DevOps Architecture

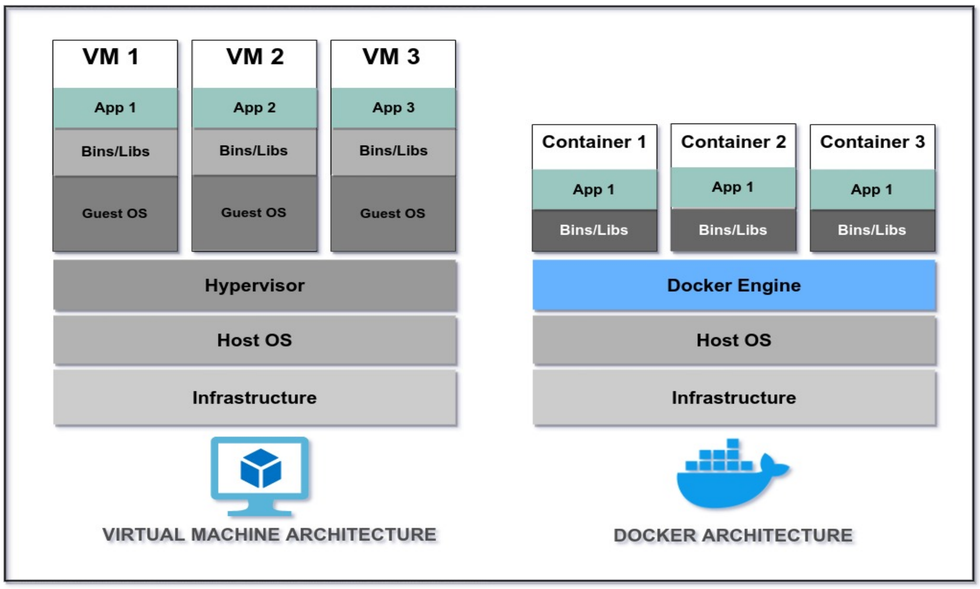
Let's understand how DevOps Platform interacts with the two fundamental technologies Docker & Kubernetes.

- Docker Technology
- Kubernetes Technology

Docker Technology

A simple explanation can be that a container is a form of operating system virtualization that contains all necessary executables, binary code, libraries and configuration files. There are some basic similarities but containers are very different from virtual machines. Compared to virtualization approaches, containers take advantage of a form of operating system (OS) virtualization in which features of the OS. In the case of the Linux kernel, namely the namespaces and cgroups are leveraged to isolate processes and control the amount of CPU, memory, and disk that those processes have access to.

Each container shares the same host OS or system kernel which makes containers more lightweight, portable and with less overhead. A container is a logical partition where users can run applications isolated from the rest of the system. Each application running in the container gets its own private network and a virtual filesystem that is not shared with other containers or the host.



A comparison between Containers and Virtual machines

Docker uses a client server architecture where a **docker client** talks to a **docker daemon** which builds, runs and distributes containers. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

A Docker **image** is a read-only template with instructions for creating a Docker container.

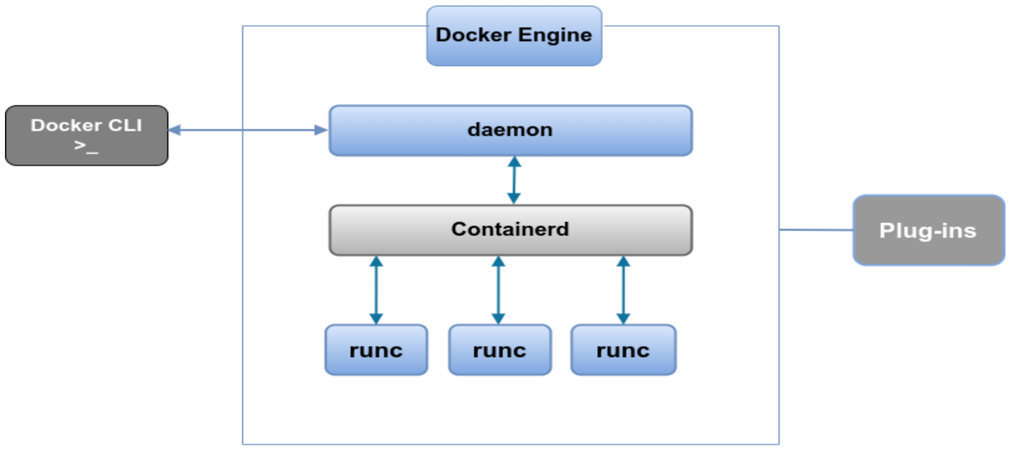
The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage

Docker services.

Docker engine is the core of Docker, the underlying client-server technology that creates and runs the container. Docker engine components are :

- A system with a running daemon process dockerd.
- APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
- A command line interface (CLI) client docker.

A logical representation of docker engine -



> **Note: runc is a small, lightweight CLI wrapper for libcontainer and a standalone container runtime tool**.

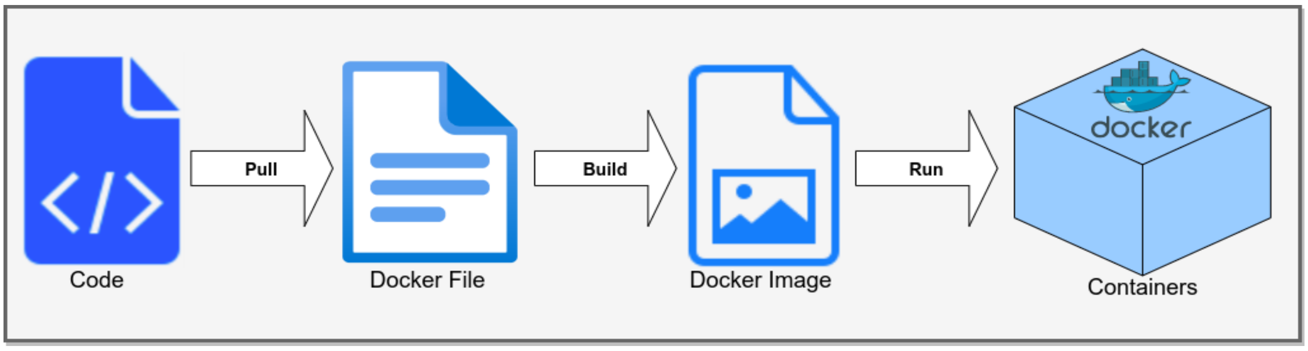
Container Images

A container image is a ready-to-run software package, containing everything needed to run an application. The code and any runtime it requires, all its software dependencies, system libraries, and default values for any essential settings that are required to run the application.

A container is a runnable instance on image. A docker container **registry** is a catalog of storage locations where you can push and pull container images. Docker hub is Docker’s own official image registry which can be a starting point for those who are new to container registry. A Docker or a container **registry** stores Docker images. **Self-Hosted Registries** are one which can be setup as an organisation's own private registry.**Third-party registry** services are fully managed offerings that also give you control over how you manage your images—but without the operational headache of running your own on-premises registry.

Container workflow

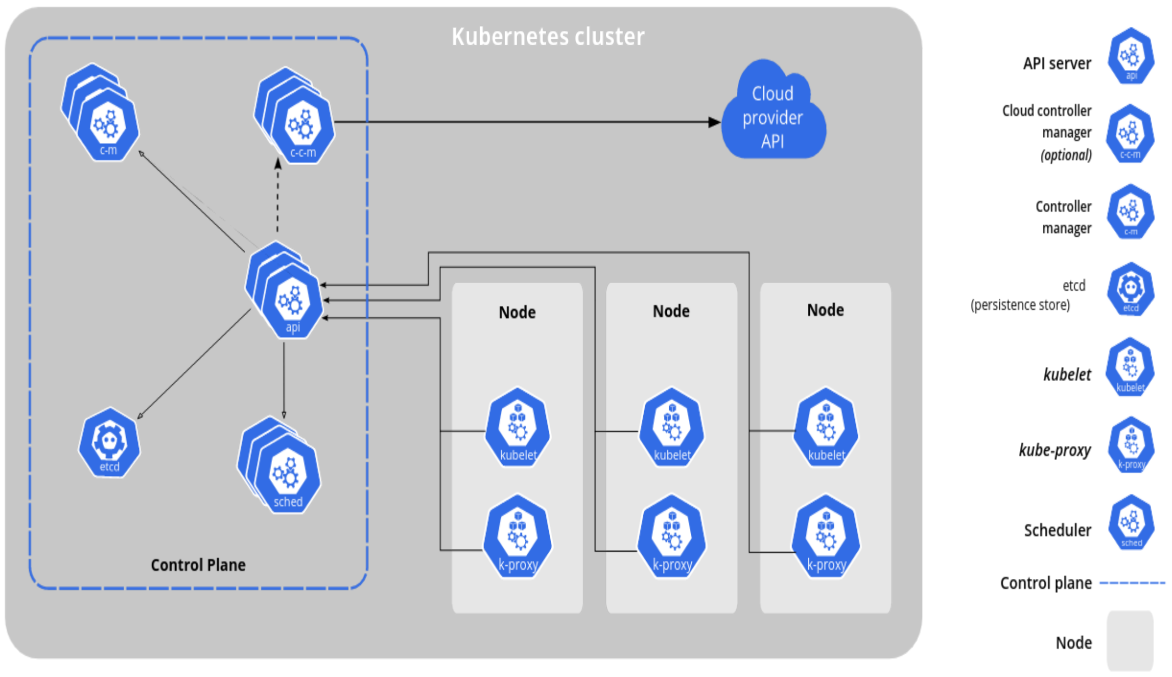
Developers build container images from Dockerfiles and distribute container images from Docker registries.



Kubernetes Technology

Kubernetes when deployed, it's basically a **cluster**. A Kubernetes cluster consists of a set of machines called **nodes**. A node can be a virtual or physical machine. A node is a worker machine. A **worker node** hosts the Pods that are the components of the application workload. The application workload runs on worker nodes.

Please refer to the below diagram which is a high level representation of a Kubernetes cluster.



A **pod** represents a set of running containers in the kubernetes cluster. A Pod is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers

> Every cluster should have at least one worker node.

The **control plane** manages the worker nodes and the Pods in the cluster. A control plane is the container orchestration layer that exposes the API and interfaces to define, deploy and manage the life cycle of a container.

Kubernetes follows master-slave architecture having

- Master Node
- Worker Node

Master Node

The master node manages the Kubernetes cluster, and it is the entry point for all the administrative tasks. You can talk/connect to the master node via the CLI, GUI, or API. For achieving fault tolerance, there can be more than one master node in the cluster.

Below are the main components of a master node-

- **kube-apiserver** - The API server is the front end for the Kubernetes control plane.The main implementation of a Kubernetes API server is kube-apiserver. All communications and operations between the control plane components and external clients are translated into RESTful API calls that are handled by the API server, which then validates the requests, then processes and executes them.

- **kube-controller-manager** - Non-terminating control loops that regulate the state of the Kubernetes cluster are managed by the Control Manager. Now, each one of these control loops knows about the desired state of the object it manages, and then they look at their current state through the API servers. The controller manager makes sure that your current state is the same as the desired state.

- **cloud-controller-manager** - The cloud-controller-manager is a Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API. The cloud-controller-manager is structured using a plugin mechanism that allows different cloud providers to integrate their platforms with Kubernetes.

- **kube-scheduler** - The kube-scheduler is the Kubernetes controller responsible for assigning pods to nodes in the cluster. It's scope is narrow but complex. The scheduler also considers the quality of service requirements, data locality, and many other such parameters. kube-scheduler selects a node for the pod in a 2-step operation known as *Filtering & Scoring*.

- **etcd** - The etcd is a distributed key-value store that is used to store the cluster state. etcd is written in the *goLang*, and it is based on the Raft consensus algorithm. Besides storing the cluster state, etcd is also used to store the configuration details such as the subnets and the config maps.

Worker Node

A worker node is a virtual or physical server that runs the applications and is controlled by the master node. The pods are scheduled on the worker nodes, which have the necessary tools to run and connect them. **The application workload runs on worker nodes**.

The worker node has three components-

- **kubelet** - Kubelet is basically an agent that runs on each worker node and communicates with the master node. The kubelet takes a set of PodSpecs `a version of Pod library` that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet connects to the container runtime using *gRPC framework*. The kubelet connects to the *container runtime interface (CRI)* to perform containers and image operations.
- **kube-proxy** - Kube-proxy is a network proxy that runs on each node in cluster.kube-proxy maintains network rules on nodes. These network rules allow network communication to Pods from network sessions inside or outside of cluster.

- **Container runtime** - The container runtime is the software that is responsible for running containers. Kubernetes supports several container runtimes: *Docker, containerd, CRI-O*, and any implementation of the Kubernetes CRI.

Kubernetes Workloads

Workloads is an application running on Kubernetes inside a pod or a set of pods. Workload resources manage a set of pods. An application can be stateless or stateful. The workload resources configure controllers that make sure the right number of the right kind of pod are running, to match the state specified by you.