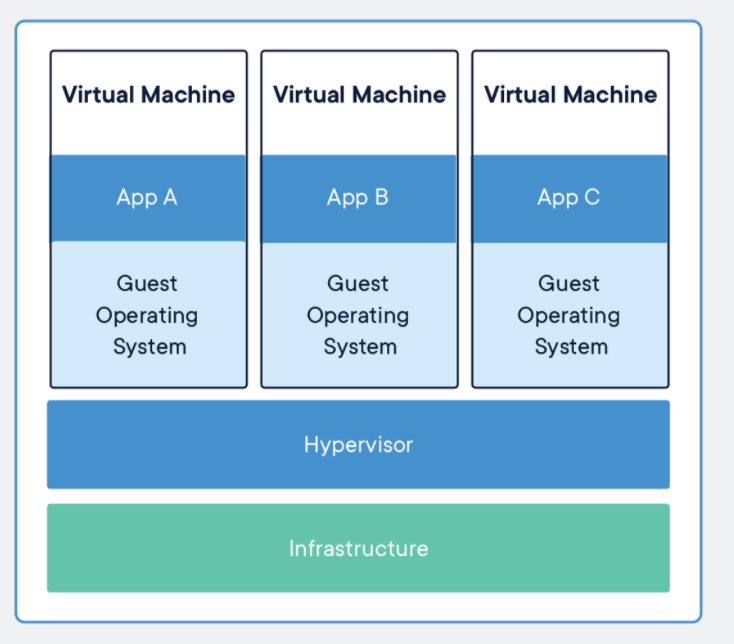
**DOCKER**

**VIRTUAL MACHINE:**

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

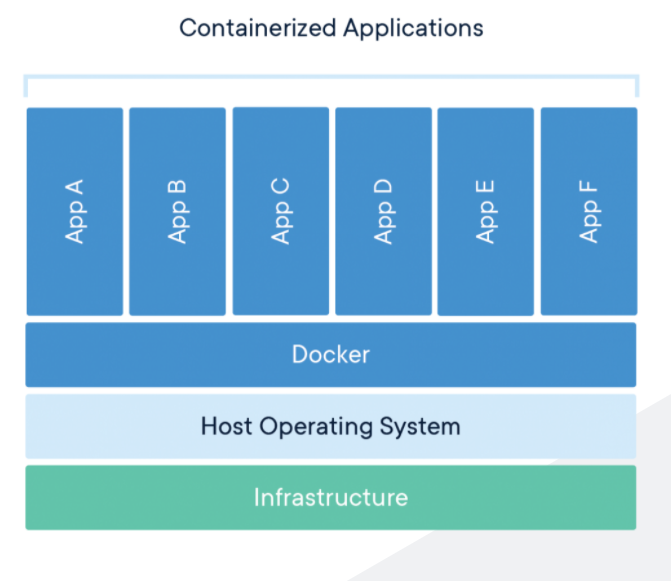


**CONTAINER**:

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

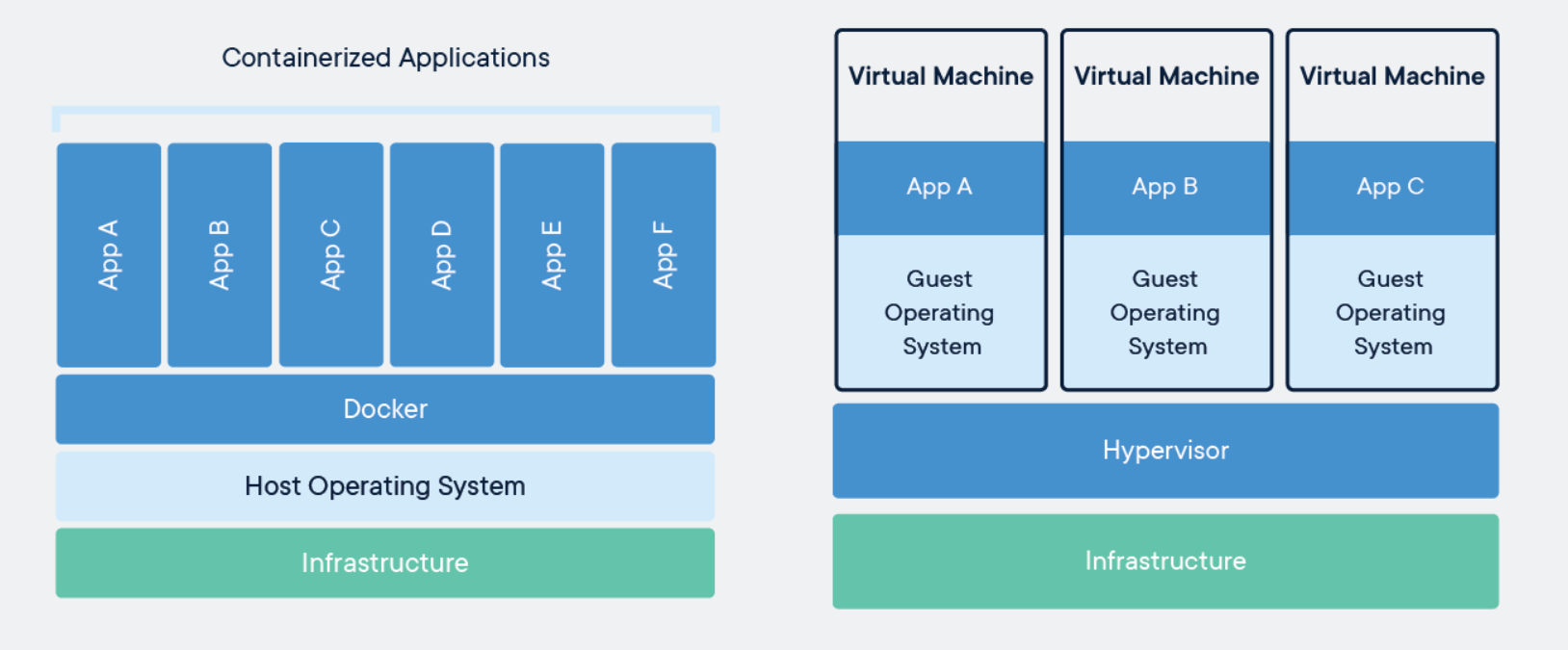
Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.



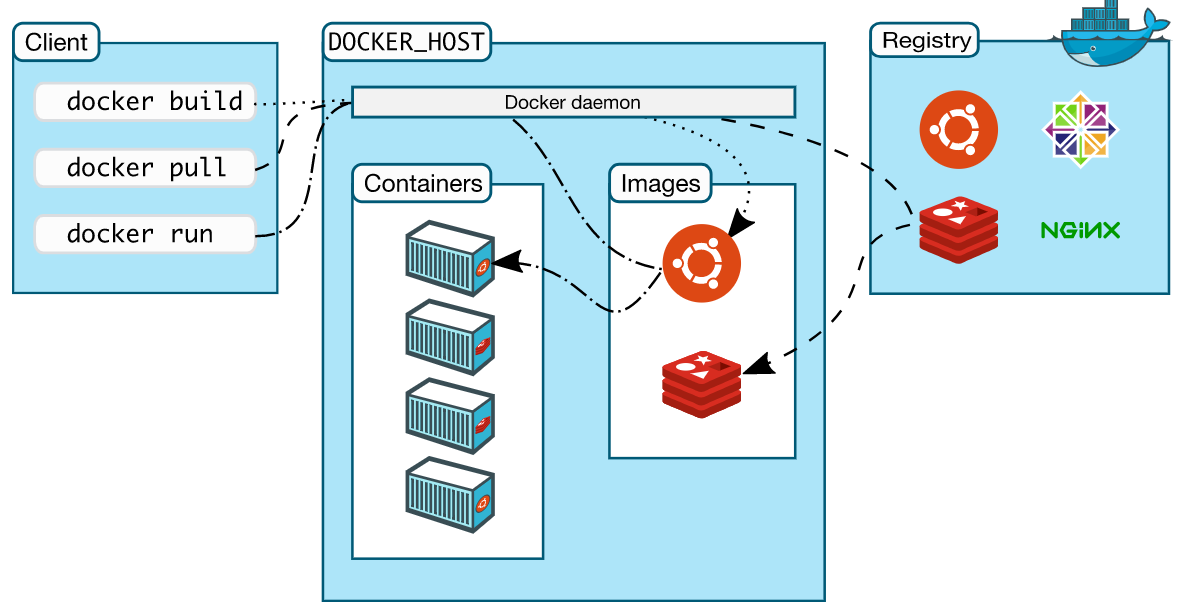
**Note**: On Windows Machine, we can run both Linux and windows based docker containers but on Linux VM, we can only run Linux containers.

**Difference between Virtual Machine and Docker**:

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.



**Docker Architecture**:



Underlying technology used for docker:

**Namespaces**:

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 ipc namespace: Managing access to IPC resources (IPC: InterProcess Communication).

The mnt namespace: Managing filesystem mount points (MNT: Mount).

The uts namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

**Control groups**:

Docker Engine on Linux also relies on another technology called control groups (cgroups). 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. For example, you can limit the memory available to a specific container.

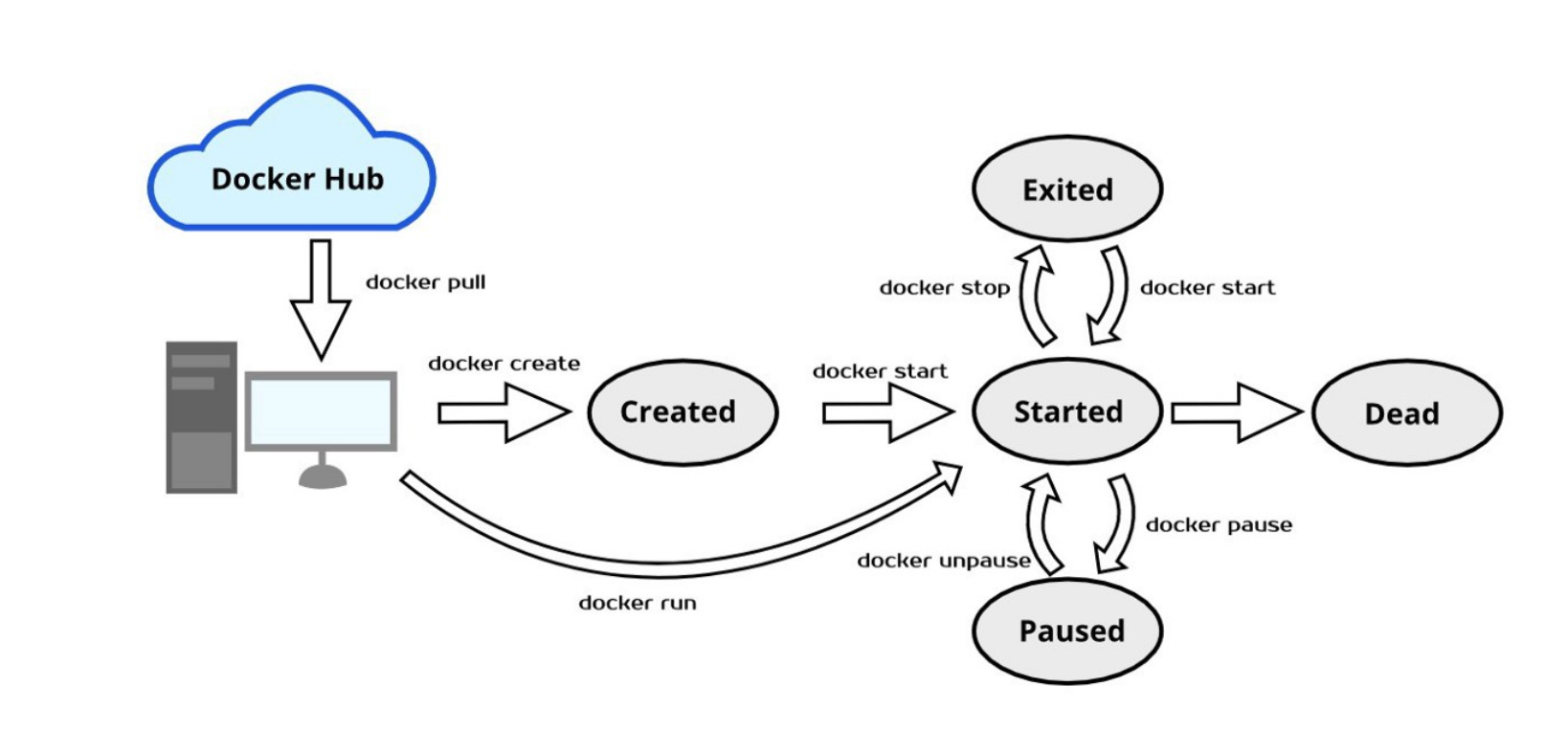
**Union file systems**:

Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

**Container format**:

Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a container format. The default container format is libcontainer. In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones.

**Container Lifecycle**:



**Links**:

<https://www.docker.com/resources/what-container>

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

<https://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-virtual-machine>

**DOCKER INSTALLATION STEPS ON UBUNTU or RED HAT/CENTOS:**

**Ubuntu**: <https://docs.docker.com/engine/install/ubuntu/>

Step 1: Uninstall the previous version of docker.

sudo apt-get remove docker docker-engine docker.io containerd runc

Step 2: Set up repository

sudo apt-get update

sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

Step 3: Add docker's official gpg key

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Step 4: Verify the key

sudo apt-key fingerprint 0EBFCD88

pub rsa4096 2017-02-22 [SCEA]

9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

uid [ unknown] Docker Release (CE deb) <docker@docker.com>

sub rsa4096 2017-02-22 [S]

Step 5: set up stable repository

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

Step 6: Install docker engine

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

**RHEL**: <https://docs.docker.com/engine/install/centos/>

Step 1: Uninstall the previous docker version

sudo yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

Step 2: Set up the repository

sudo yum install -y yum-utils

sudo yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

Step 3: Install docker engine

sudo yum install docker-ce docker-ce-cli containerd.io

**See yourself The below topics on Docker**:

1. Basic Understanding about Docker Swarm.
2. Basic Understanding about Docker-compose.