

**VM:**

Cost higher compared to container

Can’t changes kernel at runtime.

Hardware cannot change at runtime. (Need to restart if you want)

VM ISO is file.

**Container:**

Portability (Docker container will run any platform)

Speed (Container can be create within less time)

Docker Object image. It’s encrypted and compressed. Images contains layer’s, each layer is object.

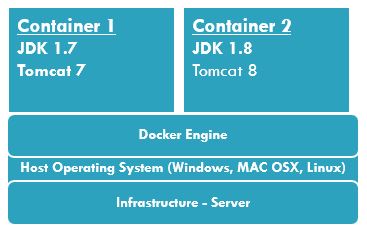
Layers (only downloads which are changed from 16.4 to 16.8 Ubuntu (incremental update)

GIT repo for code, Docker Hub for Docker image. Jfrag stores both code and Docker image.

OverlayFS is a modern union filesystem that is similar to AUFS, but faster and with a simpler implementation. Docker provides two storage drivers for OverlayFS: the original overlay, and the newer and more stable overlay2.

The overlay2 subdirectory specifically contains the various filesystem layers for images and containers. To cleanup unused containers and images, see docker system prune . There are also options to remove volumes and even tagged images, but they aren't enabled by default due to the possibility of data loss.

* Docker is a platform for developing, shipping and running applications using a container based Virtualization technique.
* The car needs an engine to run, similarly Docker needs an engine to run. Docker Engine sits on top of the host operating system.
* In short **Docker** engine, ***is a lightweight container run time***. Let us go through what it means.

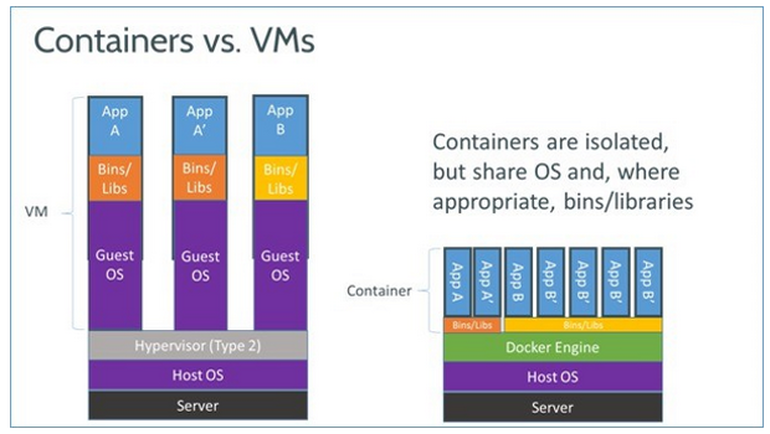


**Docker**

* Runs the same operating system as the Host OS.
* This allows it to share a lot of host operating system resources.
* In order to achieve this they originally used **LXC** (LinuX Containers) but later moved to **runC** (aka libcontainer).
* **AuFS is a layered file system**, so you can have a read only part and a write part which are merged together. One could have the common parts of the operating system as read only (and shared amongst all of your containers) and then give each container its own mount for writing.

**Virtual Machines**

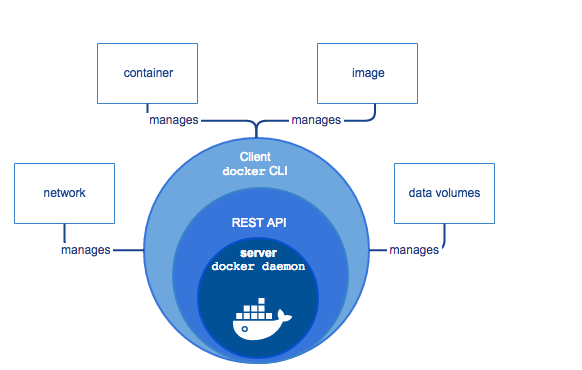
* A full virtualized system **gets its own set of resources** allocated to it, and does minimal sharing.
* You get more isolation, but it **is much heavier** (requires more resources). With docker you get less isolation, but the containers are lightweight (require fewer resources).



A hypervisor (or virtual machine monitor, VMM) is a computer software, firmware or hardware that creates and runs virtual machines. A computer on which a hypervisor runs one or more virtual machines is called a host machine, and each virtual machine is called a guest machine. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems.

**What is the Docker platform?**

Docker provides the ability to **package** and **run** an application in a **loosely isolated environment called a container.** The isolation and security allow you to run **many containers simultaneously on a given host**.



**What is the Docker Engine?**

***Docker Engine*** is a client-server application with these major components:

A **server** which is a type of long-running program called a daemon (docker d) process.

A **REST API** which specifies interfaces that programs can use to talk to the daemon and instruct it what to do. Docker d contacts container d for the further process. In one docker engine one docker d and for each container one container d will be there.

A command line **interface (CLI) client**.

**Namespaces: (limits what you can see)**

Docker makes use of kernel 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 the following namespaces on Linux:

**PID namespace** for process isolation.

**NET namespace** for managing network interfaces.

**IPC namespace** for managing access to IPC resources.

**MNT namespace** for managing filesystem mount points.

**UTS namespace** for isolating kernel and version identifiers.

**Cgroups: (limits how much you can use)**

Docker also makes use of kernel control groups for resource allocation and isolation. A cgroup 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.

Docker Engine uses the following cgroups:

**Memory cgroup** for managing accounting, limits and notifications.

**HugeTBL cgroup** for accounting usage of huge pages by process group.

**CPU group** for managing user / system CPU time and usage.

**CPUSet cgroup** for binding a group to specific CPU. Useful for real time applications and NUMA systems with localized memory per CPU.

**BlkIO cgroup** for measuring & limiting amount of blckIO by group.

net\_cls and net\_prio cgroup for tagging the traffic control.

**Devices cgroup** for reading / writing access devices.

**Freezer cgroup** for freezing a group. Useful for cluster batch scheduling, process migration and debugging without affecting prtrace.

Key Points:

**container will run only if service is available, if service goes down container will also down.**

From docker image you can create multiple containers.

Docker image contains layers, each layer is object. Layer location

ls /var/lib/docker/aufs/layers

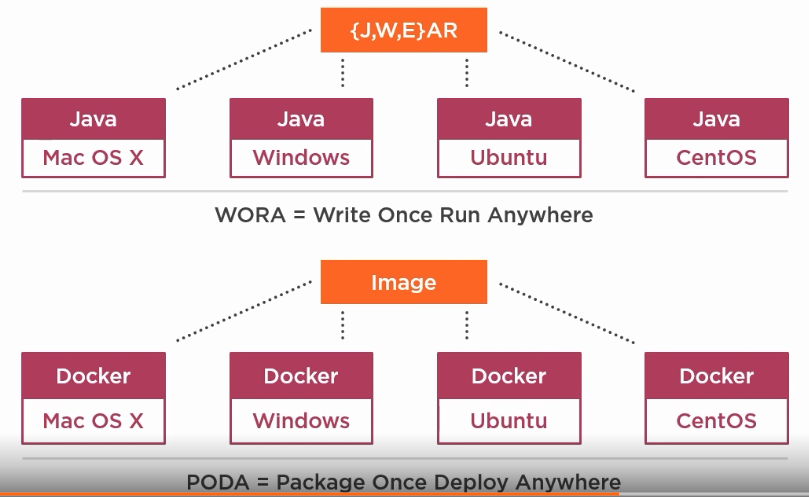
volumes: virtual disk to store and share data between containers. Not part of image.

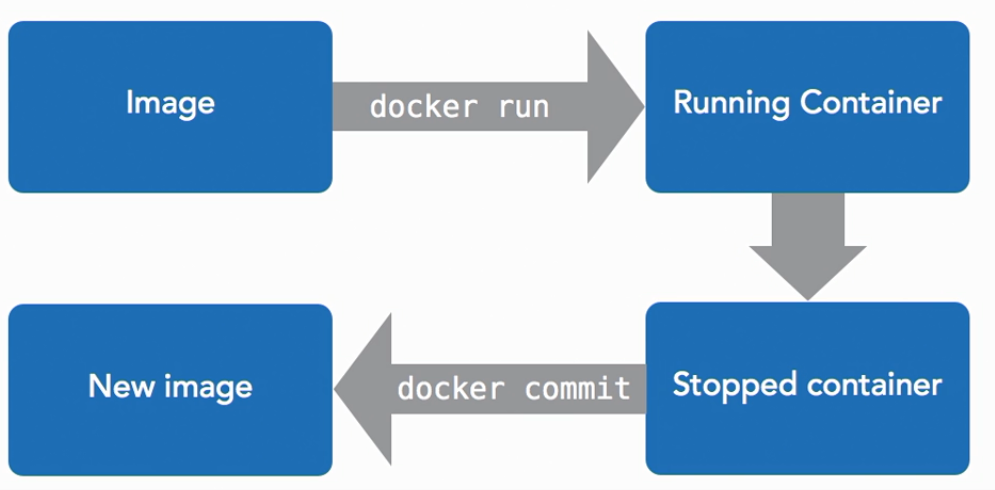
Two types persistent(will exist after container exit) and ephemeral(will not exist after exit)

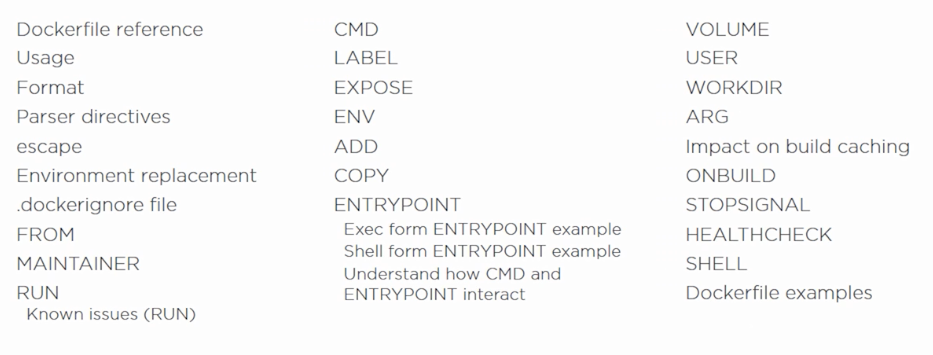
ENTRYPOINT, CMD can use both, ENTRYPOINT can’t replace while running, CMD can replace.

<https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/#sharing-promotes-smaller-images>

<https://stackoverflow.com/questions/19234831/where-are-docker-images-stored-on-the-host-machine>







**Docker Commands:**

$ docker version

$ docker –v



**MYSQL DB:**

$ docker pull mysql:5.7

Pulls the Docker image from docker hub <https://hub.docker.com/_/mysql/>

$ docker run --name mysql-standalone -e MYSQL\_ROOT\_PASSWORD=password -e MYSQL\_DATABASE=test -e MYSQL\_USER=sa -e MYSQL\_PASSWORD=password

-d mysql:5.7

Command to run the mysql container.

$ docker build . -t users-mysql

Creating a docker image from the Spring Boot Application using the Dockerfile.

$ docker run -p 8086:8086 --name users-mysql --link mysql-standalone:mysql -d users-mysql

$ docker container ls

$ docker logs <container\_name>

$ docker container rm <container\_name>

<http://www.java67.com/2018/02/5-free-docker-courses-for-java-and-DevOps-engineers.html>