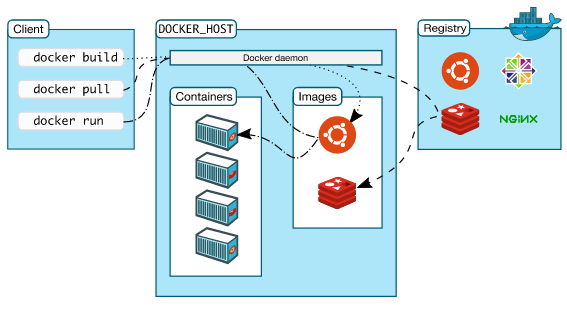
**Working with Docker images: brief introduction**

**Introduction**

**Docker images** are used to create Docker containers.

Docker images are the build component of Docker.



Picture credit : [Understand the architecture](https://docs.docker.com/v1.8/introduction/understanding-docker/)

We create Docker containers using **[base]** images. An image can be basic, with nothing but the operating-system fundamentals, or it can consist of a sophisticated pre-built application stack ready for launch.

When we build images with docker, each action taken (i.e. a command executed such as **apt-get install**) forms a new layer on top of the previous one. These base images then can be used to create new containers.

**Docker registries** hold images.

These are public or private stores from which we can upload or download images.

The public Docker registry is provided with the [Docker Hub](http://hub.docker.com/).

**What happens to an image at "docker run"?**

We use "docker run" command from client to tell the Docker daemon to run a container, for example:

$ docker run -it ubuntu:latest /bin/bash

Basically, it's a container "launch" command.

At **docker**, a Docker client is launched and at **run** subcommand, a new container will be launched.

The new container will be built from **ubuntu** base image with "latest" tag.

Here *i*: interactive, : terminal.

As described in the official document, here are the things happening under the hood:

1. **Pulls the ubuntu image**:

Docker checks for the presence of the ubuntu image and, if it doesn't exist locally on the host, then Docker downloads it from Docker Hub.

If the image already exists, then Docker uses it for the new container.

1. **Creates a new container**:

Once Docker has the image, it uses it to create a container.

1. **Allocates a filesystem and mounts a read-write layer**:

The container is created in the file system and a read-write layer is added to the image.

1. **Allocates a network and sets up an IP address**:

Creates a network interface that allows the Docker container to talk to the local host.

1. **Executes a process that we specify**:

Runs our application.

**Search images - docker search**

**docker search** searches registry for image.

$ docker search ubuntu

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

ubuntu Ubuntu is a Debian-based Linux operating s... 4416 [OK]

ubuntu-upstart Upstart is an event-based replacement for ... 65 [OK]

...

**Download images - docker pull**

**docker pull** pulls an image from registry to local machine.

$ docker pull ubuntu

latest: Pulling from ubuntu

20ee58809289: Pull complete

f905badeb558: Pull complete

119df6bf2a3a: Pull complete

94d6eea646bc: Pull complete

bb4eabee84bf: Pull complete

Digest: sha256:85af8b61adffea165e84e47e0034923ec237754a208501fce5dbeecbb197062c

Status: Downloaded newer image for ubuntu:latest

Docker images can consist of multiple layers.

In the example above, the image consists of five layers (20ee58809289,...,bb4eabee84bf).

We can use a tag to specify what to download. For example, 'latest' for tag:

$ docker pull ubuntu:latest

**Listing images - docker images**

To list the images on the host:

docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

ubuntu latest bb4eabee84bf 2 weeks ago 124.8 MB

ubuntu 16.04 bb4eabee84bf 2 weeks ago 124.8 MB

centos 7 2a332da70fd1 9 weeks ago 196.8 MB

ubuntu trusty 9bc953763843 10 weeks ago 188 MB

debian latest cea663c8c811 10 weeks ago 125.1 MB

centos latest ce20c473cd8a 9 months ago 172.3 MB

**Running container - docker run**

Now we want to launch a container:

$ docker run -it ubuntu:latest /bin/bash

We're logged in as a root:

root@859d4a27d4c8:/# whoami

root

Check the OS:

root@859d4a27d4c8:/# cat /etc/\*release

DISTRIB\_ID=Ubuntu

DISTRIB\_RELEASE=16.04

...

Let's check what processes are currently running:

root@859d4a27d4c8:/# top

...

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

1 root 20 0 18232 2032 1556 S 0.0 0.1 0:00.09 bash

14 root 20 0 36628 1704 1272 R 0.0 0.0 0:00.02 top

As we can see there are only two processes, and they are isolated ones from the host processes.

**Getting out of a container without stopping it**

"Ctrl + P + Q" will do the trick:

root@859d4a27d4c8:/#

k@laptop:~$

From the PID, we can check the "top" on the host is using different space from the "top" on the docker container:

k@laptop:~$ pa aux | grep top

root 14512 0.0 0.0 36628 1704 pts/17 S+ 22:01 0:00 top

Now, we may want to get back to our container again via "attach". To do that, we need to know the "CONTAINER ID":

k@laptop:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

859d4a27d4c8 ubuntu:latest "/bin/bash" 5 minutes ago Up 5 minutes cranky\_swartz

Let's do attach:

k@laptop:~$ docker attach 859d4a27d4c8

root@859d4a27d4c8:/#

**Stops a container**

We can stop the container and get out of it via "Ctrl + D":

root@859d4a27d4c8:/# exit

k@laptop:~$

If we issue "docker ps" command again, we see the container is not running anymore:

k@laptop:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

k@laptop:~$

We can list all containers including old ones that stopped running using "docker ps -a":

k@laptop:~$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

859d4a27d4c8 ubuntu:latest "/bin/bash" 31 minutes ago Exited (0) 4 minutes ago cranky\_swartz

...

**Running a container in a background**

To run a container in background, we use "docker start" command:

k@laptop:~$ docker start cranky\_swartz

cranky\_swartz

k@laptop:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

859d4a27d4c8 ubuntu:latest "/bin/bash" 36 minutes ago Up 4 seconds cranky\_swartz

**Check what processes are running inside a container**

We can check what processes are running inside a container using "docker top":

k@laptop:~$ docker top cranky\_swartz

UID PID PPID C STIME TTY TIME CMD

root 15680 1559 0 22:33 pts/17 00:00:00 /bin/bash

**Stop a container running in background**

To stop a container running in background, we use "docker stop" command:

k@laptop:~$ docker stop cranky\_swartz

**What's in the system**

What's in our system related to the Docker containers?

root@laptop:/# tree /var/lib/docker/containers

|-- 859d4a27d4c883db39e68590f1d1d2c340f8775a94fe721eba85111d3b79c1fe

â”‚Â Â  |-- 859d4a27d4c883db39e68590f1d1d2c340f8775a94fe721eba85111d3b79c1fe-json.log

â”‚Â Â  |-- config.json

â”‚Â Â  |-- hostconfig.json

â”‚Â Â  |-- hostname

â”‚Â Â  |-- hosts

â”‚Â Â  |-- resolv.conf

â”‚Â Â  |-- resolv.conf.hash

Note that there are no binary images for the container which makes docker consumes much less space compare to the other virtual tools.

**Detach mode run**

We can run a container in a "detach" mode, and later we can attach to it:

Note we need to use "-it" so that we can do something with the container after attaching. Also, we used "--name" to give our own name to the container.

k@laptop:~$ docker run -d -it --name=yaong ubuntu:16.04 /bin/bash

6bbba8e00d68a9b9c38bd7fdbd807dae01b9329d3b5ecd7ad2918305743bf5ea

Ok, it's started, and we can check it:

k@laptop:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

6bbba8e00d68 ubuntu:16.04 "/bin/bash" About a minute ago Up About a minute yaong

**To stop it:**

k@laptop:~$ docker stop 6bbba8e00d68

6bbba8e00d68

k@laptop:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

k@laptop:~$