec Run a command in a running container

export Export a container's filesystem as a tar archive

history Show the history of an image

images List images

import Import the contents from a tarball to create a filesystem image

info Display system-wide information

inspect Return low-level information on a container, image or task

kill Kill one or more running container

load Load an image from a tar archive or STDIN

login Log in to a Docker registry.

logout Log out from a Docker registry.

logs Fetch the logs of a container

network Manage Docker networks

node Manage Docker Swarm nodes

pause Pause all processes within one or more containers

port List port mappings or a specific mapping for the container

ps List containers

pull Pull an image or a repository from a registry

push Push an image or a repository to a registry

rename Rename a container

restart Restart a container

rm Remove one or more containers

rmi Remove one or more images

run Run a command in a new container

save Save one or more images to a tar archive (streamed to STDOUT by default)

search Search the Docker Hub for images

service Manage Docker services

start Start one or more stopped containers

stats Display a live stream of container(s) resource usage statistics

stop Stop one or more running containers

swarm Manage Docker Swarm

tag Tag an image into a repository

top Display the running processes of a container

unpause Unpause all processes within one or more containers

update Update configuration of one or more containers

version Show the Docker version information

volume Manage Docker volumes

wait Block until a container stops, then print its exit code

Run 'docker COMMAND --help' for more information on a command.

Steps:

1. Install docker using yum

yum install docker

2. download the image using docker pull

docker pull ubuntu

docker pull rhel6

3. List dockers available

docker ps → to list running containers

docker ps -a → lists all containers including killed ones

4. start a docker using

docker run -t -i ubuntu

5. To stop a docker

docker stop <container_id>

6. to start a docker

docker start <container_id>

7.to login

docker attach <>

Commands:

Docker pull is to pull images

Docker ps to display running containers

Docker ps -a to display all running containers including the dead ones

Docker inspect is to show all values

Docker stop/start – is to stop or start a container

Docker attach/detach is to attach or detach to a container

Docker run is to run the container, if image is not available docker pulls from docker hub repository

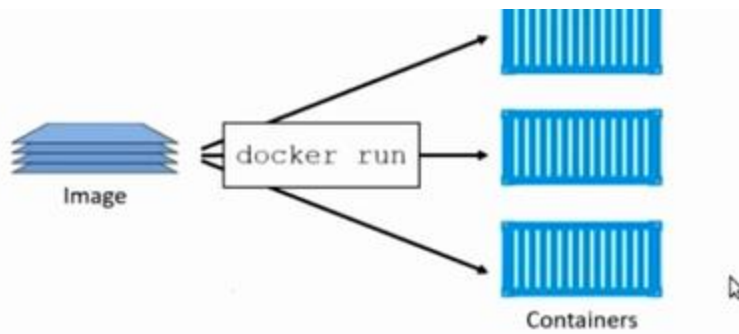


Figure 6.1

Examples:

```
[root@rscthydnet1 ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	1967d889e07f	2 weeks ago	167.9 MB

➔ ***sudo docker run -t -i ubuntu:latest*** ➔ downloads Ubuntu docker

checks local machine first, if not available downloads the image from net

```
[root@rscthydnet1 LINUX]# sudo docker run -t -i ubuntu
```

Unable to find image 'ubuntu:latest' locally

latest: Pulling from ubuntu

0847857e6401: Pull complete

f8c18c152457: Pull complete

8643975d001d: Pull complete

d5802da4b3a0: Pull complete

fe172ed92137: Pull complete

Digest: sha256:5349f00594c719455f2c8e6f011b32758dcd326d8e225c737a55c15cf3d6948c

Status: Downloaded newer image for ubuntu:latest

```
root@d47c9e2cbf15:/#
```

it will login to docker

```
root@d47c9e2cbf15:/# id
```

```
uid=0(root) gid=0(root) groups=0(root)
```

```
root@d47c9e2cbf15:/# cat /etc/*release
```

```
DISTRIB_ID=Ubuntu
```

```
DISTRIB_RELEASE=16.04
```

```
DISTRIB_CODENAME=xenial
```

```
DISTRIB_DESCRIPTION="Ubuntu 16.04.1 LTS"
```

```
NAME="Ubuntu"
```

```
VERSION="16.04.1 LTS (Xenial Xerus)"
```

```
ID=ubuntu
```

```
ID_LIKE=debian
```

```
PRETTY_NAME="Ubuntu 16.04.1 LTS"
```

```
VERSION_ID="16.04"
```

```
HOME_URL="http://www.ubuntu.com/"
```

```
SUPPORT_URL="http://help.ubuntu.com/"
```

```
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
```

```
VERSION_CODENAME=xenial
```

```
UBUNTU_CODENAME=xenial
```

```
root@d47c9e2cbf15:/#
```

We can download the docker and load it

```
docker load -i centos_docker_image.tar
```

run apt-get update to work with apt-get

```
root@d47c9e2cbf15:/# apt-get update
```

```
Get:1 http://ports.ubuntu.comubuntu-ports xenial InRelease [247 kB]
```

```
Get:2 http://ports.ubuntu.comubuntu-ports xenial-updates InRelease [95.7 kB]
```

```
Get:3 http://ports.ubuntu.comubuntu-ports xenial-security InRelease [94.5 kB]
```

```
Get:4 http://ports.ubuntu.comubuntu-ports xenial/main Sources [1103 kB]
```

```
Get:5 http://ports.ubuntu.comubuntu-ports xenial/restricted Sources [5179 B]
```

```
Get:6 http://ports.ubuntu.comubuntu-ports xenial/universe Sources [9802 kB]
```

```
Get:7 http://ports.ubuntu.comubuntu-ports xenial/main x86_64el Packages [1470 kB]
```

```
Get:8 http://ports.ubuntu.comubuntu-ports xenial/universe x86_64el Packages [9485 kB]
```

```
Get:9 http://ports.ubuntu.comubuntu-ports xenial-updates/main Sources [256 kB]
```

```
Get:10 http://ports.ubuntu.comubuntu-ports xenial-updates/restricted Sources [1872 B]
```

```
Get:11 http://ports.ubuntu.comubuntu-ports xenial-updates/universe Sources [136 kB]
```

```
Get:12 http://ports.ubuntu.comubuntu-ports xenial-updates/main x86_64el Packages [487 kB]
```

```
Get:13 http://ports.ubuntu.comubuntu-ports xenial-updates/universe x86_64el Packages [395 kB]
```

```
Get:14 http://ports.ubuntu.comubuntu-ports xenial-security/main Sources [54.7 kB]
```

```
Get:15 http://ports.ubuntu.comubuntu-ports xenial-security/restricted Sources [1872 B]
```

```
Get:16 http://ports.ubuntu.comubuntu-ports xenial-security/universe Sources [13.8 kB]
```

```
Get:17 http://ports.ubuntu.comubuntu-ports xenial-security/main x86_64el Packages [171 kB]
```

```
Get:18 http://ports.ubuntu.comubuntu-ports xenial-security/universe x86_64el Packages [55.7 kB]
```

```
Fetch: 23.9 MB in 1min 2s (382 kB/s)
```

Reading package lists... Done

Install packages

apt-get install vim

apt-get install openssh

apt-get install vim

apt-get install iputils-ping

apt-get install net-tools

apt-get install ksh

to come out of docker without stopping it control+pq

docker run – every time it creates a new container.

to get the container id

root@rscthydnet1 ~]# docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					

docker stop container_ID

docker start container_ID

docker run -d --name ubuntu-docker1 ubuntu

[root@rscthydnet1 ~]# docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
20354b3035a3	ubuntu	"/bin/bash"	6 seconds ago	Up 3 seconds	
romantic_booth					

```
docker attach 20354b3035a3
```

```
[root@rscthydnet1 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
20354b3035a3 romantic_booth	ubuntu	"/bin/bash"	8 minutes ago	Up 18 seconds	

```
[root@rscthydnet1 ~]# docker stop 20354b3035a3
```

```
20354b3035a3
```

```
[root@rscthydnet1 ~]# docker start 20354b3035a3
```

```
20354b3035a3
```

```
[root@rscthydnet1 ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
20354b3035a3 romantic_booth	ubuntu	"/bin/bash"	9 minutes ago	Up 57 seconds	
5f3e16925382 admiring_montalcini	ubuntu	"/bin/bash"	9 minutes ago	Exited (127) 9 minutes ago	
24eedbddd026 ubuntu-docker1	ubuntu	"/bin/bash"	11 minutes ago	Exited (0) 11 minutes ago	
d47c9e2cbf15 sleepy_elion	ubuntu	"/bin/bash"	3 hours ago	Exited (127) 11 minutes ago	

```
[root@rscthydnet1 ~]#
```

```
[root@rscthydnet1 ~]# docker attach d47c9e2cbf15
```

You cannot attach to a stopped container, start it first

```
[root@rscthydnet1 ~]# docker start d47c9e2cbf15
```

```
d47c9e2cbf15
```

```
[root@rscthydnet1 ~]#
```

```
[root@rscthydnet1 ~]# docker attach d47c9e2cbf15
```

```
root@d47c9e2cbf15:/#
```

```
root@d47c9e2cbf15:/#
```

```
root@d47c9e2cbf15:/#
```

```
root@d47c9e2cbf15:/# ls -lrt
```

```
total 828
```

```
drwxr-xr-x.  2 root root   6 Apr 12 2016 home
```

```
drwxr-xr-x.  2 root root   6 Apr 12 2016 boot
```

```
drwxr-xr-x. 10 root root 97 Nov  1 08:20 usr
```

```
drwxr-xr-x.  2 root root   6 Nov  1 08:20 srv
```

```
drwxr-xr-x.  2 root root   6 Nov  1 08:20 opt
```

```
drwxr-xr-x.  2 root root   6 Nov  1 08:20 mnt
```

```
drwxr-xr-x.  2 root root   6 Nov  1 08:20 media
```

```
drwxr-xr-x.  2 root root  22 Nov  1 08:21 lib64
```

```
drwxr-xr-x. 11 root root 4096 Nov  1 08:21 var
```

```
dr-xr-xr-x. 12 root root   0 Nov 14 04:07 sys
```

```
drwxr-xr-x. 2 root root 6 Nov 16 08:25 rsctfvt
drwxr-xr-x. 9 root root 4096 Nov 16 08:31 lib
-rw-r-----. 1 root root 809234 Nov 16 08:34 sg3_utils-1.41-3.fc24.x86_64.rpm
drwxrwxrwx. 2 root root 8192 Nov 16 08:36 _linux_2
drwxr-xr-x. 5 root root 74 Nov 16 08:38 run
drwxr-xr-x. 2 root root 4096 Nov 16 08:40 bin
drwxr-xr-x. 2 root root 4096 Nov 16 08:40 sbin
drwxr-xr-x. 72 root root 4096 Nov 16 08:45 etc
drwxrwxrwt. 2 root root 6 Nov 16 08:45 tmp
drwx-----. 4 root root 94 Nov 16 10:32 root
dr-xr-xr-x. 583 root root 0 Nov 16 10:44 proc
drwxr-xr-x. 5 root root 380 Nov 16 10:44 dev
root@d47c9e2cbf15:/#
root@d47c9e2cbf15:/#
root@d47c9e2cbf15:/#
```

to make container running always

```
docker run -d <docker name> sh /script.sh
```

this script,sh should be a never ending loop

something like:

```
while true
```

```
do
```

```
sleep 1
```

done

Port forwarding:

Start a container from host

And we can map the host and container's ports

Container 's app can be accessed from host machine using port forwarding concept

Example:

Create a docker and do a mapping between host port 8080 to container 8080

docker run -p 8080:8080 jenkins:latest

docker run -d -p 8080:8080 jenkins:latest -> to run in the background

docker rm <name of the container> to remove the container, first stop the container and remove.

Detaching/attaching – we can do on a running container

Attaching/detaching a shell

Stop/start – stoping a service .

Docker rmi – to remove image

Docker pull xyz:latest

Docker run -it xyz /bin/bash

Install software

Apt-get update

Apt-get install <>

Customize the image

Exit the container

To create the image to push it to Docker Hub:

```
docker commit -m "customized image name" -a "authorname"  
<containerid> <nameofaccount>/<nameof the image>:tag
```

```
docker commit -m "ubuntupython" -a  
"santosh" f56be454c884  
santoshdevops/ubuntupython:v1
```

sha256:7f093647f35a39965d101859c269da63884fef22bc574829786408513efbe7b8

```
docker commit -m "jenkinscustomzed3" -a "santosh" fd4e1be10bd2  
santoshdevops/jenkinscustomzed3:v200
```

docker login

Note:

We need to create a login in docker hub, and use the same for pushing images

Give account name and password

```
Docker push nameofdockerimage
```

```
docker push santoshdevops/ubuntupython
```

then try to delete the image from local machine

and try to pull same image

Docker File:

To automate the image creation

Create docker file

touch Dockerfile

Add the instructions to file

Command	Description
---------	-------------

ADD	Copies a file from the host system onto the container
-----	---

CMD	The command that runs when the container starts
-----	---

ENTRYPOINT	
------------	--

ENV	Sets an environment variable in the new container
-----	---

EXPOSE	Opens a port for linked containers
--------	------------------------------------

FROM	The base image to use in the build. This is mandatory and must be the first command in the file.
------	--

MAINTAINER	An optional value for the maintainer of the script
------------	--

ONBUILD	A command that is triggered when the image in the Docker file is used as a base for another image
---------	---

RUN	Executes a command and save the result as a new layer
-----	---

USER	Sets the default user within the container
------	--

VOLUME	Creates a shared volume that can be shared among containers or by the host machine
--------	--

WORKDIR	Set the default working directory for the container
---------	---

cat Dockefile

FROM ubuntu:latest

MAINTAINER Visualpath

RUN apt-get update && apt-get install -y ruby

Docker build -t newdockerimage:v3 .

Creates image

Do all the stuff

Create a new container

Remove the old container

ADD

CMD similar to run,

Run gets executed while building

CMD gets executed while running the container

USER to set user id

VOLUME – attach a directory from a host machine

WORKDIR – launch into default dir

Docker bridge ip 172.17.0.1 – advanced switch

This is the gateway for containers

Example of building image from Dockerfile:

docker build -t myimage_t .

docker run --name my_first_instance -t myimage_t

To stop all the docker containers

docker stop \$(docker ps -a -q)

to remove all the docker images

docker rm \$(docker ps -a -q)

Docker images are stored in - /var/lib/docker/devicemapper/

(based on the device used, here device used is devicemapper)

Can we directly copy the image from one machine to other?

Yes:

save the docker image as a tar file:

docker save -o <save image to path> <image name>

Then copy your image to a new system with regular file transfer tools such as cp or scp. After that you will have to load the image into docker: docker load -i <path to image tar file>

docker load -i <path to image tar file>

Can we control the cpu /memory running for a container?

Yes:

If we have 2 containers, one for the database and one more for the web server

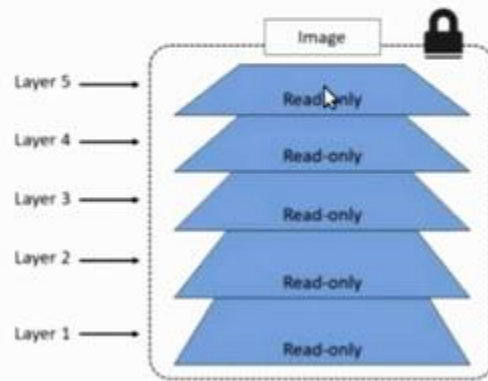
sudo docker run -c 614 -dit --name db postgres /postgres.sh

sudo docker run -c 410 -dit --name web nginx /nginx.sh

Will give 60% to the db container (614 is 60% of 1024) and 40% to the web container.

Images and layers

All Docker images are made up of one or more read-only *layers* as shown below.



Docker image has layers

If any data is added, after a container is created on top of image it will create a new layer.

Every layer will have id.

Docker inspect is command to give info about docker.

To share volumes between 2 containers:

```
[root@reviewb ~]# cd /vol2
```

```
[root@reviewb vol2]# ls -lrt
```

```
total 4
```

```
-rw-r--r--. 1 root root 92 Mar  4 10:10 Dockerfile
```

```
[root@reviewb vol2]# cat Dockerfile
```

FROM ubuntu:latest

RUN mkdir /myvol

RUN echo "hello world" > /myvol/greeting

VOLUME /myvol

docker build -t vol2 . ➔ build the image with directory /myvol

docker run -it vol2 /bin/bash ➔ creates container1

docker run -it --volumes-from <1st container id> ubuntu:latest bash ➔ creates container2 with shared directory /myvol

file /myvol will be shared across 2 dockers

Sharing file between Host and Container:

on host:

docker volume create --name DataVolume1 --> creates a docker volume

docker run -ti --rm -v DataVolume1:/datavolume1 ubuntu --> creates container, upon exit docker will be deleted

on docker:

echo "Example1" > /datavolume1/Example1.txt

on host:

docker volume inspect DataVolume1

output

```
[
  {
    "Name": "DataVolume1",
    "Driver": "local",
    "Mountpoint": "/var/lib/docker/volumes/datavolume1/_data",
    "Labels": null,
    "Scope": "local"
  }
]
```

we can start a new docker with same data volume

```
docker run --rm -ti -v DataVolume1:/datavolume1 ubuntu
```

to create volume that persists when container is removed.

```
docker run -ti --name=Container2 -v DataVolume2:/datavolume2 ubuntu
```

Docker logs:

On RHEL

```
cat /var/log/messages | grep docker
```

or

```
journalctl -u docker.service
```

Docker logs based on OS:

Ubuntu (old using upstart) - /var/log/upstart/docker.log

Ubuntu (new using systemd) - journalctl -u docker.service

Boot2Docker - /var/log/docker.log

Debian GNU/Linux - /var/log/daemon.log

CentOS - /var/log/daemon.log | grep docker

CoreOS - journalctl -u docker.service

Fedora - journalctl -u docker.service

Red Hat Enterprise Linux Server - /var/log/messages | grep docker

OpenSuSE - journalctl -u docker.service

OSX - ~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/log/docker.log

Database Use case:

```
docker run --name db -d -e  
MYSQL_ROOT_PASSWORD=123 -p 3306:3306  
mysql:latest
```

```
docker ps
```

```
docker exec -it db /bin/bash
```

```
mysql -uroot -p123
```

```
show databases;
```

```
exit
```


steps to build image:

1. docker pull ubuntu ==> downloads image
2. docker run -it ubuntu => creates container, logs into container
3. do customization, apt-get update, apt-get install python
4. exit from the container using exit
5. docker commit
6. docker login (need to sign up with docker hub first)
7. docker push

this can be automated using dockerfile

docker build -t <image name> .

Create nginx container using docker image:

docker run --name mynginx-P -d nginx

creates a container with name mynginx and runs in the detached mode

container will be running until it is stopped.

Nginx exposes the ports 80 and 443 in the container

-P tells dockers to map those ports to ports in the docker host.

Nginx (pronounced as engine-x) is open source software for web serving, caching load balancing.

Its designed for max performance and stability

Apache webserver is slow, developed in 1995, there are some issues with respect to multi tasking

Nginx was developed in 2004.

FROM nginx

RUN rm /etc/nginx/conf.d/default.conf

RUN rm /etc/nginx/conf.d/openssl.conf

COPY content /usr/share/nginx/html

COPY conf /etc/nginx

Create the image by running

docker build -t mynginximage1 .

nginx content will be from

/usr/share/nginx/html

Dockerfile to create image with customized code:

FROM nginx

RUN rm /usr/share/nginx/html/index.html

COPY content /usr/share/nginx/html

Docker Swarm:

docker swarm is cluster of machines running docker which provides scalable and reliable platform to run many containers.

Manages all the containers.

Single faced manager where user can interact with.

Docker swarm is for

load balancing: load is balanced across the containers/machines

high availability/fail-over: Webserver will be up, even if one of the node goes down

reliability: load will be shared even if the node goes down.

How to convert 4 machines to docker swarm machines:

Pre-req:

Machines are running with Docker version 1.12

Machines are in the same subnet

With ports 2377,4789 7946 open.

Login to 1st machine and make it manager:

docker swarm init --listen-addr 192.168.0.104:2377 --advertise-addr 192.168.0.104

docker node ls → to check nodes in the swarm.

Login to 2nd, 3rd, 4th nodes and make them workers

docker swarm join --token SWMTKN-1-0fywejkjqfpvmivezc5biirlqe1dkaxqbdiuv3642v4jsdkbs-0lamc59edwsqzdmqkh1gtnrwj 192.168.0.104:2377

docker node ls → will show 4 nodes

Running service on swarm:

Swarm manages individual containers, we work at higher level

Service is abstract created on top of the containers.

Service is independent function/task/application running inside the container.

Create a service with nginx server:

`docker service create --name website --publish 80:80 sixeyed/docker-swarm-walkthrough`

make it more scalable:

scale up to 10 instances:

`docker service inspect website --pretty`

`docker service update --replicas 20 website`

containers can run on manager or workers

`docker service list`

even without running the container, using second node we can access website.

Reliability:

If the node goes down, manager knows, node list shows node as down.

The containers will be moved to another node.

Dockerfile example:

```
#select the base image

FROM ubuntu

#author name

MAINTAINER ubuntu

#copies the contents from docker host to docker container

ADD /host /var/www/html

# execute a command

RUN apt-get update

RUN apt-get install -y python

# define environment variable

ENV var 1

# expose a port

EXPOSE 80

#mount

VOLUME ["/host1"]
```

Example2:

```
FROM ubuntu

ENTRYPOINT ["/bin/ping"]

CMD ["localhost"]
```

`docker run -i -t test → pings local host`

`docker run -i -t test google → pings google`

The ENTRYPOINT specifies a command that will always be executed when the container starts.

The CMD specifies arguments that will be fed to the ENTRYPOINT.

To mount a directory from host to container

`docker run -i -t -v /host1:/mnt new:v1 "bin/bash"`