What is Docker?



Docker is an open source container platform designed to make it easier to create, deploy, and run applications by using containers.

It allow developers to package up an application with all needed components, such as libraries and other dependencies, and ship it as an independent and single package called Docker Images.

The developer is sure 100% that the application will run on all docker supported machines.

Docker containers are so lightweight, a single server or virtual machine can run several containers simultaneously.

Who is Docker for?

Docker is designed to benefit both developers and system administrators, making it a part of many DevOps (developers + operations) toolchains.

For developers, they can focus on writing code without worrying about the system that it will ultimately be running on.

For operations staff, Docker gives flexibility and potentially reduces the number of systems needed because of its small footprint and lower overhead.

Purpose of Docker?

Its primary focus is to automate the deployment of applications inside docker containers and the automation of operating system level virtualization on Linux.

It's more lightweight than standard containers and boots up in seconds.

Docker can be integrated into various infrastructure tools, including:

Amazon Web Services, Google Cloud Platform, Microsoft Azure, Kubernetes, OpenStack, OpenSVC, Oracle Container Cloud Service, Ansible, Chef, Jenkins, Puppet, Vagrant.

Choosing between docker and VM:

When Application is more important than infrastructure, then we go for docker. But if OS management is more important than it is recommended to use VM.

Case study: Suppose you want to install tomcat in your BareMetal system, you need to perform the following task:

1) Installing the VM, 2) Get tomcat image, 3) Start VM, 4) install dependency for tomcat, 5) install tomcat, 6) Start tomcat.

Above process takes 1-15 days for installing the tomcat. Alternative of this is Golden image. If you have golden image, it may take day or 2 to install the tomcat, but in docker, we can do it in 2 minutes.

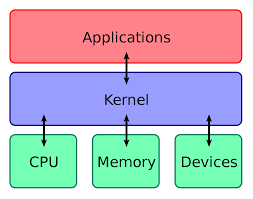
Steps for installing tomcat in docker: (2 Minute activity)

docker run -it --rm tomcat:9.0

<http://container-ip:8080>

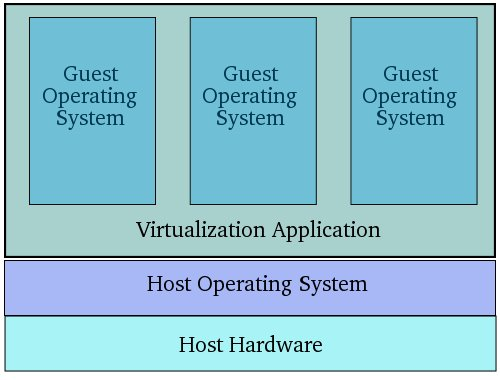
Which kind of memory management docker use?- Docker use dynamic memory management. Earlier we assigned 128 MB RAM to each container, but now these days all is managed dynamacially.

Kernel: The Linux® kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the 2, managing resources as efficiently as possible.



Host operating system- A host OS is the software installed on a computer that interacts with the underlying hardware and is usually used to describe an operating system used in a virtualized server to differentiate it from the guest operating system.

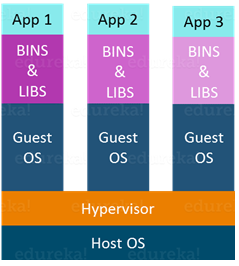
Guest Operating system- A guest operating system (guest OS) is an operating system (OS) that is secondary to the OS originally installed on a computer, which is known as the host operating system. The guest OS is either part of a partitioned system or part of a virtual machine (VM) setup. A guest OS provides an alternative OS for a device.



Virtualization

Virtualization is the technique of importing a Guest operating system on top of a Host operating system. This technique was a revelation at the beginning because it allowed developers to run multiple operating systems in different virtual machines all running on the same host. This eliminated the need for extra hardware resource. The advantages of Virtual Machines or Virtualization are:

* Multiple operating systems can run on the same machine
* Maintenance and Recovery were easy in case of failure conditions
* Total cost of ownership was also less due to the reduced need for infrastructure



In the above diagram, you can see there is a host operating system on which there are 3 guest operating systems running which is nothing but the virtual machines.

As you know nothing is perfect, Virtualization also has some shortcomings. Running multiple Virtual Machines in the same host operating system leads to performance degradation. This is because of the guest OS running on top of the host OS, which will have its own kernel and set of libraries and dependencies. This takes up a large chunk of system resources, i.e. hard disk, processor and especially RAM.

Another problem with Virtual Machines which uses virtualization is that it takes almost a minute to boot-up. This is very critical in case of real-time applications.

Following are the disadvantages of Virtualization:

* Running multiple Virtual Machines leads to unstable performance
* Hypervisors are not as efficient as the host operating system
* Boot up process is long and takes time

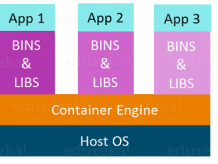
These drawbacks led to the emergence of a new technique called Containerization. Now let me tell you about Containerization.

Containerization

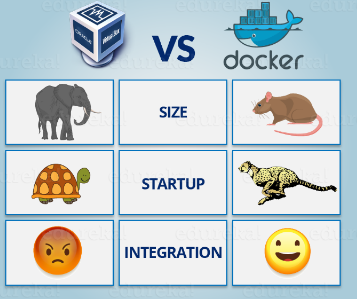
Containerization is the technique of bringing virtualization to the operating system level. While Virtualization brings abstraction to the hardware, Containerization brings abstraction to the operating system. Do note that Containerization is also a type of Virtualization. Containerization is however more efficient because there is no guest OS here and utilizes a host’s operating system, share relevant libraries & resources as and when needed unlike virtual machines. Application specific binaries and libraries of containers run on the host kernel, which makes processing and execution very fast. Even booting-up a container takes only a fraction of a second. Because all the containers share, host operating system and holds only the application related binaries & libraries. They are lightweight and faster than Virtual Machines.

Advantages of Containerization over Virtualization are:

* Containers on the same OS kernel are lighter and smaller
* Better resource utilization compared to VMs
* Boot-up process is short and takes few seconds

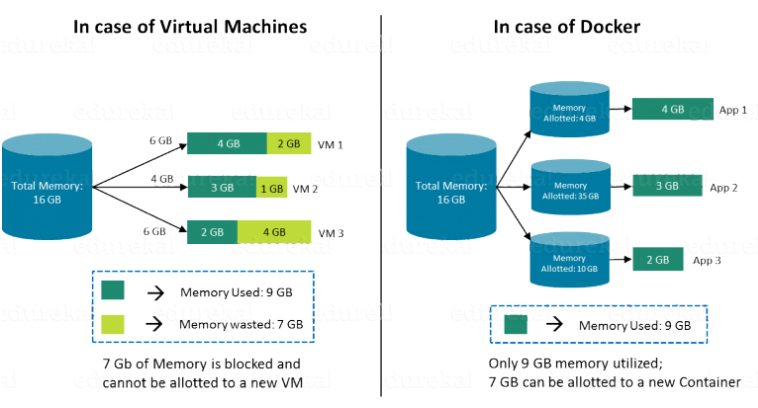


Virtual Machine and Docker Container are compared on the following three parameters:



Size

The following image explains how Virtual Machine and Docker Container utilizes the resources allocated to them.



Consider a situation depicted in the above image.  I have a host system with 16 Gigabytes of RAM and I have to run 3 Virtual Machines on it. To run the Virtual Machines in parallel, I need to divide my RAM among the Virtual Machines. Suppose I allocate it in the following way:

* 6 GB of RAM to my first VM,
* 4 GB of RAM to my second VM, and
* 6 GB to my third VM.

In this case, I will not be left with anymore RAM even though the usage is:

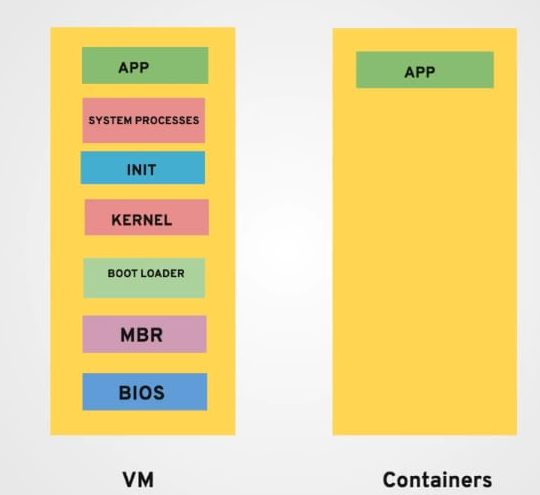
* My first VM uses only 4 GB of RAM – Allotted 6 GB – 2 GB Unused & Blocked
* My second VM uses only 3 GB of RAM – Allotted 4 GB – 1 GB Unused & Blocked
* My uses only 2 GB of RAM – Allotted 6 GB – 4 GB Unused & Blocked

## **Start-Up**

When it comes to start-up, Virtual Machine takes a lot of time to boot up because the guest operating system needs to start from scratch, which will then load all the binaries and libraries. This is time consuming and will prove very costly at times when quick startup of applications is needed. In case of Docker Container, since the container runs on your host OS, you can save precious boot-up time. This is a clear advantage over Virtual Machine.

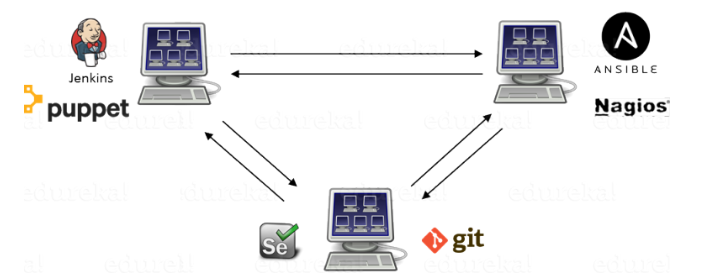
Consider a situation where I want to install two different versions of Ruby on my system. If I use Virtual Machine, I will need to set up 2 different Virtual Machines to run the different versions. Each of these will have its own set of binaries and libraries while running on different guest operating systems. Whereas if I use Docker Container, even though I will be creating 2 different containers where each container will have its own set of binaries and libraries, I will be running them on my host operating system. Running them straight on my Host operating system makes my Docker Containers lightweight and faster.

So Docker Container clearly wins again from Virtual Machine based on Startup parameter.

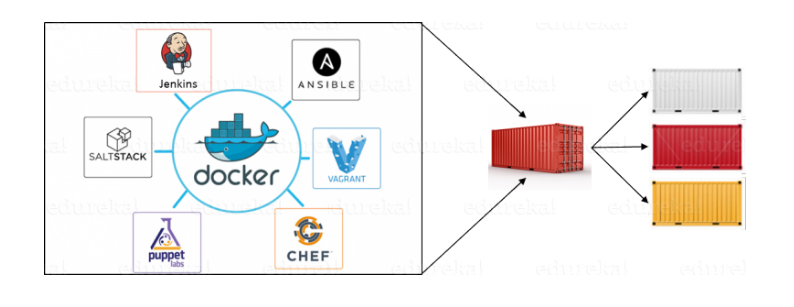
****

**Integration**

Integration of different tools using Virtual Machine maybe possible, but even that possibility comes with a lot of complications.



I can have only a limited number of DevOps tools running in a Virtual Machine. As you can see in the image above, If I want many instances of Jenkins and Puppet, then I would need to spin up many Virtual Machines because each can have only one running instance of these tools. Setting up each VM brings with it, infrastructure problems. I will have the same problem if I decide to setup multiple instances of Ansible, Nagios, Selenium and Git. It will also be a hectic task to configure these tools in every VM.This is where Docker comes to the rescue. Using Docker Container, we can set up many instances of Jenkins, Puppet, and many more, all running in the same container or running in different containers which can interact with one another by just running a few commands. I can also easily scale up by creating multiple copies of these containers. So configuring them will not be a problem.



Docker written in which language- Go language.

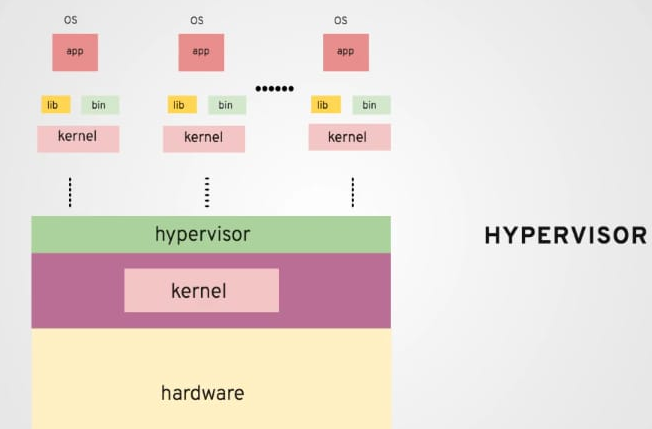
Docker support which OS- Linux, window and MAC OS.

## What is a Physical Server? For most, the physical server is a well-understood part of the IT infrastructure that has been around since the very beginning. A physical server is a hardware you can **touch** and **feel**. A typical server is sometimes referred to as “bare-metal”. It generally includes all physical hardware components contained in the physical server case that allows it to function. Physical servers typically have a CPU, RAM, and some type of internal storage from which the operating system is loaded and is booted. It may or may not have general-purpose storage outside of the storage used for the operating system.

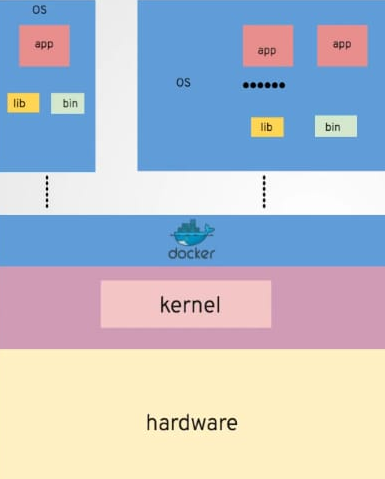
## 

What is VM? Virtual Machines are made possible by installing a hypervisor on top of a “bare-metal” server. A common approach for many popular hypervisors today, such as VMware vSphere and Microsoft Hyper-V, is to virtualize the hardware of the underlying physical server and present this virtualized hardware to the operating system. The hypervisor generally has a CPU scheduler of some sort that brokers requests from the client operating systems running in guest virtual machines with the physical CPU installed in the underlying physical host.

Virtual machines provide many advantages over a physical server in terms of provisioning, management, configuration, and automation. While a new physical server may take days or weeks to acquire, provision, and configure, a new virtual machine can generally be spun up in minutes and even seconds in some cases.



What is Container? - they run as a process in container engine.

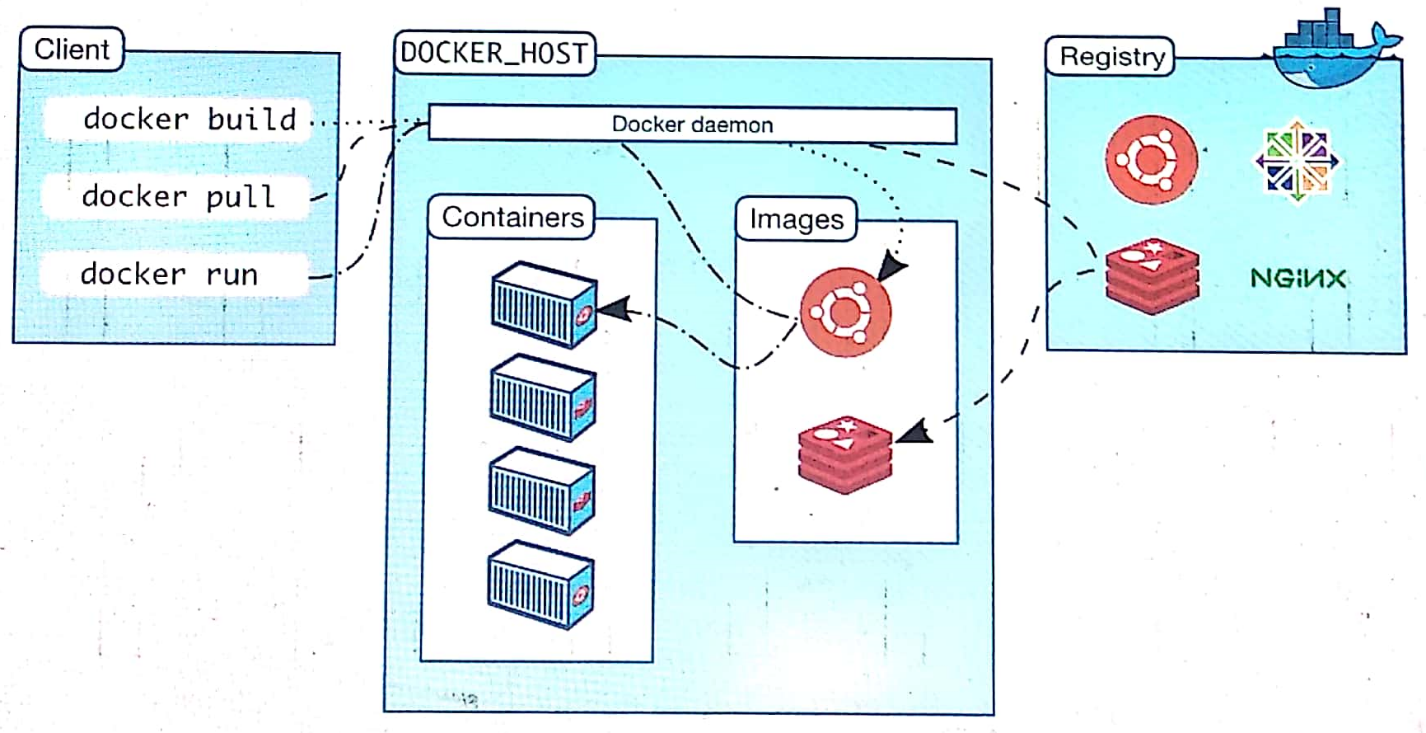


Docker can be considered as PAAS for application.

## What happen when we do a double click on notepad++? When we double click on notepad++, processor will allocate CPU and RAM to process, and PID file will be generated then we can do a common operation.

What is Docker? Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

## Docker architecture Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



### The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

### The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

### Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

### Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

#### Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images, or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

#### Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container’s network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

##### Example docker run command

The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs /bin/bash.

$ docker run -it ubuntu /bin/bash

When you run this command, the following happens (assuming you are using the default registry configuration):

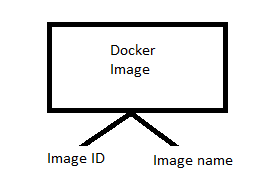
1. If you do not have the ubuntu image locally, Docker pulls it from your configured registry, as though you had run docker pull ubuntu manually.
2. Docker creates a new container, as though you had run a docker container create command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.
4. Docker creates a network interface to connect the container to the default network, since you did not specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine’s network connection.
5. Docker starts the container and executes /bin/bash. Because the container is running interactively and attached to your terminal (due to the -i and -t flags), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type exit to terminate the /bin/bash command, the container stops but is not removed. You can start it again or remove it.

Installation on ubuntu

$ curl -fsSL https://get.docker.com -o get-docker.sh

$ sh get-docker.sh

Image attributes Image ID and Image name



Can you run multiple containers from image, and how many? yes, depend on system capacity.

# This script is meant for quick & easy install via:

# $ curl -fsSL https://get.docker.com -o get-docker.sh

# $ sh get-docker.sh

What is docker client?- Whenever we are running a command such as “docker info” these commands known as docker client.

What happen when we install docker? Docker daem on and docker client will be installed.

Docker info command- This command will give you the overview of infrastructure.

root@ip-172-31-20-17:~# docker info

Client:

Debug Mode: false

Server:

Containers: 2

Running: 1

Paused: 0

Stopped: 1

Images: 1

Server Version: 19.03.8

What is image: it contains all information about running an application.

What is happening inside a container?- docker container provide run time environment to application, it will have all the necessary things that a docker can have. Container provide same infra as OS provide.

What is userspace in Linux- Whenever we are creating any user in Linux, they allocate some memory to it.(home directory)

All docker technology works on below things-

1) Kernel

2) Namespace

3) C group- control group

4) Overlay FS

5) Copy on write.

Namespace :

[Docker](https://foxutech.com/category/docker/) uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker Engine uses namespaces such as the following on Linux:

* The pid namespace: Process isolation (PID: Process ID).
* The net namespace: Managing network interfaces (NET: Networking).
* The user namespace:
* uts namespace

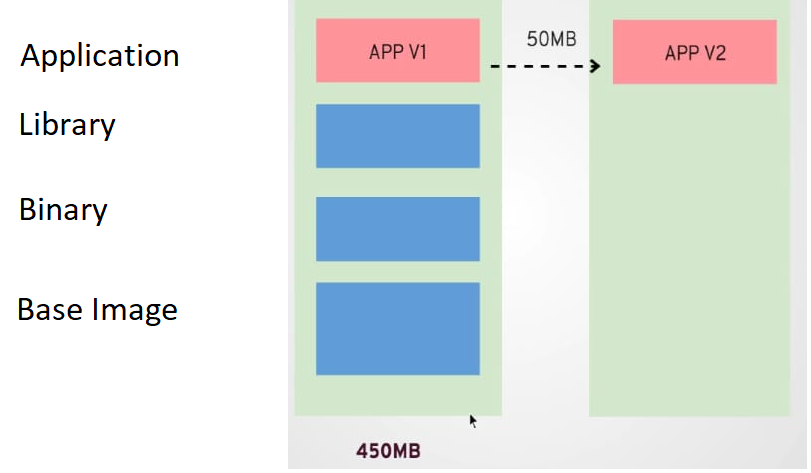
C-group: **cgroups(Control Group):** A croup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available to a specific container.

Primarily, Cgroup has 2 major responsibility:

1) Control resource utilization, 2) Collect data for monitoring (container monitoring.)

Casestudy: Suppose, 1 container has memory leak, it may corrupt to entire system, for avoiding these kind of issue **Google has given “Control group concept”**

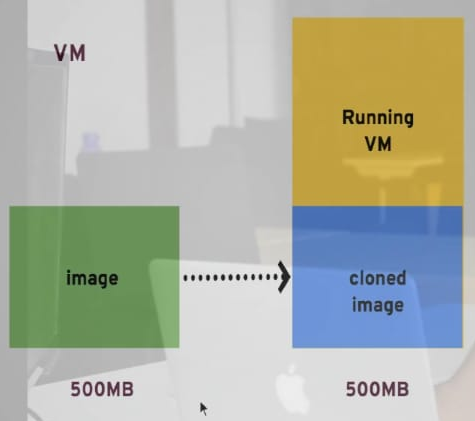
Overlay file system (Cricinfo scorecard for refresh)

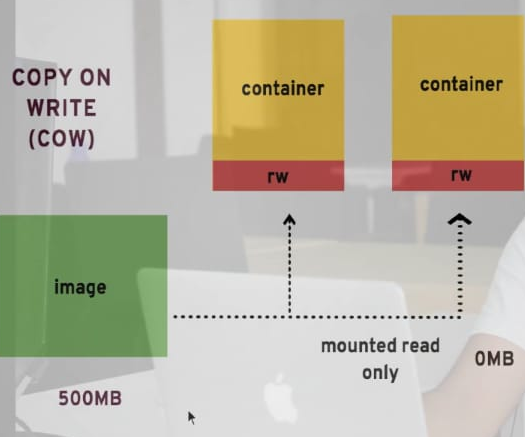


Layering concept of docker image: Here, only latest changes would be copied, not all image. latest: Pulling from library/jenkins

55cbf04beb70: Pull complete 1607093a898c: Extracting [=============================================> ] 9.83MB/10.74MB 9a8ea045c926: Download complete d4eee24d4dac: Download complete c58988e753d7: Download complete 794a04897db9: Download complete 9a8ea045c926: Download complete d4eee24d4dac: Download complete 9a8ea045c926: Download complete d4eee24d4dac: Download complete c58988e753d7: Download complete 794a04897db9: Download complete

Copy on write:





Feature of Docker over VM: 1) Density, 2) Consistency, and 3) portability.

Density: **it** refers to the number of objects that a single physical server can run at one time. As docker is a lightweight solution, so it provides high density.

Consistency: Build >> Ship >> Deploy. Here, Developer packaged code with dependency, so easily we can maintain the consistency.

Portability: One of the main benefits in using Docker and container technology is the portability of applications. It’s possible to spin up an application on-site or in a public cloud environment in a matter of minutes. What enables this is a storage technology that implements a layered “copy on write” approach to storing data in the file system used by Docker itself. Let's take a look at how it works.

Docker as Devops too example Serverless computing (AWS Lambda):

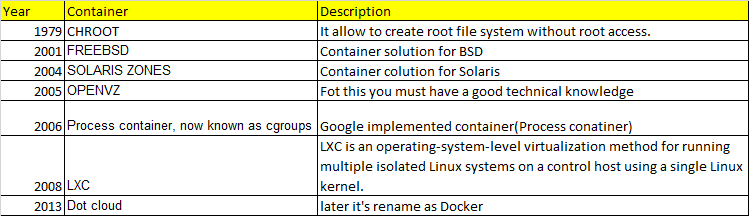
Serverless computing is also known as FAAS (Function as service), Here we implement faction those launches container once it’s called.

Example: Suppose, in e-commerce side you want to implement FAAS, so whenever user’s call a any function (let’s say checkout) a new container would be launched, and it will serve the request.

Docker: on demand processing: Microservice implementation.

Container portability means the ability to move an application, in other words, port it from one host environment to another. The new host environment [could](https://wiki.aquasec.com/display/containers/Containers+and+Cloud+Computing) be a different kind of [operating system](https://wiki.aquasec.com/display/containers/Container+Operating+Systems), different version of the same operating system or a different type of hardware platform.

Docker Evolution:



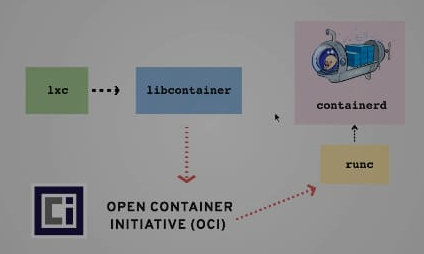
Based on LXC many organizations started building their own container, and dot cloud is 1 of them.

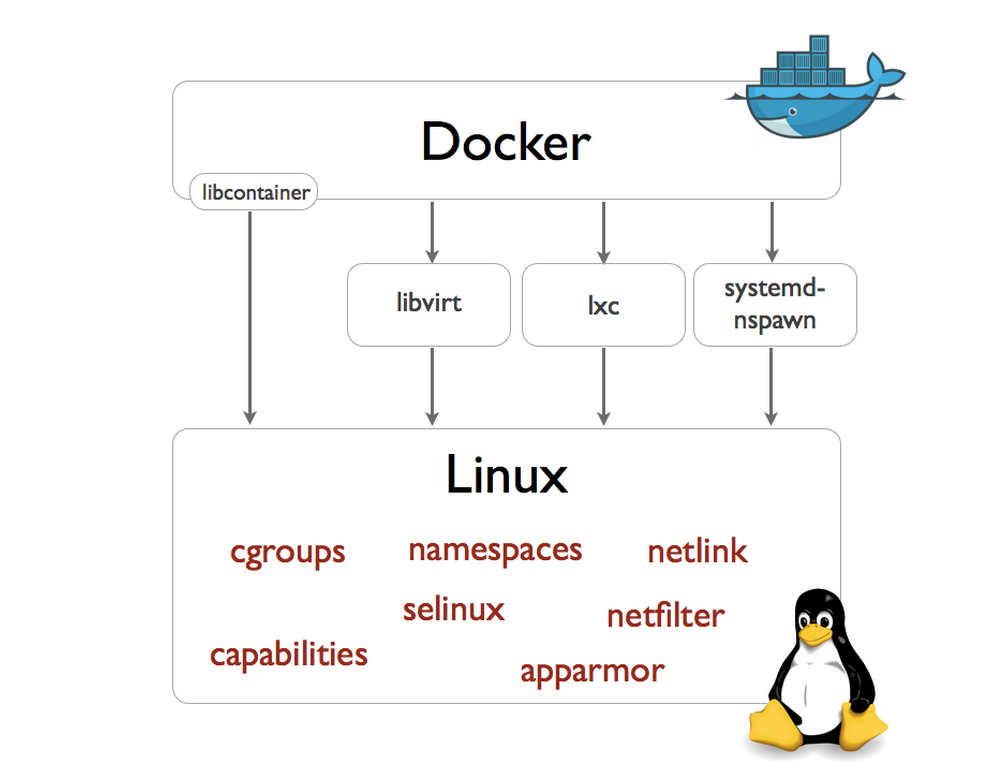
LXC was external project, so dot cloud builds their own system called libcontainer.

As there were multiple company who started duplicating their effort, so open standard found, and dot cloud contributed their code to open standard, and it called run.

runC: Universal run time container.

In top of runC, docker has built container demon.





Docker version check:

root@ip-172-31-41-185:~# docker --version

Docker version 20.10.1, build 831ebea

root@ip-172-31-41-185:~# docker -v

Docker version 20.10.1, build 831ebea

$ docker version : it gives client and server version details.

Client:

Version: 19.03.8

API version: 1.40

Go version: go1.12.17

Git commit: afacb8b

Built: Wed Mar 11 01:21:11 2020

OS/Arch: darwin/amd64

Context: default

Experimental: true

Server:

Engine:

Version: 19.03.8

API version: 1.40 (minimum version 1.12)

Go version: go1.12.17

Git commit: afacb8b

Built: Wed Mar 11 01:29:16 2020

OS/Arch: linux/amd64

Experimental: true

containerd:

Version: v1.2.13

GitCommit: 7ad184331fa3e55e52b890ea95e65ba581ae3429

runc:

Version: 1.0.0-rc10

GitCommit: dc9208a3303feef5b3839f4323d9beb36df0a9dd

docker-init:

Version: 0.18.0

GitCommit: fec3683

$ docker info : it gives environment details

Client:

Context: default

Debug Mode: false

Server:

Containers: 14

Running: 3

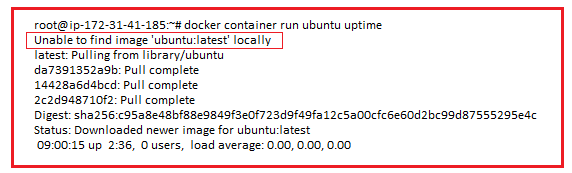
Paused: 1

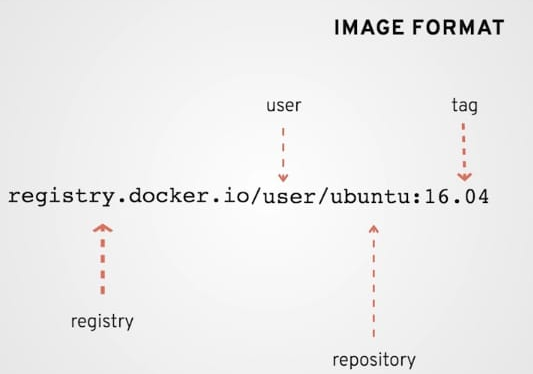
Stopped: 10

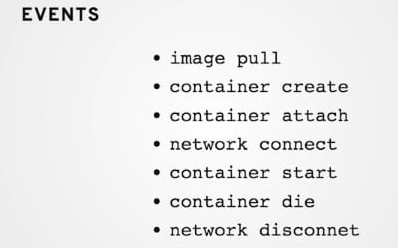
Images: 52

Creating a container: This command first tries to find the image locally, if not then it will go to registry.



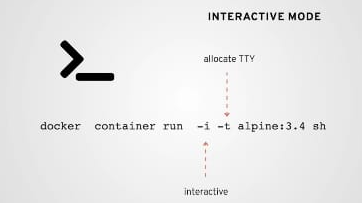






Note: Default tag is latest.

Running container interactive: it will open the shell terminal.

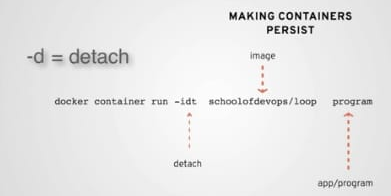


Namespace concept of Docker:



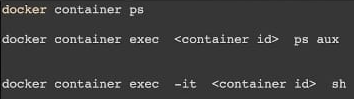
Docker ps command: Docker container ps : listing the docker processes. Docker container ps -l : listing docker last running process. Docker container ps -n 2 : listing last 2 running process. Docker container ps -a : listing all the process

Making container persist:

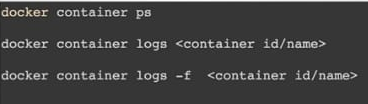


For making container persist, we need to make application persist. Let say we just run uptime command in container, so post execution of it container would be deleted.

Connecting to a container:



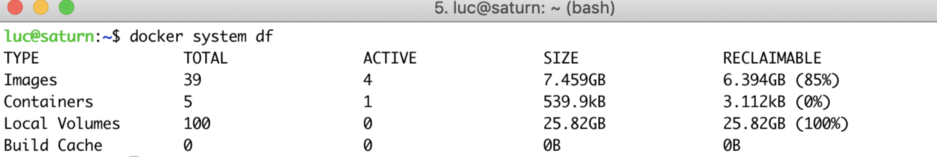
Checking container log



Docker operations on container:





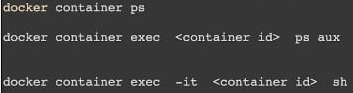


docker system prune

Remove all unused containers, networks, images (both dangling and unreferenced), and optionally, volumes.

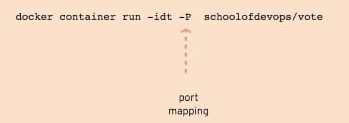
Advance Container operation:

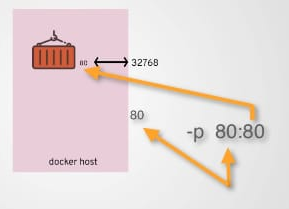
1) Connecting to a container:

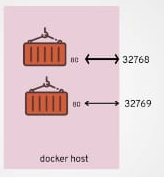


2) Port mapping Port mapping means connecting with container to outside world.

-P/-p : They both are same, -P means it looks in docker file whereas -p looks in runtime.







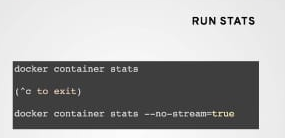
Container operation:

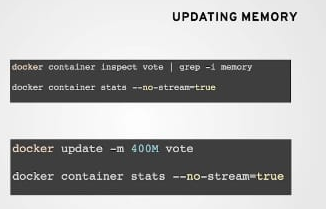


Attaching/detaching a process:









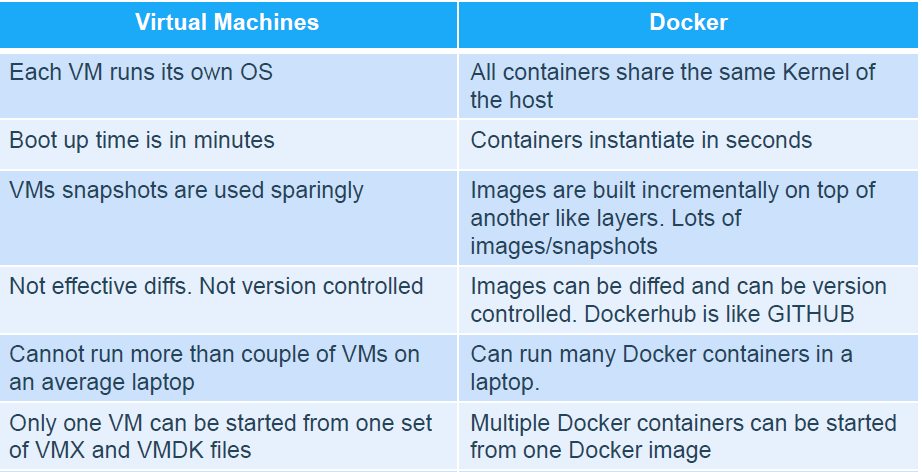
--cpu-shares

Set this flag to a value greater or less than the default of 1024 to increase or reduce the container’s weight, and give it access to a greater or lesser proportion of the host machine’s CPU cycles. This is only enforced when CPU cycles are constrained. When plenty of CPU cycles are available, all containers use as much CPU as they need. In that way, this is a soft limit. --cpu-shares does not prevent containers from being scheduled in swarm mode. It prioritizes container CPU resources for the available CPU cycles. It does not guarantee or reserve any specific CPU access.

Advantage of container over VM: Speed, cost, integration.

Advantage of VM over container: security, complete OS not custom.

Docker VS VM:



Note: In container technology container can used security feature of VM.