What is Docker and Container?

- What is Docker?
- What is a Container?
- Docker Container Architecture
- Docker Components
- Deployment Era
- Microservices

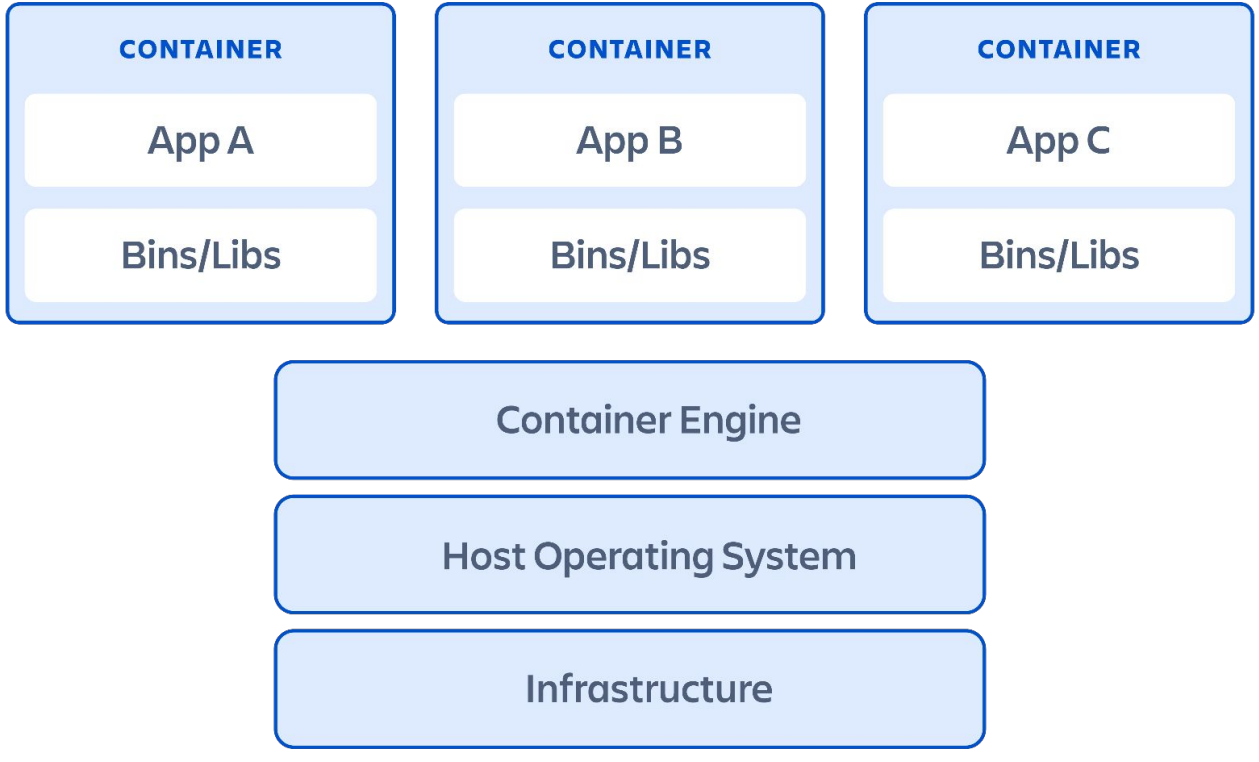
Virtualization Recap

- Virtualization is the act of dividing shared computational resources: CPU, RAM, Disk, and Networking into isolated resources that are unaware of the original shared scope.
- When virtualizing a machine using either virtual machines (VMs) or Containers, the host machine's resources are essentially cut into slices for the virtualized components to use.

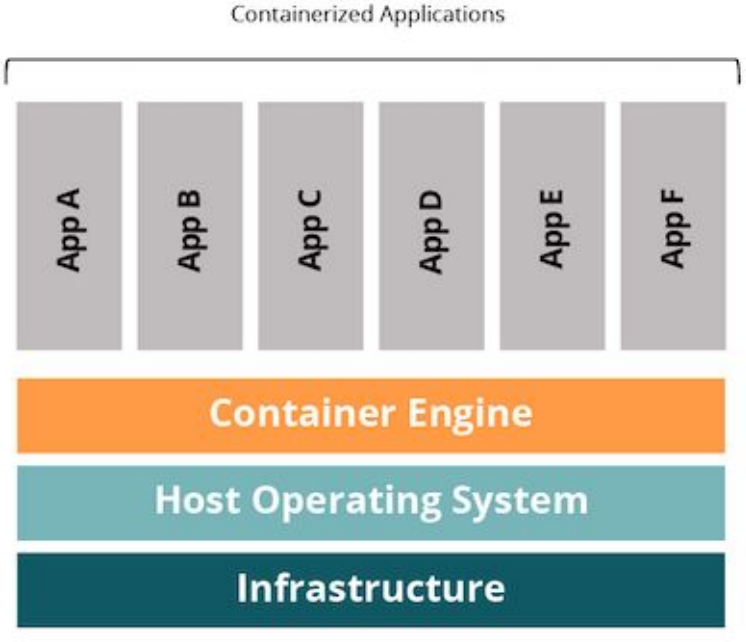
What is Docker and Container?

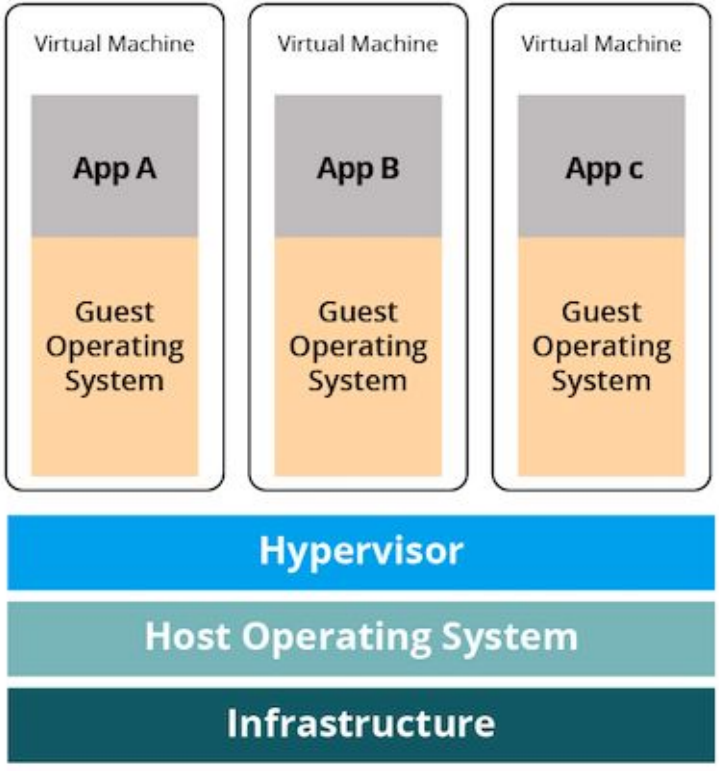
- **Docker** is an open source platform that's used to build, ship and run distributed services.
- **Containers** are an encapsulation of an application with its dependencies.
- Containers are an application deployment technology that performs a similar role to virtual machines (VMs).
  - A **container** holds an isolated instance of an operating system, which is used to run various other applications.
  - A container is a package of software that includes all **dependencies: *code, runtime, configuration, and system libraries*** so that it can run on any host system.
  - Containers allow the packaging of your application (and everything that you need to run it) in a "**container image" / "docker image**" (similar to EC2 AMI)
  - Inside a container you can include a **base operating system, libraries, files and folders, environment variables, volume mount-points, and your application binaries.**
  - Once a container is created, the execution is managed by the **container runtime**. You can interact with the container runtime through the "**docker**" commands.
  - The **reduced size and simplicity of containers** also means they can **stop and start more quickly** than VMs.

Containers

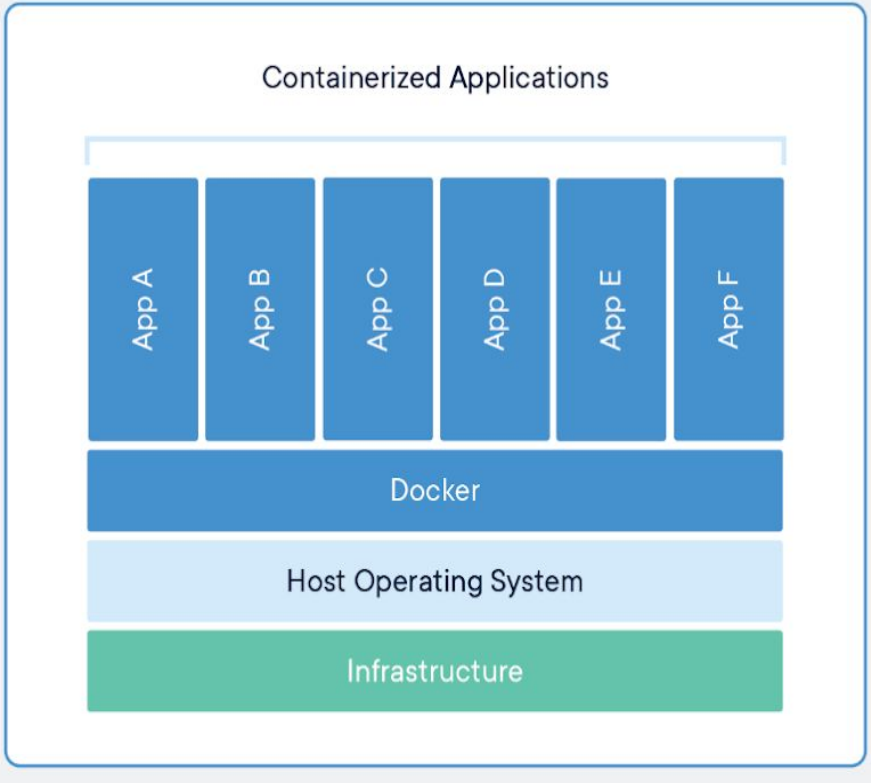

CONTAINER
App A
Bins/Libs
CONTAINER
App B
Bins/Libs
CONTAINER
App C
Bins/Libs
Container Engine
Host Operating System
Infrastructure

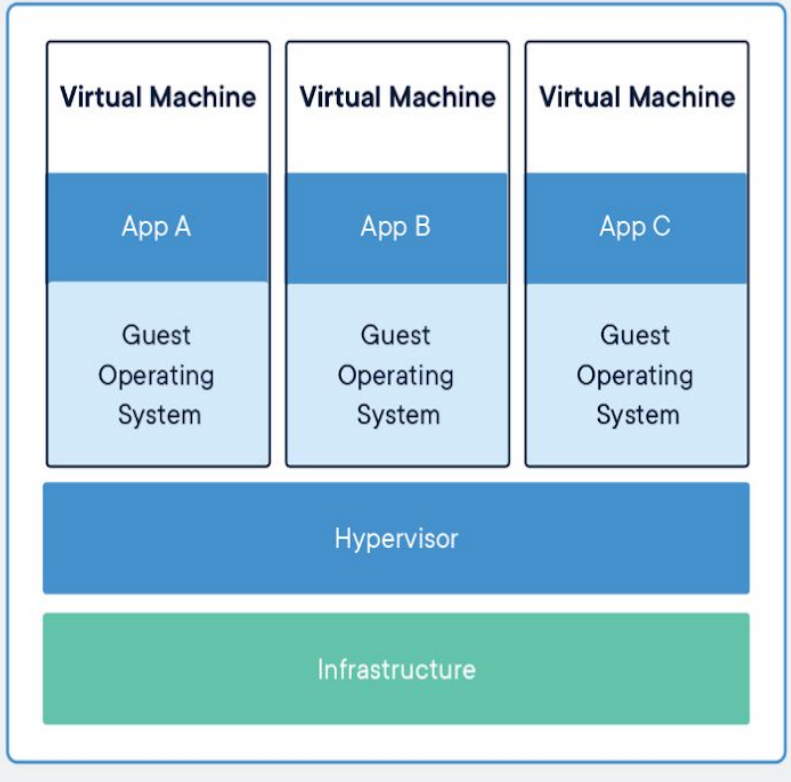
What are Containers?





Containers vs VM





Containers vs VM

- Containers and VMs are very similar in their goals.
- They both help distribute application software in a repeatable isolated package.
- Where they differ is in how much of the hardware stack they attempt to virtualize.
- VM's simulate the entire machine and operating system.
- At runtime, the container is also granted its own isolated slice of operating system resources like CPU, RAM, Disk, and Networking.
- VMs use a hypervisor to emulate fully fledged guest operating systems, containers share the kernel of the host operating system with other containers.
- This means VMs have simulated CPU, RAM, Filesystems and Network resources.
- Containers only virtualize the user space of an existing operating system.
- In this sense, containers are much more lightweight than VMs.
- Containers can be utilized in an existing host operating system.
- EC2 Instance is a running environment of a specified **AMI**
- Docker Container is a running environment of a specified **Docker Image**

Docker Components

1. Docker Daemon
2. Docker Client
3. Docker Images
4. Docker Registries
5. Docker Containers

Docker Components

- **Docker daemon** (generally referred to as **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 client** (also called '**docker**'), with which many Docker users interact with Docker, can communicate with more than one daemon.
- **Docker Images :** A Docker image is made up of a collection of files that bundle together all the essentials – such as **installations, application code, and dependencies** – required to configure a fully operational container environment. A Docker image is a snapshot, or template, from which new containers can be started. **(Similar to AMI in AWS)**
- **Docker registry** stores Docker images. (**Remote Docker Image Repository Storage**)
  - Docker Hub is a public image registry that anybody can use, and Docker daemon is configured to look for images on Docker Hub, by default.
  - Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.
  - Google Container Registry (GCR)
  - Azure Container Registry (ACR)

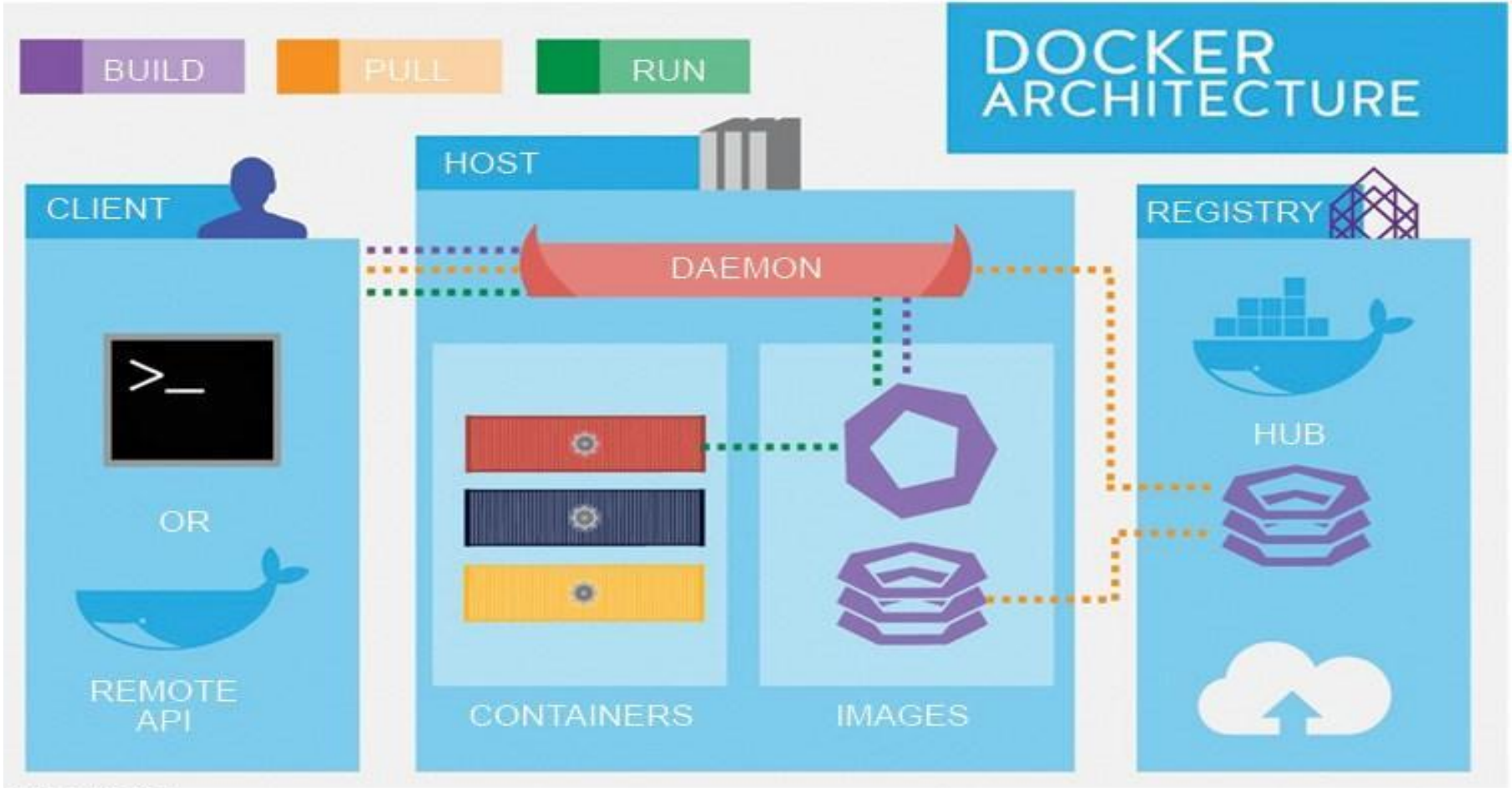
Docker Components

- **Docker Container**
  - Docker containers are basically runtime instances of Docker images.
  - Docker containers will contain all the binaries and libraries required for application
  - So, application is present inside a container. Here, the same container can be used in the Test and Prod environment.

Docker Components

- **Note**: Docker **pull** and **push** are images commands that pull an image/push an image to configured registry.
- Docker Objects are basically images, containers, networks, volumes, plugins and other various such objects that we create and use.
- Thus containers are fundamentally changing the way we develop, distribute and run software on a daily basis.
- These developments and advantages of containers help in the advancement of microservices technology.
- If an image specified in the Docker run command is not present in the Docker host, by default, the Docker daemon will download the image from the Docker public registry (Docker hub).

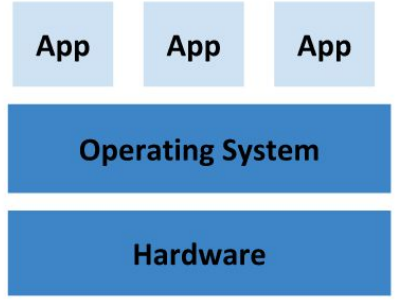
Docker Container Architecture



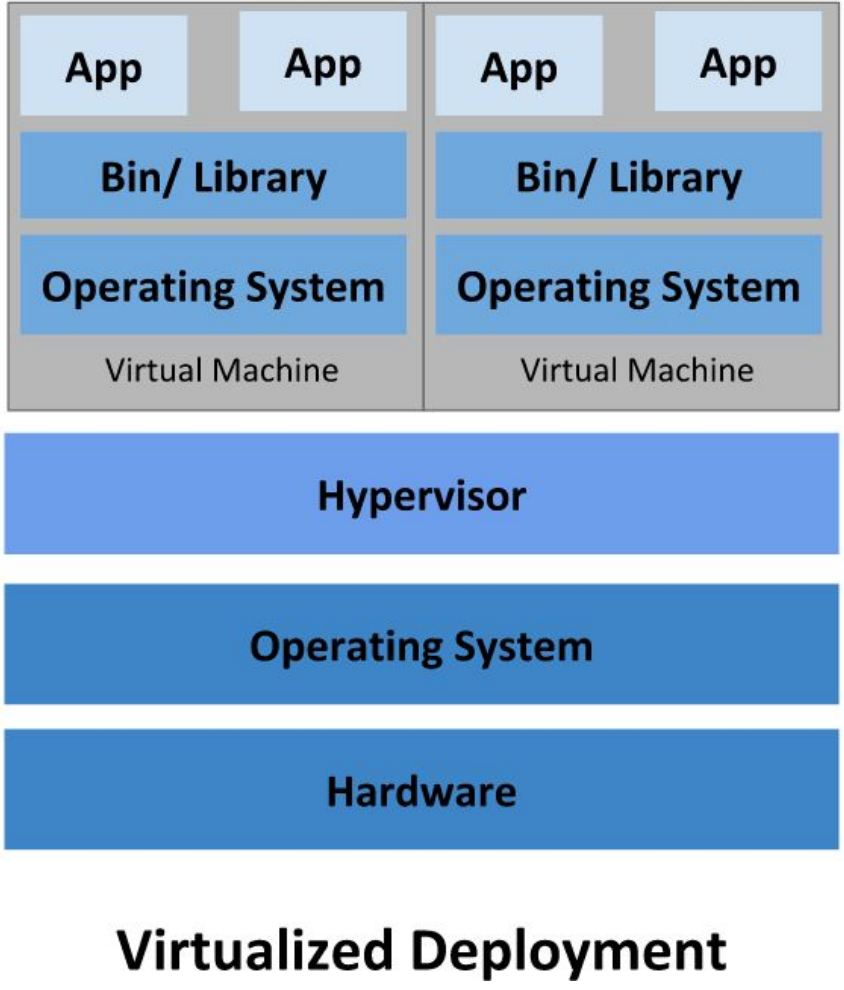
Deployment Era

Traditional deployment :

- Early on, organizations ran applications on physical servers.
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.
- For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform.
- A solution for this would be to run each application on a different physical server.
- But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.



Traditional Deployment



Virtualized Deployment

Virtualized deployment:

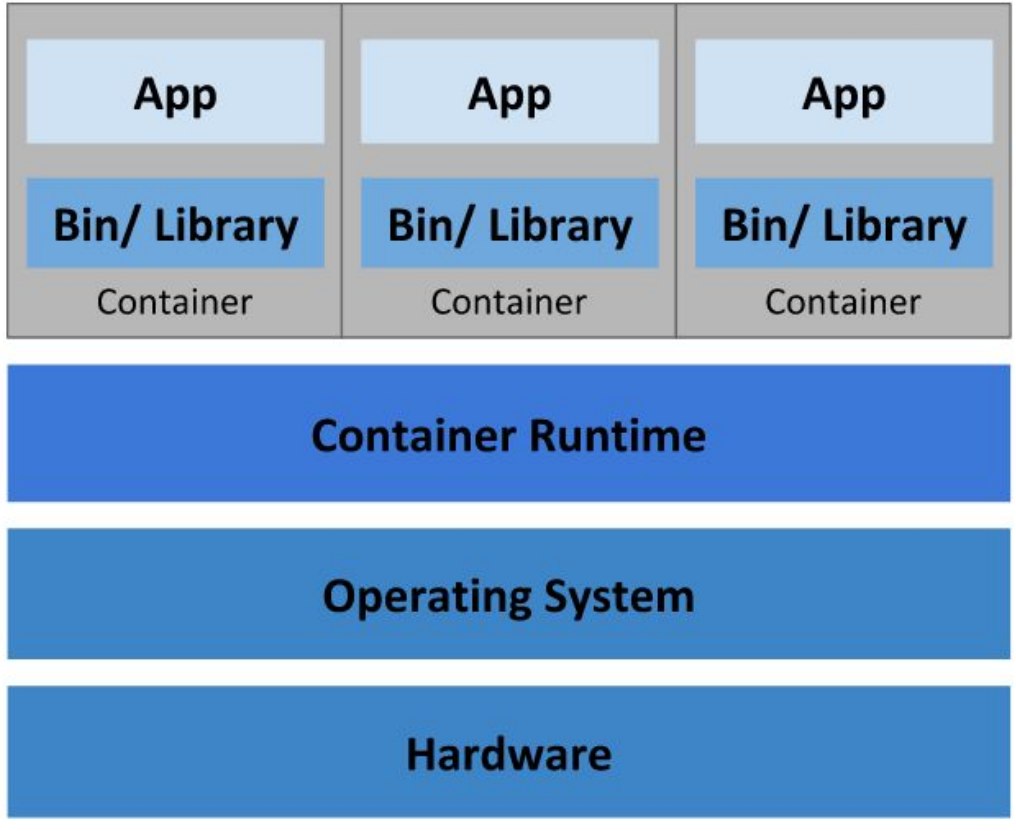
- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.
- Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.
- Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

Container deployment era:

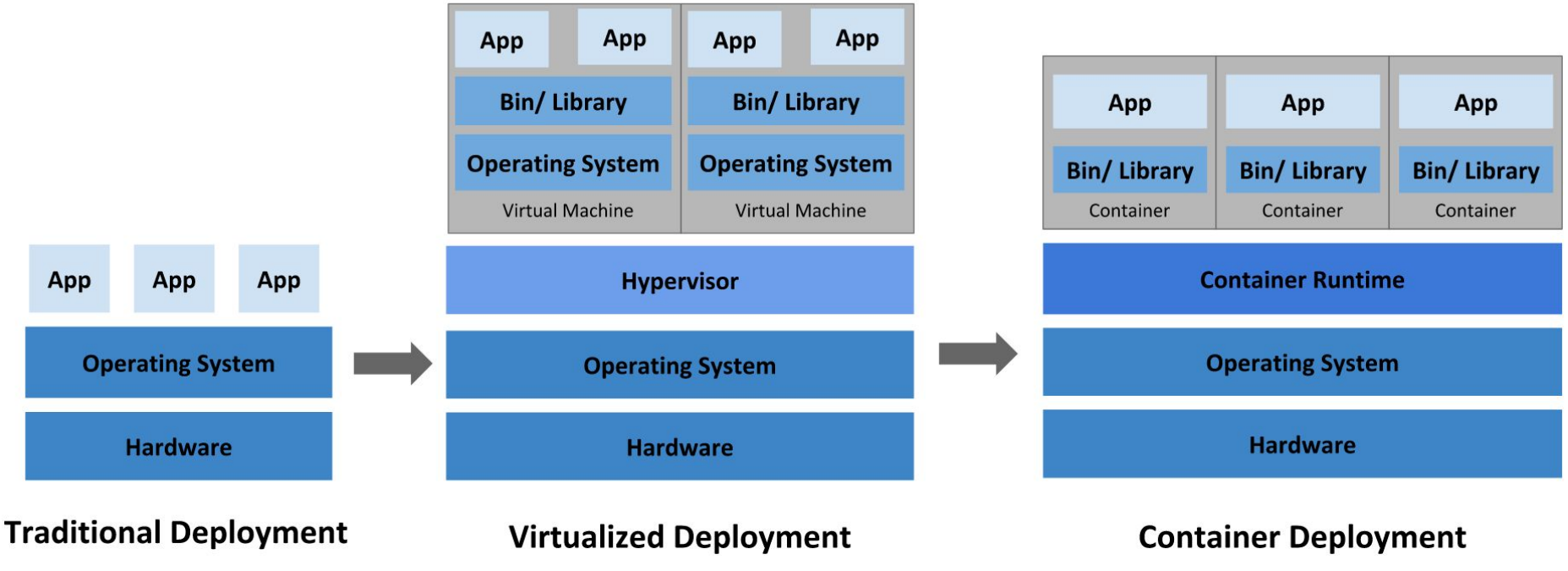
- Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications.
- Therefore, containers are considered lightweight.
- Similar to a VM, a container has its own file system, CPU, memory, process space, and more.
- As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Containers have become popular because they provide extra benefits :

- Agile application creation and deployment
- Continuous development, integration, and deployment
- Dev and Ops separation of concerns
- Environmental consistency across development, testing, and production
- Cloud and OS distribution portability
- Application-centric management
- Loosely coupled, distributed, elastic, liberated micro-services
- Resource isolation
- Resource utilization



Container Deployment


App
App
App
Operating System
Hardware
Traditional Deployment
App
App
App
App
Bin/ Library
Bin/ Library
Operating System
Operating System
Virtual Machine
Virtual Machine
Hypervisor
Operating System
Hardware
Virtualized Deployment
App
App
App
Bin/ Library
Bin/ Library
Bin/ Library
Container
Container
Container
Container Runtime
Operating System
Hardware
Container Deployment

VM v/s Container

VM

- **Separate OS**
- Larger size compared to Container
- Minutes to be created or provisioned
- Minutes to start/stop
- Fewer can be created on physical machine compared to container

Container

- Share OS Kernel
- Smaller size compared to VM
- Seconds to be created or provisioned
- Seconds to start/stop
- Many can be created on a VM compared to VM

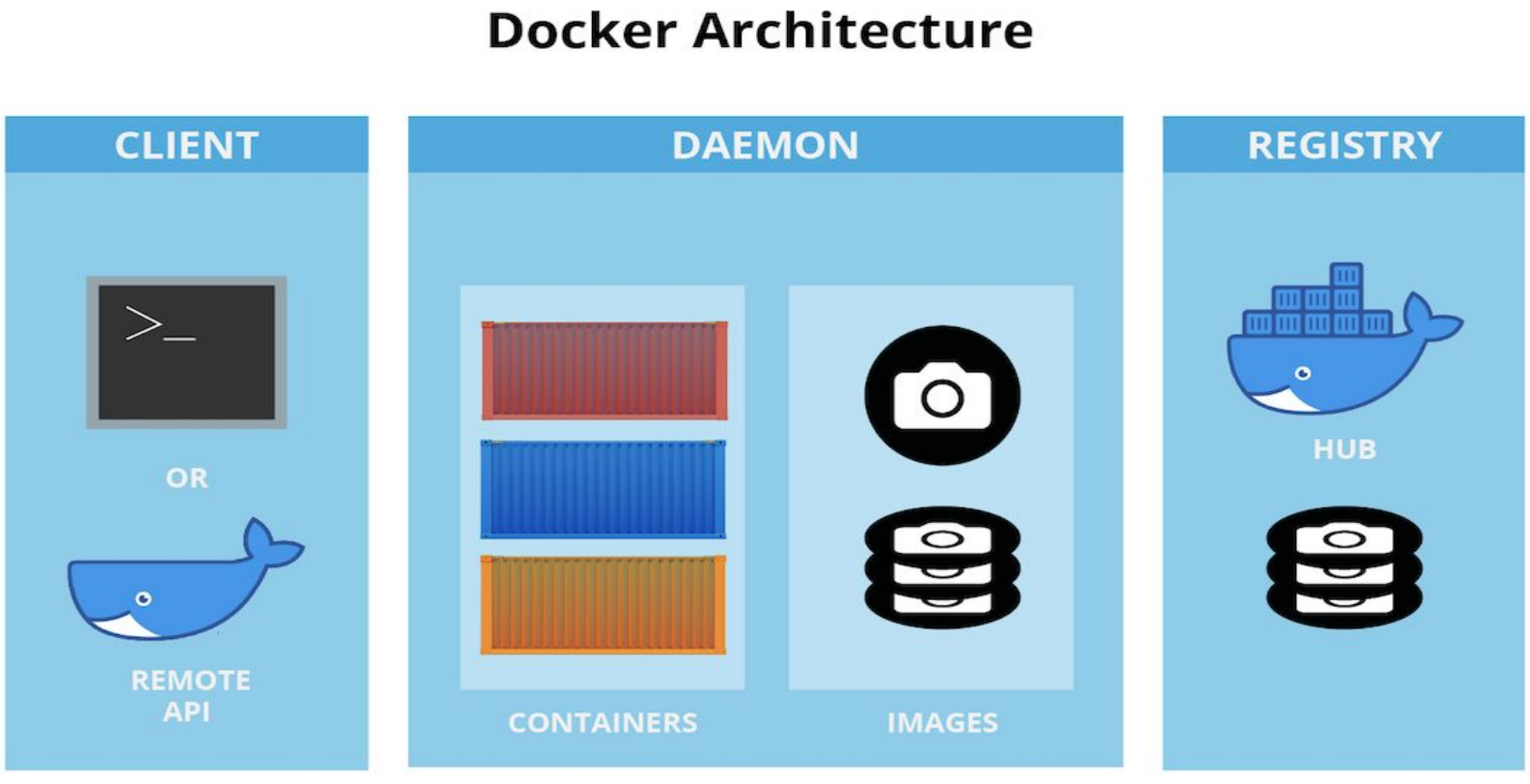
Understanding Docker

- Docker container is based on an image
- Images are provided through **Docker Registry** or **Docker Hub**

Running Docker is a simple process:

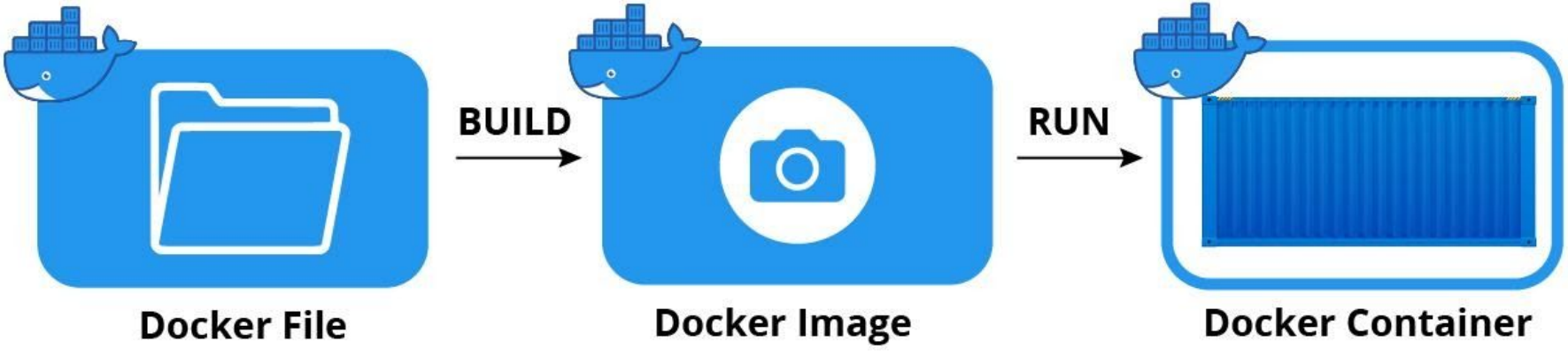
- The docker service must be running.
- Use **docker search** to search a container
- Use **docker pull** to pull an image to local image store
- Use **docker run** to run your container and start a command within
- Use **docker create** to create a new container
- Use **docker exec** to run a command in an already running container.

Docker Container Architecture

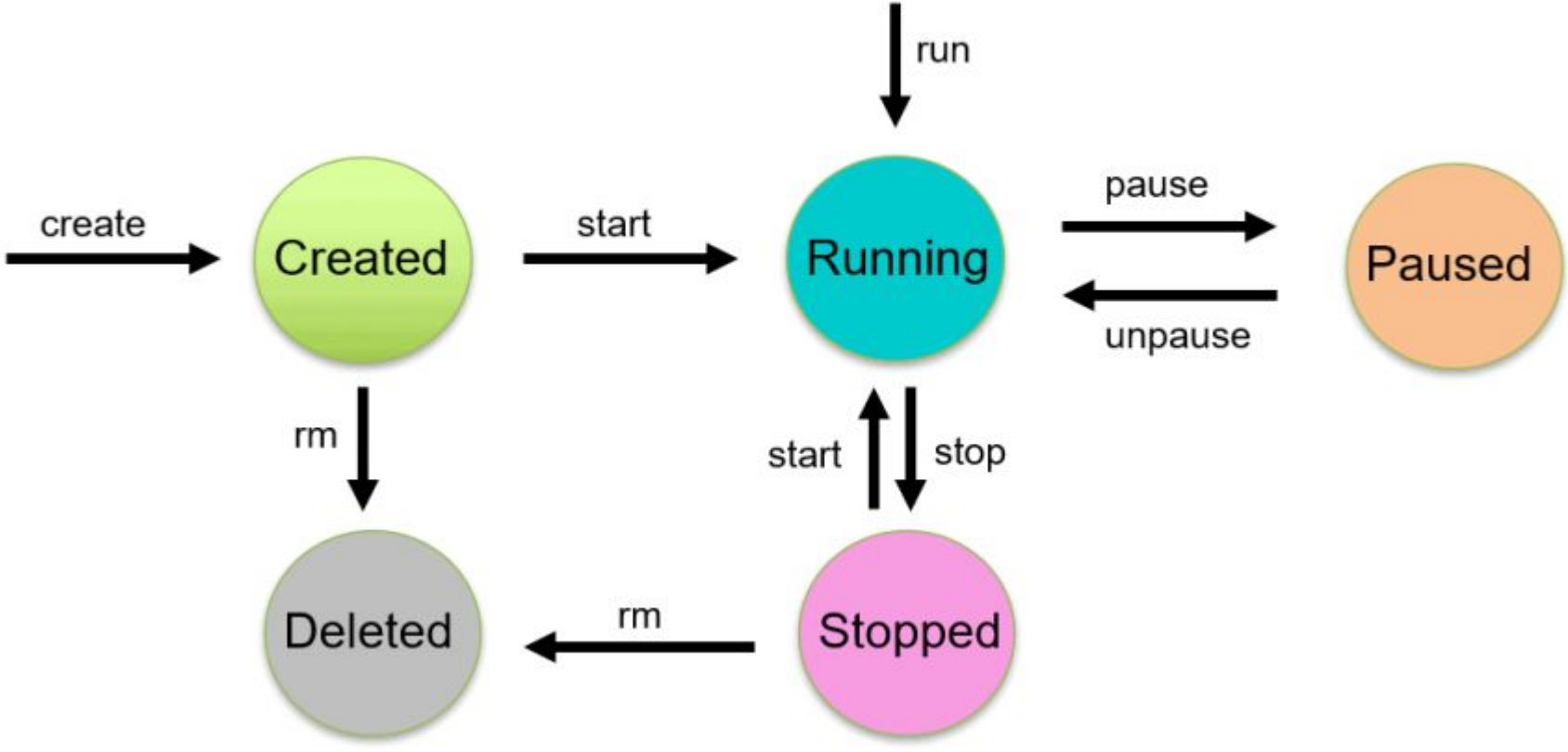


Docker Image : BUILD , RUN

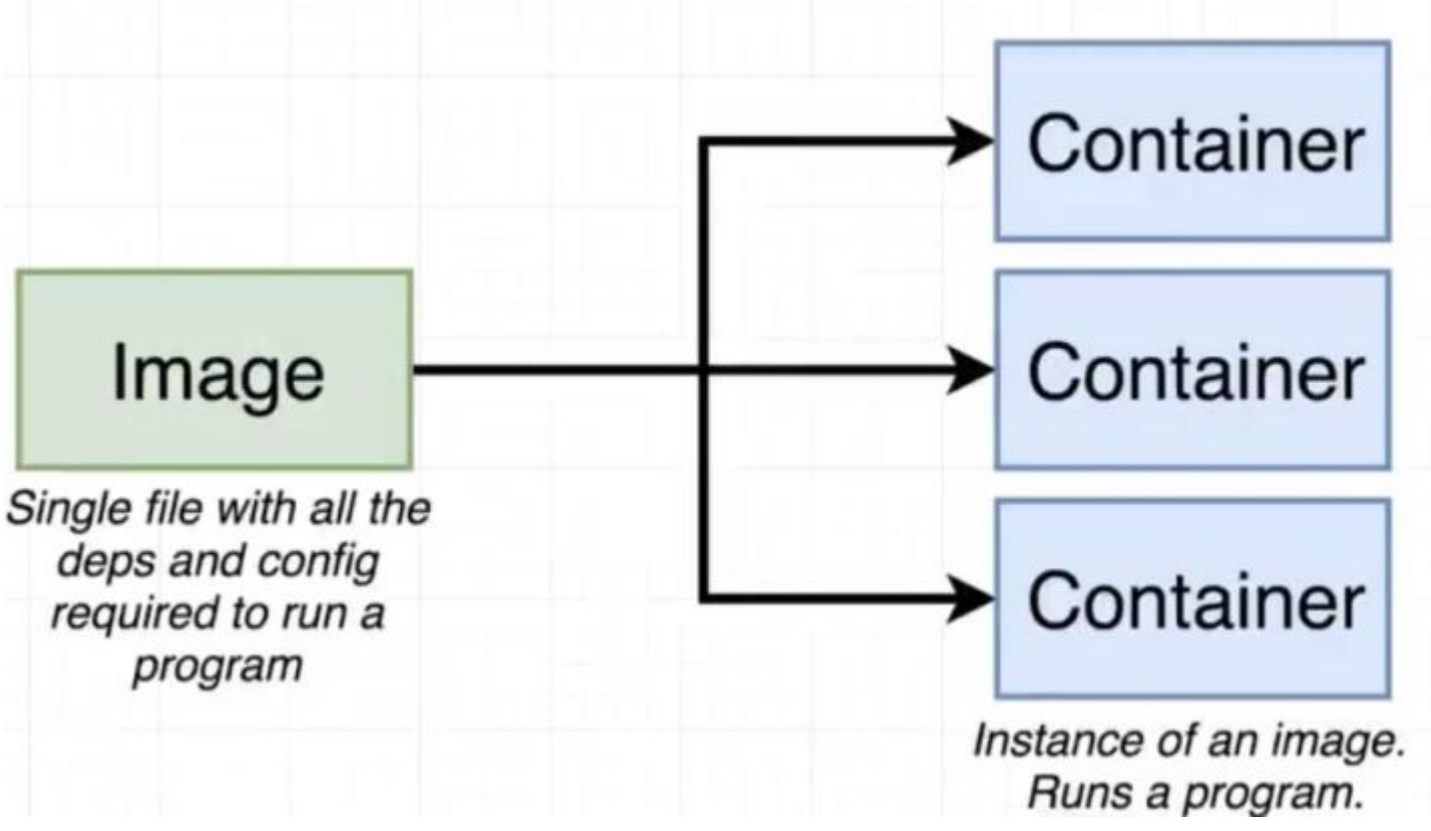
1) Write a Dockerfile
2) Use **docker build** command
3) This will create a Docker Image using instruction specified in Dockerfile
4) Use **docker run** command to create a container using image created.



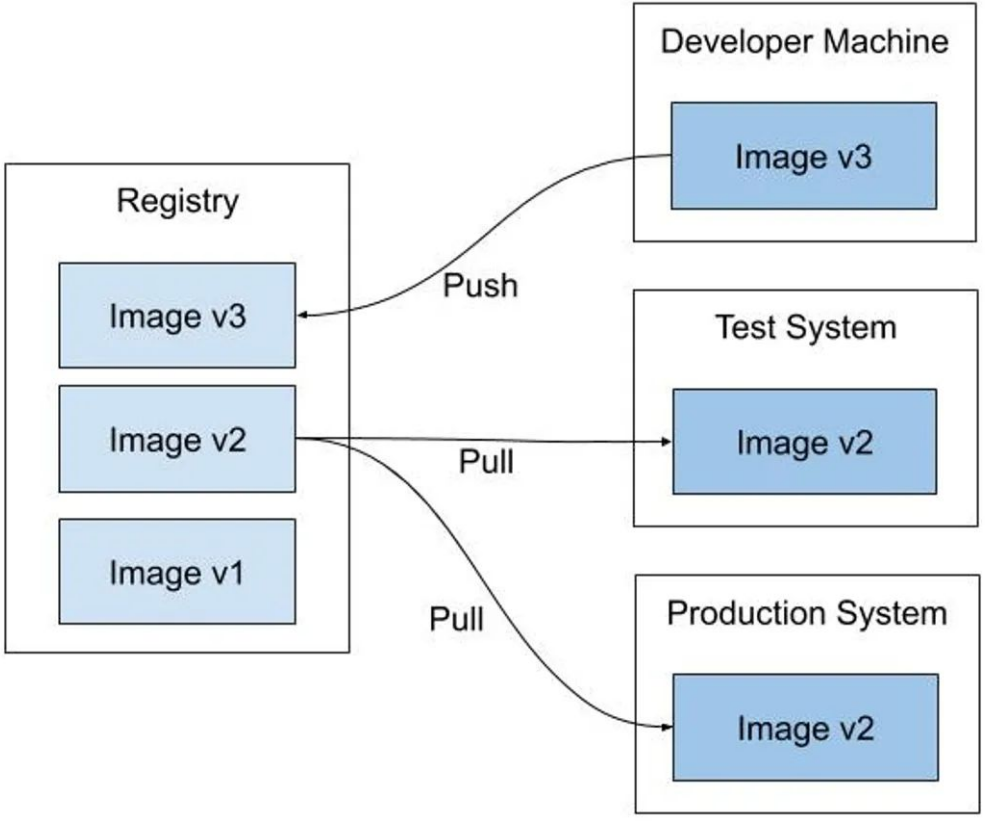
Container Lifecycle



Docker Image , Container



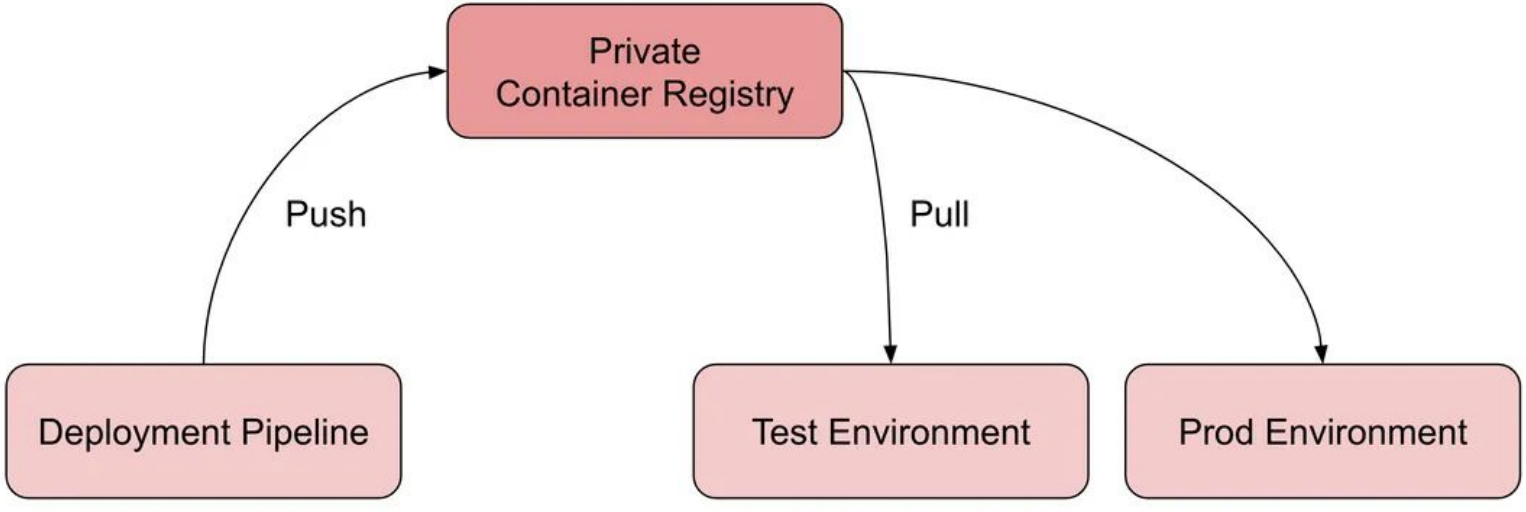
What is a container registry?



- A **container registry** is a central place to store and distribute container images. A container image includes all the data needed to start a container—for example, the operating system, libraries, runtime environments, and the application itself.

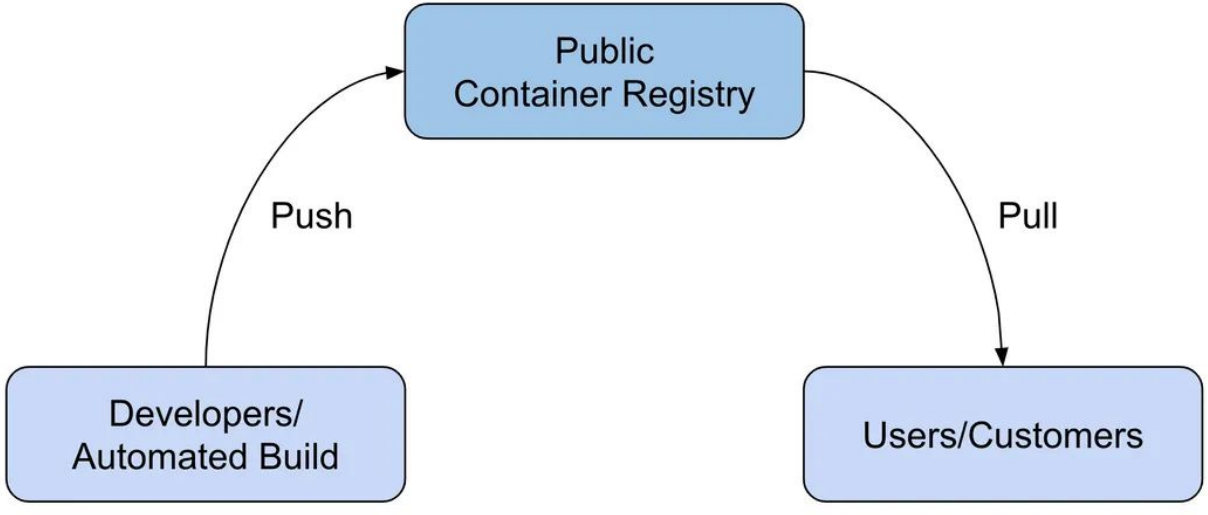
Private Container Registry

- If you need to share your service with internal developers or teams so they can run containers, a private container registry could be selected that allows you to distribute images within your organization.



Public Container Registry

- If you need to share your service with multiple users or customers available outside your organization, a public container registry can be used that anyone can access to pull your images. Here, only Developers or Automated Build Environments are allowed to push images to the registry.



Docker Registry Comparison

	Amazon ECR	Docker Hub
Public Repository	✅ yes	✅ yes
Private Repository	✅ yes	✅ yes
Pricing (Public Repository)	💰 Storage: $0.10 per GB	🆓 $0
Pricing (Private Repository)	💰 Storage: $0.10 per GB, Data Transfer: $0.09 per GB	💰💰💰 >= $7 per user/month[^2]
Authentication	AWS IAM	Password or Access Token
MFA for Image Push/Pull	✅ yes	❌ no
SLA Availability	✅ 99.9%	❌ n/a
General Available	✅ yes	✅ yes
Immutable Images	✅ yes	❌ no
Image Scanning	✅ yes	✅ yes (paid plans only)
Regions	Choose between one of 25 regions worldwide	⚠️ unknown
Rate Limits	Pull: 1,000 per second, Push: 10 per second	Pull: 100/200 (Free Plan), unlimited (Paid Plan)

Some Docker vocabulary

Containers	Images
How you **run** your application	How you **store** your application

Docker Image

The basis of a Docker container. Represents a full application

Docker Container

The standard unit in which the application service resides and executes

Docker Engine

Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

Registry Service (Docker Hub or Docker Trusted Registry)

Cloud or server based storage and distribution service for your images

Instruction	What it does
ADD	Copies files from the build host to the image[a]
ARG	Sets variables that can be referenced during the build but not from the final image; not intended for secrets
CMD	Sets the default commands to execute in a container
COPY	Like ADD, but only for files and directories
ENV	Sets environment variables available to all subsequent build instructions and containers spawned from this image
EXPOSE	Informs **dockerd** of the network ports exposed by the container
FROM	Sets the base image; must be the first instruction
LABEL	Sets image tags (visible with **docker inspect**)
RUN	Runs commands and saves the result in the image
STOPSIGNAL	Specifies a signal to send to the process when told to quit with **docker stop**; defaults to SIGKILL
USER	Sets the account name to use when running the container and any subsequent build instructions
VOLUME	Designates a volume for storing persistent data
WORKDIR	Sets the default working directory for subsequent instructions

Container Use Cases

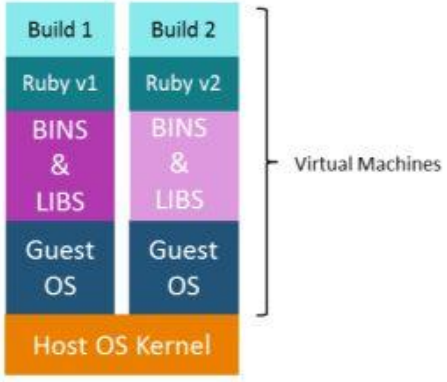
- Containers solve one of the classic causes of software distribution failure: “it doesn’t works on my machine.”
- In this unfortunate scenario, application code behaves as expected on one machine, but when executed on another, it fails.
- This is usually due to subtle differences between the two machines. These differences can be mismatched dependency versions or other configuration discrepancies.
- Containers solve this by creating a static repeatable package of everything the application code needs to execute.

Benefits of Containers

- Enables the app to run efficiently across different computing environments.
- Holds everything essential to an app – from code to system tools to settings.
- Multiple containers are managed by a single OS, making them more lightweight than VMs.

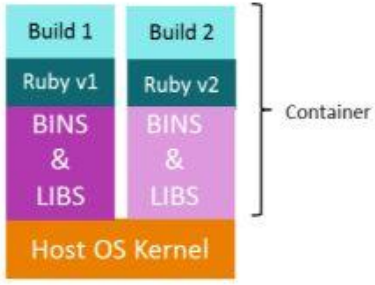
Before Docker

In case of Virtual Machines



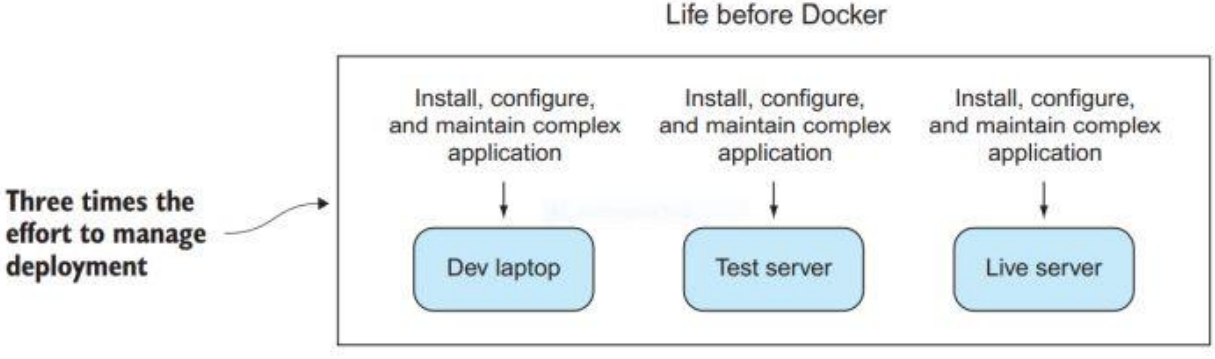
New Builds → Multiple OS → Separate Libraries → Heavy → **More Time**

In case of Docker



New Builds → Same OS → Separate Libraries → Lightweight → **Less Time**

Life before Docker



After Docker