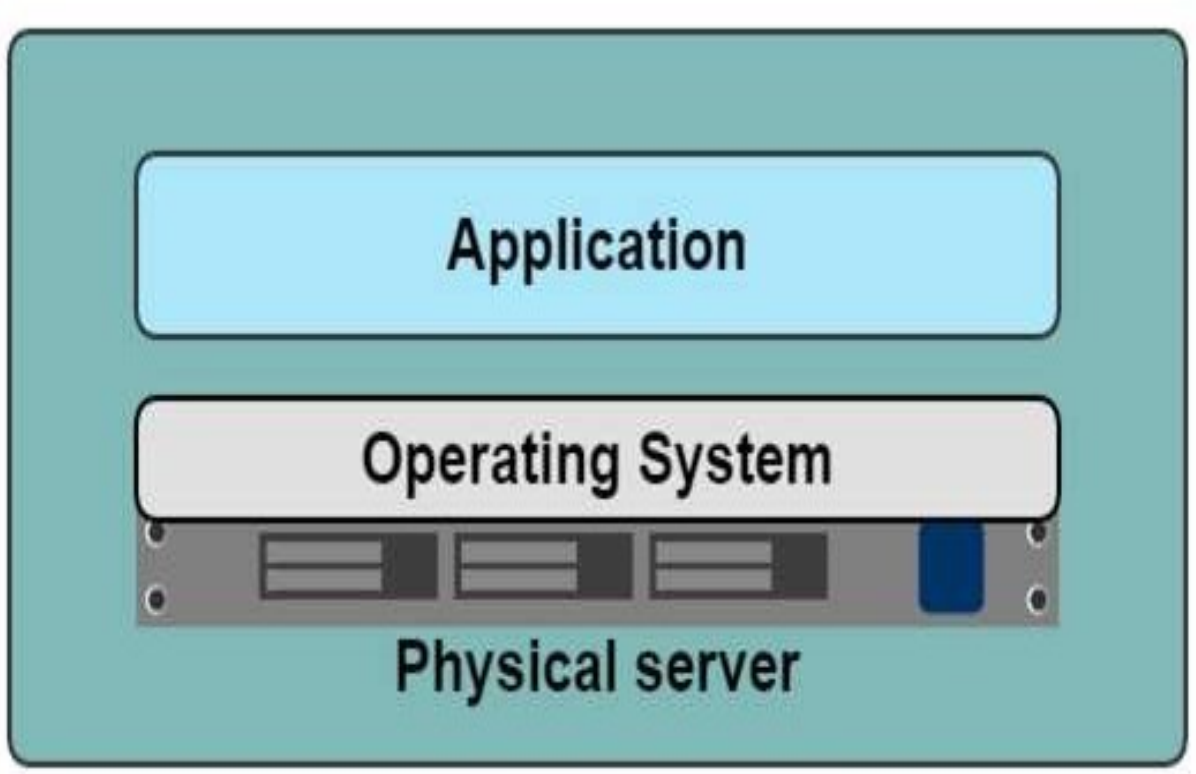
22AIE305: CLOUD COMPUTING

Historical limitations of application deployment

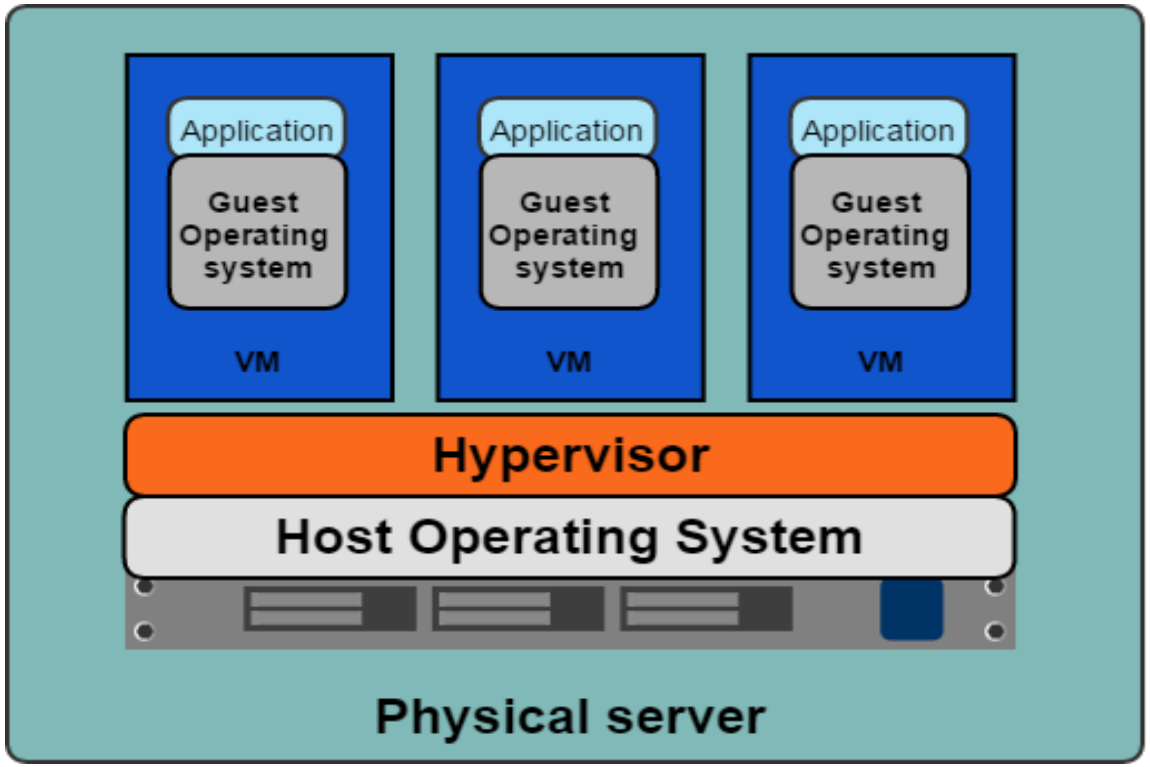
- Slow deployment times
- Huge costs
- Wasted resources
- Difficult to scale
- Difficult to migrate
- Vendor lock in



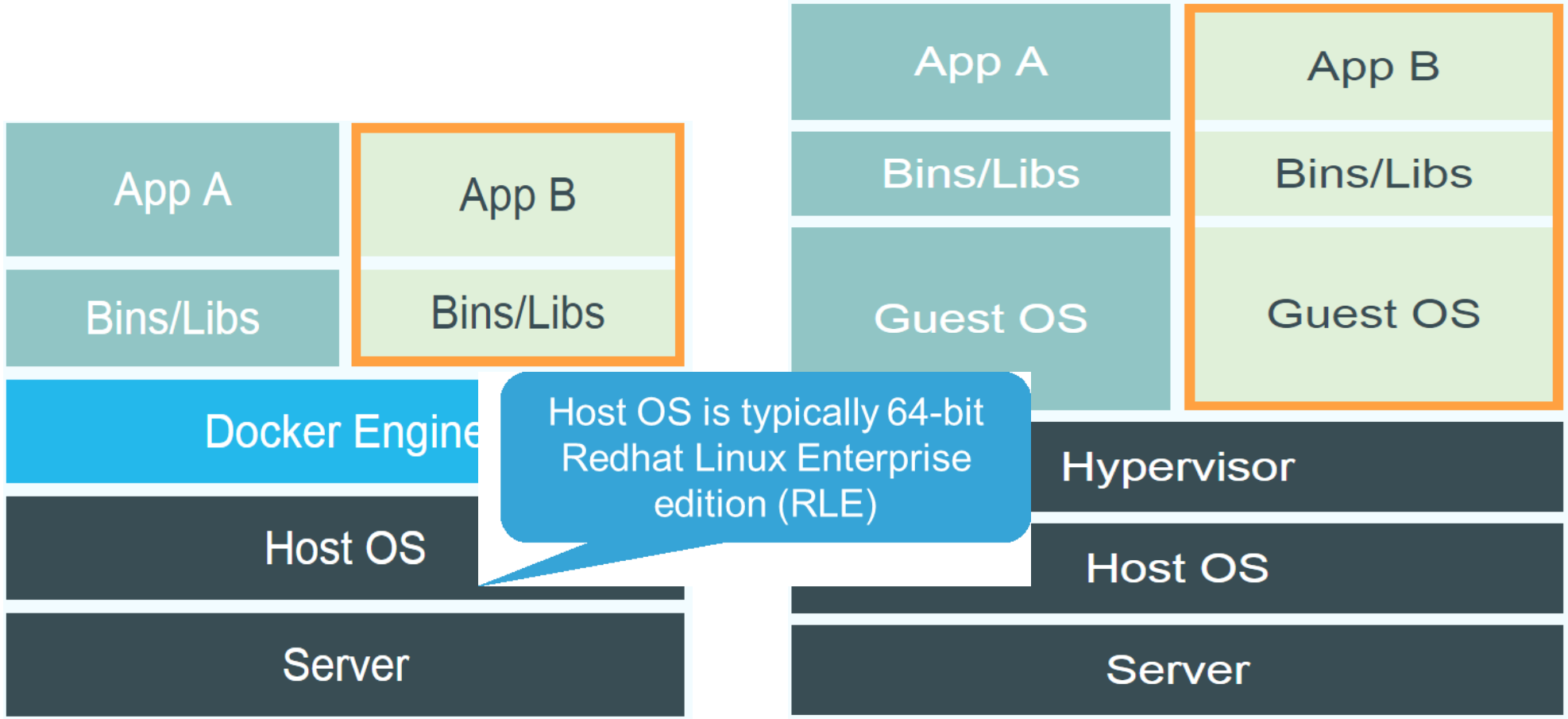
A History Lesson

Hypervisor-based Virtualization

- One physical server can contain multiple applications
- Each application runs in a virtual machine (VM)



Docker vs. Virtual Machine



Source: https://www.docker.com/whatisdocker/

EMULATORS

Virtual machines (VMs), running on physical server hardware, can be used to implement virtual servers.

VMs isolate guest OS from intensive use of host hardware.

The hypervisor provides a hardware emulation that simulates the operation of the underlying hardware.

Several of these hardware emulators share the physical hardware and run in parallel.

You can run an OS and then install server software on each hardware emulator.

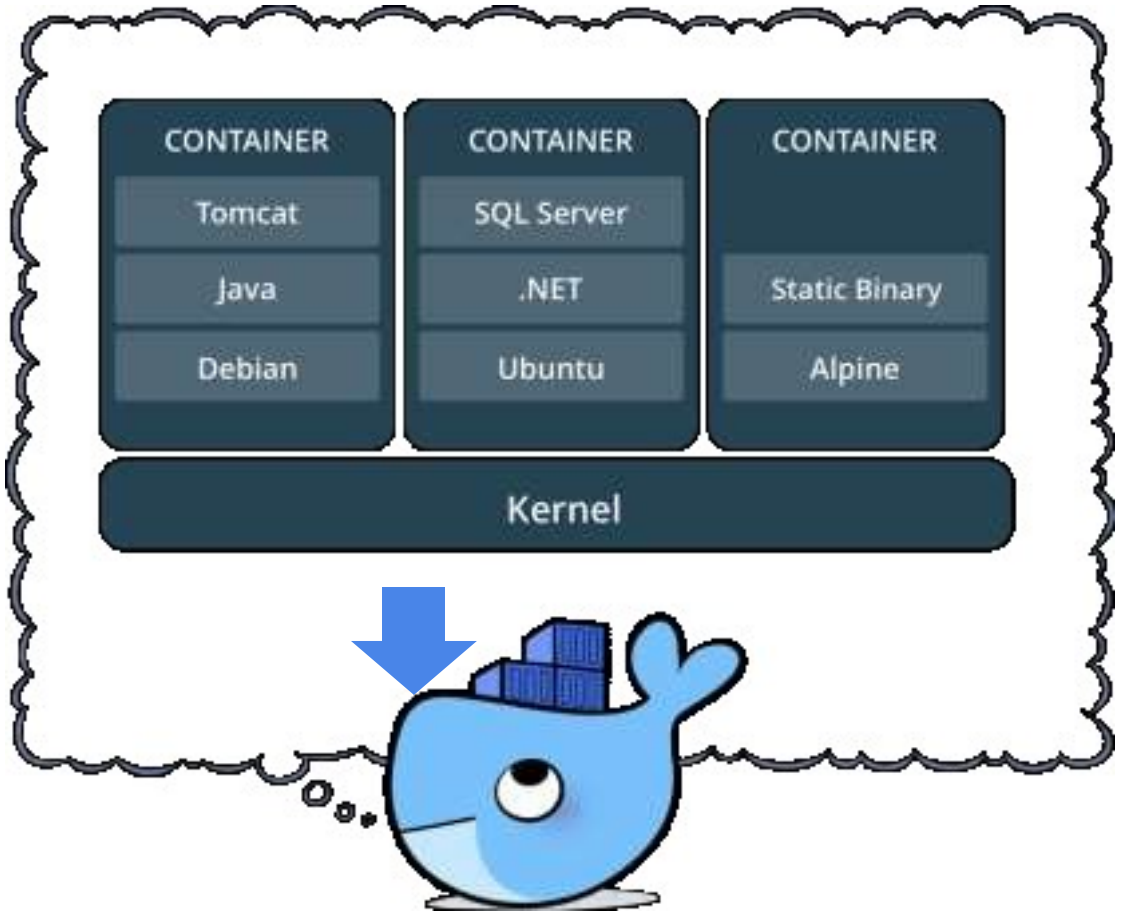
Benefits of VMs

- Better resource pooling
  - One physical machine divided into multiple virtual machines
- Easier to scale
- VMs in the cloud
  - Rapid elasticity
  - Pay as you go model

Limitations of VMs

- Each VM stills requires
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system
- The more VMs you run, the more resources you need
- Guest OS means wasted resources
- Application portability not guaranteed

What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works with all major Linux and Windows Server

VM vs Container

In many cases, you don't really need the generality of a virtual machine. If you are running a cloud-based system with many instances of applications or services, these all use the same operating system. To cater to this situation, a simpler, lightweight, virtualization technology called "containers" may be used.

VIRTUAL MACHINE VS CONTAINERS

- Using containers dramatically speeds up the process of deploying virtual servers on the cloud.
- Containers are usually megabytes in size, whereas VMs are gigabytes.
- Containers can be started up and shut down in a few seconds rather than the few minutes required for a VM.
- Many companies that provide cloud-based software have now switched from VMs to containers because containers are faster to load and less demanding of machine resources.

Containers are an operating system virtualization technology that allows independent servers to share a single operating system. They are particularly useful for providing isolated application services where each user sees their own version of an application.

As we can see, the VMs have the hypervisor layer as well as individual operating systems for each application, thereby having more layers between the application and the host operating system. On the other hand, it is easy to see that there are fewer demands on the server in the case of containers. This is a key point. It simply means that we can fit in many more containers in the same server than VMs and continue to enjoy superior performance! As we can see, the containers do not have their individual operating systems; it simply uses the host operating system kernel to get their work done.

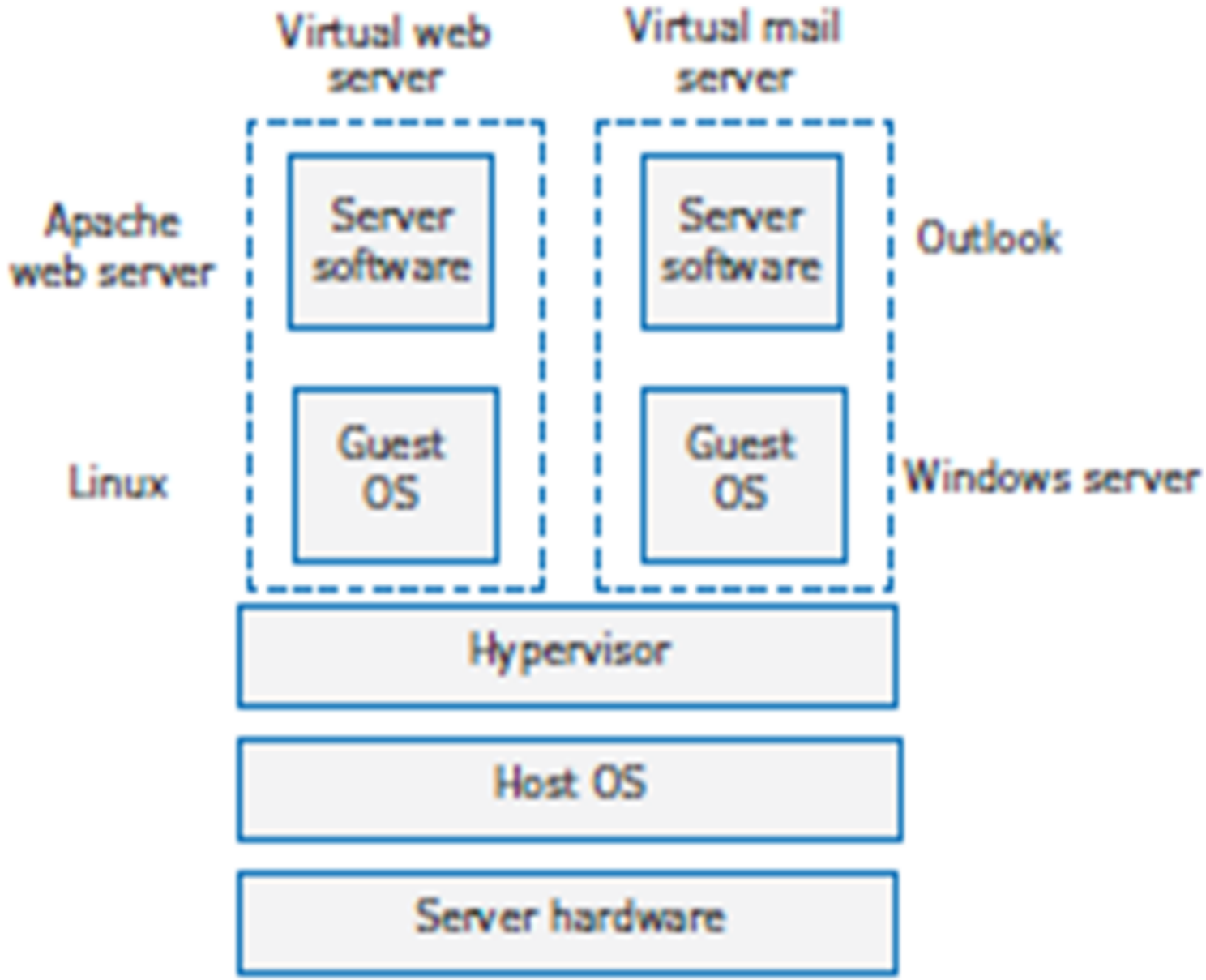
The second benefit that the container provides us is because they are so lightweight, given the fact that they do not have a dedicated operating system for each container, they are much more nimble and agile than a traditional VM as far as starting and stopping go. As mentioned earlier, we can launch containers in sub-second times, and that is quite understandable-simply since there is no operating system to boot, our app starts loading immediately.

Container vs Virtual Machine

What's the Diff: VMs vs Containers

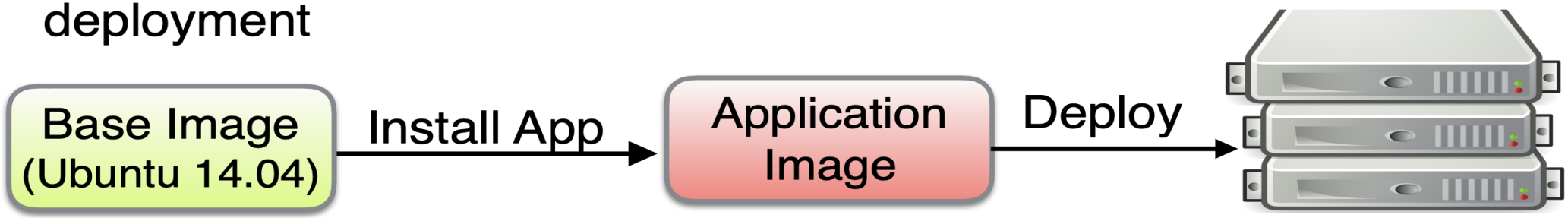
VMs	Containers
Heavyweight	Lightweight
Limited performance	Native performance
Each VM runs in its own OS	All containers share the host OS
Hardware-level virtualization	OS virtualization
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

Figure 5.2 Implementing a virtual server as a virtual machine



Performance comparison

- Getting applications from development to production involves creating disk images
- Fast image creation enables rapid testing and continuous deployment



Container images have only application layer

VMs contain entire OS, and have larger images

Time (s)	VM (Vagrant)	Docker
MySQL	236	129
NodeJS	304	49

- Docker: 2-6x faster

Size Comparison

Image size	VM	LXC	Docker
MySQL	1.68 GB	0.4 GB	**112 KB**
NodeJS	2.05 GB	0.6 GB	**72 KB**

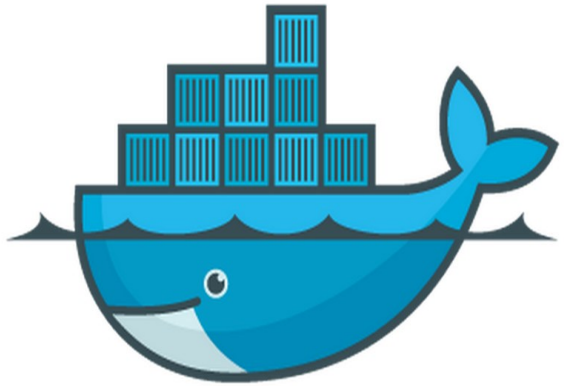
Docker: 2-6x smaller

- VMs contain entire OS, and have larger images
- Docker stores only differences (application layer)

History of Docker

2004
Solaris Containers / Zones technology introduced

2008
Linux containers (LXC 1.0) introduced



2013
Solomon Hykes starts Docker as an internal project within dotCloud

Mar 2013
Docker released to open source

Feb 2016
Docker introduces first commercial product – now called Docker Enterprise Edition

2022
Open source community includes:

- 17+ million developers
- 83,000+ customers
- 20,000+ forks

DOCKER (HTTPS://DOCS.DOCKER.COM/ENGINE/CONTAINERS/MULTI-SERVICE_CONTAINER/)

It is an open-source containerization platform by which you can pack your application and all its dependencies into a standardized unit called a container.

Containers are light-weight which makes them portable and they are isolated from the underlying infrastructure and from each other container.

You can run a Docker image as a docker container in any machine where docker is installed without depending on the OS.

WHAT IS DOCKER?

Docker is a container-based system that allows users to define the

software to be included in a container as a Docker image.

It also includes a run-time system that can create and manage containers using these Docker images

Kubernetes is used to manage multiple containers running simultaneously.

DOCKER ADVANTAGES

Portability: Docker facilitates the developers in packaging their applications with all dependencies into a single lightweight containers. It facilities in ensuring the consistent performance across different computing environments.

Reproducibility: Through encapsulating the applications with their dependencies within a container it ensures in software setups remaining consistent across the development, testing and production environments.

Efficiency: Docker through its container based architecture it optimizes the resource utilization. It allows the developers to run the multiple isolated applications on a single host system.

Scalability: Docker's scalability features facilitated the developers in making easier of their applications handling at time of workloads increment.

DOCKER ADVANTAGES

Resource Efficiency : Docker helps in maximizing the resource utilization by running the multiple containers on a single host.

It helps in reducing the infrastructure costs and improves the efficiency.

Version Control: It simples the versioning for the applications and their dependencies ensuring the consistency and making easier of collaboration across the teams.

Microservices Agility: It enables the adoption of microservices architecture, promoting the scalability, flexibility and fault isolation agile application development.

DOCKER ADVANTAGES

Portability: Docker facilities with creation of lightweight portable containers that can be unable on any machine regardless of the underlying operating systems.

Isolation: Docker through containers provides a high level of isolation with enabling the applications to run independently of each other addressing the issues that one container doesn't impact on other.

Reproducibility: With, Docker developers can easily package their applications and their dependencies into a reusable images. It allows for consistent and repoduciable builds across the development, testing and production environments.

DevOps Integration : It promotes the collaboration and automation across the software development life cycle in handing the increasing workloads.

- Docker makes development quick, simple and cross-platform portable while assisting developers in getting rid of tedious, recurring configuration activities.
- The entire Docker ecosystem consists of UIs, CLIs, APIs, and security that are designed to function in concert with one another throughout the complete application delivery lifecycle.
- **Improved portability that works smoothly**
- **Create containers automatically**
- **Reuse of Docker containers**
- **Libraries for shared containers**

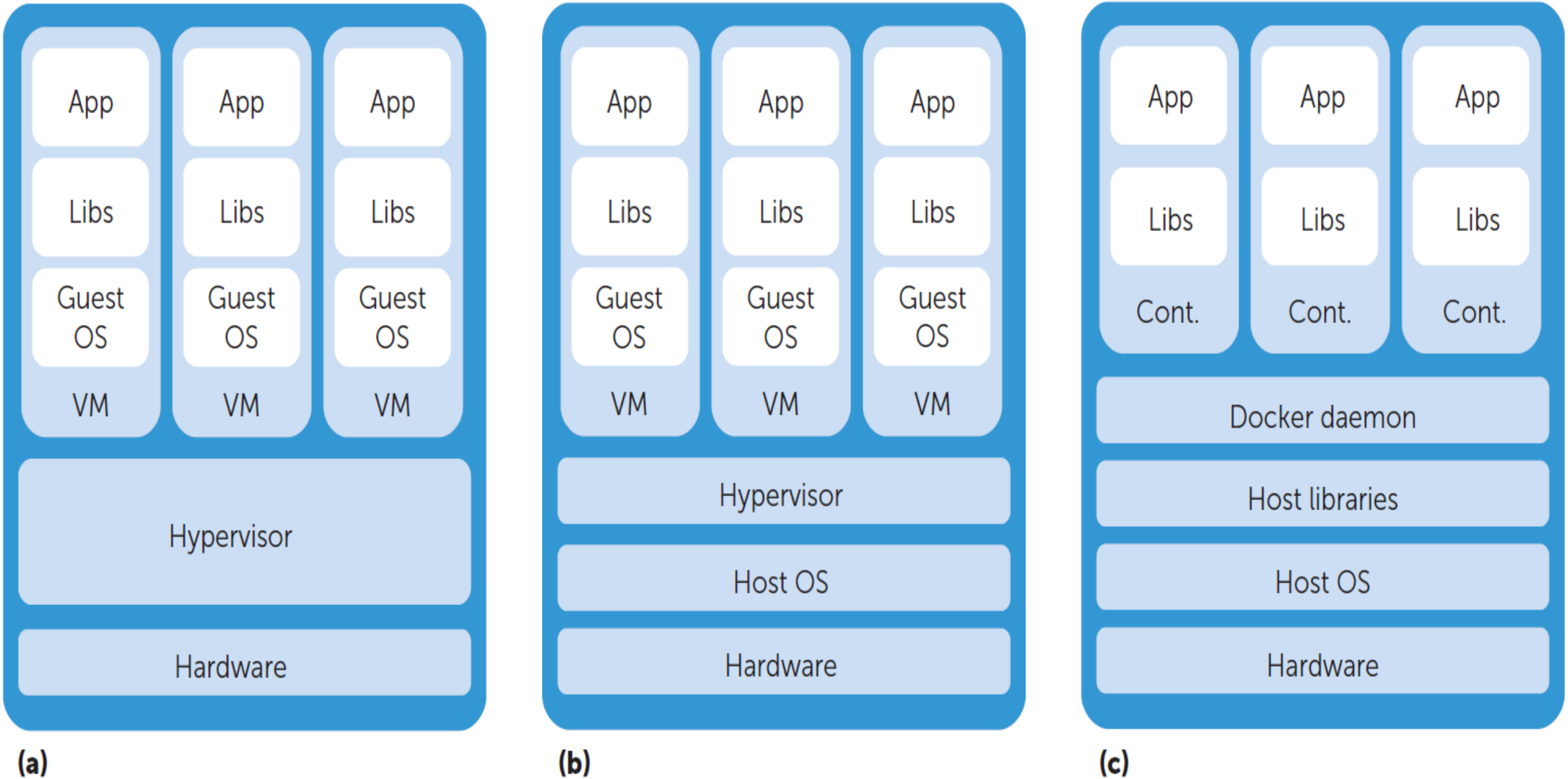


FIGURE 1. Comparing various application runtime models: (a) a type 1 hypervisor, (b) a type 2 hypervisor, and (c) a container.

KEY COMPONENTS OF DOCKER

Docker Engine: It is a core part of docker, that handles the creation and management of containers.

Docker Image: It is a read-only template that is used for creating containers, containing the application code and dependencies.

Docker Hub: It is a cloud based repository that is used for finding and sharing the container images.

Dockerfile: It is a script that containing instructions to build a docker image.

Docker Registry : It is a storage distribution system for docker images, where you can store the images in both public and private modes.

DOCKER DAEMON

Docker is produced by a San Francisco-based company called Docker Inc.

Moby is an open-source Github project created by Docker Inc to advance software containerization movement.

Docker daemon handles starting/stopping processes with:

Attach logic

Logs

TTY management

Docker run options

Events and container state

C/S ARCHITECTURE

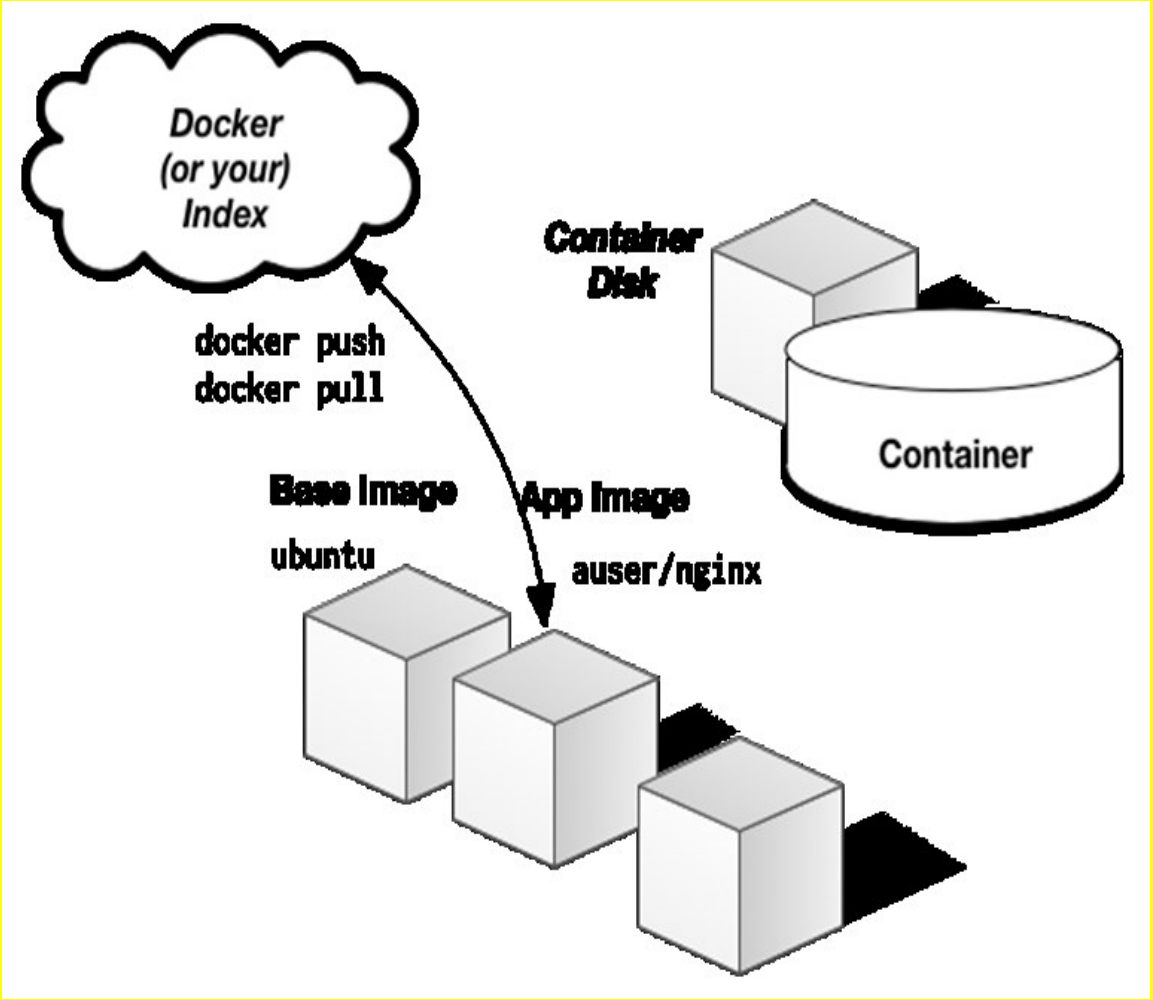
Docker is similar to Client/Server architecture.

Docker Daemon (dockers) is the server. Clients connect to the Daemon (dockerd) and requests service.

Daemon can run either on the local machine or on remote server.

Client requests are processed by the docker daemon by listening to a prespecified port number.

Docker provides an environment for automation of container creation and an interface for container monitoring and management. An application or microservice on the Cloud is represented as a series of inter-related containers.



Build. Ship. Run.

Build

- Build an ***image*** of one application
- Images are (almost) immutable.
  They cannot be modified once built

Ship

- Hosting of the image on a ***Registry***
  (e.g., Docker Hub, GitLab Registry, …)
- Public registry VS private registry

Run

- Start you application in a container
- Isolation of processes, network, filesystem, users/groups, resource control

DOCKER ENGINE

The software that hosts the containers is named Docker Engine. Docker Engine is a client-server based application. The docker engine has 3 main components:

Server: It is responsible for creating and managing Docker images, containers, networks, and volumes on the Docker. It is referred to as a daemon process.

REST API : It specifies how the applications can interact with the Server and instructs it what to do.

Client: The Client is a docker command-line interface (CLI), that allows us to interact with Docker using the docker commands.

DOCKER IMAGE

It is a file, comprised of multiple layers, used to execute code in a Docker container. They are a set of instructions used to create Docker containers.

Docker Image is an executable package of software that includes everything needed to run an application. This image informs how a container should instantiate, determining which software components will run and how. Docker Container is a virtual environment that bundles application code with all the dependencies required to run the application. The application runs quickly and reliably from one computing environment to another.

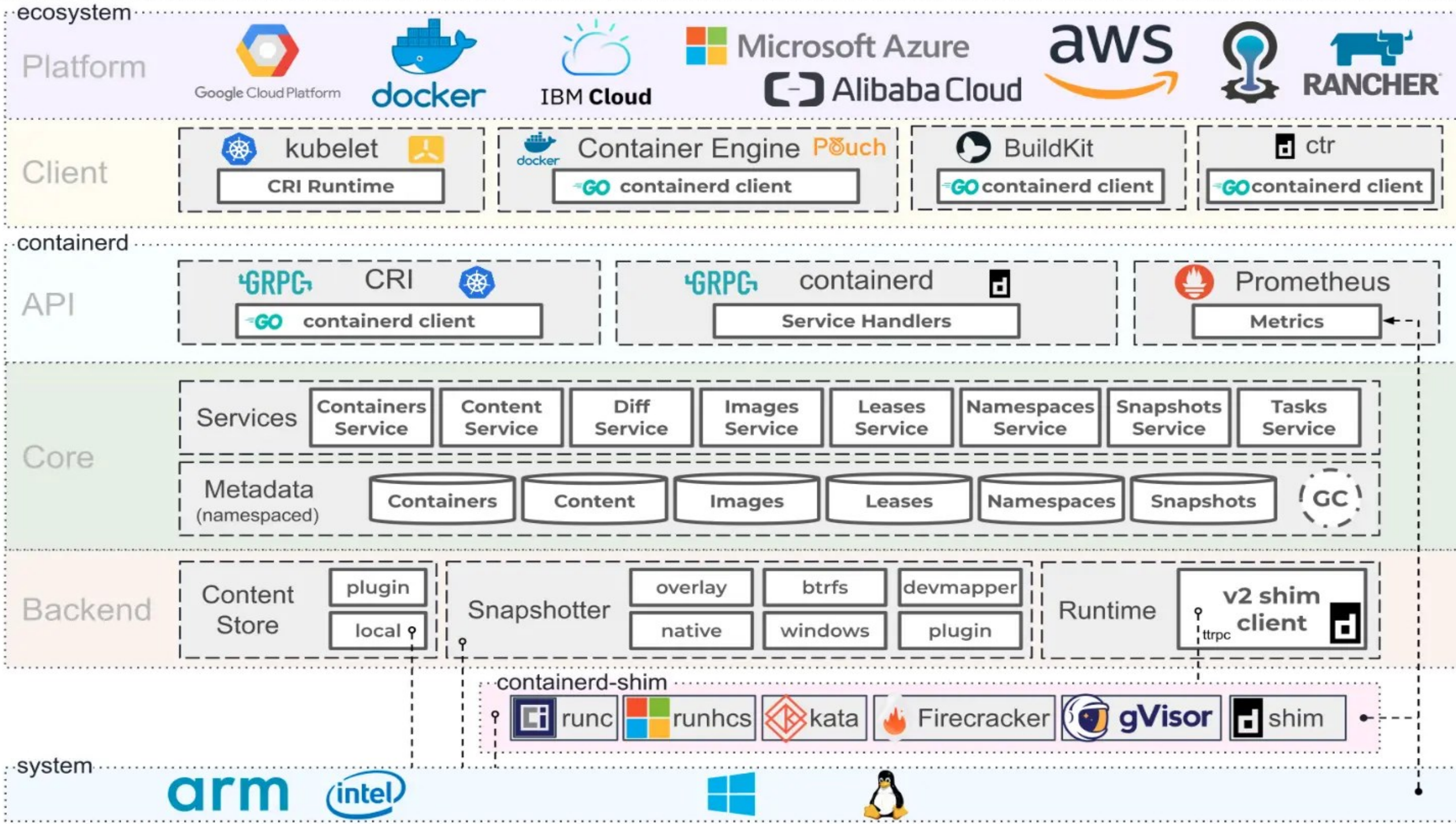
MINIMAL DOCKER

Docker adds an application deployment engine on top of a virtualised container execution environment.

You can get started with Docker on a minimal host running nothing but a compatible Linux kernel and a Docker binary.

Docker works on a copy-on-write model so that making changes to an application is incredibly fast.

As hypervisor overhead is removed, you can pack more Dockers into your hosts and make the best possible use of resources.


ecosystem
Platform
Google Cloud Platform
docker
IBM Cloud
Microsoft Azure
Alibaba Cloud
aws
RANCHER
Client
kubelet
CRI Runtime
Container Engine
containerd client
BuildKit
containerd client
ctr
containerd client
containerd
API
GRPC
CRI
containerd client
GRPC
containerd
Service Handlers
Prometheus
Metrics
Core
Services
Containers Service
Content Service
Diff Service
Images Service
Leases Service
Namespaces Service
Snapshots Service
Tasks Service
Metadata
(namespaced)
Containers
Content
Images
Leases
Namespaces
Snapshots
GC
Backend
Content Store
plugin
local
Snapshotter
overlay
btrfs
devmapper
native
windows
plugin
Runtime
v2 shim client
ttrpc
containerd-shim
runc
runhcs
kata
Firecracker
gVisor
shim
system
arm
intel

DOCKER IMAGE

Deploying code to the Cloud requires skills about the cloud environment and its various components. The environment is a collection of computing, storage, and networking resources that work together to provide a service.

The Docker image must contain an /etc/passwd file with an entry for the root user (superuser or Admin). In addition, the home directory and the shell for that root user must be present in the image file system.

The total size of the Docker image file system layers must not exceed the disk quota for the app. The maximum disk allocation for apps is set by the Cloud Controller. The default maximum disk quota supported by several providers is 2 GB per app.

WHAT IS A DOCKER FILE

The Dockerfile is a text file that uses DSL (Domain Specific Language) and contains instructions for generating a Docker image. Dockerfile will define the processes to quickly produce an image.

While creating your application, you should create a Dockerfile in order, since the **Docker daemon** runs all of the instructions from top to bottom.

(The Docker daemon, often referred to simply as "Docker," is a background service that manages Docker containers on a system.)

DOCKER VS CONTAINER

Containerization started with Linux Container (LXC) format.

LXC is an OS-level virtualization method to run multiple isolated Linux systems using a single Linux kernel.

Containers don't need its own OS so that it is much more light-weight

Containers are standalone, and executable packages that include everything needed to run a piece of software, including the code, runtime, libraries, and system tools.

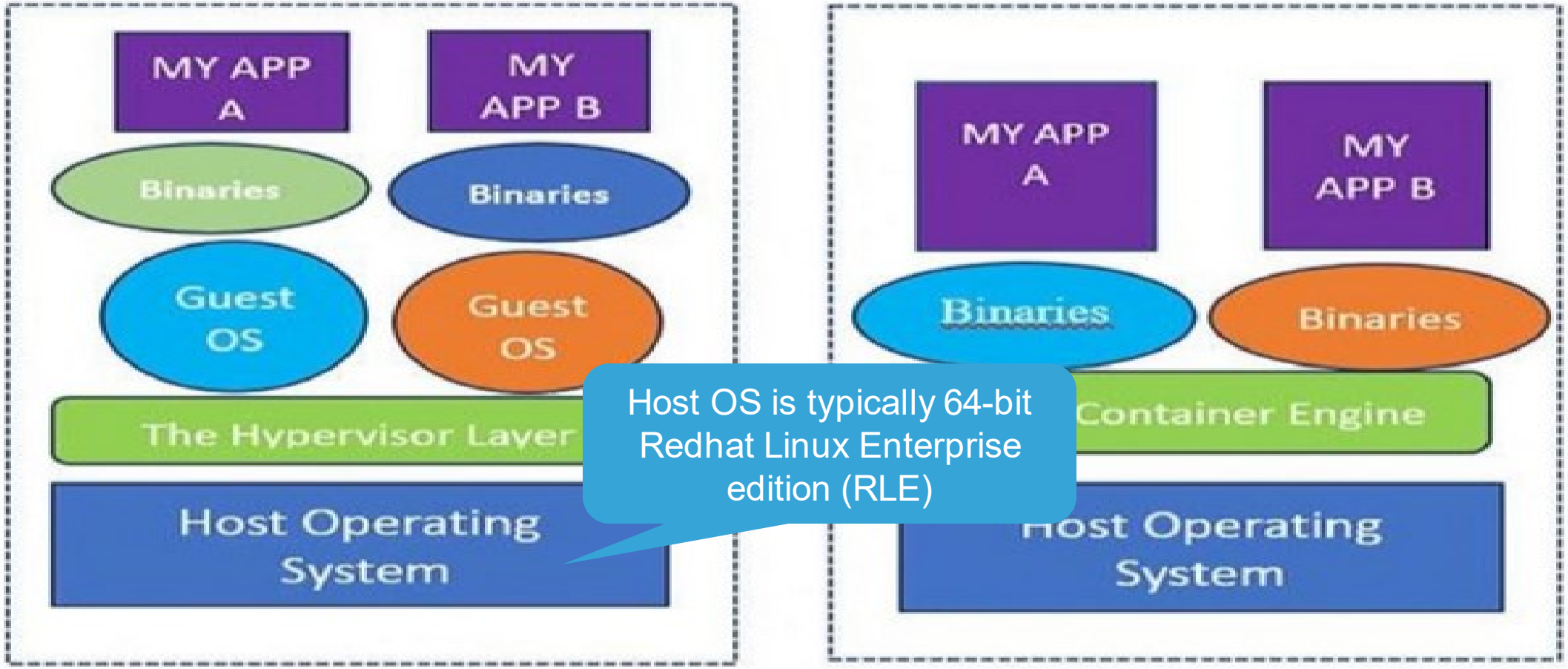
Docker is a platform for developing, shipping, and running applications in containers. Docker automates the deployment of applications in containers.

Docker Images

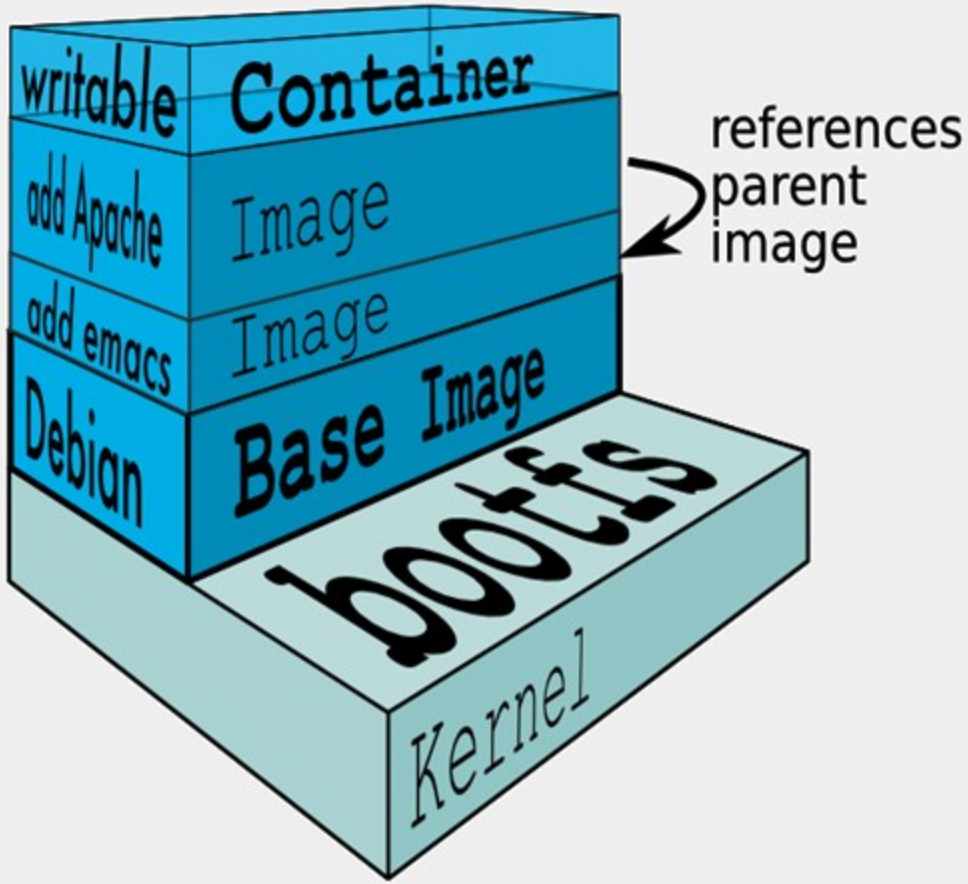
- **containerd** starts a container from a Docker image
- **dockerd** is a full RESTful API to interact with docker daemon.
- One image can launch multiple containers
- An image is built from a Dockerfile that specifies the image's attributes, files, commands, etc.
- Consider the following analogy:

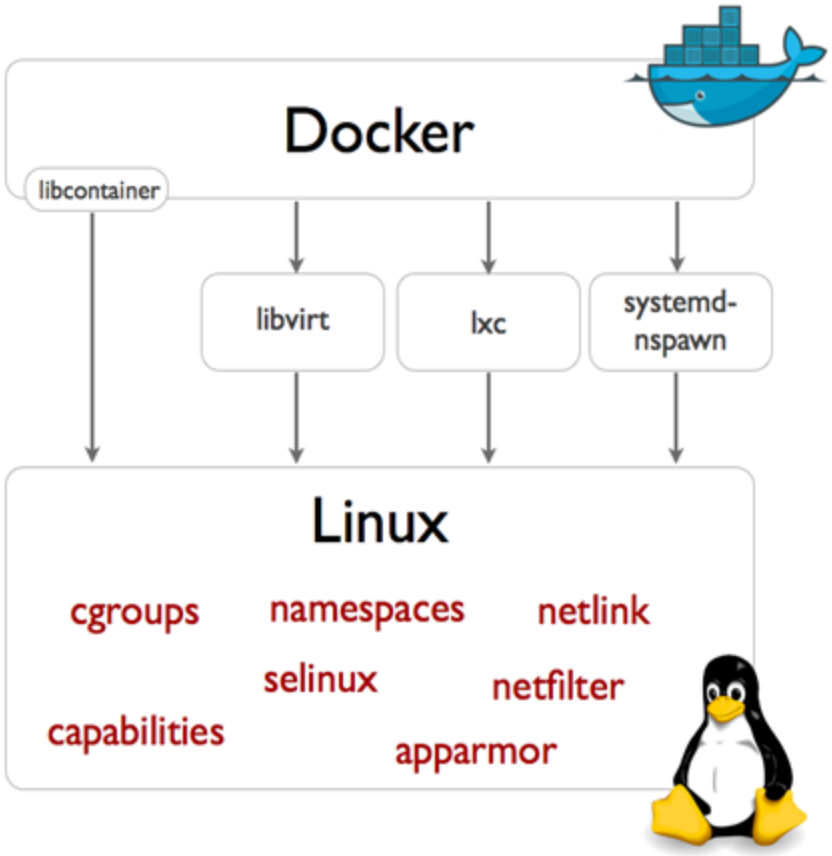
Program	Executable Binary	Process
Dockerfile	Image	Container

So, lets now take a step back and see how containerization benefits us. As discussed earlier, compared to **virtual machines (VMs)**, containers have a significantly smaller resource consumption footprint than a VM. Since the containers share the same OS kernel, we can pack in many more containers in the same host or server as virtual machines. Thus, it's really a no brainer that containers any day will be a more prudent choice for application development compared to VMs. Let's have a look at the following graphic. The graphic makes a minimalistic resource usage comparison between a virtual machine and a container.



Docker Components





Docker Resource Isolation

Cgroups : Isolation and accounting

- cpu
- memory
- block i/o
- devices
- network
- numa
- freezer

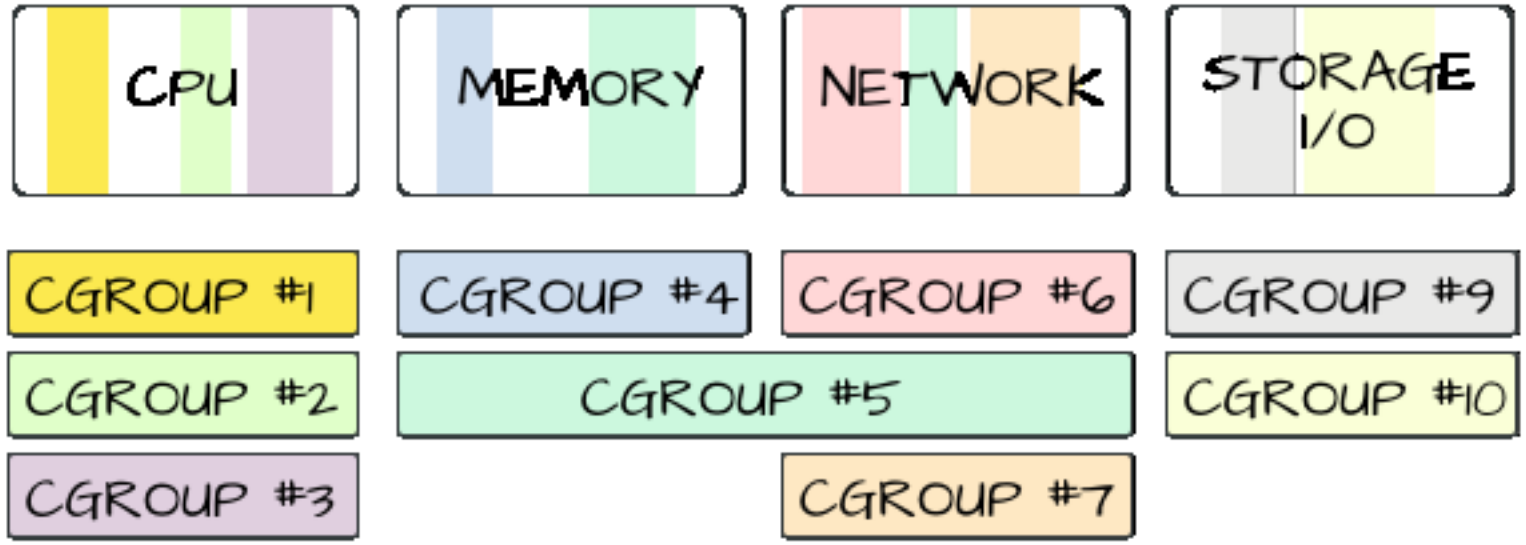


image credit: mairin

DOCKER IMAGES AND CONTAINERS

Docker Images:

- An image is a lightweight, standalone, and executable package that includes everything needed to run a piece of software.

Docker Containers:

- A container is an instance of a Docker image, running as a process in an isolated environment on the host machine.
- **Docker Hub** is a cloud-based registry service that allows you to link to code repositories, build your images, and share them with others.

HOW ARE CONTAINERS DEPLOYED?

There are a variety of tools available for container deployment.

For example, Docker is a popular container platform and runtime that people use to build and deploy containers.

The starting point for using Docker for a container deployment is to build a Docker image for your container.

You can also source an existing Docker image from Docker Hub repository, where people share prebuilt images for popular services and application needs.

DOCKERFILE

A Dockerfile is used when dockerizing a microservice/app.

The Dockerfile is used to create container images.

The Dockerfile is a text file consisting of CLI commands, used in conjunction with the 'docker build' CLI command.

An image is a binary that includes all of the requirements for running a single container, as well as metadata describing its needs and capabilities.

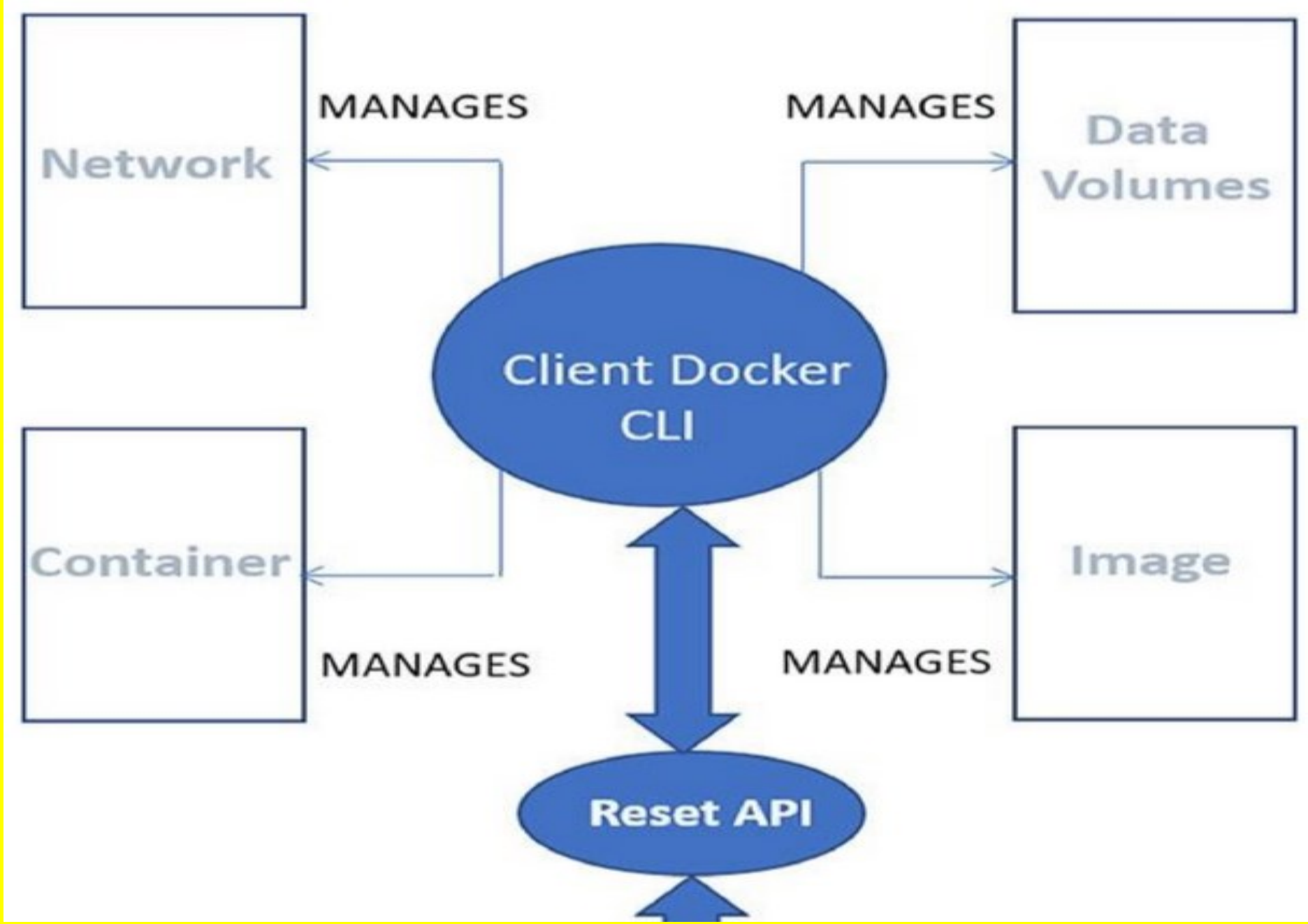
HTTPS://DOCS.DOCKER.COM/INSTALL/OVERVIEW

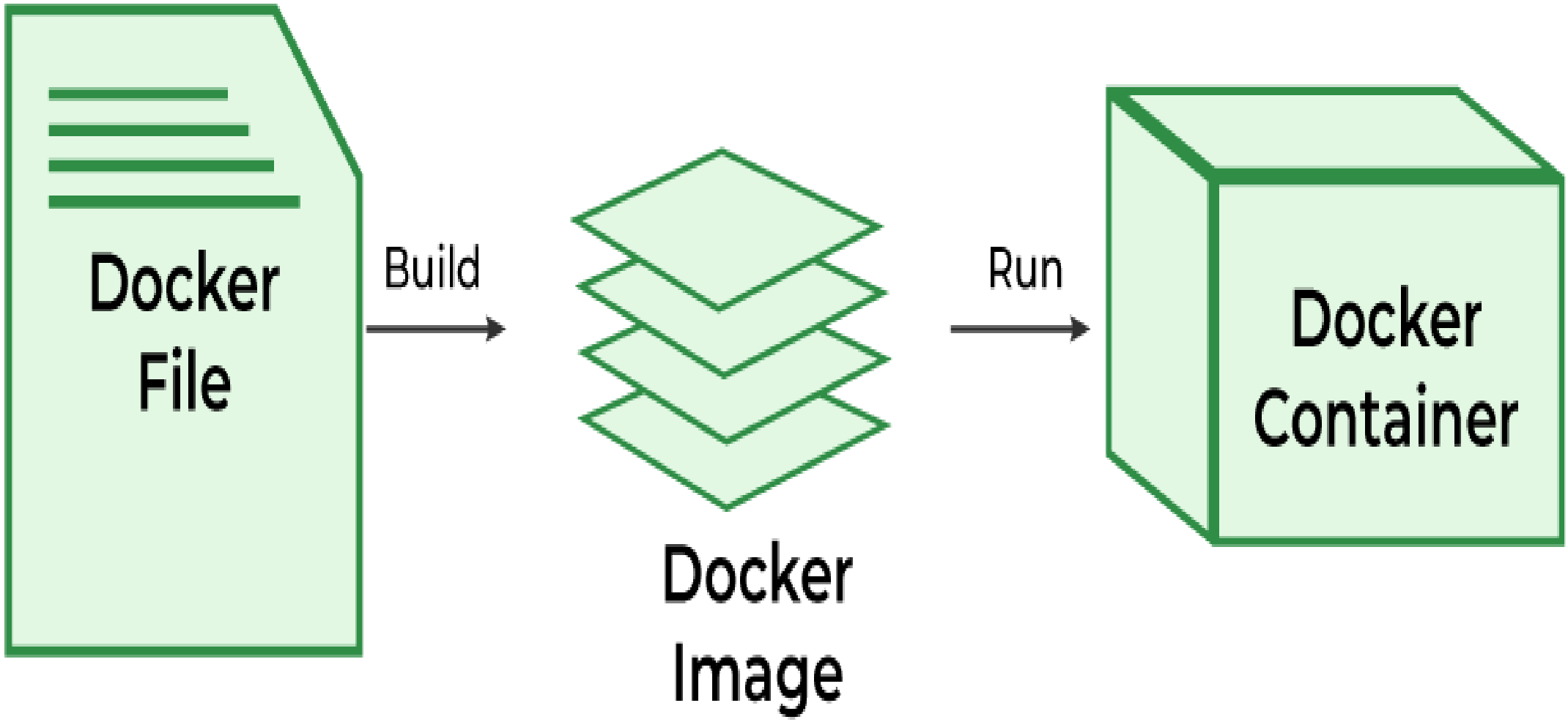
Docker can be used to pack the application and its dependencies which makes it lightweight and easy to ship the code faster with more reliability.

Docker makes it very easy to run the application in the production environment.

Docker container can be platform independent if the docker engine is installed in the machine.

Docker is the most powerful tool to run the application in the form of containers. Docker container are light in weight and can be run on any operating system.




Docker
File
Build
Docker
Image
Run
Docker
Container

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

DOCKER ARCHITECTURE

Docker makes use of a client-server architecture. The Docker client talks to the docker daemon which helps to build, run, and distribute docker containers.

The Docker client runs with the daemon on the same system or we can connect the Docker client with the Docker daemon remotely.

With the help of REST API over a UNIX socket or a network, the docker client and daemon interact with each other.

DOCKER CONTAINER

Docker container is a runtime instance of an image. Allows developers to package applications with all parts needed such as libraries and other dependencies. Docker Containers are runtime instances of Docker images. Containers contain the whole kit required for an application, so the application can be run in an isolated way. For eg.- Suppose there is an image of Ubuntu OS with NGINX SERVER when this image is run with the docker run command, then a container will be created and NGINX SERVER will be running on Ubuntu OS.

DOCKER HUB

Docker Hub is a repository service and it is a cloud-based service where people push their Docker Container Images and also pull the Docker Container Images from the Docker Hub anytime or anywhere via the internet. Generally it makes it easy to find and reuse images. It provides features such as you can push your images as private or public registry where you can store and share Docker images.

Mainly DevOps team uses the Docker Hub. It is an open-source tool and freely available for all operating systems. It is like storage where we store the images and pull the images when it is required. When a person wants to push/pull images from the Docker Hub they must have a basic knowledge of Docker. Let us discuss the requirements of the Docker tool.

DOCKER COMPOSE

Docker Compose will execute a YAML-based multi-container application.

The YAML file consists of all configurations needed to deploy containers Docker Compose , which is integrated with Docker Swarm , and provides directions for building and deploying containers.

With Docker Compose, each container is constructed to run on a single host.

DOCKER COMMANDS

Docker Run: It used for launching the containers from images, with specifying the runtime options and commands.

Docker Pull: It fetches the container images from the container registry like Docker Hub to the local machine.

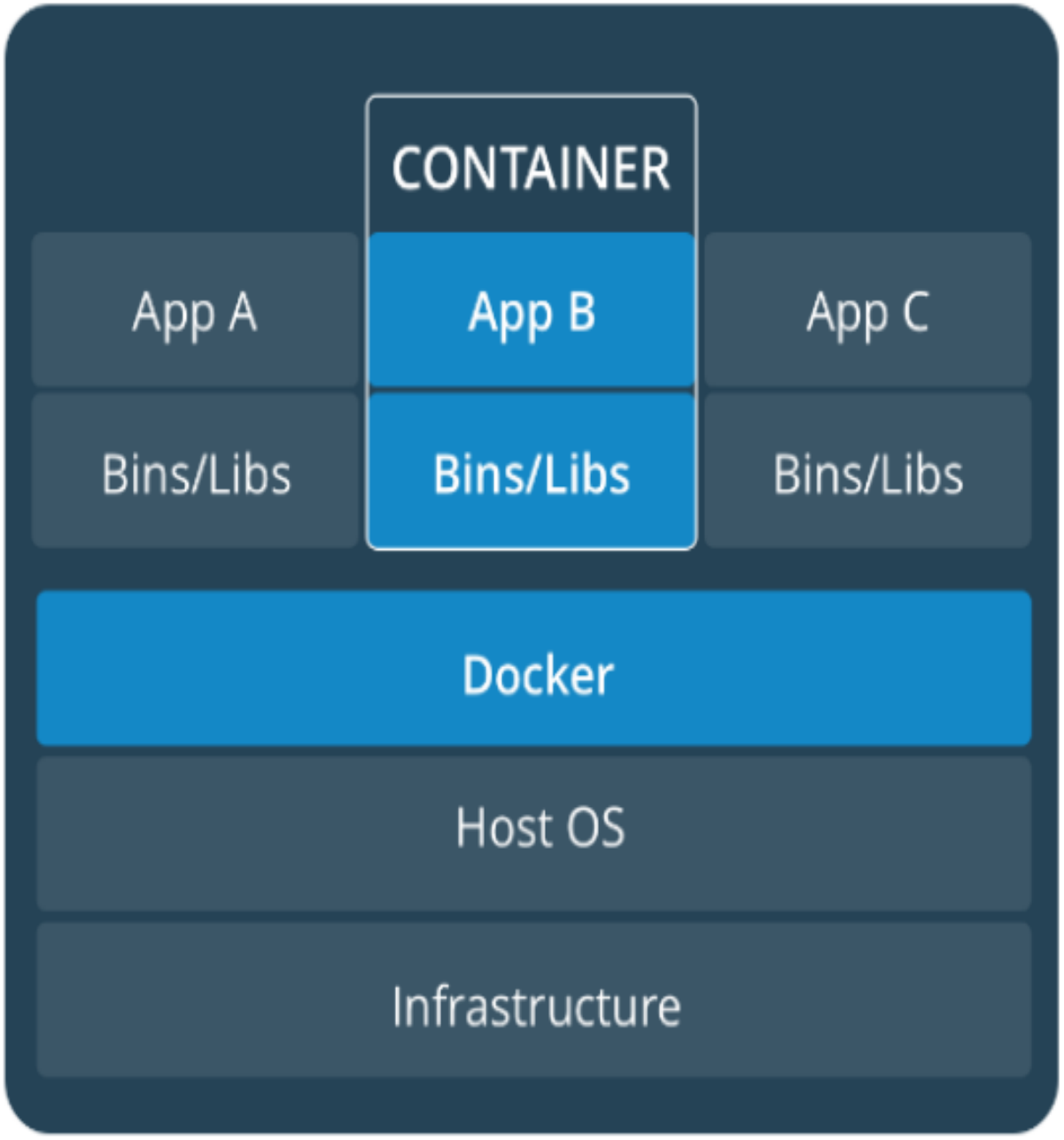
Docker ps : It helps in displaying the running containers along with their important information like container ID, image used and status.

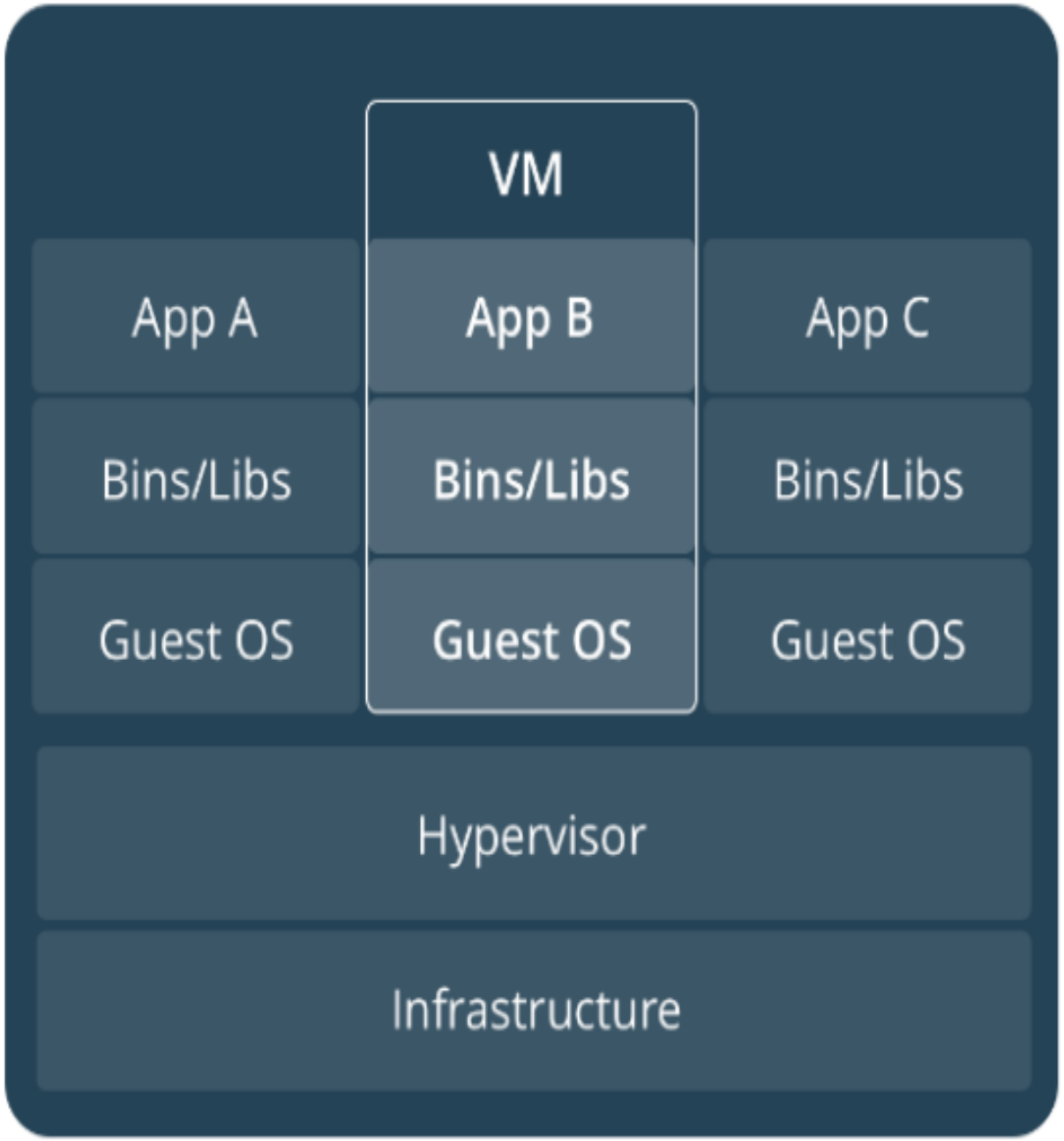
Docker Stop : It helps in halting the running containers gracefully shutting down the processes within them.

Docker rm: Removes a stopped container.

Docker Start: It helps in restarting the stopped containers, resuming their operations from the previous state.

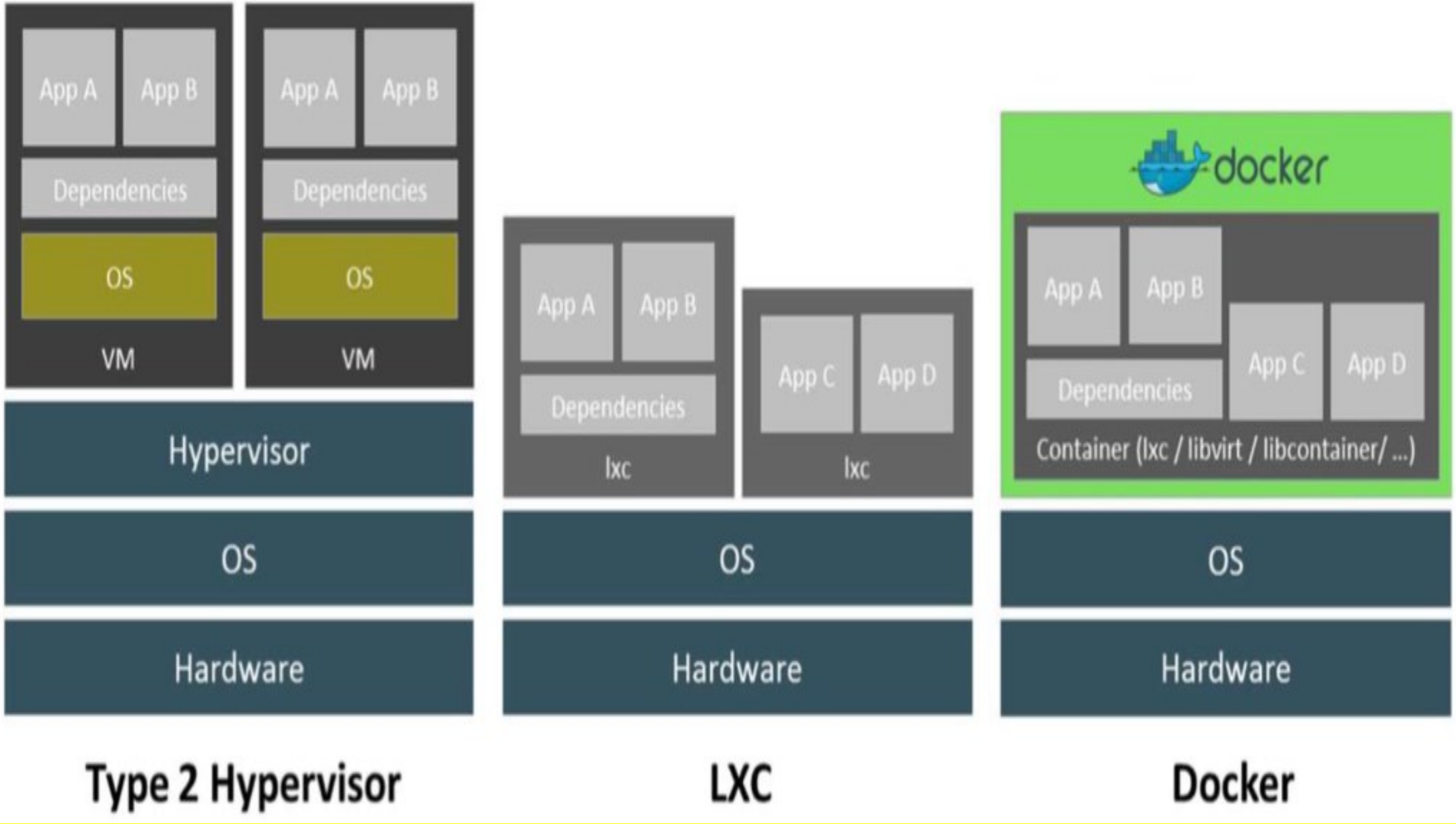
Docker Login: It helps to login in to the docker registry enabling access to private repositories.


CONTAINER
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker
Host OS
Infrastructure


VM
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure

Docker Containers	Virtual Machines
Docker Containers contain binaries, libraries, and configuration files along with the application itself.	Virtual Machines (VMs) run on Hypervisors, which allow multiple Virtual Machines to run on a single machine along with its own operating system.
They don't contain a guest OS for each container and rely on the underlying OS kernel, which makes the containers lightweight.	Each VM has its own copy of an operating system along with the application and necessary binaries, which makes it significantly larger and it requires more resources.
Containers share resources with other containers in the same host OS and provide OS-level process isolation.	They provide Hardware-level process isolation and are slow to boot.

Feature	Docker	Kubernetes
Primary Purpose	Containerization platform	Container orchestration platform
Functionality	Creates and manages containers	Manages and scales containerized applications
Setup Complexity	Simple to set up and use	More complex setup and configuration
Scalability	Limited to single-node scaling	Designed for large-scale, multi-node environments
Networking	Basic networking capabilities	Advanced networking with service discovery and load balancing
State Management	Stateless; manages individual containers	Stateful; manages container clusters and services
Use Case	Development and testing environments	Production environments with high scalability and reliability requirements


App A
App B
Dependencies
OS
VM
App A
App B
Dependencies
OS
VM
Hypervisor
OS
Hardware
Type 2 Hypervisor
App A
App B
Dependencies
lxc
App C
App D
lxc
OS
Hardware
LXC
docker
App A
App B
Dependencies
App C
App D
Container (lxc / libvirt / libcontainer/ ...)
OS
Hardware
Docker

ALTERNATIVES TO DOCKER

Podman : Offers a Docker-compatible container engine with a focus on security and compatibility, ideal for environments where Docker is not preferred or available.

rkt: A lightweight container runtime developed by CoreOS, designed for simplicity, security, and composability, offering an alternative to Docker's container runtime.

LXC (Linux Containers): Provides operating-system-level virtualization for running multiple isolated Linux systems (containers) on a single host, offering a lightweight alternative to Docker for certain use cases.

Containerd: An industry-standard core container runtime developed by Docker, Inc., offering a minimal and stable platform for building containerized applications, often used as a lower-level alternative to Docker for more advanced container orchestration systems like Kubernetes.

Points to Remember

- Containers encapsulate the run-time environment along with the application, thus providing a consistent environment for the applications to run.
- Containers are much more lightweight than virtual machines, thus making them far easier to spin up, run, and maintain.
- *Docker Inc.*, the company, provided ready-to-use container technology for commercial use.
- Docker runs both on Windows and Linux.
- Microservices and Docker are a great combination.
- Docker has built-in clustering and orchestrating technologies.
- Docker, in general, provides a fundamentally secure setup off the shelf.

QUANTUM COMPUTER ON CLOUD

Some cloud providers are already offering Quantum computers on shared basis, however the pricing differs from conventional computers

IBM, Google, Microsoft, Amazon, Alibaba and many others have CLOUD-BASED Quantum computers in selected countries.

This will speed-up Cloud-based applications by a factor of 10^5

QUANTUM ON THE CLOUD

IBM Quantum is a pioneer in the QCaaS landscape.

Azure Quantum, developed by Microsoft, is another key player in the QCaaS market.

AWS Braket, Amazon Web Services' quantum computing service, provides a comprehensive environment for exploring and developing quantum algorithms.

SPINQ QUANTUM COMPUTER COSTS $5000/-