ec2-user

sudo su –

<https://www.adaptivecomputing.com/blog-hpc/an-introduction-to-basic-docker-commands/>

what is docker?

**Docker** is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package

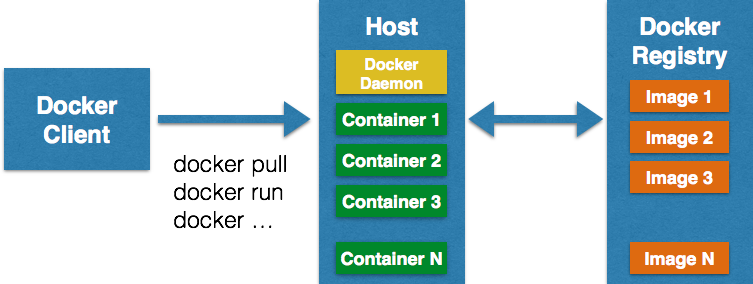
**Docker** hub acts like git, where you can build your images locally on your laptop, commit it and then can be pushed to the **Docker** hub. Container is the execution environment for **Docker**. Containers are created from images. It is a writable layer of the image.

**Docker Components:**

Docker is composed of following four components

1. Docker Client and Daemon.
2. Images
3. Docker registries
4. Containers

Docker has a client-server architecture. Docker Daemon or server is responsible for all the actions that are related to containers. The daemon receives the commands from the Docker client through CLI or REST API’s. Docker client can be on the same host as a daemon or it can be present on any other host.



mages are the basic building blocks of Docker. Containers are built from images. Images can be configured with applications and used as a template for creating containers. It is organized in a layered fashion. Every change in an image is added as a layer on top of it.

**Docker registry** is a repository for Docker images. Using Docker registry, you can build and share images with your team. A registry can be public or private. Docker Inc provides a hosted registry service called Docker Hub. It allows you to upload and download images from a central location. If your repository is public, all your images can be accessed by other Docker hub users. You can also create a private registry in Docker Hub. Docker hub acts like git, where you can build your images locally on your laptop, commit it and then can be pushed to the Docker hub.

**Container** is the execution environment for Docker. Containers are created from images. It is a writable layer of the image. You can package your applications in a container, commit it and make it a golden image to build more containers from it. Two or more containers can be linked together to form tiered application architecture. Containers can be started, stopped, committed and terminated. If you terminate a container without committing it, all the changes made to the container will be lost.

**Commands and scenarios**

yum install -y docker

service docker start

docker version

Client:

Version: 18.06.1-ce

API version: 1.38

Go version: go1.10.3

Git commit: e68fc7a215d7133c34aa18e3b72b4a21fd0c6136

Built: Fri Jun 28 23:16:08 2019

OS/Arch: linux/amd64

Experimental: false

Server:

Engine:

Version: 18.06.1-ce

API version: 1.38 (minimum version 1.12)

Go version: go1.10.3

Git commit: e68fc7a/18.06.1-ce

Built: Fri Jun 28 23:17:39 2019

OS/Arch: linux/amd64

Experimental: false

docker info => Get detailed information about docker installed on the system including the kernel version, number of containers and images, etc.

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Server Version: 18.06.1-ce

Storage Driver: overlay2

Backing Filesystem: extfs

Supports d\_type: true

Native Overlay Diff: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

Volume: local

Network: bridge host macvlan null overlay

Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: 468a545b9edcd5932818eb9de8e72413e616e86e

runc version: 69663f0bd4b60df09991c08812a60108003fa340

init version: fec3683

Security Options:

seccomp

Profile: default

Kernel Version: 4.14.123-86.109.amzn1.x86\_64

Operating System: Amazon Linux AMI 2018.03

OSType: linux

Architecture: x86\_64

CPUs: 1

Total Memory: 985.8MiB

Name: ip-172-31-36-110

ID: QKWM:SQ2G:CSNR:K2NT:5GCR:B5H4:MGLA:NESG:VU2E:B553:6N4G:4STF

Docker Root Dir: /var/lib/docker

Debug Mode (client): false

Debug Mode (server): false

Registry: https://index.docker.io/v1/

Labels:

Experimental: false

Insecure Registries: 127.0.0.0/8

Live Restore Enabled: false

docker pull hello-world

Using default tag: latest

latest: Pulling from library/hello-world

1b930d010525: Pull complete

Digest: sha256:6540fc08ee6e6b7b63468dc3317e3303aae178cb8a45ed3123180328bcc1d20f

Status: Downloaded newer image for hello-world:latest

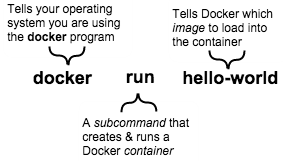
docker images => name of the image which is used to run the container

REPOSITORY TAG IMAGE ID CREATED SIZE

hello-world latest fce289e99eb9 7 months ago 1.84kB

docker run hello-world => download the **hello-world** image, if it is not already present, and run

the **hello-world** as a container.



Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

docker ps=> list all of the currently running containers on the machine

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

docker ps -a => list all of the containers on the system

**─a** − It tells the **docker ps** command to list all of the containers on the system.

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

bf4f2f3164db hello-world "/hello" 12 min ago Exited (0) dazzling\_euler

12 minutes ago

docker rm bf4f2f3164db => This command is used to delete a container.

docker images => This command is used to display all the images currently installed on the system.

REPOSITORY TAG IMAGE ID CREATED SIZE

hello-world latest fce289e99eb9 7 months ago 1.84kB

docker history ImageID

docker history fce289e99eb9 =>you can see all the commands that were run with an image via a container.

IMAGE CREATED CREATED BY SIZE COMMENT

fce289e99eb9 7 months ago /bin/sh -c #(nop) CMD ["/hello"] 0B

<missing> 7 months ago /bin/sh -c #(nop) COPY file:f77490f70ce51da2… 1.84kB

docker run ubuntu:14.04 echo "Hello world"

Unable to find image 'ubuntu:14.04' locally

14.04: Pulling from library/ubuntu

a7344f52cb74: Pull complete

515c9bb51536: Pull complete

e1eabe0537eb: Pull complete

4701f1215c13: Pull complete

Digest: sha256:2f7c79927b346e436cc14c92bd4e5bd778c3bd7037f35bc639ac1589a7acfa90

Status: Downloaded newer image for ubuntu:14.04

Hello world

docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu 14.04 2c5e00d77a67 3 months ago 188MB

hello-world latest fce289e99eb9 7 months ago 1.84kB

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

f9a88357cff2 ubuntu:14.04 "echo 'Hello world'" 6 min ago Exited (0) 6 min ago affectionate\_varahamihira

docker start f9a88357cff2

docker ps

docker rm f9a88357cff2

docker images

docker run ubuntu:14.04 ps ax

PID TTY STAT TIME COMMAND

1 ? Rs 0:00 ps ax

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

13fc794668bd ubuntu:14.04 "ps ax" 55 seconds ago Exited (0) 54 seconds ago trusting\_jennings

46042671ea40 ubuntu:14.04 "ps ax" About minute ago Exited(0)About a minute ago peaceful\_yalow

docker rm 13fc794668bd

Container with terminal

docker run -i -t ubuntu:14.04

Where -i flag tells docker to connect to STDIN on the container

-t flag specifies to get a pseudo-terminal

useradd devops

id devops

apt-get update

apt-get install curl

exit

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

4b04b7feffe2 ubuntu:14.04 "/bin/bash" 2 minutes ago Exited (0) 32 seconds ago determined\_chaplygin

docker start 4b04b7feffe2

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

4b04b7feffe2 ubuntu:14.04 "/bin/bash" 4 minutes ago Up 15 seconds determined\_chaplygin

docker attach 4b04b7feffe2

id devops

ctrl +p +q => image is still running, but out of the container

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

4b04b7feffe2 ubuntu:14.04 "/bin/bash" 9 minutes ago Up 5 minutes determined\_chaplygin

ps -ef | grep docker

where -f options tells ps to provide detailed and full information about each process ,

-e all processes

docker attach 4b04b7feffe2

exit

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

4b04b7feffe2 ubuntu:14.04 "/bin/bash" 12 min ago Exited (9) 13 seconds ago determined\_chaplygin

docker rm 4b04b7feffe2

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu 14.04 2c5e00d77a67 3 months ago 188MB

The **location** of the images vary depending on the driver **Docker** is using for storage. Actually, **Docker** images are **stored** in two **files** as shown by following command. /var/lib/**docker**/aufs/diff/<id> has the **file** contents of the images. /var/lib/**docker**/repositories-aufs is a JSON **file** containing local image information.

CONTAINER ID

2 IDs – Long ID and Short ID

docker run -i -t ubuntu:14.04

exit

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

b4565f411467 ubuntu:14.04 "/bin/bash" 40 seconds ago Exited (0) 16 seconds ago priceless\_banach

Short ID

docker inspect b4565f411467

Long ID

[

{

"Id": "b4565f41146761368eac5ba94bbfe96814302d26ab6cc5245117e9cd87f04330",

"Created": "2019-08-21T00:26:59.928203336Z",

"Path": "/bin/bash",

"Args": [],

"State": {

"Status": "exited",

"Running": false,

"Paused": false,

"Restarting": false,

"OOMKilled": false,

"Dead": false,

"Pid": 0,

"ExitCode": 0,

"Error": "",

"StartedAt": "2019-08-21T00:27:00.30725437Z",

"FinishedAt": "2019-08-21T00:27:22.167247893Z"

},

"Image": "sha256:2c5e00d77a67934d5e39493477f262b878f127b9c01b491f06d8f06f78819578",

"ResolvConfPath": "/var/lib/docker/containers/b4565f41146761368eac5ba94bbfe96814302d26ab6cc5245117e9cd87f04330/resolv.conf",

"HostnamePath": "/var/lib/docker/containers/b4565f41146761368eac5ba94bbfe96814302d26ab6cc5245117e9cd87f04330/hostname",

"HostsPath": "/var/lib/docker/containers/b4565f41146761368eac5ba94bbfe96814302d26ab6cc5245117e9cd87f04330/hosts",

"LogPath": "/var/lib/docker/containers/b4565f41146761368eac5ba94bbfe96814302d26ab6cc5245117e9cd87f04330/b4565f41146761368eac5ba94bbfe96814302d26ab6cc5245117e9cd87f04330-json.log",

"Name": "/priceless\_banach",

"RestartCount": 0,

"Driver": "overlay2",

"Platform": "linux",

"MountLabel": "",

"ProcessLabel": "",

"AppArmorProfile": "",

"ExecIDs": null,

"HostConfig": {

"Binds": null,

"ContainerIDFile": "",

"LogConfig": {

"Type": "json-file",

"Config": {}

},

"NetworkMode": "default",

"PortBindings": {},

"RestartPolicy": {

"Name": "no",

"MaximumRetryCount": 0

},

"AutoRemove": false,

"VolumeDriver": "",

"VolumesFrom": null,

"CapAdd": null,

"CapDrop": null,

"Dns": [],

"DnsOptions": [],

"DnsSearch": [],

"ExtraHosts": null,

"GroupAdd": null,

"IpcMode": "shareable",

"Cgroup": "",

"Links": null,

"OomScoreAdj": 0,

"PidMode": "",

"Privileged": false,

"PublishAllPorts": false,

"ReadonlyRootfs": false,

"SecurityOpt": null,

"UTSMode": "",

"UsernsMode": "",

"ShmSize": 67108864,

"Runtime": "runc",

"ConsoleSize": [

0,

0

],

"Isolation": "",

"CpuShares": 0,

"Memory": 0,

"NanoCpus": 0,

"CgroupParent": "",

"BlkioWeight": 0,

"BlkioWeightDevice": [],

"BlkioDeviceReadBps": null,

"BlkioDeviceWriteBps": null,

"BlkioDeviceReadIOps": null,

"BlkioDeviceWriteIOps": null,

"CpuPeriod": 0,

"CpuQuota": 0,

"CpuRealtimePeriod": 0,

"CpuRealtimeRuntime": 0,

"CpusetCpus": "",

"CpusetMems": "",

"Devices": [],

"DeviceCgroupRules": null,

"DiskQuota": 0,

"KernelMemory": 0,

"MemoryReservation": 0,

"MemorySwap": 0,

"MemorySwappiness": null,

"OomKillDisable": false,

"PidsLimit": 0,

"Ulimits": [

{

"Name": "nofile",

"Hard": 4096,

"Soft": 1024

}

],

"CpuCount": 0,

"CpuPercent": 0,

"IOMaximumIOps": 0,

"IOMaximumBandwidth": 0,

"MaskedPaths": [

"/proc/acpi",

"/proc/kcore",

"/proc/keys",

"/proc/latency\_stats",

"/proc/timer\_list",

"/proc/timer\_stats",

"/proc/sched\_debug",

"/proc/scsi",

"/sys/firmware"

],

"ReadonlyPaths": [

"/proc/asound",

"/proc/bus",

"/proc/fs",

"/proc/irq",

"/proc/sys",

"/proc/sysrq-trigger"

]

},

"GraphDriver": {

"Data": {

"LowerDir": "/var/lib/docker/overlay2/94e9bdf76684c3148d428b99c9daace92d7b4bf8cf2a424200ed453f051c7e81-init/diff:/var/lib/docker/overlay2/2c8549a507b319e142aed501bd67c188009b2330be008d30f60812b3bdcf47f0/diff:/var/lib/docker/overlay2/07726752c43c73f4a3e74659f97627b53fe7d5f4c5c32e32936139dea379fb4a/diff:/var/lib/docker/overlay2/6157d8878d87a3b26d7cdf0ef82049bebcffa15ab3b4b48a559f04411e0b969d/diff:/var/lib/docker/overlay2/3fdf9b851a2efaf35df1a350a9201ffa36ee556a4ad661613d8db453a3325114/diff",

"MergedDir": "/var/lib/docker/overlay2/94e9bdf76684c3148d428b99c9daace92d7b4bf8cf2a424200ed453f051c7e81/merged",

"UpperDir": "/var/lib/docker/overlay2/94e9bdf76684c3148d428b99c9daace92d7b4bf8cf2a424200ed453f051c7e81/diff",

"WorkDir": "/var/lib/docker/overlay2/94e9bdf76684c3148d428b99c9daace92d7b4bf8cf2a424200ed453f051c7e81/work"

},

"Name": "overlay2"

},

"Mounts": [],

"Config": {

"Hostname": "b4565f411467",

"Domainname": "",

"User": "",

"AttachStdin": true,

"AttachStdout": true,

"AttachStderr": true,

"Tty": true,

"OpenStdin": true,

"StdinOnce": true,

"Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

],

"Cmd": [

"/bin/bash"

],

"ArgsEscaped": true,

"Image": "ubuntu:14.04",

"Volumes": null,

"WorkingDir": "",

"Entrypoint": null,

"OnBuild": null,

"Labels": {}

},

"NetworkSettings": {

"Bridge": "",

"SandboxID": "bd998ab16de9eefe2a04bf1b2e8e8f3cef3831d4c1807220645de6cdf1e4177a",

"HairpinMode": false,

"LinkLocalIPv6Address": "",

"LinkLocalIPv6PrefixLen": 0,

"Ports": {},

"SandboxKey": "/var/run/docker/netns/bd998ab16de9",

"SecondaryIPAddresses": null,

"SecondaryIPv6Addresses": null,

"EndpointID": "",

"Gateway": "",

"GlobalIPv6Address": "",

"GlobalIPv6PrefixLen": 0,

"IPAddress": "",

"IPPrefixLen": 0,

"IPv6Gateway": "",

"MacAddress": "",

"Networks": {

"bridge": {

"IPAMConfig": null,

"Links": null,

"Aliases": null,

"NetworkID": "46131385031c8d01f2b504f4ccf19556d730ee79d0d951db9abf288a6fadd3a0",

"EndpointID": "",

"Gateway": "",

"IPAddress": "",

"IPPrefixLen": 0,

"IPv6Gateway": "",

"GlobalIPv6Address": "",

"GlobalIPv6PrefixLen": 0,

"MacAddress": "",

"DriverOpts": null

}

}

}

}

]

docker run -d centos:7 ping 127.0.0.1 -c 30

where -d is detach

-c number of times to be executed i.e. 30 times

To run a Docker container in the **background**, use the use **-d=true** or just **-d** option. First, stop it from the **foreground mode** by pressing **[Ctrl+C]**

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

c5a570ebc911 centos:7 "ping 127.0.0.1 -c 30" 29 seconds ago Up 28 seconds pensive\_cray

docker logs -f pensive\_cray

64 bytes from 127.0.0.1: icmp\_seq=26 ttl=255 time=0.025 ms

64 bytes from 127.0.0.1: icmp\_seq=27 ttl=255 time=0.025 ms

64 bytes from 127.0.0.1: icmp\_seq=28 ttl=255 time=0.027 ms

64 bytes from 127.0.0.1: icmp\_seq=29 ttl=255 time=0.025 ms

64 bytes from 127.0.0.1: icmp\_seq=30 ttl=255 time=0.026 ms

--- 127.0.0.1 ping statistics ---

30 packets transmitted, 30 received, 0% packet loss, time 29686ms

rtt min/avg/max/mdev = 0.015/0.025/0.035/0.006 ms

where –f To follow docker log output or tail continuously

docker ps

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

c5a570ebc911 centos:7 "ping 127.0.0.1 -c 30" 2 mins ago Exited (0) About a min ago pensive\_cray

b4565f411467 ubuntu:14.04 "/bin/bash" 9 mins ago Exited (0) 9 minutes ago

priceless\_banach

docker rm c5a570ebc911 b4565f411467

docker ps

docker images

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

[root@ip-172-31-13-157 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

centos 7 67fa590cfc1c 4 hours ago 202MB

ubuntu 14.04 2c5e00d77a67 3 months ago 188MB

docker rmi 67fa590cfc1c 2c5e00d77a67

docker images

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

Images

docker run -it centos:7

yum install -y httpd

exit

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ac302644adf6 centos:7 "/bin/bash" 2 minutes ago Exited (0) 25 seconds ago hopeful\_mahavira

docker commit ac302644adf6 centos/httpd:2.2

docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

centos/httpd 2.2 10c4146827b0 24 seconds ago 346MB

centos 7 67fa590cfc1c 2 days ago 202MB

docker rm ac302644adf6

docker run -it centos/httpd:2.2

whereis httpd

exit

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

9c3ce5b75d90 centos/httpd:2.2 "/bin/bash" 3 minutes ago Exited (0) 3 seconds ago jovial\_ardinghelli

docker rm 9c3ce5b75d90

# Dockerfile

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image. Using **docker** build users can create an automated build that executes several command-line instructions in succession. This page describes the commands you can use in a **Dockerfile,**

The following points need to be noted about the above file −

* The first line "#This is a sample Image" is a comment. You can add comments to the Docker File with the help of the **#** command
* The next line has to start with the **FROM** keyword. It tells docker, from which base image you want to base your image from. In our example, we are creating an image from the **ubuntu** image.
* The next command is the person who is going to maintain this image. Here you specify the **MAINTAINER** keyword and just mention the email ID.
* The **RUN** command is used to run instructions against the image. In our case, we first update our Ubuntu system and then install the nginx server on our **ubuntu** image.
* The last command is used to display a message to the user.

vi Dockerfile

FROM centos:7

RUN yum install -y httpd

RUN yum install -y tomcat

docker build -t apatom:1.0 . where . is path i.e. build context , image is build using current path, t stands for tag

builds the file and downloads httpd , tomcat

docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

apatom 1.0 865a02557f92 16 seconds ago 661MB

centos/httpd 2.2 10c4146827b0 27 minutes ago 346MB

centos 7 67fa590cfc1c 2 days ago 202MB

CMD Instruction

vi Dockerfile

FROM centos:7

CMD ping 127.0.0.1 -c 30

docker build -t apatom:1.1 .

docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

apatom 1.1 d19562bb6659 30 seconds ago 202MB

apatom 1.0 865a02557f92 9 minutes ago 661MB

centos/httpd 2.2 10c4146827b0 36 minutes ago 346MB

centos 7 67fa590cfc1c 2 days ago 202MB

docker run apatom:1.1

docker ps

docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

9690a39d5b72 apatom:1.1 "/bin/sh -c 'ping 12…" 52 seconds ago Exited (0) 22 seconds ago vigilant\_einstein

65e6c64a7115 apatom:1.1 "/bin/sh -c 'ping 12…" 2 minutes ago Exited (0) About a minute ago agitated\_heisenberg

586cd0a6490a e24847bd8e09 "/bin/sh -c 'yum ins…" 16 minutes ago Exited (1) 16 minutes ago zen\_goodall

docker rm $(docker ps -a -q) Delete all containers that are not currently running.



mkdir test

cd test

vi Dockerfile

FROM ubuntu:14.04

RUN apt-get update

RUN apt-get install –y curl

RUN apt-get install –y vim

docker build –t testimage:1.0 .

docker images

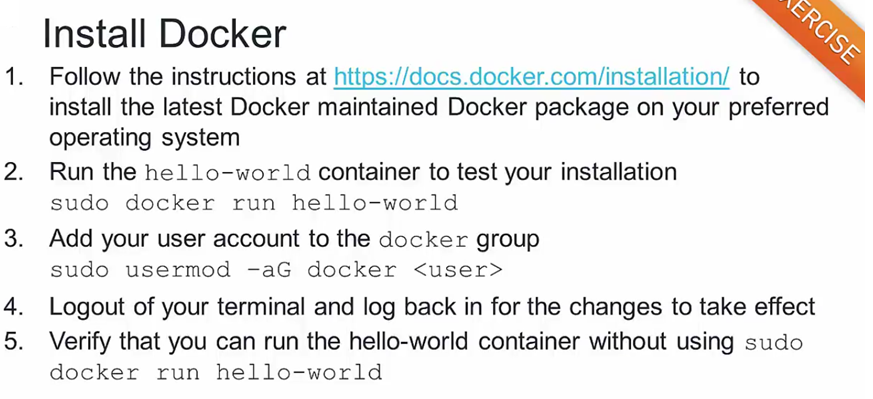
docker run -it testimage:1.0

whereis curl

whereis vim

exit

docker ps-a



ENTRY POINT

In Dockerfiles there are two commands that look similar to me: CMD and**ENTRYPOINT** . ... The main purpose of a CMD is to provide defaults for an executing container. and for **ENTRYPOINT** : An **ENTRYPOINT** helps you to configure a container that you can run as an executable.

The **ENTRYPOINT** instruction works very similarly to CMD in that it is **used** to specify the command executed when the container is started. However, where it differs is that **ENTRYPOINT** doesn't allow you to override the command. Instead, anything added to the end of the **docker** run command is appended to the command.

vi Dockerfile

FROM centos:7

ENTRYPOINT [“Ping”]

docker build -t apatom:1.2 .

docker run apatom:1.2 google.co.in

docker run apatom:1.2 google.co.in -c 10

Getting Terminal Access –Bin/Bash

docker run -d -p 8080:8080 tomcat:7

docker ps

docker exec -it ff4 /bin/bash

exit => does not terminate the container

docker ps

Removing Images

docker images

docker rmi apatom:1.0

docker ps

docker rmi tomcat:7

Conflict as tomcat:7 is running

docker rm $(docker ps -a -q) => remove all the containers

error as one of the container is running

docker ps

docker stop ff4 => stop the running container

docker rm $(docker ps -a -q)

docker rmi tomcat:7

Push images to Hub Repository

docker images

docker tag a2a15febcdf3 raghuopsdev/ubuntu:1.0= > renaming the image before pushing to the Docker hub

docker images

docker login

username

password

docker push raghuopsdev/ubuntu:1.0

Volumes

mkdir data

cd data/

touch abc.txt

pwd

/home/ec2-user/data

ll

docker run -it -v /home/ec2-user/data:/data centos:7

df –h

cd /data/

ll

touch welcome.txt

exit

df –h

ps -ef | grep docker

docker ps

docker ps –a

docker rm ba59a280dd70

ps -ef | grep docker

vi Dockerfile

FROM ubuntu:14.04

RUN apt-get update

RUN apt-get install -y nginx

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]

docker build -t nginx:1.0 .

docker images

docker run -d -p 80:80 nginx:1.0 /bin/bash

docker ps

curl ifconfig.co

Open the website to confirm nginx is running

ps -ef | grep docker

docker ps

docker attach 7116d8add578

ps -ef | grep nginx

/etc/init.d/n

/etc/init.d/nginx stop

ps -ef | grep nginx

vi Dockerfile

FROM centos:7

RUN yum install -y httpd

CMD ["/usr/sbin/httpd", "-k", "start"]

docker build -t apache:1.0 .

docker images

docker run -d -p 80:80 centos:7

docker ps

docker stop 183ef48d1888

docker ps -a

docker rm 183

docker images

docker run -d -p 80:80 apache:1.0

docker ps

docker ps -a

vi Dockerfile

FROM centos:7

RUN yum install -y httpd

CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]

docker build -t apache:1.0 .

docker run -d -p 80:80 apache:1.0

docker ps

refresh the web page

ps -ef | grep docker

docker ps

docker stop 0b4301cbd42b

docker ps -a

docker rm $(docker ps -a -q)

Linking containers

docker run -it --name source centos:7

df –h

ctrl+P+Q

docker ps

docker inspect 7ac55aca0c06 | grep IPAddress

"SecondaryIPAddresses": null,

"IPAddress": "172.17.0.2",

"IPAddress": "172.17.0.2",

docker inspect 7ac55aca0c06

docker ps

docker run -it --name rcpt --link source:src centos:7

ping src

PING src (172.17.0.2) 56(84) bytes of data.

ctrl+P+Q

docker ps

docker attach rcpt

cat /etc/hosts

127.0.0.1 localhost

172.17.0.2 src 7ac55aca0c06 source

172.17.0.3 fdc0bd12d5fa

Ctrl+P+Q =>read escape sequence

docker ps

docker attach source

ping rcpt

ping: rcpt: Name or service not known

cat /etc/hosts

ping 172.17.0.3

Ctrl+P+Q =>read escape sequence

docker network ls

docker ps

docker ps -a

docker rm e704412d6550

docker run -d -p 80:80 nginx:1.0

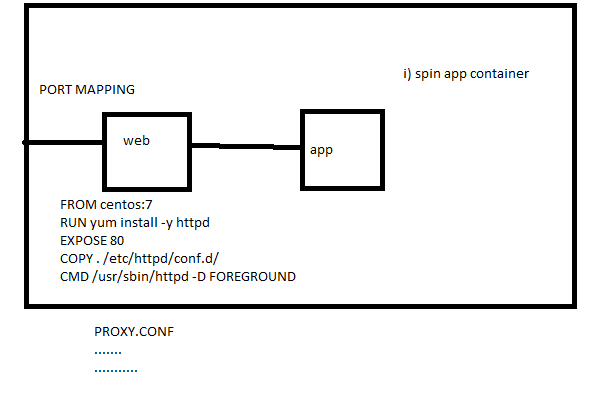
docker ps

docker exec -it 297 /bin/bash

ps -ef | grep nginx

/etc/init.d/n

/etc/init.d/nginx stop



docker run -d --name app tomcat:7

docker ps

vi proxy.conf

ProxyPass /examples <http://app:8080/examples>

ProxyPassReverse /examples http://app:8080/examples

vi Dockerfile

FROM centos:7

RUN yum install -y httpd

COPY proxy.conf /etc/httpd/conf.d/

EXPOSE 80

CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]

docker build -t apache:2.2 .

docker images

docker run -d --link app:app -p 80:80 apache:2.2

docker ps

curl ifconfig.co

OPEN THE WEBPAGE

docker inspect e8610281dff4 | grep IPAddress => Gives IP address of the image

vi proxy.conf

ProxyPass /examples <http://IPAddress:8080/examples>

ProxyPassReverse /examples <http://IPAddress:8080/examples>

LOGS FILES

mkdir /container

cd /container

mkdir logs

cd logs

mkdir nginx

cd ../../

docker run -d -P -v /container/logs/nginx:/var/log/nginx nginx

cd /container/logs/nginx

ll

-rw-r--r-- 1 root root 0 Aug 26 00:47 access.log

-rw-r--r-- 1 root root 0 Aug 26 00:47 error.log

DOCKERs in Continuous Integration

vi /etc/sysconfig/docker

docker info | grep seccomp

seccomp

grep SECCOMP /boot/config-$(uname -r)

CONFIG\_HAVE\_ARCH\_SECCOMP\_FILTER=y

CONFIG\_SECCOMP\_FILTER=y

CONFIG\_SECCOMP=y

Download the Lab for JSON files

yum install -y git

mkdir lab

cd lab

git clone https://github.com/docker/labs

ll

cd labs/

ll

cd security/

cd seccomp/

pwd

ll

ll seccomp-profiles/

-rw-r--r-- 1 root root 141 Aug 26 01:24 allow.json

-rw-r--r-- 1 root root 24451 Aug 26 01:24 default.json

-rw-r--r-- 1 root root 24225 Aug 26 01:24 default-no-chmod.json

-rw-r--r-- 1 root root 141 Aug 26 01:24 deny.json

docker run --rm -it --cap-add ALL --security-opt apparmor=unconfined --security-opt seccomp=seccomp-profiles/deny.json alpine sh

cat seccomp-profiles/deny.json

{

"defaultAction": "SCMP\_ACT\_ERRNO",

"architectures": [

"SCMP\_ARCH\_X86\_64",

"SCMP\_ARCH\_X86",

"SCMP\_ARCH\_X32"

],

"syscalls": [

]

}

docker run --rm -it --security-opt seccomp=unconfined debian:jessie sh

ls

bin dev home lib64 mnt proc run srv tmp var

boot etc lib media opt root sbin sys usr

exit

yum install strace

whoami

strace -c -f -S name whoami 2>&1 1>/dev/null | tail -n +3 | head -n -2 | awk '{print $(NF)}''

96 strace -c -f -S name whoami 2>&1 1>/dev/null | tail -n +3 | head -n -2 | awk '{print $(NF)}'

97 strace -c -f -S name ls 2>&1 1>/dev/null | tail -n +3 | head -n -2 | awk '{print $(NF)}'

ll

vi seccomp-profiles/default-no-chmod.json

docker run --rm -it --security-opt seccomp=./seccomp-profiles/default-no-chmod.json alpine sh

ls

pwd

chmod ls

cd tmp

touch abc.txt

chmod 777 abc.txt

exit

vi seccomp-profiles/default-no-chmod.json

{

“name”: “chmod”,

“action”: “SCMP\_ACT\_ALLOW”,

“args”: []

}

docker run --rm -it --security-opt seccomp=./seccomp-profiles/default-no-chmod.json alpine sh

ls

cd tmp

touch abc.txt

chmod 777 abc.txt

exit

vi seccomp-profiles/default-no-chmod.json

Docker – Compose

Docker Compose is used to run multiple containers as a single service. For example, suppose you had an application which required NGNIX and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately.

Docker Compose ─ Installation

The following steps need to be followed to get Docker Compose up and running.

sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-c -s)-$(uname -m)" -o /usr/local/bin/docker-compose

The above command downloads the version **1.24.1** of Docker Compose and stores in the directory

**/usr/local/bin/**

Provide **execute privileges** to the downloaded Docker Compose file, using the following command

sudo chmod +x /usr/local/bin/docker-compose

Command to see the **compose** version

docker-compose version

Creating Your First Docker-Compose File

mkdir composetest

pwd

/root

cd composetest/

ll

vi app.py

import time

import redis

from flask import Flask

app = Flask(\_\_name\_\_)

cache = redis.Redis(host='redis', port=6379)

def get\_hit\_count():

retries = 5

while True:

try:

return cache.incr('hits')

except redis.exceptions.ConnectionError as exc:

if retries == 0:

raise exc

retries -= 1

time.sleep(0.5)

@app.route('/')

def hello():

count = get\_hit\_count()

return 'Hello World! I have been seen {} times.\n'.format(count)

vi requirements.txt

flask

redis

ll

app.py

requirements.txt

vi Dockerfile

FROM python:3.7-alpine

WORKDIR /code

ENV FLASK\_APP app.py

ENV FLASK\_RUN\_HOST 0.0.0.0

RUN apk add --no-cache gcc musl-dev linux-headers

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

COPY . .

CMD ["flask", "run"]

ll

app.py

Dockerfile

requirements.txt

vi docker-compose.yml

version: '3'

services:

web:

build: .

ports:

- "5000:5000"

redis:

image: "redis:alpine"

ll

app.py

docker-compose.yml

Dockerfile

requirements.txt

docker-compose up -d

docker images

docker ps

curl ifconfig.co

http://IPAddress:5000/

docker-compose down

2) 2nd example- install httpd and tomcat7

cd ..

mkdir webtest

cd webtest/

vi proxy.conf

ProxyPass /examples <http://app:8080/examples>

ProxyPassReverse /examples http://app:8080/examples

vi Dockerfile

FROM centos:7

RUN yum install -y httpd

COPY proxy.conf /etc/httpd.conf.d/

EXPOSE 80

CMD /usr/sbin/httpd -D FOREGROUND

vi docker-compose.yml

version: '3'

services:

apache:

build: .

ports:

- "80:80"

app:

image: "tomcat:latest"

docker-compose up -d

docker images

docker ps

curl ifconfig.co

http://IPAddress:8080/

docker-compose down

mkdir SQLtest

cd SQLtest/

vi docker-compose.yml

version: '3.3'

services:

db:

image: mysql:5.7

volumes:

- db\_data:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: somewordpress

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

restart: always

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

WORDPRESS\_DB\_NAME: wordpress

volumes:

db\_data: {}

docker-compose up -d

docker images

docker ps

curl ifconfig.co

http://IPAddress

docker-compose down

Managing Images and Containers