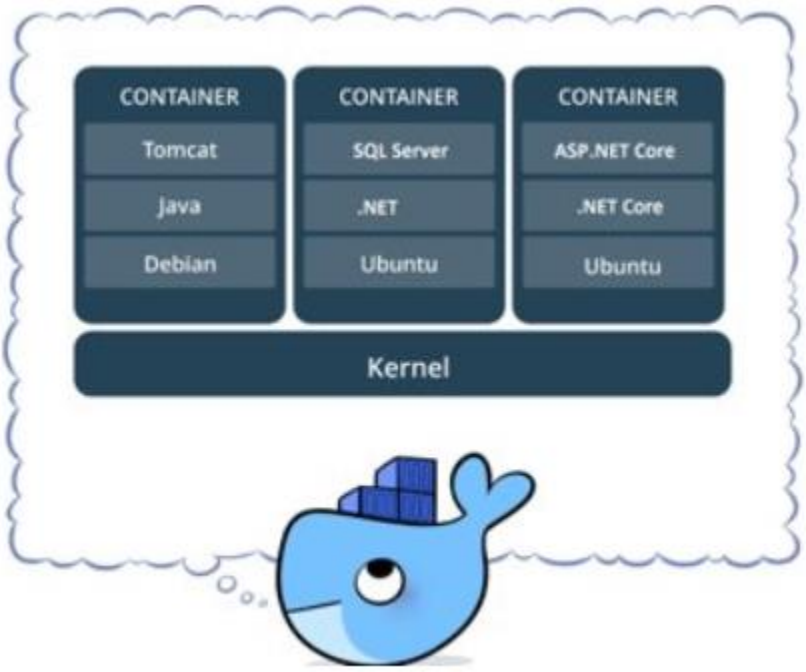
Containers

What is a Container?

Containers bundle application code & dependencies, allowing for rapid and separate execution. Allows for smooth migration between computing environments by using a shared operating system kernel that is compatible with both Windows and Linux Server.



Why Containers?

For Developers

Containers make application deployment easier, save expenses, and automate testing. They handle platform compatibility difficulties, ensuring smooth operation across various environments. Containers also enable the building of microservices, which improves the agility and scalability of applications.

For Administrators

Containers increase release speed and frequency, assuring consistent deployments. They improve the application lifecycle by making setups more efficient and repeatable, as well as guaranteeing consistency across development, testing, & production. Containers also provide scalable systems that adapt effortlessly to shifting workload needs.

Benefits of Containers

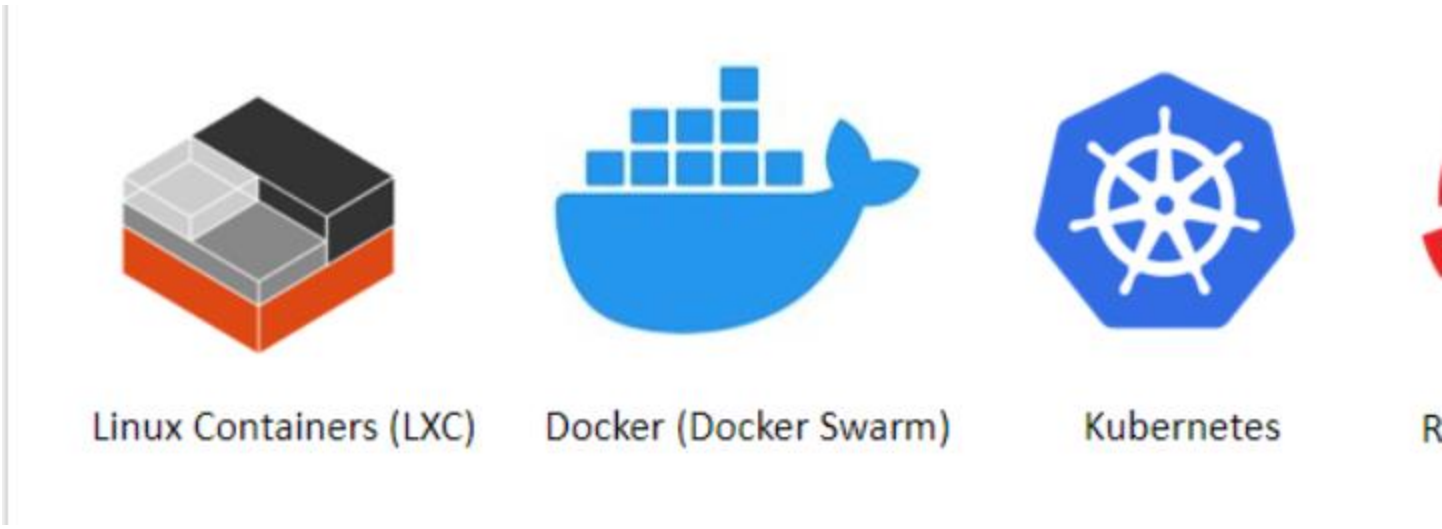
- Effective resource management and control over network interfaces.
- The workload transfer code has been minimized.
- Application and OS isolation ensures consistent environments.
- Platform independence in Linux, Windows, and Mac.
- Effective resource sharing and isolation.
- Quick container functionality control.
- Scalability is seamless.
- Increased developer productivity and pipeline.

Limitation of Containers

- Depends on the host system's kernel.
- Limited to guest operating systems based on Linux.
- Lacks a complete virtualization stack, such as Xen or KVM.
- Security is heavily reliant on the host system, exposing weaknesses.

What is a Container Platform?

Software that makes managing containerized apps possible is called a container platform. A container platform often contains features like orchestration, monitoring, governance, security, & automation.



Containers vs. Container Platforms

Containers, such as Docker, enable lightweight and portable application encapsulation for adaptable deployment. Kubernetes is a container orchestration platform that automates the deployment, scaling, and management of containerized applications at scale.

Which platform types are supported by containers?

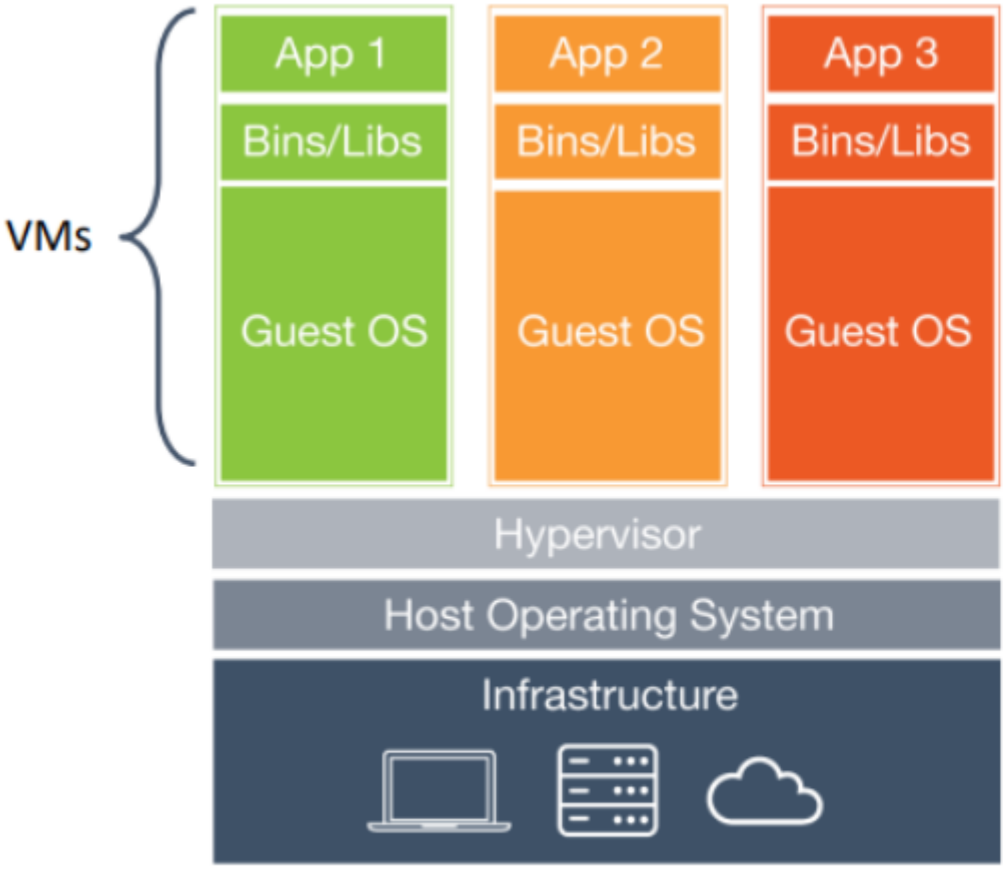
Containers are compatible with the following platforms:

- Windows 10
- Windows Server 2016 (Native Container Support)
- Windows Server 2019 (Native Container Support)
- Mac OS (Native Container Support)
- Linux
- Azure
- Amazon AWS
- Google Cloud

Virtual Machines vs. Containers

- Virtual machines have their own guest OS, which makes them heavier, whereas Docker containers share the host OS, making them lighter.
- VMs provide high isolation, which improves security, whereas containers share the host kernel, which introduces possible safety risks.

- Containers are lighter and use fewer resources, therefore they perform better than virtual machines.
- Containers are easily portable due to the lack of a separate operating system, however, virtual machines are more difficult to migrate due to their standalone operating system.



Source : www.docker.com

Advantages of Containerization over Virtualization

- Lighter and smaller than virtual machines, with a shared operating system kernel for improved resource management and faster boot-up.
- Separate app-specific libraries improve performance and efficiency.
- In comparison to virtualization, this method provides more control and faster implementation.
- Faster boot-up minimizes downtime and increases agility.

- The modular design maximizes resource utilization and facilitates deployment.

Uses of Containers and Virtual Machines

- Choose virtual machines (VMs) for complete OS capability and executing numerous apps on a server.
- Choose Containers to maximize application complexity and resource efficiency.

Docker

What is a Docker?

Docker is a containerization tool that stores everything as images and applications. Using this, we can minimize the normal concerns like dependencies, clashes, and incompatible environments that are a difficulty for distributed applications in which we need to install or upgrade numerous nodes with the same configuration.

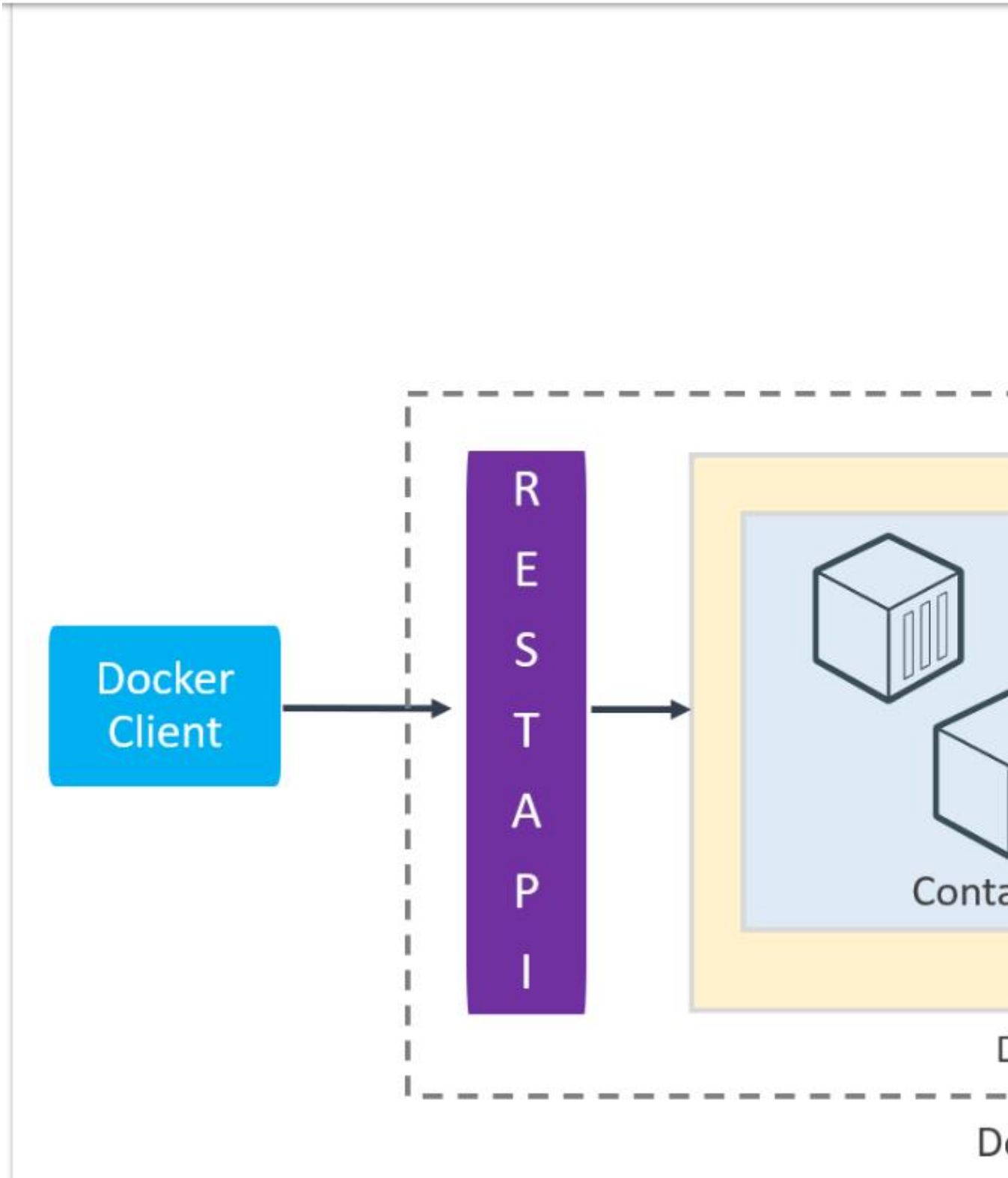
Benefits of Docker

- Cost Savings for Infrastructure
- Standardization and Productivity
- Isolation
- Security
- Improves app lifecycle efficiency and consistency.
- Continuous Testing and Deployment
- Scaling on demand
- Platform Support for Multiple Clouds

Docker Components

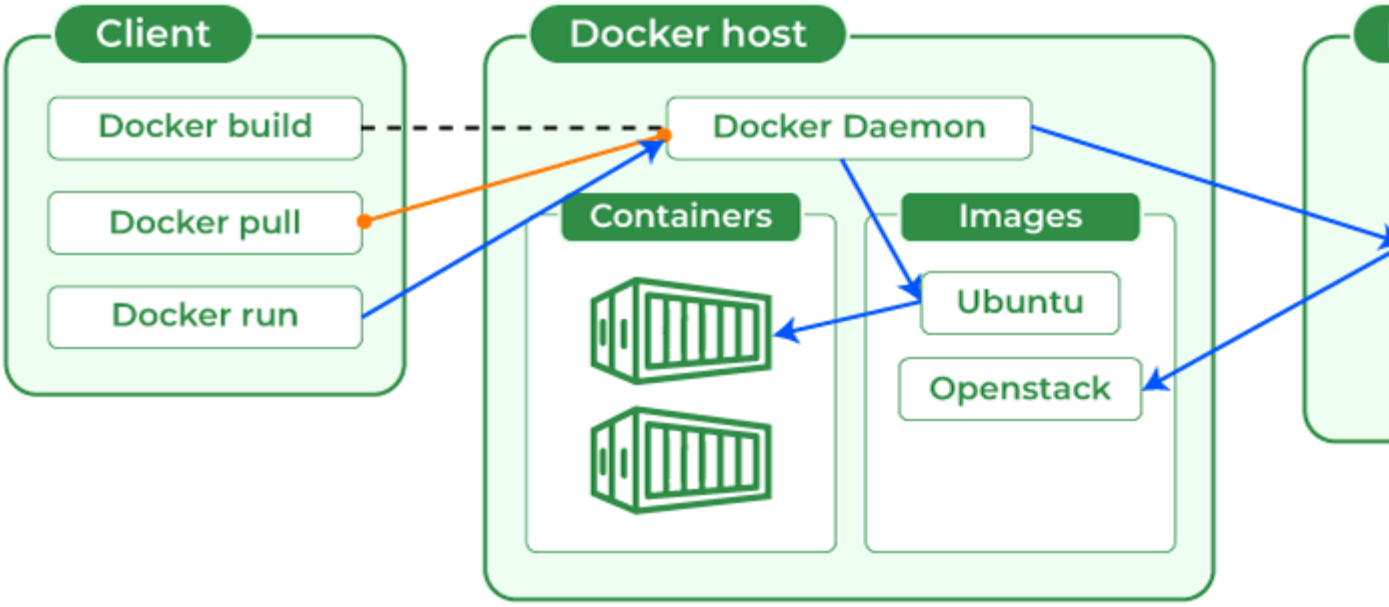
- **Docker Client:** Communicates with Docker.
- **Docker Host:** An environment containing the Docker daemon, containers, images, networks, and storage.
- **Docker Daemon:** Controls Docker components and handles API calls.

- **Docker Registry:** A repository for Docker images that is frequently used in conjunction with Docker Hub.



Working of Docker

- Docker uses a client-server architecture for container management.
- The client can run locally or connect remotely to the daemon.
- Interaction via the REST API over a UNIX socket or network.
- The daemon executes container duties as specified by the client.



Why do we use Docker?

Docker can be used to bundle an application and its dependencies, making it lightweight and easier to release code faster and more reliably. Docker makes it simple to run applications in a production environment. If the Docker engine is installed on the machine, the container can be platform-independent.

Advantages of Docker

- **Cross-platform Consistency:** Docker makes development easier by ensuring consistency across environments.

- **Serverless Storage:** Docker is highly efficient in the cloud, removing the need for large server farms.
- **High-speed build and deployment:** Docker's stacked containers allow for rapid automation and deployment.
- **Flexibility and Scalability:** Docker allows for language freedom and easy scaling for optimal performance.

Disadvantages of Docker

- **Outdated Documentation:** Documentation is out of current due to Docker's rapid development, making it difficult to access useful information.
- **Steep Learning Curve:** Docker transformation takes a long time and effort because of its complexity and regular updates.
- **Security Concerns:** Docker's shared OS poses security vulnerabilities, necessitating additional isolation techniques.
- **Limited Orchestration:** Unlike Kubernetes, Docker lacks robust automation and requires third-party management solutions.

Benefits of Docker Desktop

- Docker Desktop enables rapid, safe development and delivery of containerized applications.
- Includes the Docker App, Developer Tools, and synchronization with the production Docker engine and Kubernetes.
- Supports verified templates and pictures in a variety of languages and tools.
- Docker Hub simplifies development by providing safe storage, auto-building, CI, and collaboration.

What is included in the installation of the Docker Desktop?

The Docker desktop installation includes:

- Docker Engine
- Docker CLI client
- Docker Compose
- Docker Machine
- Dashboard

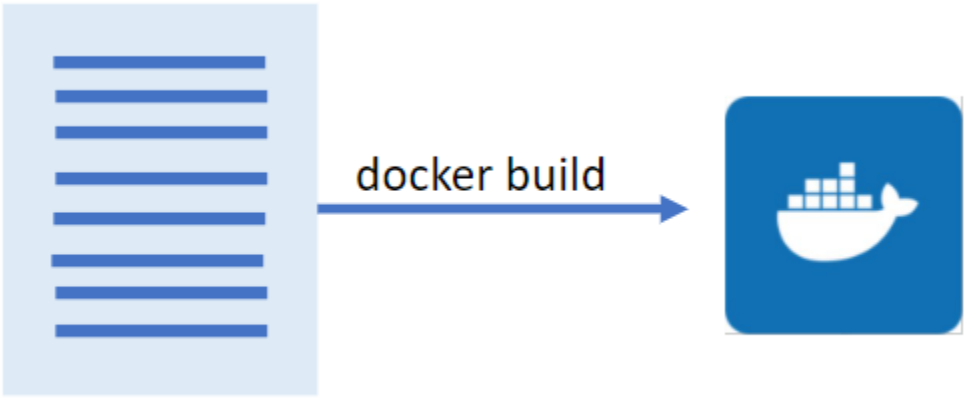
Docker Community vs. Docker Enterprise

- **Docker Community Edition:** Free, stable releases every three months, with monthly upgrades; allows application creation, building, and deployment using Docker's CLI and API.
- **Docker Enterprise:** Subscription-based; provides greater management and security features; uses the Universal Control Plane for application management.

Docker Basics

Docker File

The text file contains a sequence of commands for creating a Docker image. Its content structure and command syntax identify it as a Dockerfile even without an extension.



Dockerfile commands

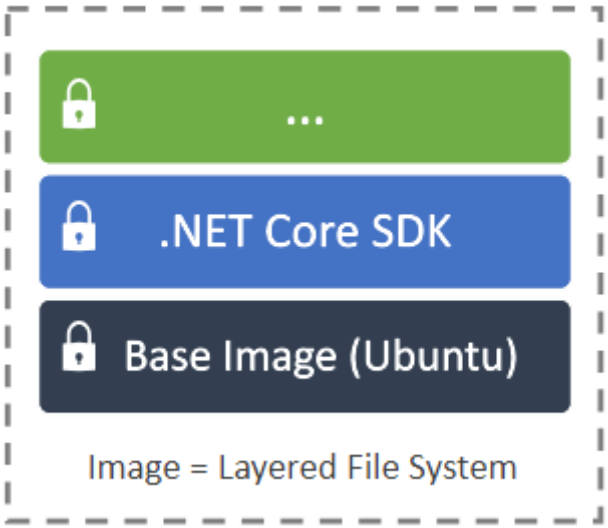
Following are the commands of Dockerfile:

- FROM
- COPY
- ADD
- RUN

- CMD
- ENTRYPOINT
- MAINTAINER
- VOLUME
- ENV

Docker Image

A Docker image is a small, self-contained package that has all of the necessary components to run an application. It is assembled using the Docker build command and stored in registries such as Docker Hub, which contain layered references to other images.



Create a Docker Image

To define the Docker machine, run the command: eval "$(docker-machine env default)" to create modified Docker images.

Pull a Docker Image

To pull the Docker image, use the command "# docker pull <repository>".

Change the Docket Image Name

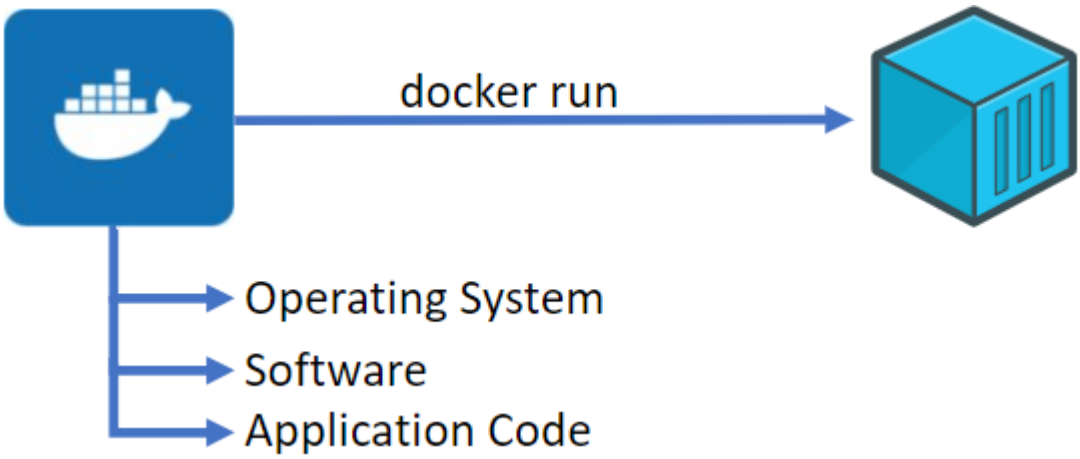
To modify the name of a Docker image, use the command "$ docker tag image id new name".

Remove the Docker image

To remove a Docker image, use the command: $ docker rmi image id.

Docker Container

A container is a running version of a Docker image. A docker container is a separate and secure shipping container. Run using the docker run command.



Convert an Image to a Container

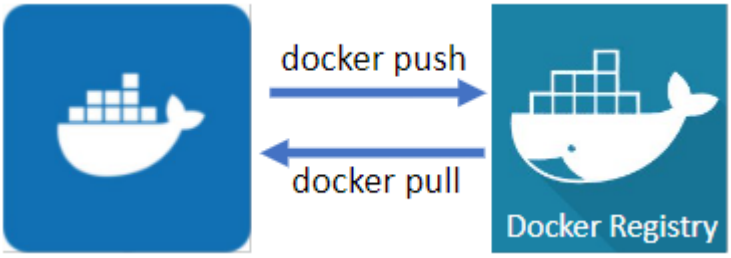
To convert an image into a container, execute the command "#docker run --image name".

Set a Connection to the Docker Containers

The Docker network functionality allows you to connect to Docker containers using either user-defined bridges or the default bridge.connect.

What is Docker Registry?

A Docker registry is a system that stores and distributes Docker images under particular names.



Different types of Registries

- **Docker Hub** - (Free for public images and Paid for private images)
- **Docker Trusted Registry** - (on-prem or on-cloud)

Docker Registry Uses

- The Docker registry allows us to save our images.
- The development process can be automated.
- Image can be secured using a private docker registry.

Types of Docker Registries

- DockerHub
- Amazon Elastic Container Registry (ECR)
- Google Container Registry (GCR)
- Azure Container Registry (ACR)

Working on Docker Registry

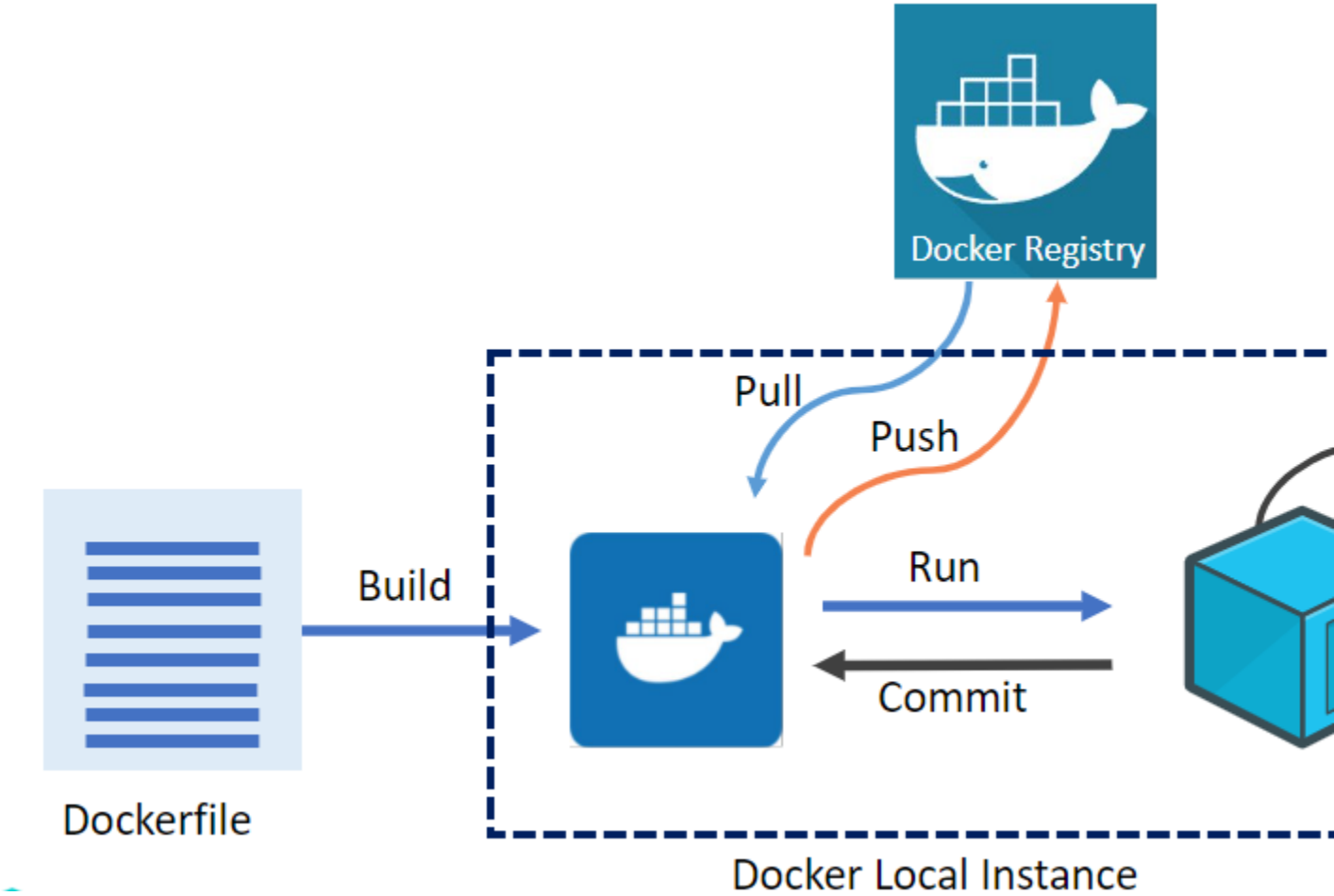
- The Docker Registry is where Docker images are stored and distributed.
- Users contribute photos that include version and name tags.
- Others seek for and download files from the registry.
- It is possible to host it yourself or use a cloud service.

Docker Hub

Docker Hub, a cloud-based repository, provides a platform for both public and private container images, allowing for easy development, testing, storage, and distribution.

Docker Container Life Cycle

Docker containers have a lifecycle that includes creation, running stopping, and removal. Containers are built from Docker images, run as separate instances, can be terminated or paused, and removed when no longer required.



Process (lifecycle) of a Docker Container

The Docker container life cycle method is listed below:

- Create a container
- Run the container
- Pause the container (optional)
- Unpause the container (optional)
- Start the container
- Stop the container

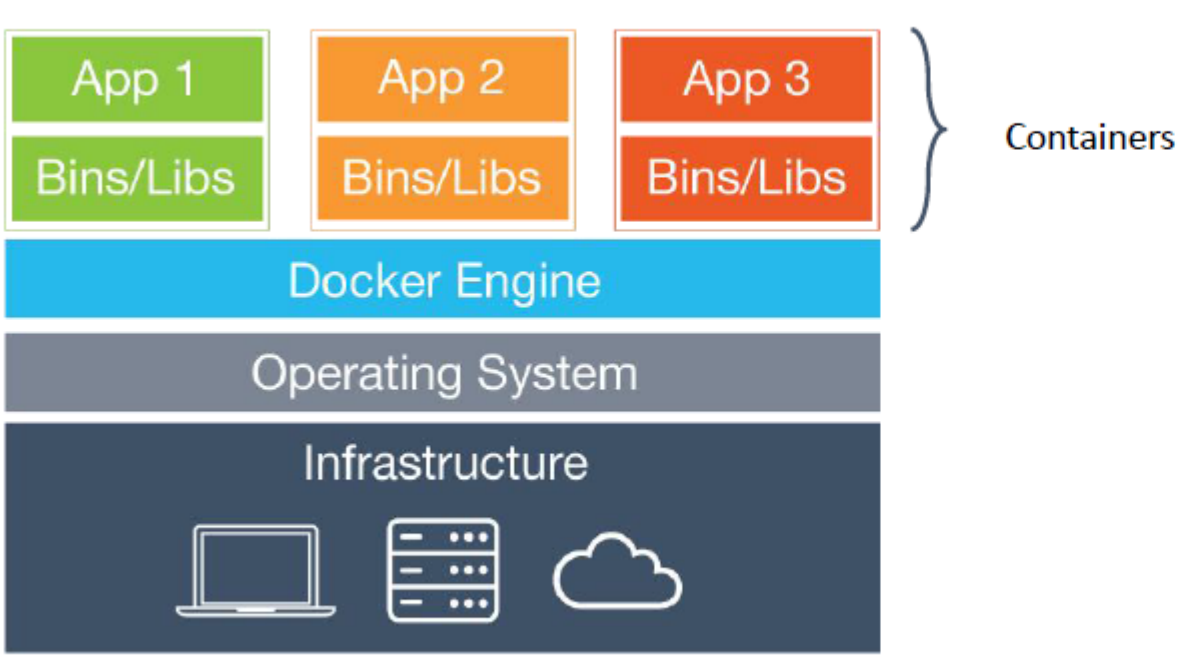
- Restart the container
- Kill the container
- Destroy the container

Valuable States of Docker Container

- Running
- Paused
- Restarting
- Exited

Docker Engine

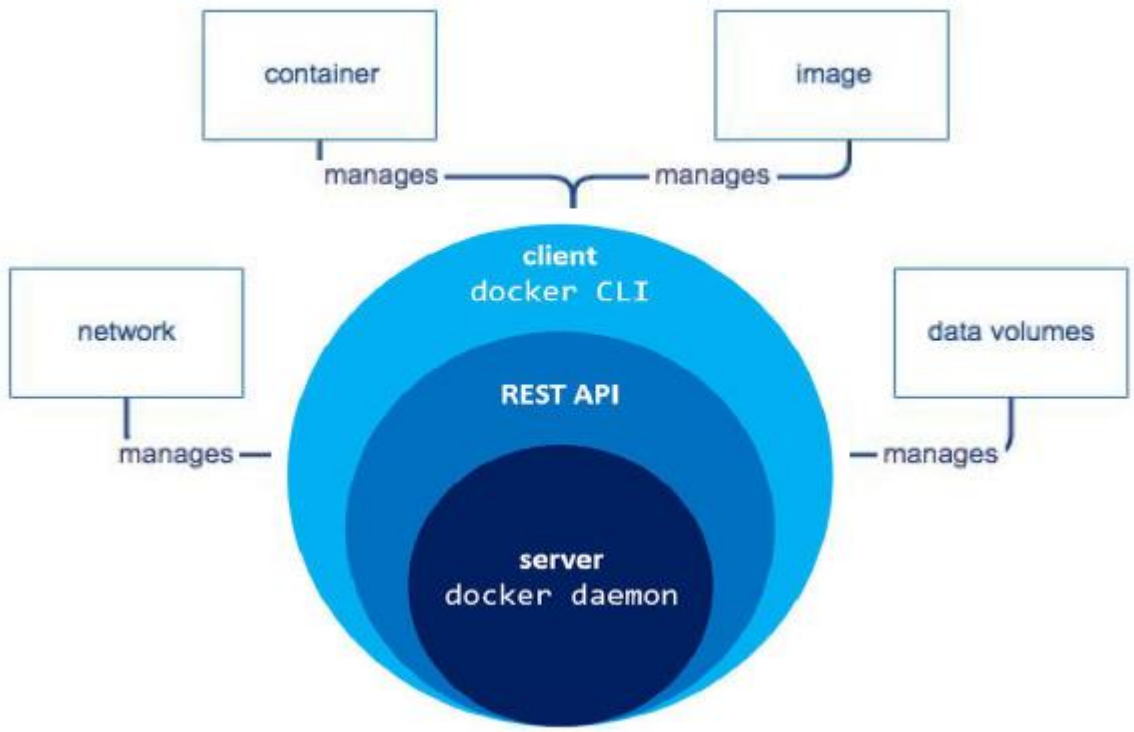
The Docker engine, sometimes known as Docker, is a client-server application that creates and runs containers using Docker components.



Source : www.docker.com

What is the Docker CLI?

Docker CLI is a command-line interface that allows you to interact with the Docker engine to create, build, and execute Docker images, containers, volumes, and networks. Docker CLI is used to interact with the Docker Engine through APIs.



Basic Docker Commands

There are a few Docker commands. If we simply run docker from the console, we can access all of the commands.

- Attach
- Build
- Commit
- Cp
- Create
- Diff

Benefits of Docker CLI

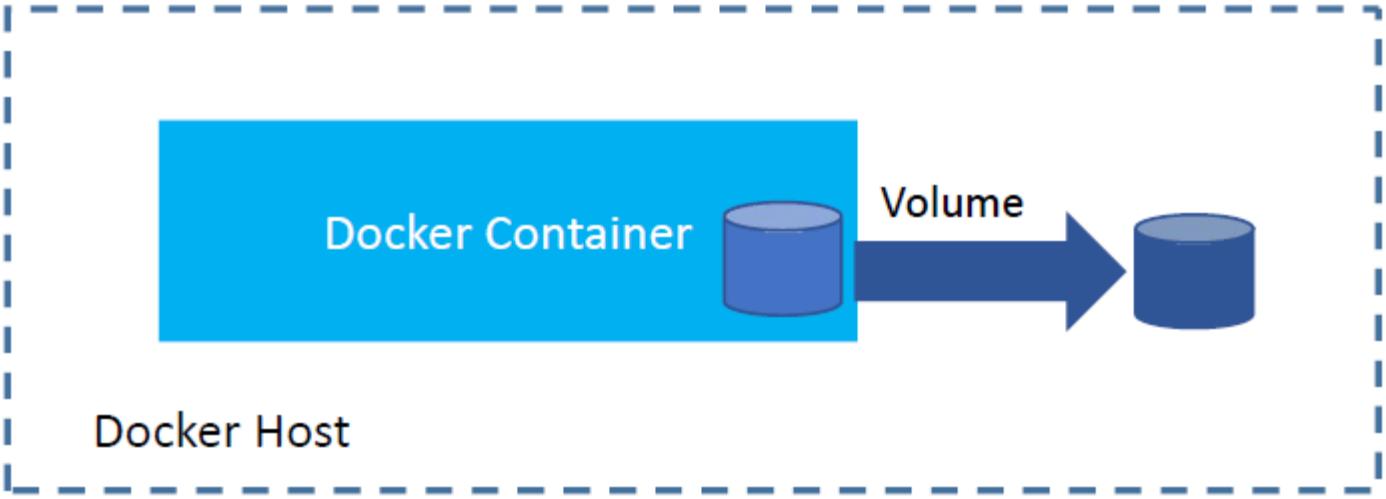
- **Portability**: Guarantees consistency in different environments.
- **Efficiency**: Graphical applications that use fewer resources than traditional ones.
- **Scriptable**: It can be used in conjunction with other command-line tools to automate tasks.
- **Flexibility**: Enables more advanced commands and choices that are not available in the GUI.
- **Customization**: Users can develop their scripts and workflows to fit their needs.

Limitations of Docker CLI

- **Learning curve**: It may be challenging for beginning programmers to use.
- **Security**: Since it is frequently run in root mode, it may cause unintended side effects.
- **Limited Visual Feedback**: It is difficult to execute and use applications that require continual visual feedback.
- **Debugging**: Without visual help, issues can be more difficult to troubleshoot.
- **Manual Updates**: To receive the most recent features and security patches, users must manually upgrade the Docker CLI.

Docker Volumes

Docker volumes store container data, allowing several containers and the host system to share information without increasing container size. They survive container deletion, ensuring data continuity & flexibility in storage management. Docker volumes are useful for running databases as containers to store data.



The command for creating and deleting volumes in Docker

We can create a volume using the following command

$ docker volume create <volume name>

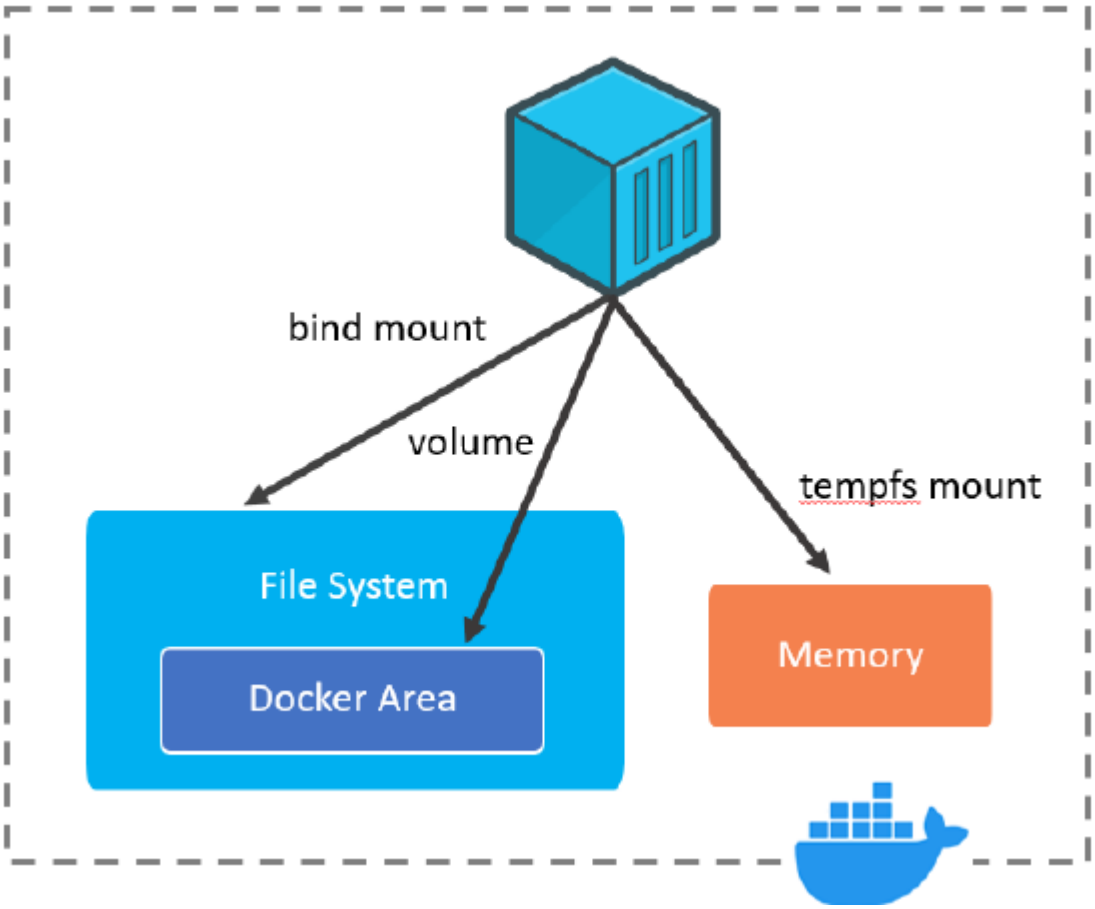
We may remove the volume by using the command below

$ Docker volume prune

Different Kinds of Volume Mount Types in Docker

The volume mount type used by Docker is as follows:

- Bind mounts
- Volumes
- tmpfs mounts



To verify if a volume has been properly created and mounted in Docker

To ensure that the volume was properly built and mounted, use docker inspect devtest.

Docker Build and Docker Engine

What is Docker Build?

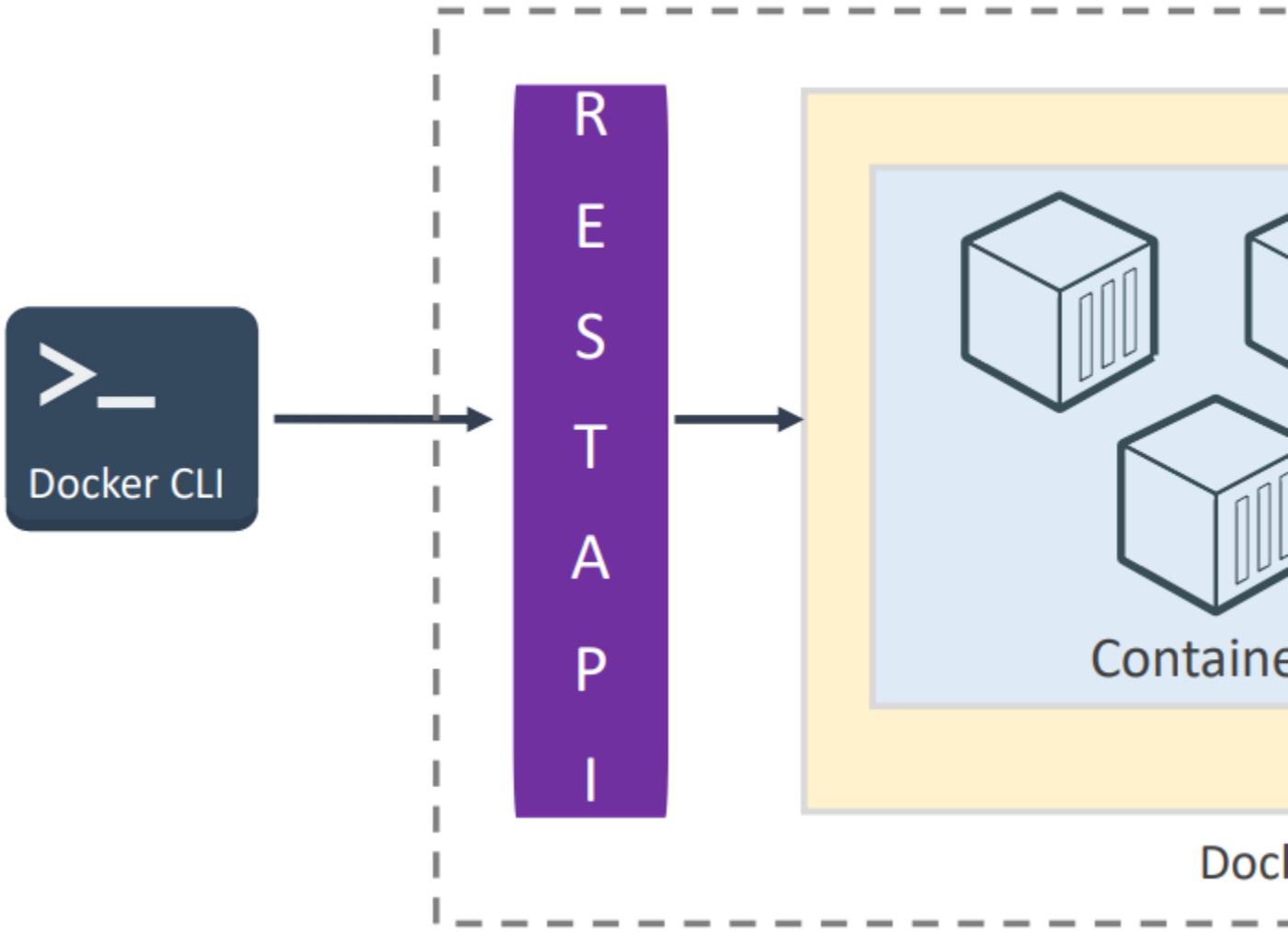
The docker build command creates Docker images using a Dockerfile and a "context". The set of files in the supplied PATH or URL constitutes the build's context.

Docker Build Best Practices

- Choose the smallest base image (alpine images) possible.
- Reduce the amount of clutter in your images.
- Create multi-stage builds.
- Try to make images with common layers.
- Tagging with semantic versioning.
- Try not to install unneeded items or dependencies.
- Create a ".dockerignore" file to remove unnecessary things from the build context.

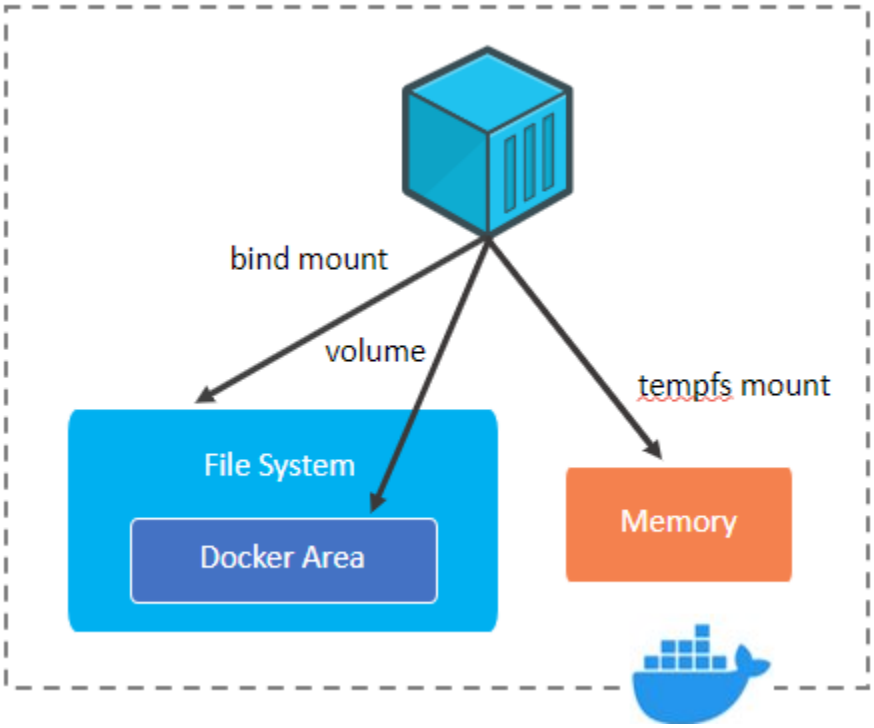
Docker Engine

The Docker engine, sometimes known as Docker, is a client-server application that creates and runs containers using Docker components. Runs on different Linux (CentOS, Debian, Fedora, Oracle Linux, RHEL, SUSE, and Ubuntu) and Windows Server OS.



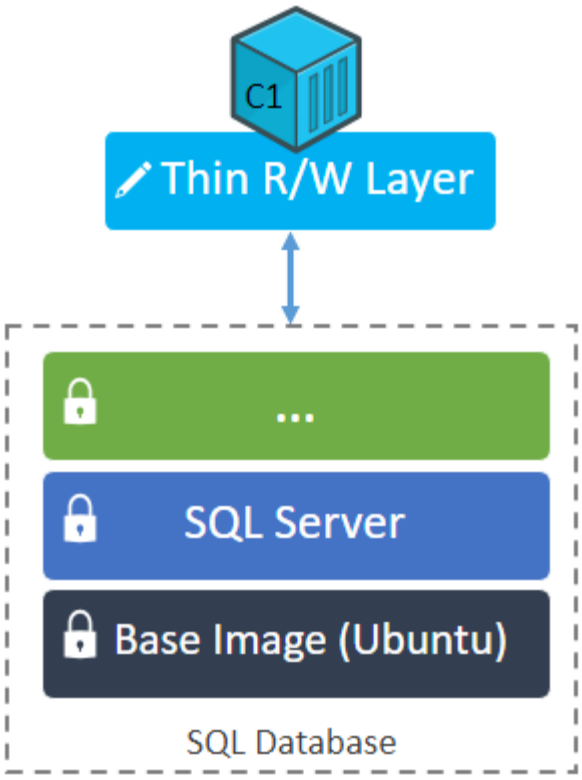
Docker Storage

Containers, as opposed to images, store data in writable layers. When a container is erased, the readable layer is removed, leaving the underlying image intact. Docker volumes enable data exchange between containers, maintaining consistency and accessibility across deployments.



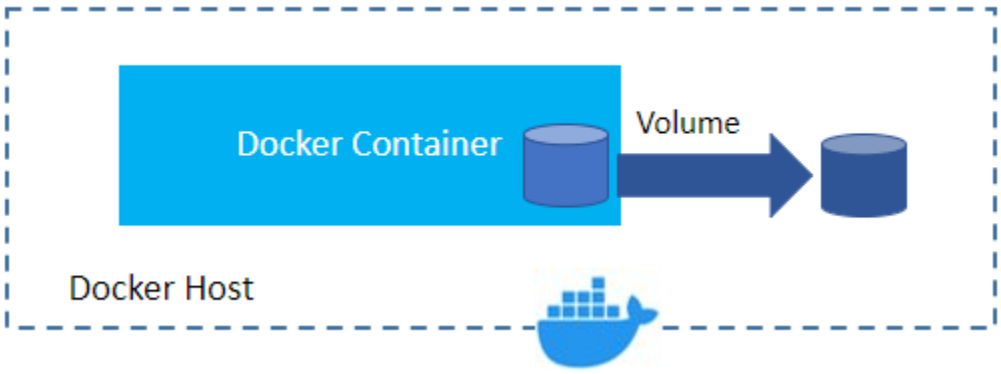
Container R/W Layer

When we construct a Docker container, we add a read-write (R/W) layer on top of the image's read-only (R/O) layer. This prepares the container to run the programme by pulling the image, setting the environment variables, configuring the entry point, and so on.



Docker Volumes Use Case

Docker volumes are helpful when you run a database as a container for storing your data.



Docker Networking

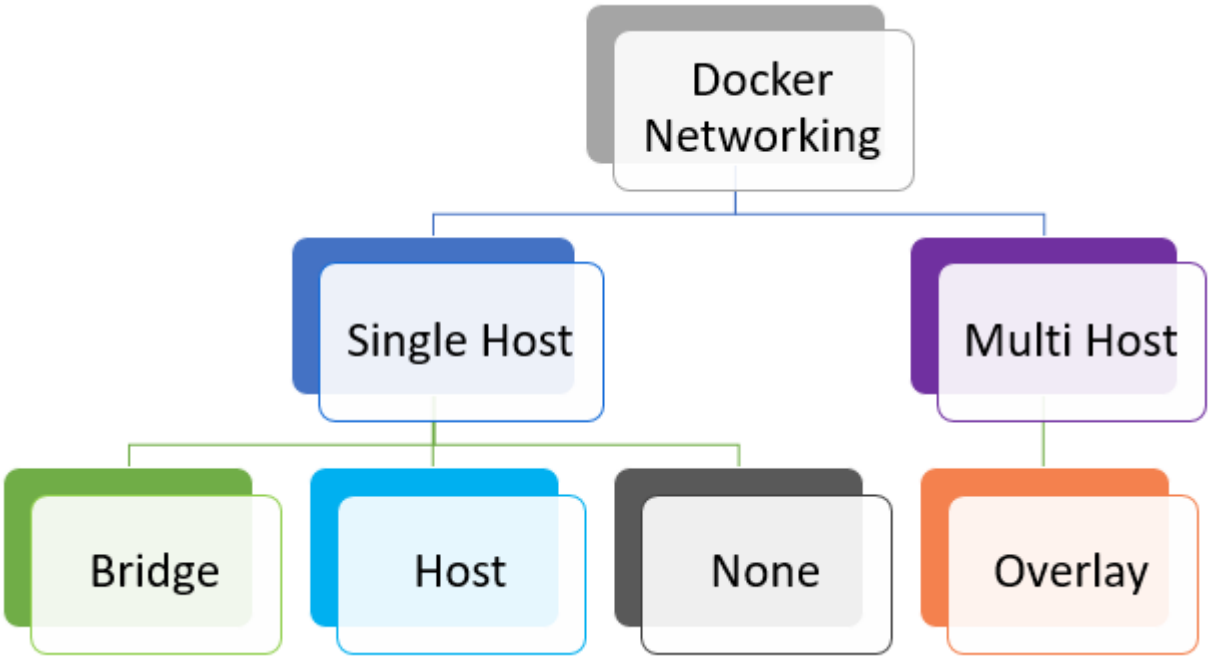
Docker Networking

A network is a collection of two or more devices that can connect, whether physically or virtually. The Docker network is a virtual network developed by Docker to facilitate communication between Docker containers.

Different types of Docker Networking

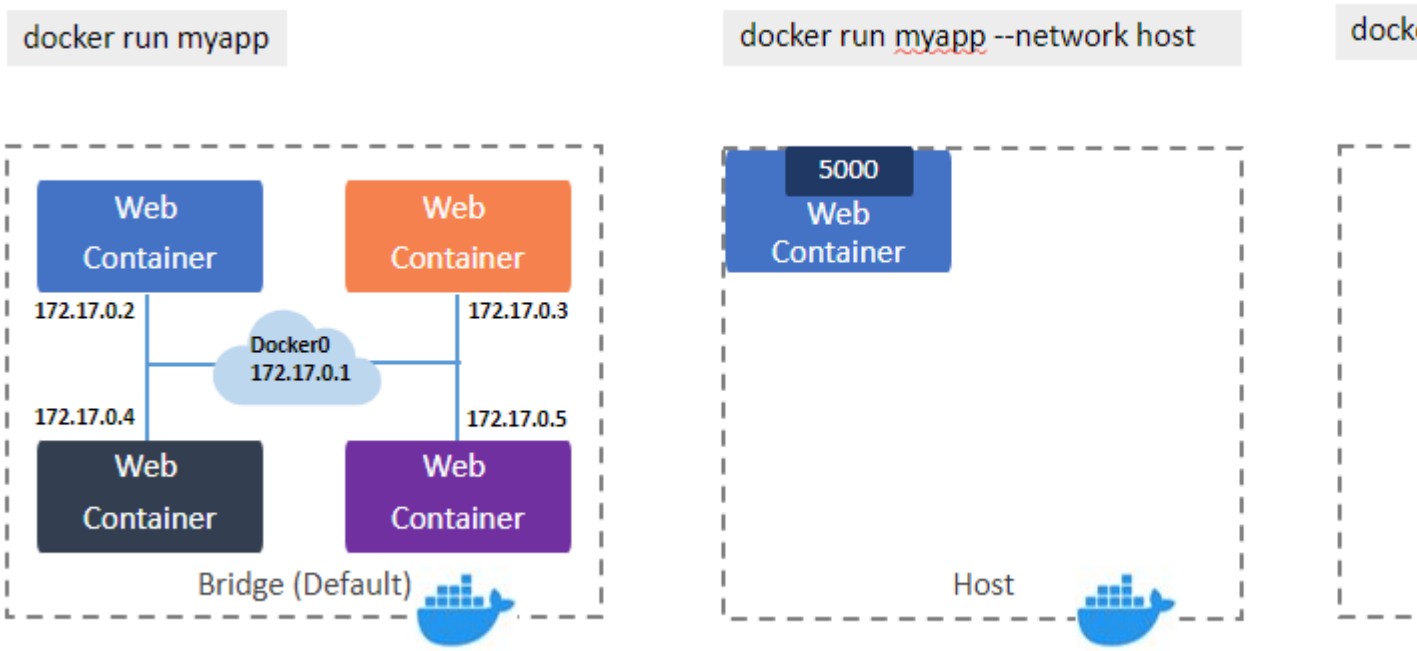
Following are the types of Docker Networking:

- Single Host
- Multi-Host



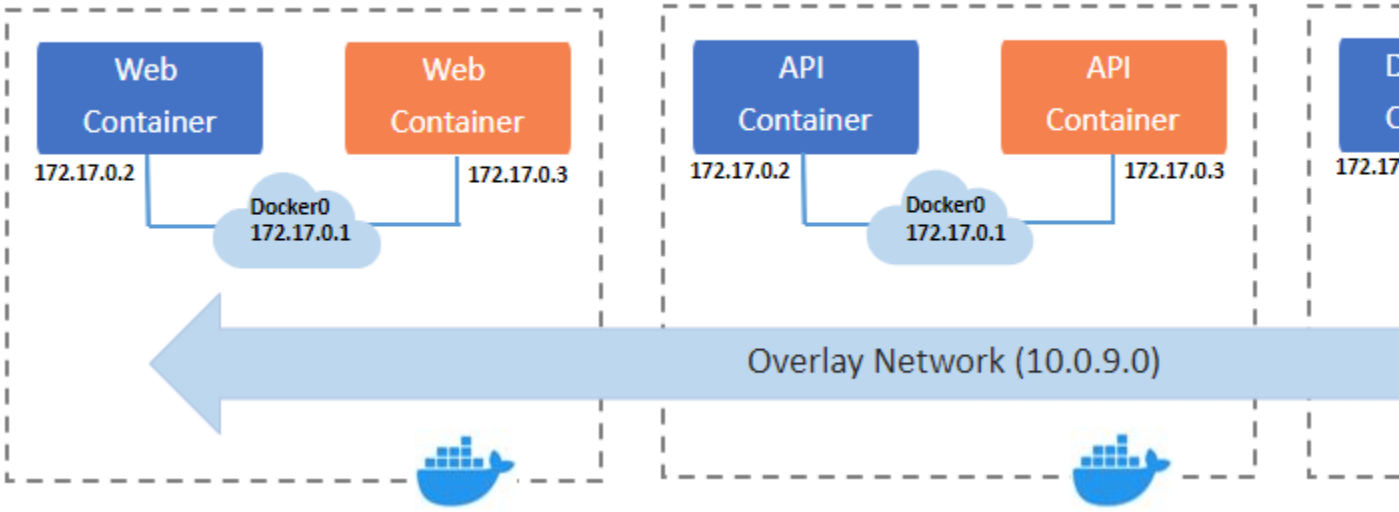
Docker Single Host Networking

Docker Single Host Networking provides communication between containers on the same host via a virtual network bridge, allowing for smooth interaction while isolating them from external networks.



Docker Multi-Host Networking

Docker Multi-Host Networking enables containers running on various hosts to communicate with one another via overlay networks, offering seamless connectivity across distributed settings while abstracting underlying network difficulties.



Network Drivers

Docker network drivers define networking behavior, allowing containers to communicate with one another.

Types of Network Drivers

- Bridge
- Host
- Overlay
- Ipvlan
- Macvlan

Docker Network Working

- Docker uses your host's network stack, altering iptables rules for traffic routing to containers to ensure isolation.
- On Linux, packet filtering is controlled by iptables, and Docker automatically manages container traffic rules.
- Each Docker container has its network namespace for isolation, with virtual network interfaces allowing communication across the host's network.

IP Address

An IP address consists of four numbers separated by periods. In general, it looks like 147.181.183.86. The numbers will range from 0 to 255. So, 255 is the greatest number in an IP address, and 0 is the smallest value.

Subnet

Similar to an IP address, and used to group IP addresses. The common subnet mask is 255.255.255.0. These numbers range from zero to 255.

```
192.168.1.0 - Subnet Address
192.168.1.1 - usually the gateway
192.168.1.2
192.168.1.3
192.168.1.4
192.168.1.5
192.168.1.6
...
192.168.1.252
192.168.1.253
192.168.1.254
192.168.1.255 - Broadcast Address
```

Bridge

The default bridge driver in Docker creates a private network for a single host, with each container having its network namespace. It allows containers to communicate within the bridge but limits communication between containers on different bridges, while also permitting external access via port mapping.

Host

The Host Network Driver allows containers to use the host's network stack directly, removing network isolation between the host and the containers. It is useful for deploying one or more containers on a single host while ensuring that each container uses a unique port.

Overlay

The Overlay driver enables communication between Docker daemons in a swarm, resulting in simple and secure multi-host networking. It facilitates seamless communication between all containers in the overlay network.

Ipvlan

IPvlan in Docker allows each container to have its own unique MAC address and IP address on the same subnet, allowing network isolation while directly accessing the host's physical network interface.

Macvlan

Macvlan in Docker assigns a unique MAC address to each container on the host's physical network, allowing them to communicate with external networks as if they were physical devices. This provides network isolation and boosts performance for containerized apps.

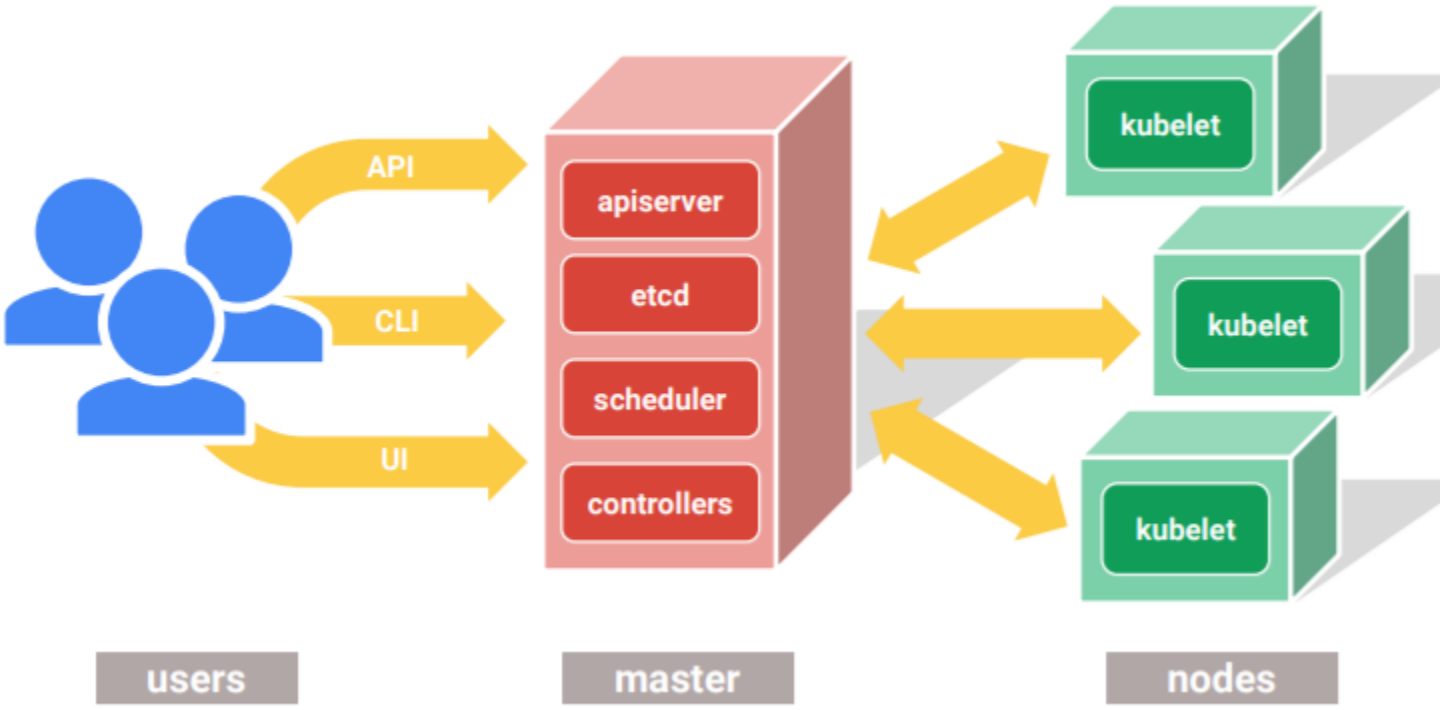
Docker Network Commands

- docker network ls
- docker network create
- docker network inspect
- docker run --network
- docker network connect
- docker network disconnect
- docker network rm
- docker inspect -f "{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}"

Introduction to Kubernates

What is Kubernetes?

Kubernetes is an open-source container orchestration solution that streamlines container deployment, scalability, and load balancing. It supports a variety of platforms, including on-premises bare metal, OpenStack, and major public clouds such as Google, Azure, and AWS.



Why use Kubernetes?

Microservices provide higher adaptability than monolithic services, which is pushing their acceptance in the rapidly evolving IT world. Deploying applications in containers offers robustness, with Kubernetes managing containers for zero downtime, making it necessary for both micro and monolithic apps.

Kubernetes Features

- Automated Scheduling
- Self-Healing Capabilities
- Automated rollouts & rollback
- Horizontal Scaling & Load Balancing
- Offers enterprise-ready features
- Application-centric management
- Auto-scalable infrastructure

Kubernetes Disadvantages

- Kubernetes can be an overkill for simple applications

- Kubernetes is very complex and can reduce productivity
- The transition to Kubernetes can be complicated
- Kubernetes control plane only supports the Linux platform
- Only Kubernetes V1.14 is supported by Windows Server 2019
- Supports clusters with up to 5000 nodes
- Supports 150000 total pods
- Supports 300000 total containers
- Supports 100 pods per node
- 2 GiB or more of RAM per machine for Kubernetes
- At least 2 CPUs on the machine that you use as a control-plane node
- Kubernetes allows administrators to set quotas, in namespaces

Object types in Kubernetes

Kubernetes supports the following key objects:

- Pod
- Node
- Service
- Replica Set
- Namespace
- Replication Controller
- Deployments
- Volume
- Secret
- Kubectl

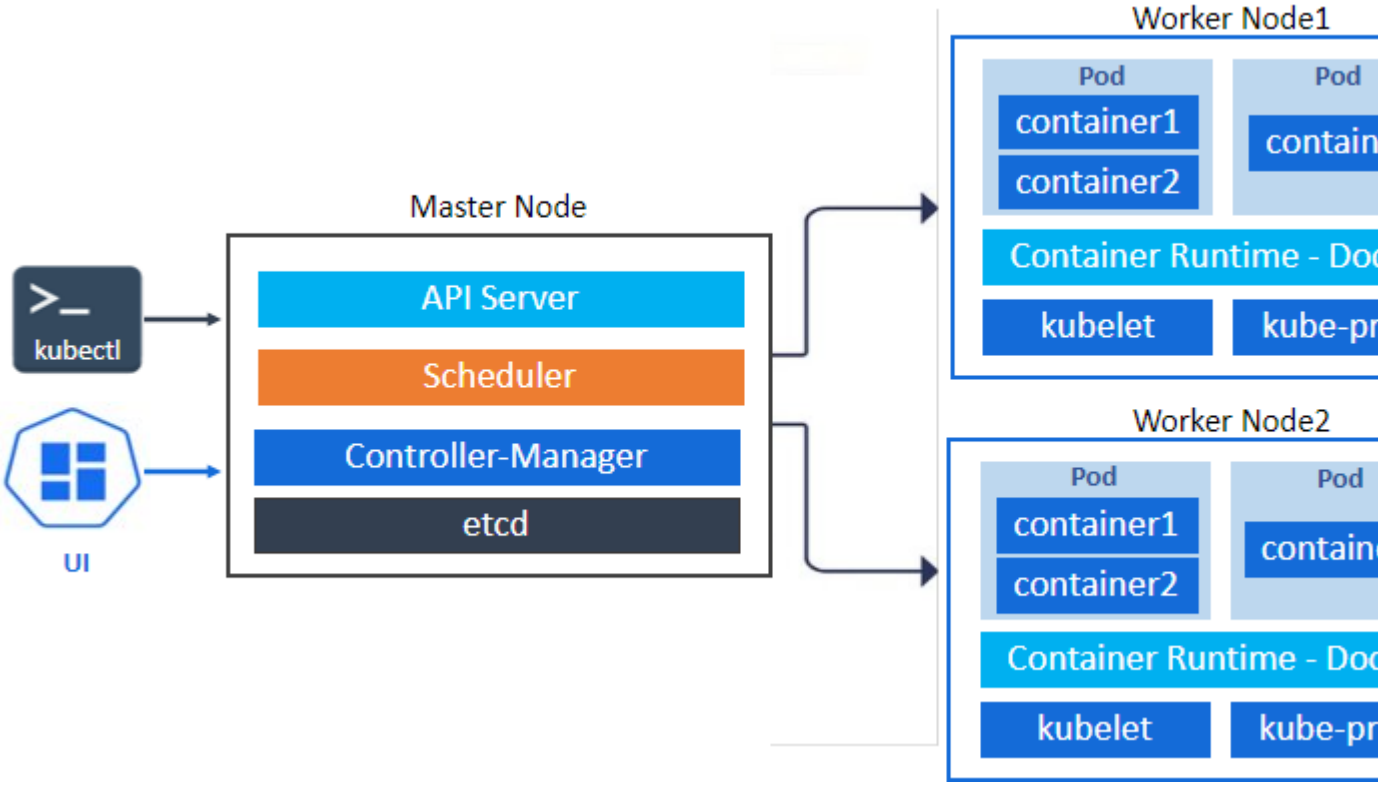
Kubernetes Alternatives

- Rancher
- Nomad
- Docker Swarm
- Cloud Foundry
- AWS Fargate
- Apache Mesos
- OpenStack
- Docker Compose

Kubernetes Architecture

- Kubernetes uses a client-server architecture, with a master on one system and nodes on other Linux machines.

- In a master-slave system, the master manages Docker containers across several nodes.
- A Kubernetes cluster consists of master and worker nodes.
- The Kubernetes master helps developers as they deploy apps in Docker containers.



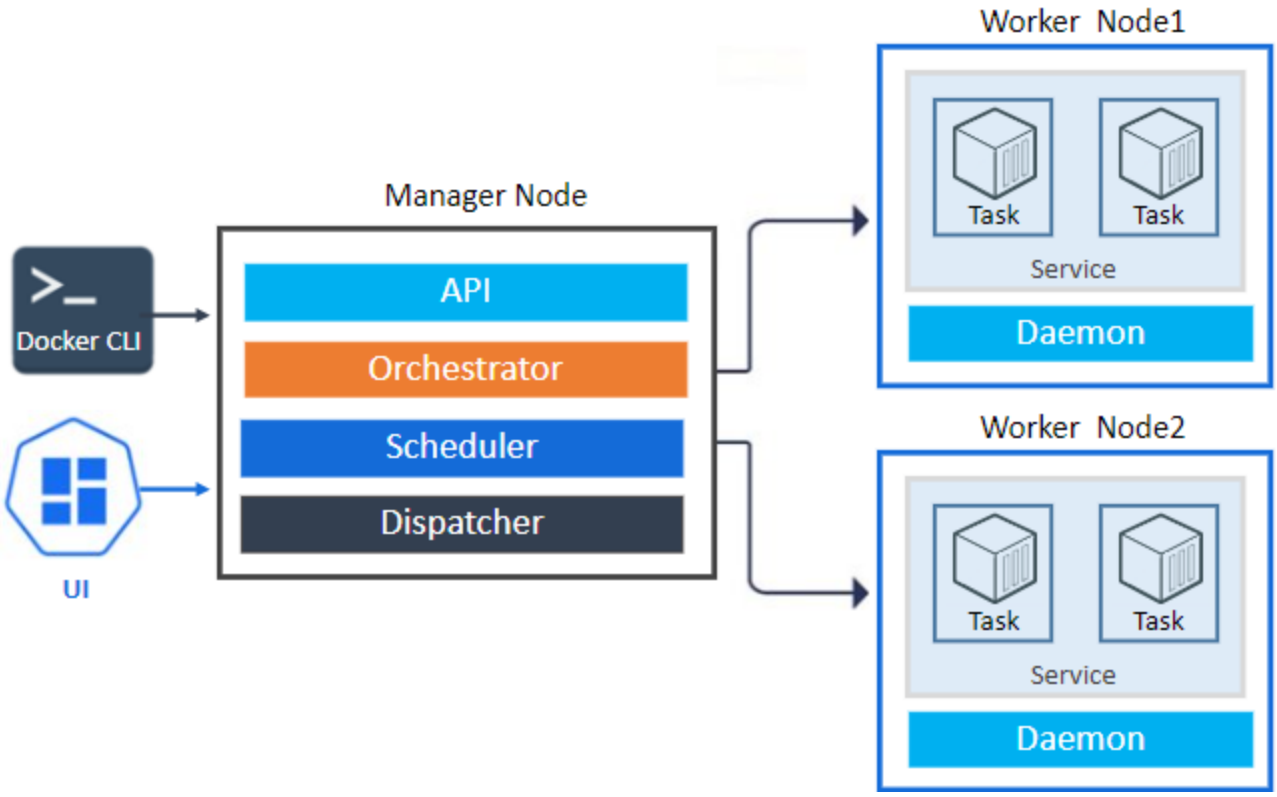
Kubernetes Applications

- Microservice Architecture.
- Cloud-Based Development.
- Continuous integration & delivery.
- Hybrid and multi-cloud deployments.
- High-performance computing.
- Edge computing.

Docker Swarm

Docker Swarm is a native Docker clustering tool that converts a collection of Docker servers into a single virtual host with a management and worker nodes. It helps with

application deployment, scalability, service discovery, and load balancing across several hosts, easing container administration operations.



Docker Swarm Architecture

- **Manager Nodes**: Manage orchestration, clusters, and task distribution.
- **API**: accepts CLI commands and generates service objects.
- **Orchestrator**: Handles service object reconciliation and task creation.
- **Scheduler**: Assigns nodes to tasks according to resource availability.
- **Dispatcher**: Supervises worker nodes to ensure task completion.
- **Worker Nodes**: Execute containers and complete given tasks.
- **Tasks**: Tasks are individual containers maintained by the Swarm.
- **Service**: A collection of containers that share configurations using Docker's Swarm mode.

Docker Swarm Advantages

- Docker Swarm is simple to set up, making it perfect for beginners in container orchestration.
- Lightweight, with automated load balancing within containers.

- Integrates seamlessly with the Docker CLI and existing Docker tools such as Docker Compose.
- Enables intelligent node selection for optimal container deployment.
- Provides its own Swarm API for management.

Docker Swarm Disadvantages

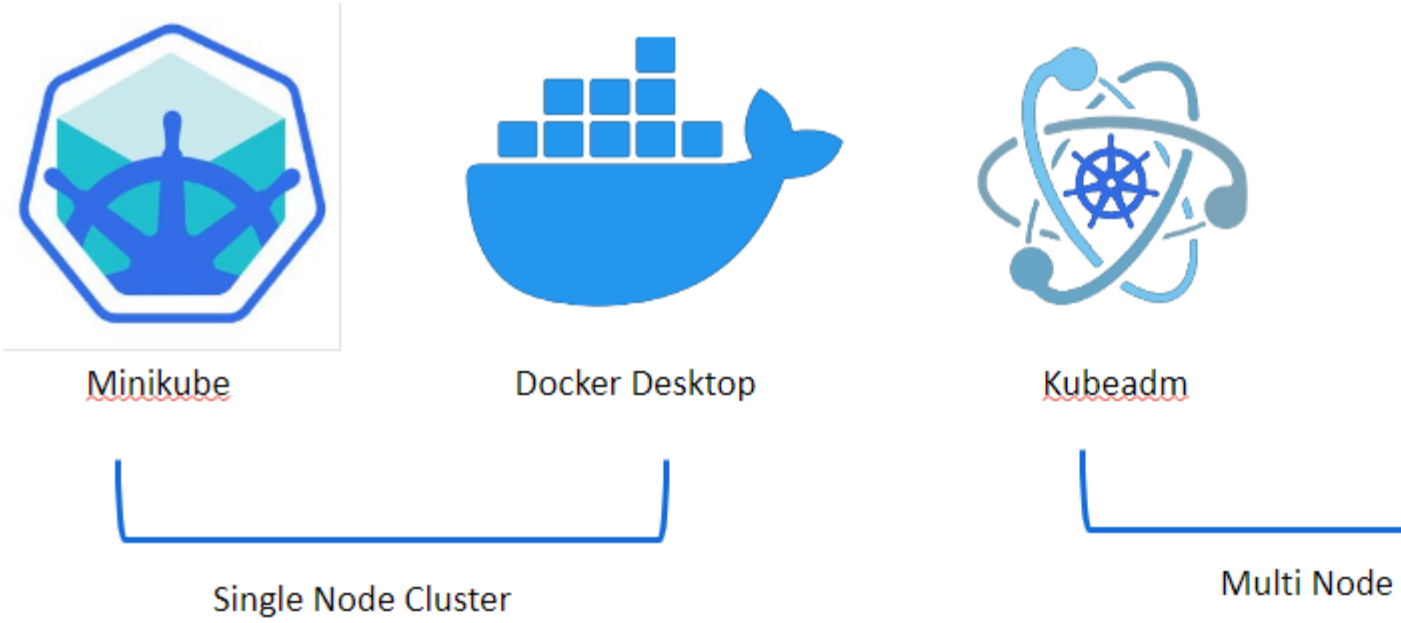
- Scalability is limited when compared to other orchestration technologies.
- Less feature-rich than Kubernetes.
- Less community and ecosystem support.
- A lack of advanced networking functions.
- There is limited support for complex deployment situations.

Docker Swarm vs. Kubernetes

- **Installation**: Kubernetes is simpler than Docker Swarm.
- **GUI Support**: Kubernetes has a built-in Web UI, whereas Docker Swarm requires additional configuration for GUI support.
- **Cluster Configuration**: Kubernetes requires manual setup and planning, but Docker Swarm is simpler and allows for post-configuration node addition.
- **API Functionality**: Unlike Docker Swarm, which relies on Docker's standards, Kubernetes creates its own API and YAML specifications.
- **Scalability**: Kubernetes expands quickly in large clusters, whereas Docker Swarm focuses on robust cluster state guarantees.
- **High Availability**: Unlike Docker Swarm, which relies on manager nodes, Kubernetes offers several master nodes for availability.

Configure Kubernetes

- **Single-node Cluster:** Single-node clusters are suitable for simple experimentation or learning purposes. A Kubernetes cluster can be set up on a local desktop using tools such as Minikube.
- **Multi-node Cluster:** Multi-node clusters are ideal for production environments or when you require more resources than a single node can give. Tools such as Kubeadm can help you set up multi-node clusters on bare metal or virtual machines.



Minikube

Run a single-node cluster on your home desktop for development or testing.

Kubeadm

Used to create a multi-node cluster on bare metal or virtual machines. This technique provides greater control and is appropriate for production use scenarios.

Cloud Provider Tools

Cloud providers such as Google Cloud Platform (GCP), Azure, and Amazon Web Services (AWS) offer specific tools for constructing and administering Kubernetes clusters.

YAML

YAML

YAML is a human-friendly data serialization standard often used for configuration files or data storage/transmission within programs. It was first released in 2001. It uses file extensions.yml, often known as.yaml, is case-sensitive and allows for indentation with spaces rather than tabs.

Advantages of YAML

Makes configuration comprehension and changing easier.

Reduces verbosity in complex arrangements.

Expresses desired states, facilitating infrastructure as code practices.

Git facilitates collaboration and change tracking.

Disadvantages of YAML

- Indentation problems can lead to syntax concerns.
- A lack of comprehensive validation may result in runtime errors.
- YAML may not meet all sophisticated configuration requirements.
- Troubleshooting YAML can be tricky, particularly for complicated structures.

YAML

YAML (YAML Ain't Markup Language) is a format for serializing data that is human-readable. YAML is the preferred format for configuration files in Kubernetes since it is easy to understand and simple.

JSON (JavaScript Object Notation)

A human-readable data format that is structured similarly to JavaScript objects. JSON is readable, but it can become verbose in complex installations. Kubernetes also understands JSON configuration files.

XML

XML (Extensible Markup Language) is a structured format that uses tags to specify elements and attributes. XML is typically more verbose than YAML or JSON and is less commonly used in Kubernetes configuration files.

XML vs. JSON vs. YAML

YAML is a great choice for specifying resources like Pods, Deployments, and Services due to its readability and ease of comprehension. JSON is also used in some instances, however, XML is less popular because of its language and complexity.

```xml
<Servers>
    <Server>
        <name>server1</name>
        <location>india</location>
        <status>active</status>
    </Server>
</Servers>
```

```json
{
    "Servers": [
        "Server": {
            "name": "server1",
            "location": "india",
            "status": "active"
        }
    ]
}
```

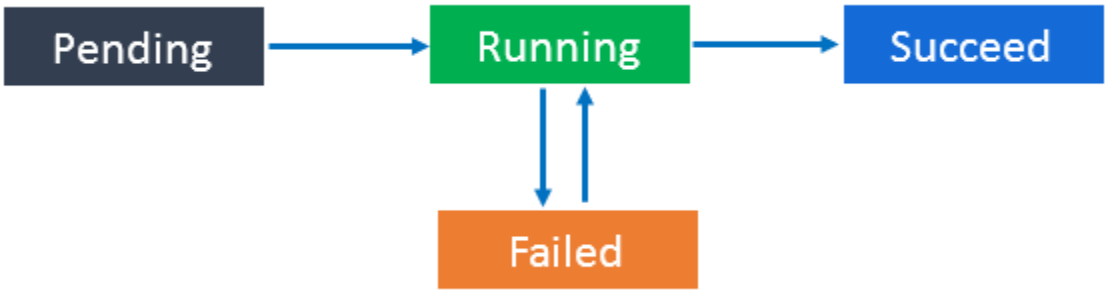
PODS And Services

What is Pod?

Kubernetes uses pods as logical containers that store numerous application containers & shared volumes, ensuring co-location and stability. Kubernetes manages using pods rather than individual containers, ensuring that containers within the same pod share networking, storage, and lifecycle on the same machine, resulting in efficient operation.

Pod States

- Pending
- Running
- Successful (Succeeded)
- Failed
- Unknown



Need for pods in Kubernetes

We need pods in clusters because they allow for data sharing and communication among their members. So we may access pod applications using the same IP and port.

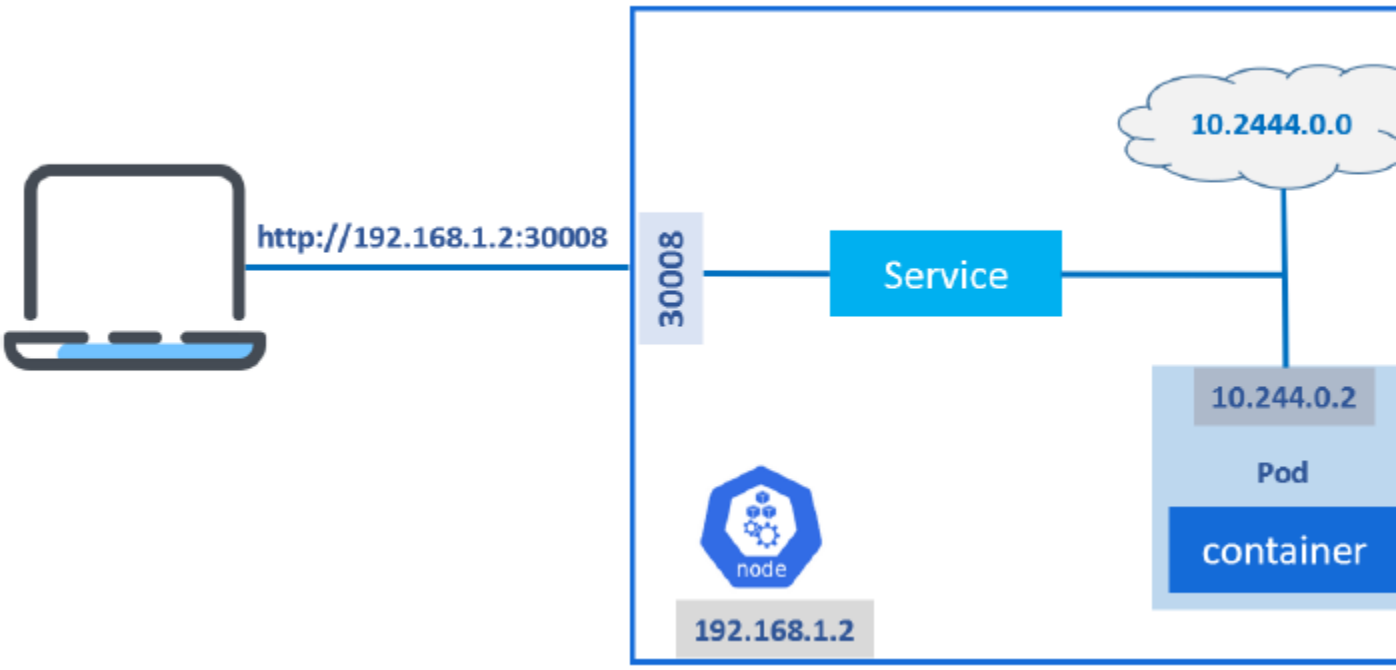
Different types of multiple-container pods

Three different types of multi-container pods.

1. Sidecar
2. Adapter
3. Ambassador

Kubernetes service

Kubernetes Services makes network access to a collection of Pods easier by categorizing them based on labels. Requests to a Service are sent to all Pods in the cluster that matches the Service's selector, ensuring smooth communication throughout the Kubernetes ecosystem.



Working of Kubernetes Service

Services are programmes that maintain the list of pod IP addresses for a deployment. Services choose pods depending on their labels. When a network request is sent to the service, it selects all Pods that match the service's selector, chooses one, and returns the network request.

Different types of services in Kubernetes

There are four types of service in Kubernetes:

1. ClusterIP
2. NodePort
3. LoadBalancer
4. ExternalName