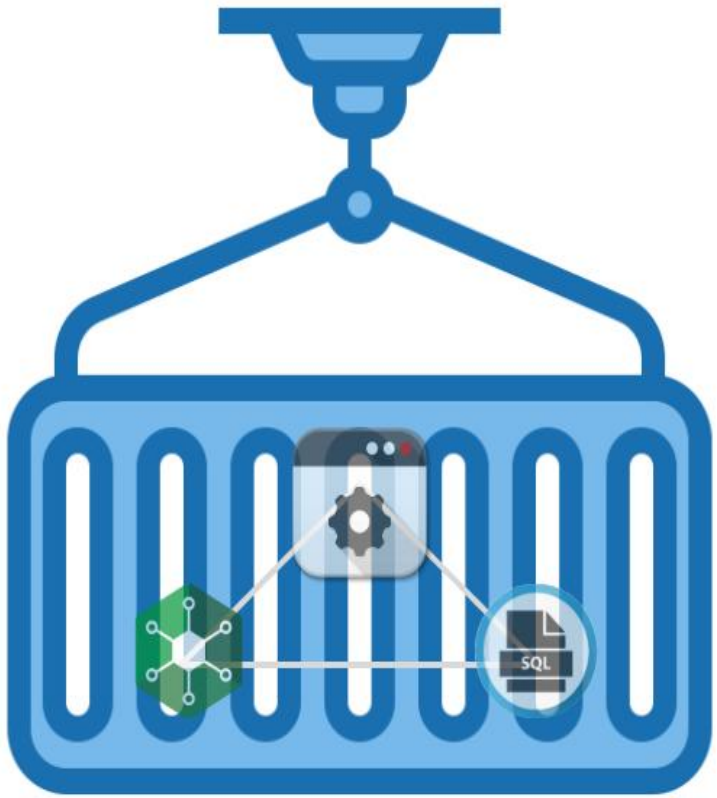
Containerization using Docker

What is Containerization?

It is the process of packaging the application and all its dependencies to execute them in an efficient and hassle-free way across different environments. A single unit of this bundled package is known as a **Container**



What is a Container?

- It is an Operating System Level Virtualization technology
- Detaches the application and its dependencies from the rest of the system
- utilizes **namespaces(Na)** and **cgroups(CG)** feature of Linux Kernel to isolate processes

Why use Containers?

Performance Overhead

Containers work on top Host OS's Kernel, therefore, there is little to no performance overhead

Easily Manageable

Since all the dependencies run inside an isolated instance, it is easy to manipulate and make changes to the application

Platform Independent

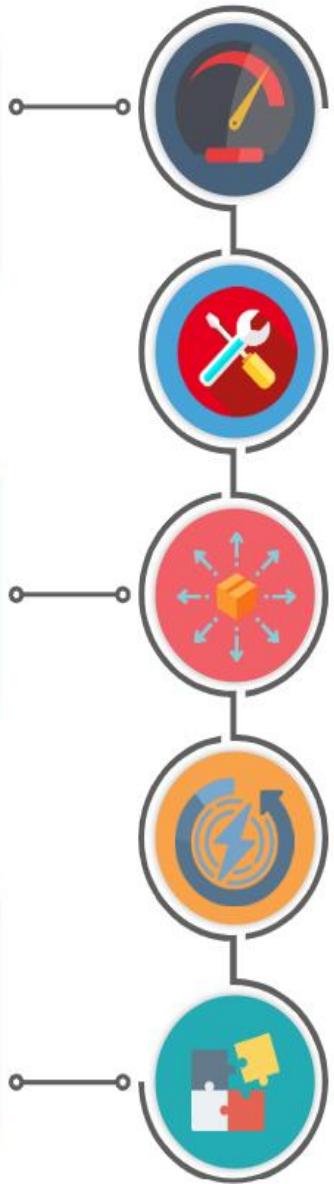
Containers can be deployed to platforms with different network topologies and security policies without any hassle

Instant Boot

Containerized application has zero boot time making them available instantaneously

Modularity

Depending upon the approach, containers work seamlessly in a monolithic as well as the microservice environment



Container: Working

Containers utilize two of the Linux Kernel features:

CG

CGROUPS

- Control Groups is a Linux Kernel feature
- Allows segregating the processes and the required resources
- Manages the isolated resources and processes as a single module

Na

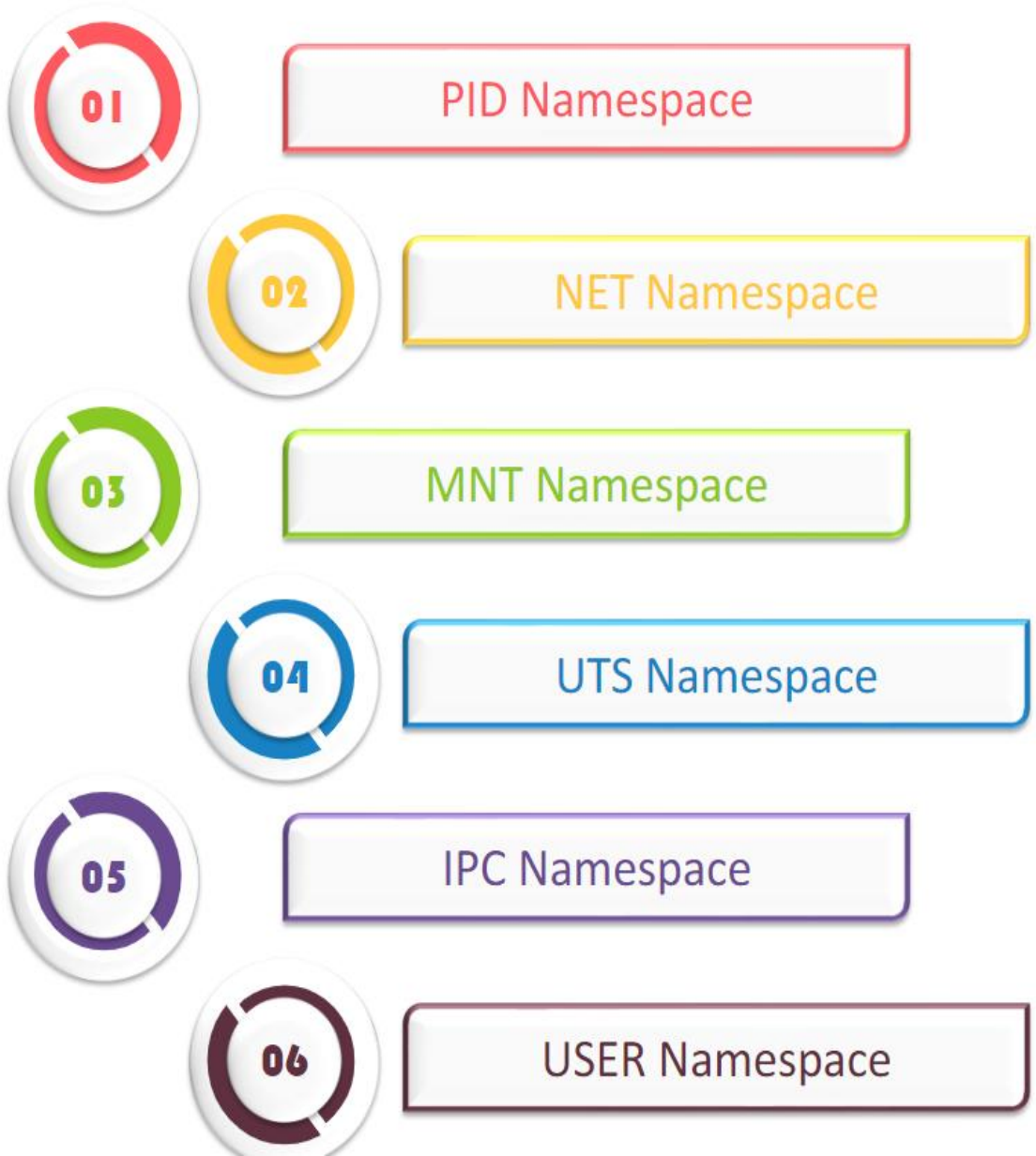
{...}

NAMESPACES

- Used to limit process's access to view the system
- Process is unaware of everything running outside of its own namespace

Namespaces

- Provides a system resource, a layer of abstraction
- Application presumes it has an isolated instance of the resource
- There are six types of namespaces



Types of Namespaces

01

PID Namespace

- Processes within PID namespaces can access/view other processes within that namespace only
- Each PID namespace starts from PID1
- If PID1 is killed, all the child processes are terminated

02

NET Namespace

- Segregates the processes by providing them their own private network
- Private network can include its own routing table, iptables, sockets and so on

03

MNT Namespace

- Enables processes to have their own root file system
- Mounts can be private or shared depending upon application requirements

04

UTS Namespace

- Provides the processes a separate copy of hostname to work with
- This isolates kernel for the processes and system identifiers

05

IPC Namespace

- Used to provide access to IPC resources
- Isolates the inter-process communication resource
- Resources such as IPC semaphores, IPC message queues, IPC shared memory can be accessed.

06

USER Namespace

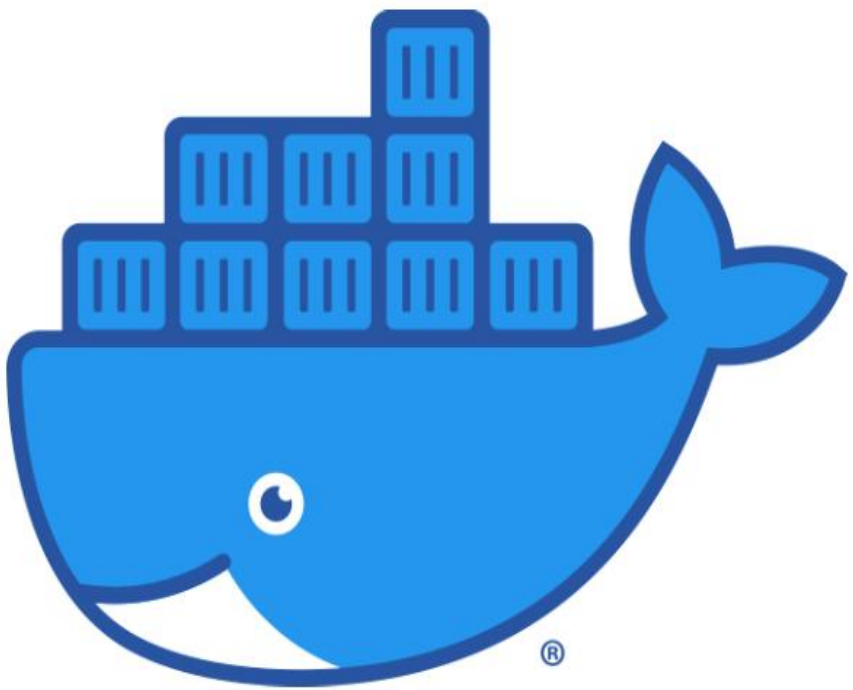
- Provides identification and isolation to a set of processes
- Different UIDs and GIDs can be mapped to the processes

Containers vs Virtual Machines

	Containers	Virtual Machines
Number of Apps on Server	Can run a limited number of applications, depending on VM Configuration	Can run multiple instances of different applications together
Resource Overhead	Light-weight with little to no overhead	Highly resource-intensive with performance overhead of managing multiple Oss
Speed of Deployment	Very fast deployment. Container images require seconds to deploy new instances	Very slow Deployment. Requires setting up OS, app dependencies, etc.
Security	Users usually require root level permissions to execute container related tasks	Provides better security
Portability	Highly portable with emphasis on consistency across environments	Very limited portability

Introduction to Docker

Docker is one of the most popular Container engines today because of the way it handles containers



Docker: Features

Fast Configuration

- The set up process is very quick and easy
- It separates the requirements of the infra from the requirements of the application

Application Isolation

- Provides applications/service isolation
- Applications inside containers execute independent of the rest of the system

Productivity

- Rapid deployment, in turn, increases the productivity
- Reduced resource utilisation is another factor increasing the productivity

Swarm

- It helps clustering and scheduling docker containers
- It enables controlling cluster of docker hosts from a single point

Docker: Features (Contd.)

Services

- Services use a set of tasks to define the state of a container inside a cluster
- Swarm manager takes the defined service as input and schedules it on the nodes

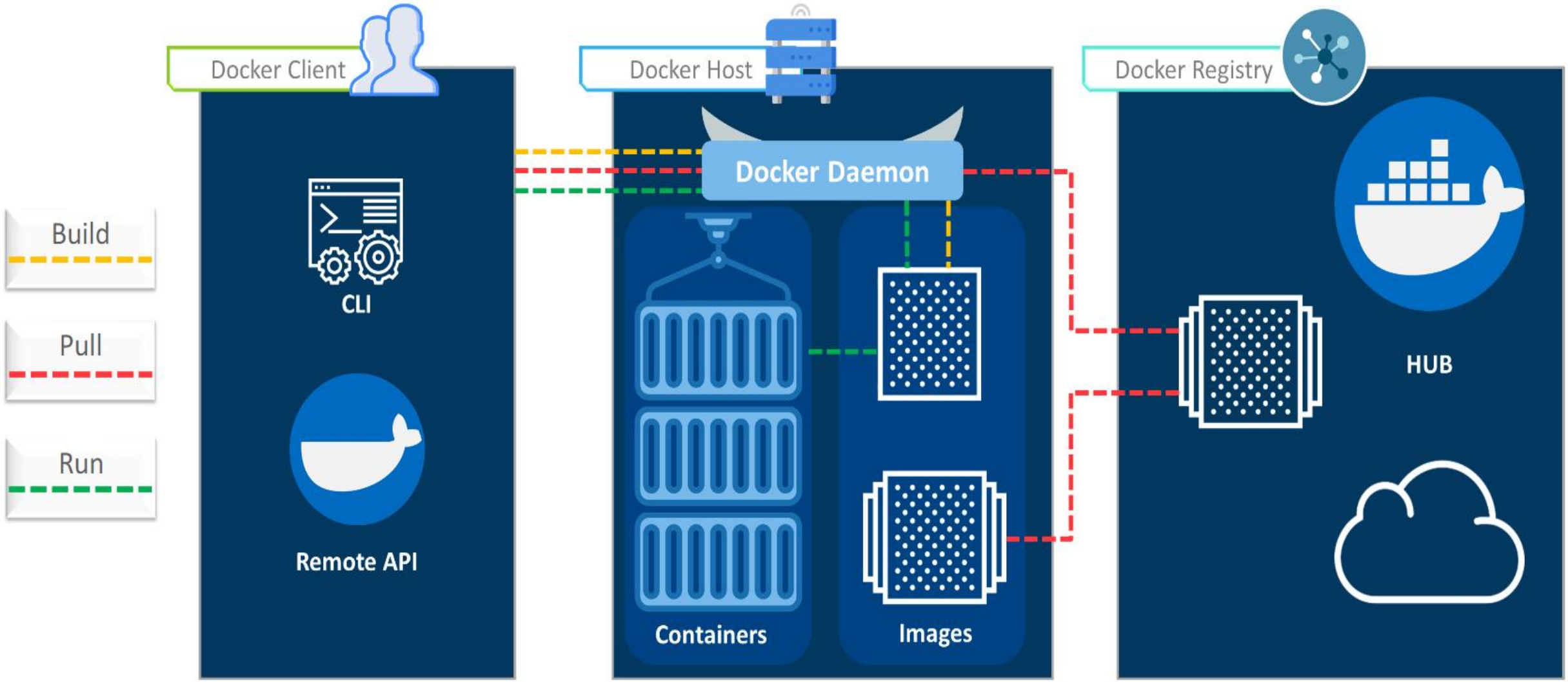
Security

- Allows saving secrets inside the swarm cluster
- These secrets can then be accessed by the required service

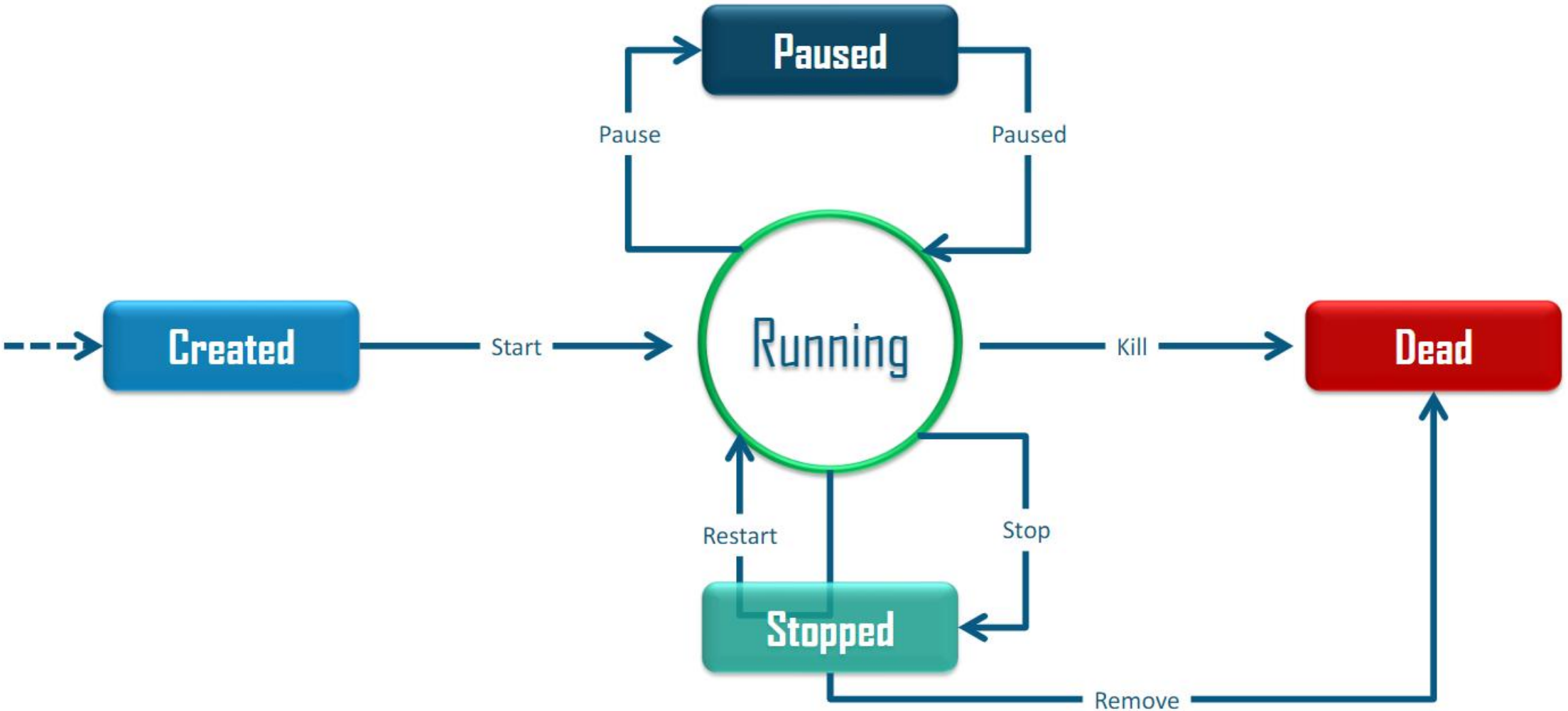
Service Discovery

- Mesh allows service discovery through the same port on all the nodes in swarm
- It is possible to reach the node even if the service is not deployed on it

Docker Architecture



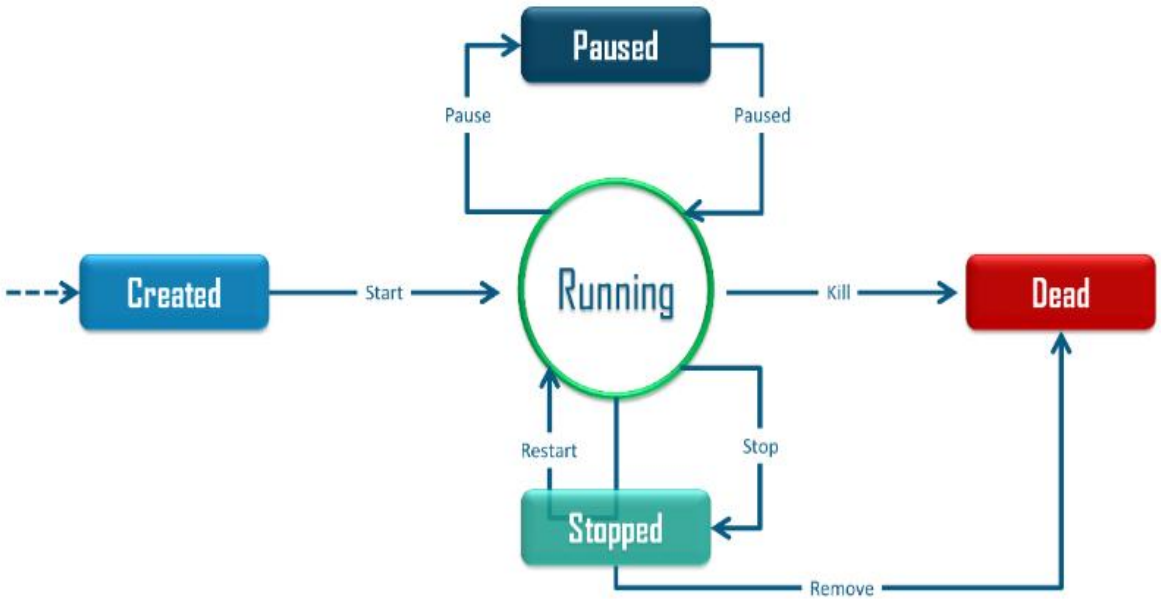
ontainer Lifecycle



Container Lifecycle

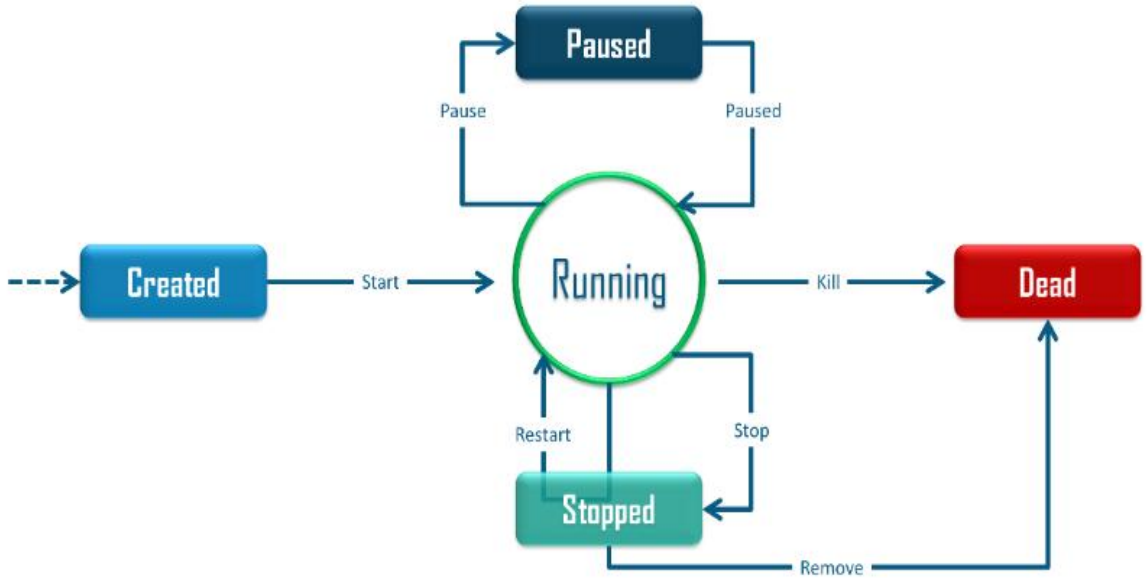
A newly created container can be in one of six states:

- Created
- Running
- Paused
- Stopped
- Restarted
- Dead



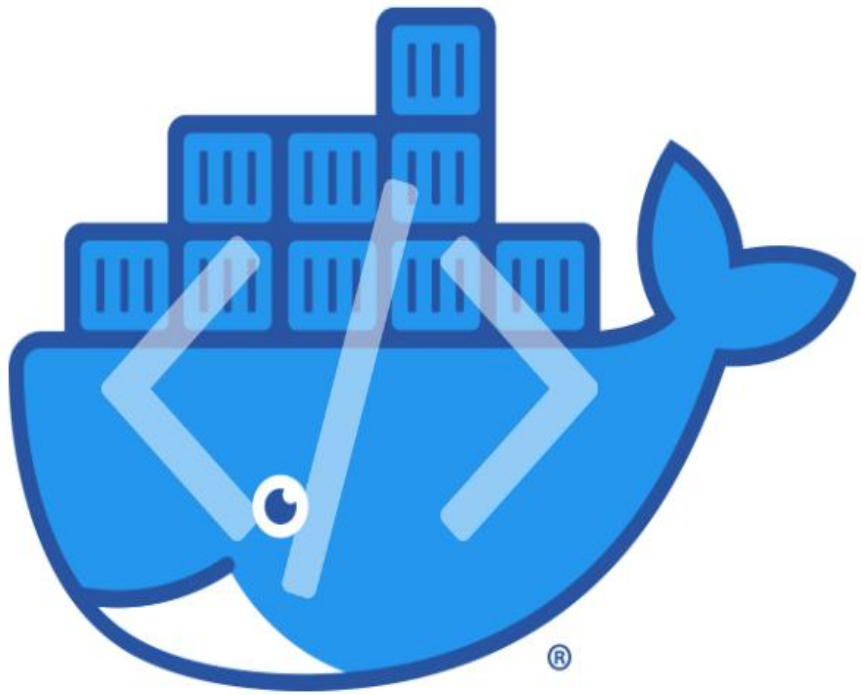
Container Lifecycle

State	Description
Created	A container that has been created but not yet started
Running	A container in its normal working state
Paused	Container whose processes have been paused at the moment
Stopped	A container that is no longer working. Also known as Exited
Restarting	A previously stopped container which is now being restarted
Dead	A container which is no longer in use and is discarded



Docker Command Line Interface (CLI)

Docker offers a Command Line Interface (CLI) to manage and interact with containers. The CLI can also be used to manage remote server operations and the Docker Hub repository operations



Common Docker Commands

Command	Description
`docker run`	Creates a container from an image
`docker start`	Starts an already stopped container(s)
`docker stop`	Stops an active container
`docker build`	Builds a docker image from a dockerfile
`docker pull`	Pulls pre-created images from a specified repository
`docker push`	Push images to the specified repository
`docker export`	Exports containers filesystem to a .tar archive file

Common Docker Commands

Command	Description
`docker images`	Lists the docker images currently on the local system
`docker search`	Searches repository for the specified image
`docker ps`	Lists all active containers running on the system
`docker kill`	Kills an active container without any grace period to shut down its processes
`docker commit`	Creates a new image out of an already active container
`docker login`	Command to login to docker hub repository