

* **DOCKER CONTAINERS:**
* A container is a **runnable instance of an image.** You can create, start, stop, move, or delete it.
* Use containers to Build, Share and Run your applications.
* It is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
* 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.
* Docker Containers Are Everywhere: Linux, Windows, Data center, Cloud, Serverless, etc.

**DOCKER CONTAINERS ARE:**

* **Flexible :** Even the most complex applications can be containerized.
* **Lightweight :** Containers leverage and share the host kernel.
* **Interchangeable :** Can deploy updates and upgrades on-the-fly.
* **Portable :** Can build locally, deploy to the cloud, and run anywhere.
* **Scalable :** Can increase and automatically distribute container replicas.
* **Stackable :** Can stack services vertically and on-the-fly





CONTAINERS

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.

VIRTUAL MACHINES

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 LIFECYCLE:

The complete lifecycle of a docker container revolves around five phases:

Create phase

Running phase

Paused phase/unpause phase

Stopped / Exited phase

Killed phase



ummary of these states before we deep dive into it:

created - This is the initial state of a Docker container after it has been created, but before it has been started.

running - When a Docker container is started, it transitions to the running state. In this state, the container is actively executing its processes.

paused - If a container is paused, it is temporarily stopped from running its processes, but it is not terminated.

exited - If a container's main process completes, the container stops and transitions to the exited state.

dead - If a container fails to start, it is in the dead state. Containers in this state cannot be restarted and must be recreated.

Running containers:

Docker runs processes in isolated containers. A container is a process which runs on a host. The host may be local or remote.

When you execute docker run, the container process that runs is isolated in that it has its own file system, its own networking, and its own isolated process tree separate from the host.

SYNTAX: $docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

Options: -i : Gives us an interactive shell into the running container

 -t : Will allocate a pseudo-tty

 -d : The daemon mode / background process

 --name: Give the container name

For example, you can specify sh as the [COMMAND], combined with the -i and -t flags, to start an interactive shell in the container (if the image you select has an sh executable on PATH).

$docker run -it IMAGE sh

Foreground and background:

When you start a container, the container runs in the foreground by default. If you want to run the container in the background instead, you can use the --detach (or -d) flag. This starts the container without occupying your terminal window.

 docker run -d IMAGE

$docker run -d nginx

$docker ps

To check the container logs:

$ docker logs -n 5 0246aa4d1448

MANIPULATING DOCKER IMAGES:

#docker run -i -t <imagename>:<tag> /bin/bash

Options: -i : Gives us an interactive shell into the running container

 -t : Will allocate a pseudo-tty

 -d : The daemon mode

#docker run -i -t centos:latest /bin/bash

#docker ps (or) #docker container ls [List running containers] [on another terminal]

#docker ps -a [all containers]

#docker stop cid [Application shutdown gracefully]

#docker start -a cid [-a attach mode]

Rename Container:

#docker rename <current\_container\_name> <new\_container\_name>

Container stats:

#docker stats <container\_name>

#docker stats cid

Monitor Container:

#docker top cid/name

Container Pause:

#docker container pause containerID

Container Unpause:

#docker container unpause cid

Kill one or more Containers:

#docker kill cid [no time for proper shutdown of Application]

Removing Containers:

#docker container rm cid (or) #docker rm cid

Removing all stopped containers:

#docker container prune

Docker System and stat:

#docker logs cid

#docker logs -f cid

#docker system

#docker system df

#docker system events [Two terminals] #docker stop containerID / Launch new container

#docker system prune [Delete all stopped and unused containers]

#docker system prune -a [Delete all images and stopped containers]

PORT FORWARDING:

#docker run -d -p <host\_port>:<container\_port> <image>:<tag>

#docker pull nginx

#docker run --name mynginx -d -p 8080:80 nginx/imageid

Go To Web Browser

<http://10.10.10.10:8080/>

#docker logs cid

#docker logs -f cid

Allocate Memory for Container:

#docker status ContID

#docker run -d -p 8000:80 -m 512m nginx [Container memory limit 512m max]

#docker run -d -p 8000:80 -m 512m --cpu-quota=50000 nginx [cpulimit 50000(50%) total cpu size 100thousend]