Applied Virtual Networks

COMP 4912

Instructor: **Dawood Sajjadi**

PhD, SMIEEE, CISSP

ssajjaditorshizi@bcit.ca

Winter-Spring 2025

Week #2

BCIT™

Cloud Computing (Cloud Technology)

RECAP

What is Cloud Technology?

Cloud Technology enables access to computing resources like storage, databases, servers, and software over the internet.

Types of Cloud Models

Public Cloud: Shared resources provided by third-party vendors.
Private Cloud: Dedicated resources for one organization.
Hybrid Cloud: Combination of public and private clouds.
Multi-Cloud: Use of multiple cloud services from different providers.

Benefits

Scalability: Adjust resources based on demand.
Cost Efficiency: Pay-as-you-go pricing models.
Flexibility: Access from anywhere with internet.
Collaboration: Shared environments for teams and organizations.

BCIT™

Cloud Computing (Cloud Technology)

RECAP

Cloud service models provide different levels of management, from full applications to bare infrastructure

1. **SaaS (Software-as-a-Service)**:

Minimal user management (just access and settings).

NETFLIX Gmail Office 365

2. **PaaS (Platform-as-a-Service)**:

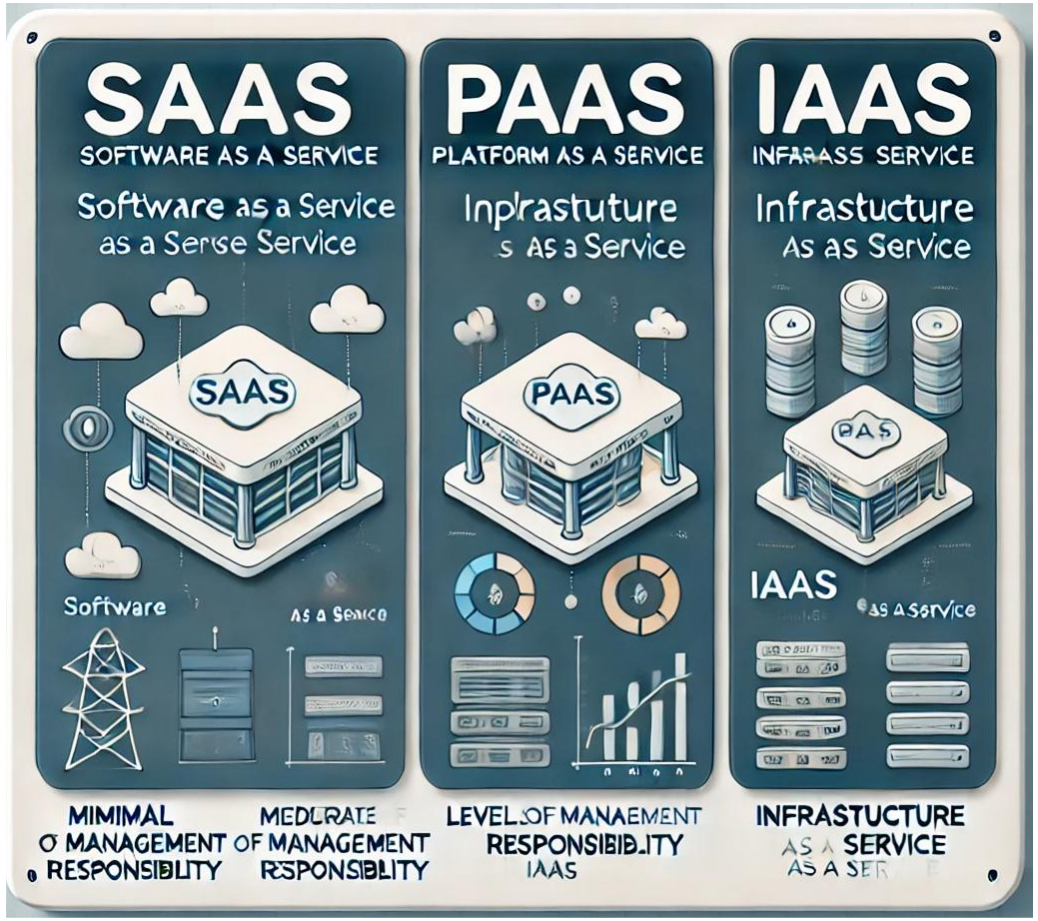
User manages applications and data, but not infrastructure.

HEROKU App Engine OPENSHIFT

3. **IaaS (Infrastructure-as-a-Service)**:

User manages applications, data, and operating systems, but the provider manages hardware.

BCIT™ Amazon EC2 Google Cloud Platform Azure



A Brief Intro to Linux

RECAP

- ✓ Master the Basics/Concepts
- ✓ Learn by Building Small Projects
- ✓ Embrace Curiosity
- ✓ Take Advantage of Online Materials
- ✓ Join Linux Communities

Some basic commands

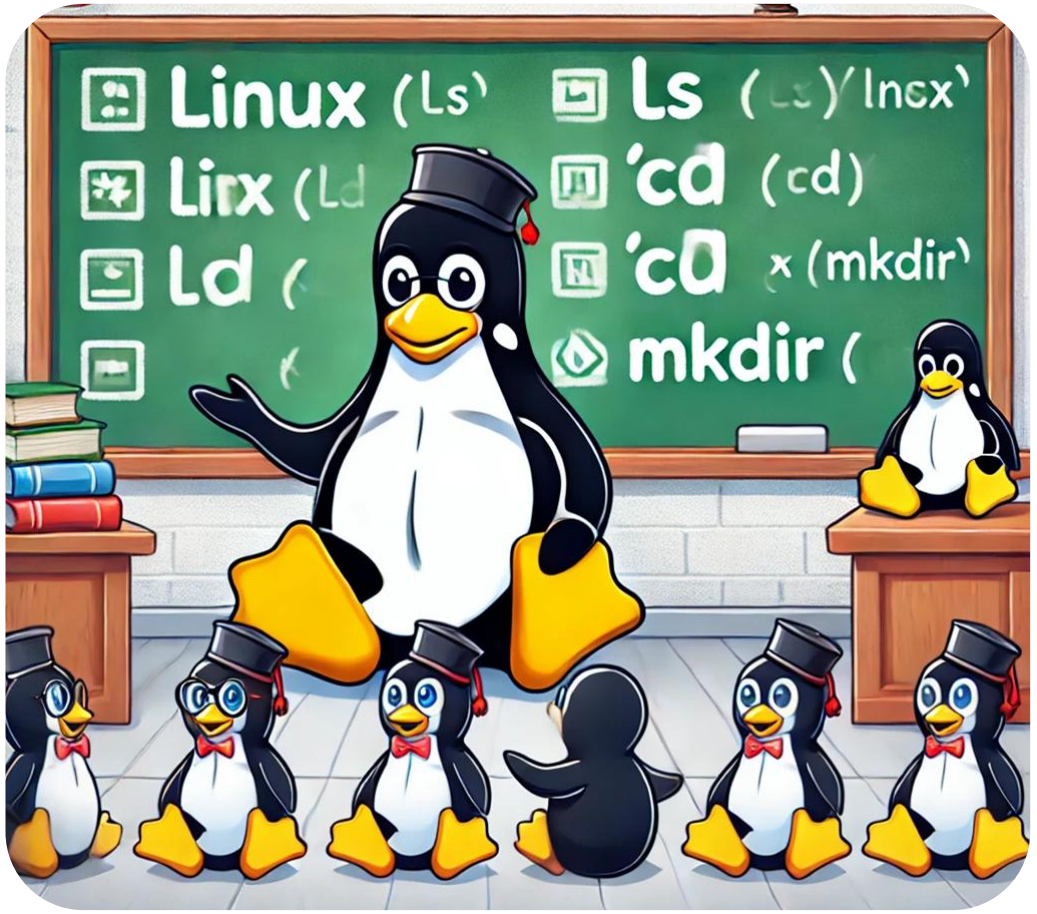
cd, ls, pwd, hostname
cp, mv, rm, mkdir, rmdir, man
cat, vim, nano, touch, find, grep

uname, df, du, tar, fdisk, whoami

top, ps, kill, jobs, init, uptime, date

sudo, yum/apt, ssh, scp, history

curl, wget, ping, mtr, ip, ifconfig
netstat, iptables



Some Nice References

https://linuxcommand.org/

https://linuxjourney.com/

https://linuxsurvival.com/

BCIT™

Learning Outcomes of Week #2

1. Explain the key applications of Virtualization technologies.
2. Name dominant Virtualization technologies.
3. Describe how can create and manage Virtual Machines (VMs).
4. Explain Container concept and its application.
5. Compare in details pros/cons of VMs and Containers.
6. Learn about Docker and basic commands to run/create a container.

BCIT™

Historical Overview

Multitenancy, containers and virtualization

1. (1960s-1970s) Time sharing and virtual memory;
2. (1970s) First hypervisor and VMs in IBM mainframes
3. (1980s-1990s) VMs on non-mainframe systems;
4. (2000 - 2005) FreeBSD Jails, Linux VServer, Virtuozzo/OpenVZ;
5. cgroups (~2006), LXC (2008), LXC-based Docker (2013), LXD (2014)

Virtualization

Operating Systems share HW resources between processes

Hypervisors share HW resources between Virtual Machines (VMs)

- ✓ each VM has independent OS, utilities, libraries.
- ✓ sharing HW across VMs improves utilization.
- ✓ allows quick recovery of OS/Applications.
- ✓ limits users' access to specific roles (not root/admin).
- ✓ allows to run applications regardless of underlying hardware or OS (more portability).

Virtualization does have performance impacts

- ✓ Contention between VMs has nontrivial overheads
- ✓ Un-tuned VMs may miss important memory features
- ✓ Mismatched scheduling of VMs can slow multi-CPU runs
- ✓ I/O virtualization is still costly

BCIT™

The Need for Virtualization and Containers

Sharing hardware via an operating system

Multi-tenant hosts:

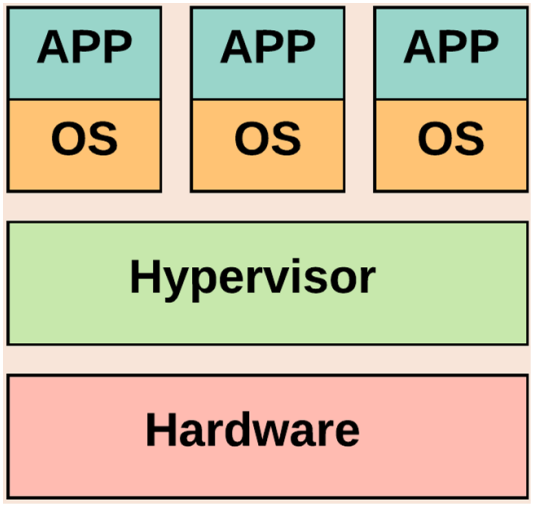
- Multiple users
- Processes can be attributed to different users and groups
- Processes share resources of an operating system (CPU, RAM, network devices, I/O devices, etc. via OS kernel)
- IP address sharing via ports

Tenant separation:

- Privileges are applied via an effective user and group and capabilities
- File system permissions isolate access to regular files, directories, special files
- Each process gets its own virtual memory
- Context switches save/restore CPU state

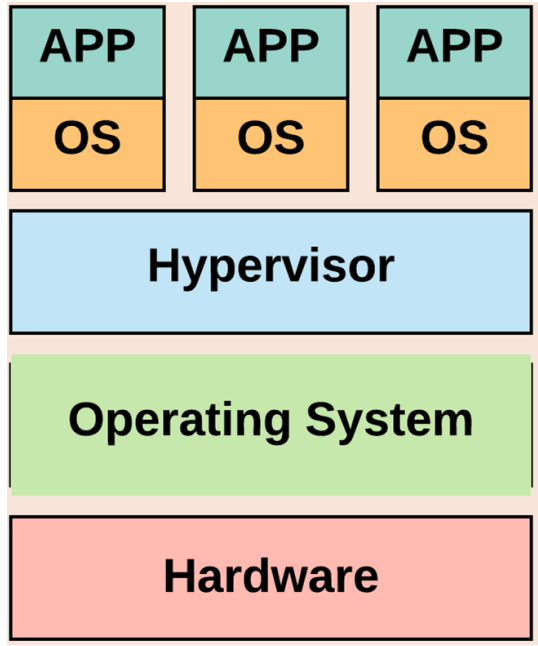
Virtual Machines

Types of Hypervisors



Type-1

Bare-metal



Type-2

Run guest-OS on top of host-OS

BCIT™

KVM

VMware vSphere®

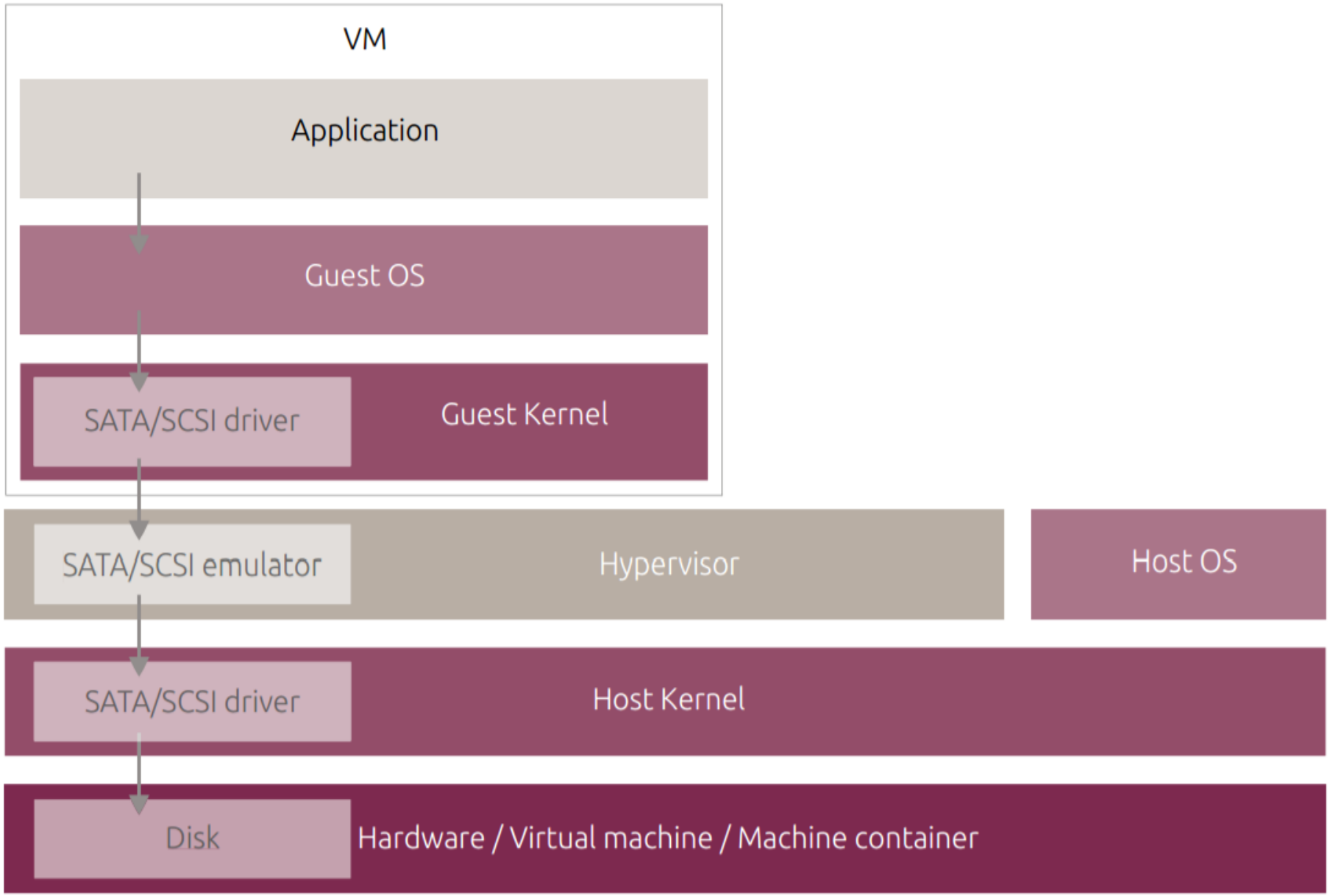
Hyper-V

Xen™

VirtualBox

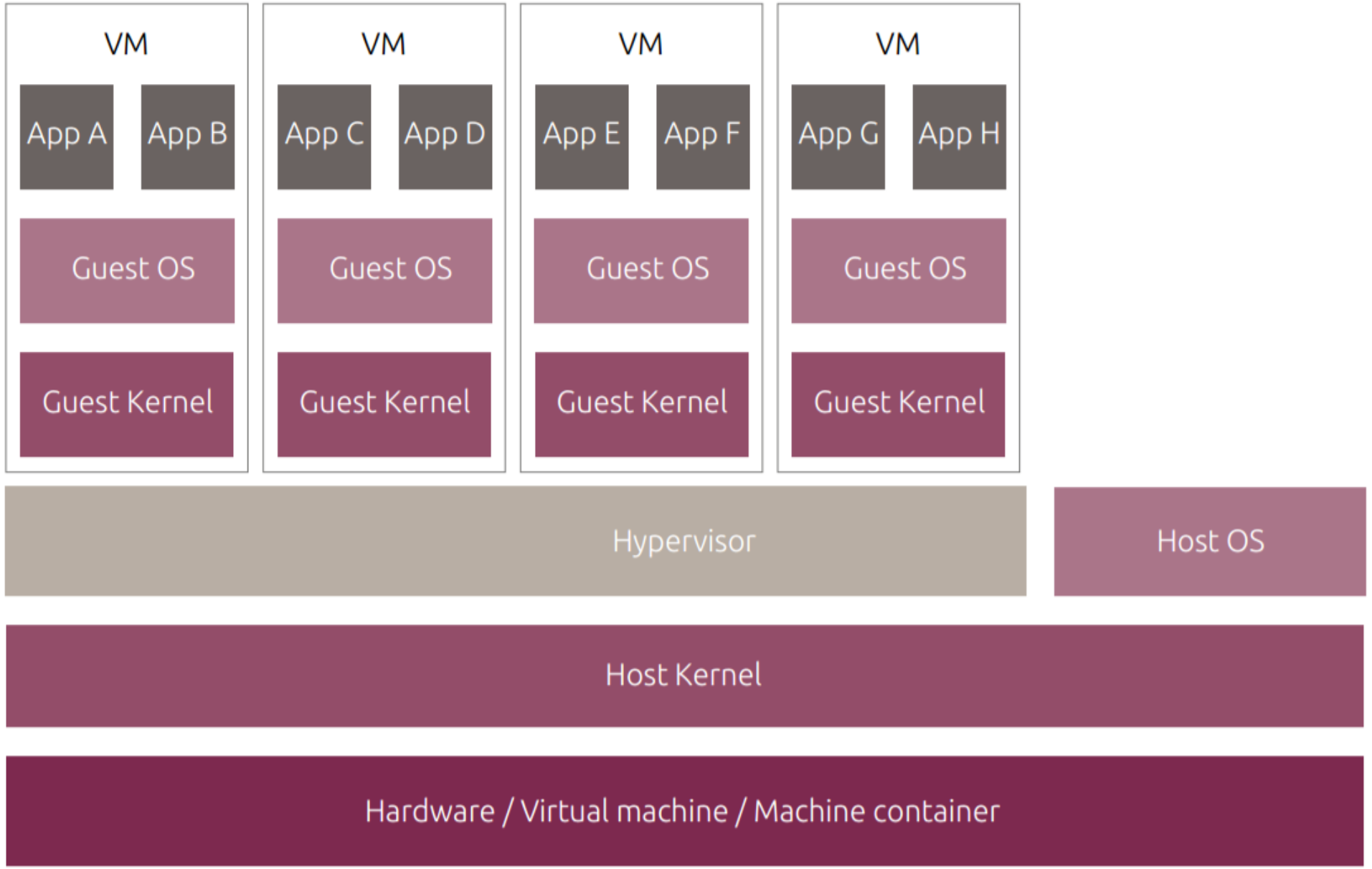
QEMU

Virtual Machines



BCIT™

Virtual Machines



BCIT™

Virtual Machines (VMware)

VMware Workstation Professional

- ✓ VMware Workstation Professional is a **desktop virtualization software** that allows you to run multiple operating systems (OS) on a single physical machine.
- ✓ It creates a **Virtual Environment** where you can test, develop, and run different OSs without affecting your main system.
- ✓ Supports popular operating systems like **Windows**, **Linux**, and **macOS** (via virtual machine emulation).

VMWARE WORKSTATION PRO™ 17

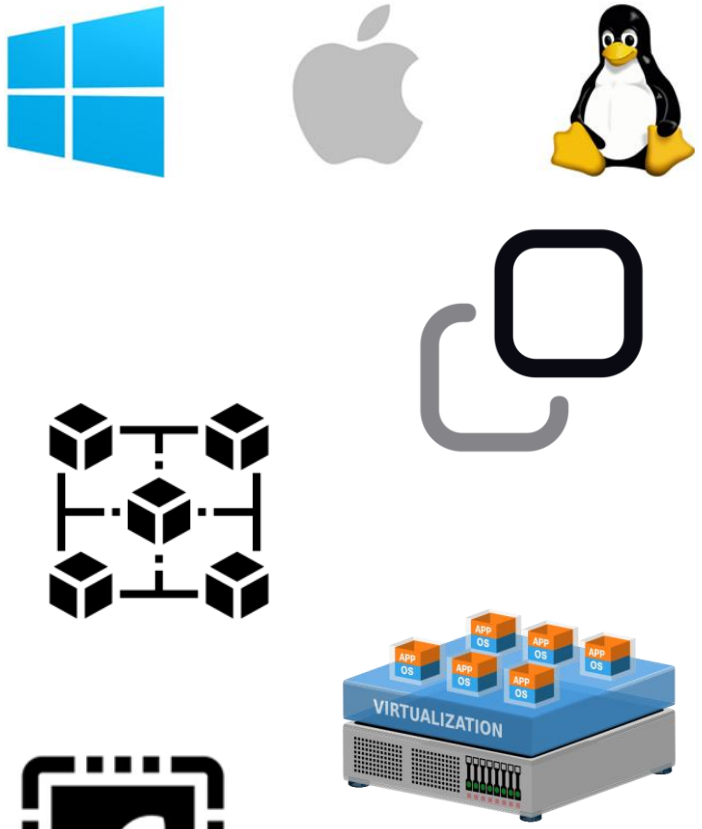
VMWARE FUSION® PRO 13

BCIT™

Virtual Machines (VMware)

Key Features (VMware Workstation Professional)

- ✓ **Multiple OS Support:** Run multiple operating systems simultaneously (e.g., Windows, Linux)
- ✓ **Snapshots & Cloning:** Create and restore snapshots of VMs, clone VMs to replicate environments.
- ✓ **Virtual Networking:** Test networking configurations with virtual networks, including NAT/Bridged/Host-only modes.
- ✓ **Hardware Virtualization:** Leverage advanced CPU and memory features for optimal performance.
- ✓ **3D Graphics Acceleration:** Run graphic-intensive applications, including 3D software/games, inside VMs.



BCIT™

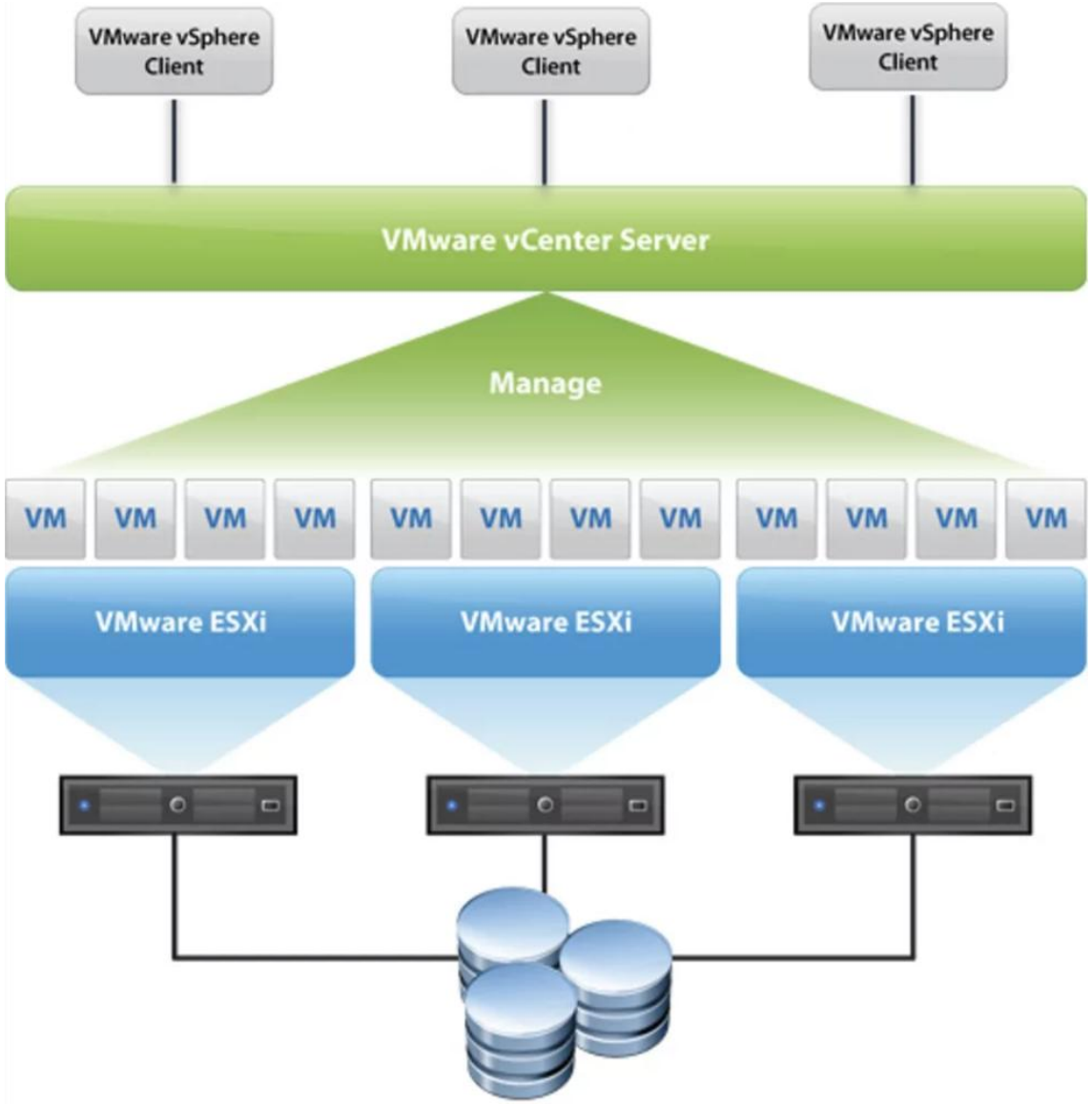
Virtual Machines (VMware)

Virtualization Capabilities in VMware Workstation

Isolated Testing Environments: Safe environments for testing software or operating systems without risk to your main OS.

Networking Simulation: Create complex networks with virtual routers, switches, and virtual machines.

Integration with VMware vSphere: Seamlessly move virtual machines between VMware Workstation and vSphere, enabling hybrid cloud environments.

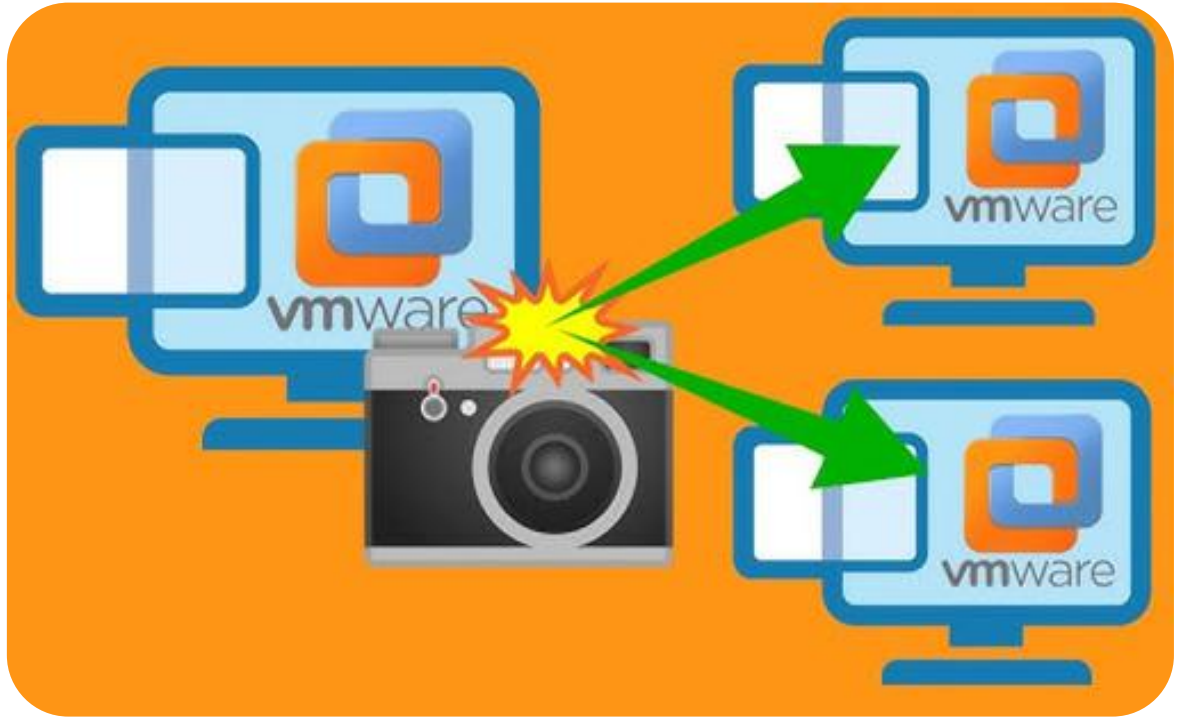


BCIT™

Virtual Machines (VMware)

Snapshots/Cloning

- ✓ **Snapshots:** Take a snapshot of your VM at any time to save its state. Revert to it later if needed.
- ✓ **Cloning:** Create an exact copy of a virtual machine for testing different configurations or versions of software.
- ✓ **Use Cases:** Test software updates, security patches, or other changes without affecting your primary environment.



BCIT™

Virtual Machines (VMware)

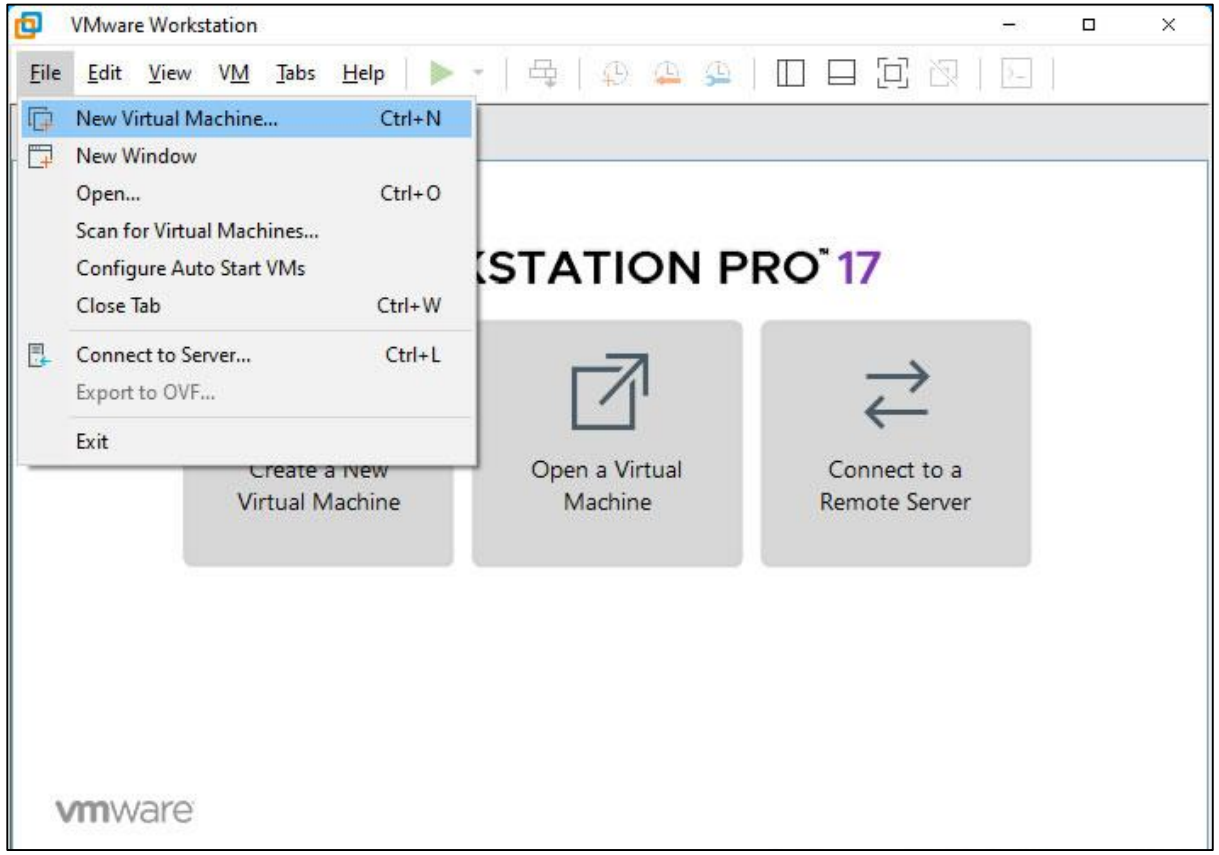
Why use VMware Workstation Professional?

Safety: Test new software or configurations in a safe, isolated environment.

Cost-effective: No need for multiple physical machines. Save on hardware costs.

Flexibility: Easily switch between different OSes without rebooting.

Learning Tool: Perfect for students, developers, and IT professionals to learn and experiment with various operating systems and configurations.

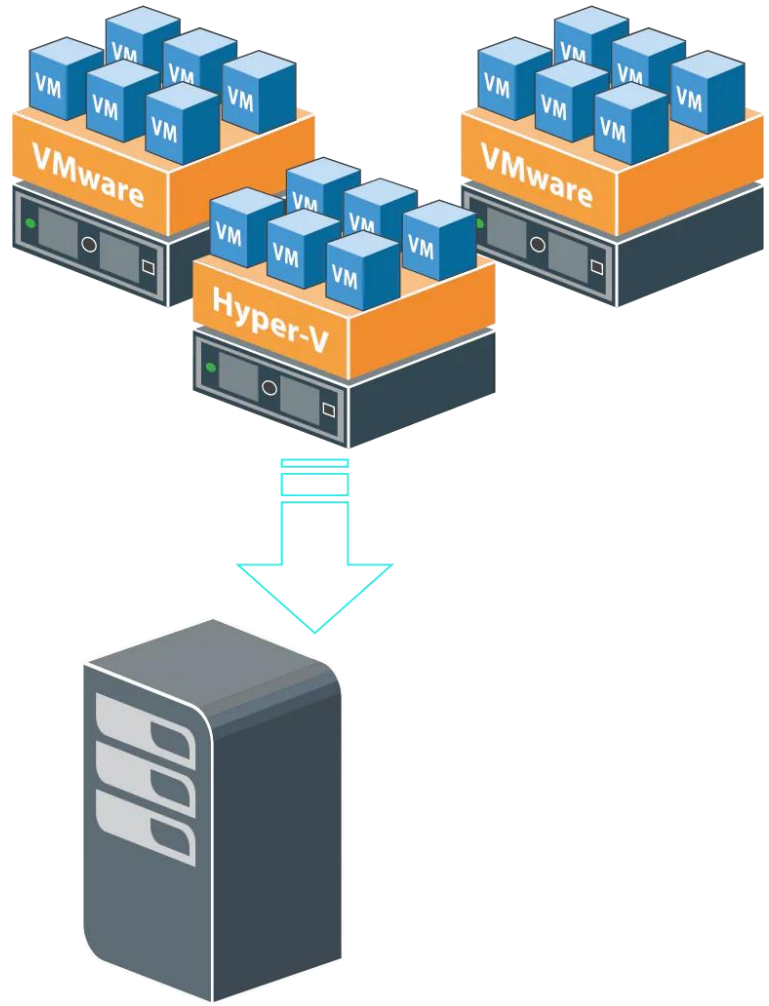


BCIT™

Virtual Machines (VMware)

System Requirements for VMware Workstation

- ✓ A compatible 64-bit x86/AMD64 CPU launched in 2011 or later
- ✓ 1.3GHz or faster core speed
- ✓ AMD CPU requires AMD-V. Intel CPU requires VT-x
- ✓ 2GB RAM minimum/ 4GB RAM or more recommended
- ✓ VMware Workstation Pro and Player run on most 64-bit Windows or Linux host OS
- ✓ VMware Workstation 17 supports hundreds of 32-bit and 64-bit guest OS. Here is a list of the most popular:
    - **Windows**
    - **Ubuntu**
    - **RedHat**
    - **SUSE**
    - **Oracle Linux**
    - **Debian**
    - **Fedora**
    - **OpenSUSE**
    - **Mint**
    - **CentOS**
    - **Solaris, FreeBSD, and various other Linux Distros**



BCIT™

Reference: https://docs.vmware.com/en/VMware-Workstation-Pro/17.5.2/rn/vmware-workstation-1752-pro-release-notes/index.html

Virtual Machines (VMware)

Download CentOS 8 VMware Image:
https://www.linuxvmimages.com/images/centos-8/

In Windows, ensure 7z (https://7-zip.org/download.html) installed.

BCIT™

Interactive Demo

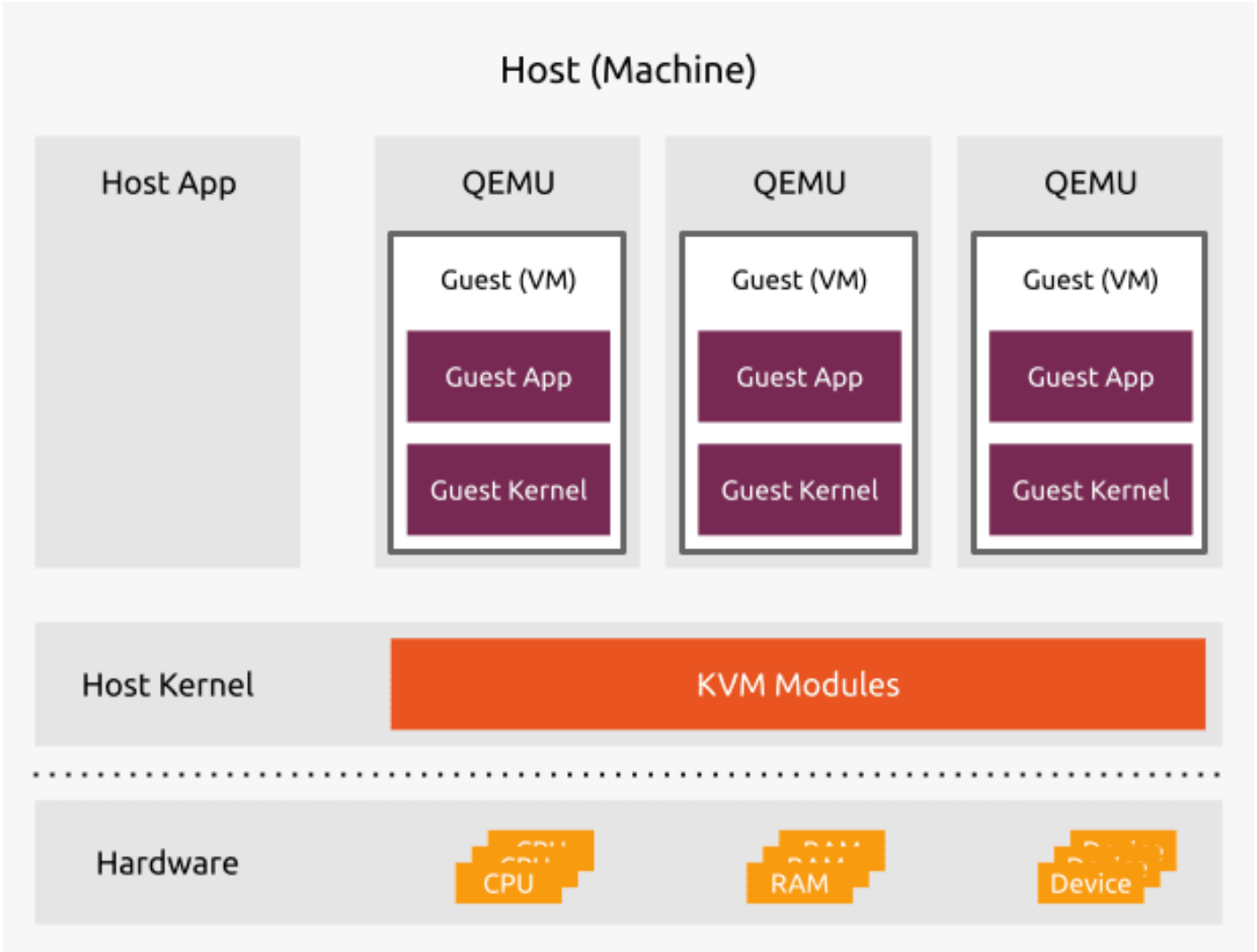
&

1 Breakout Rooms

Virtual Machines (KVM)

KVM hypervisor enables full virtualization capabilities. It provides each VM with all typical services of the physical system, including virtual BIOS/Hardware, such as processor, memory, storage, and network cards. As a result, every VM completely simulates a physical machine.

KVM is available as a **Linux Kernel module**. It plugs directly into the kernel's code and allows it to function as a **Hypervisor**. Every VM runs as a separate Linux process with dedicated virtual hardware. KVM can only be used on a CPU with extensions such as Intel-VT or AMD-V.



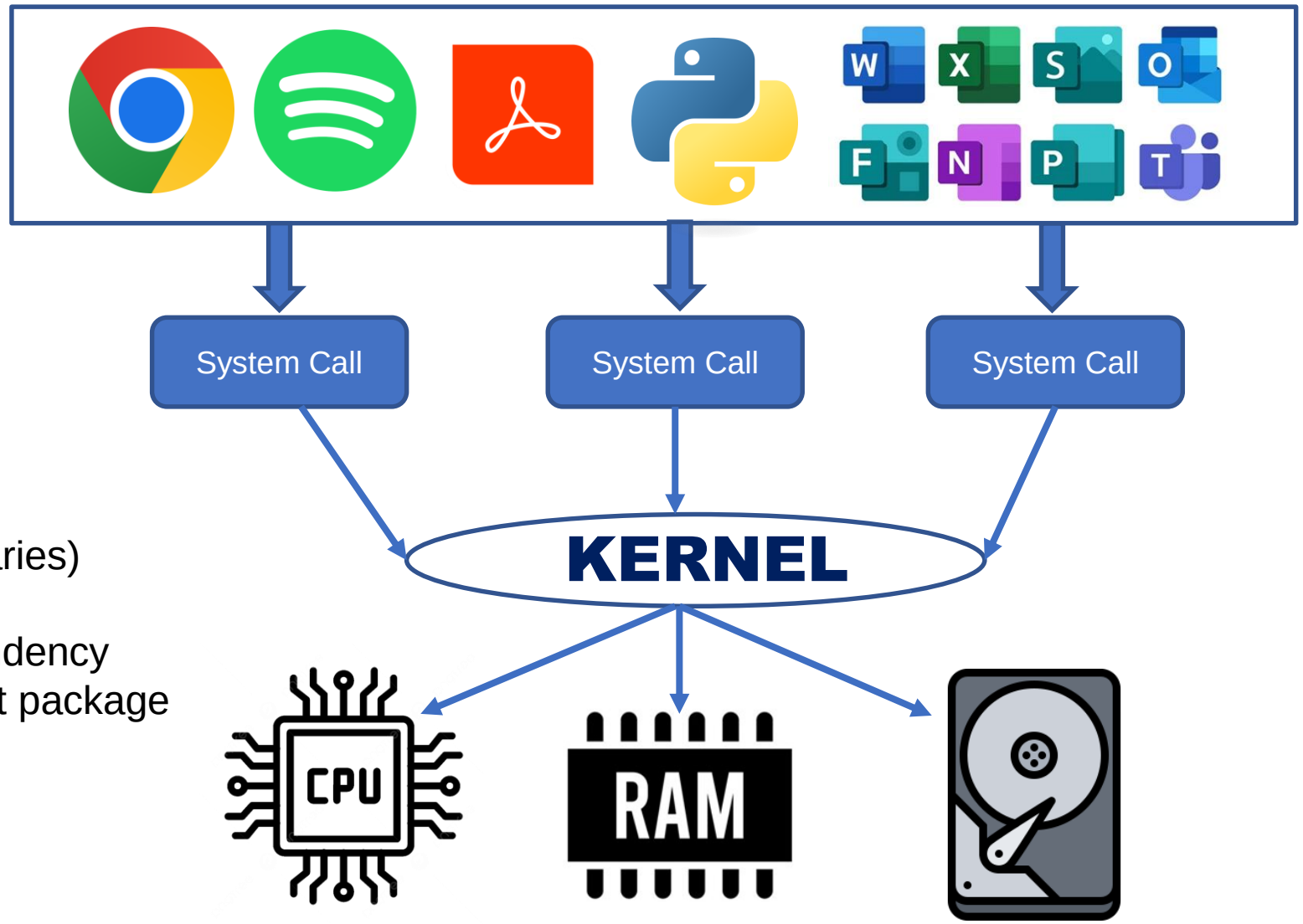
Will discuss it more in the next Lecture.

BCIT™

What is Container and Why use it?

BCIT™

A Typical Coding Day

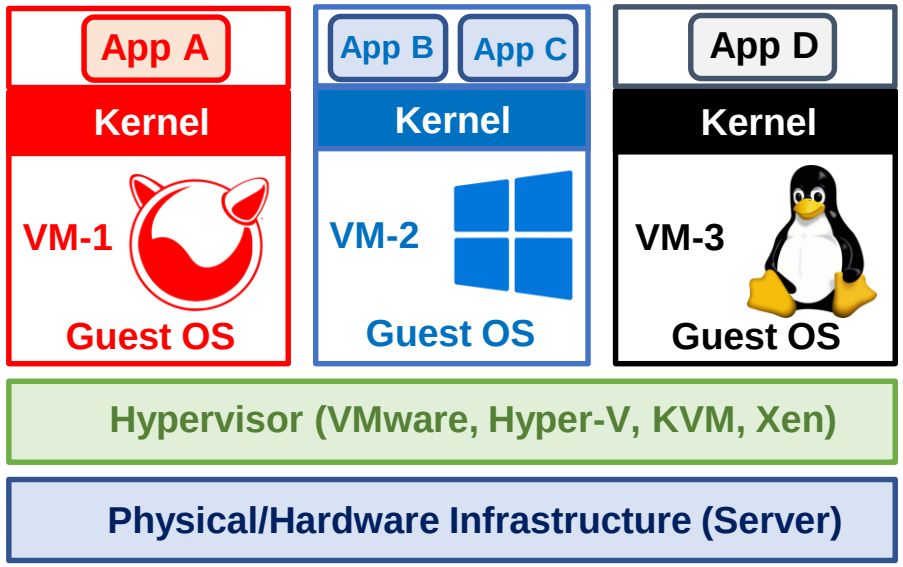


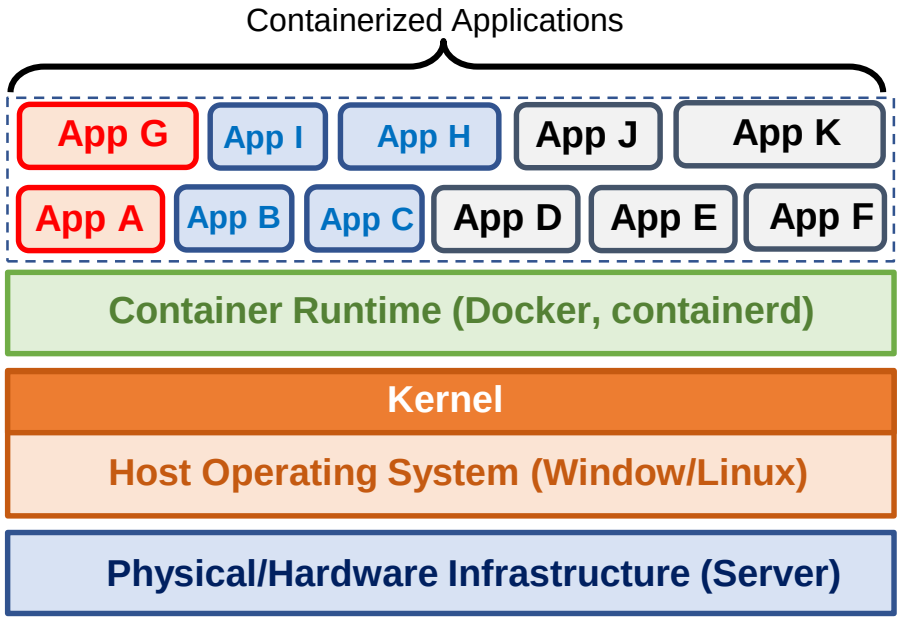
1. Build code (4x languages, many libraries)
2. Doesn’t work; install missing library
3. Requires different version of a dependency
4. Install new version, breaking different package

BCIT™

Definition of a Container

A **Container** is a **process or a process tree** isolated from other processes via Linux kernel mechanisms such as **namespaces** and constrained in resource usage via **cgroups** and **filesystem**-specific features.





BCIT™

Containers vs Virtual Machines

Virtual Machines

Containers

BCIT™

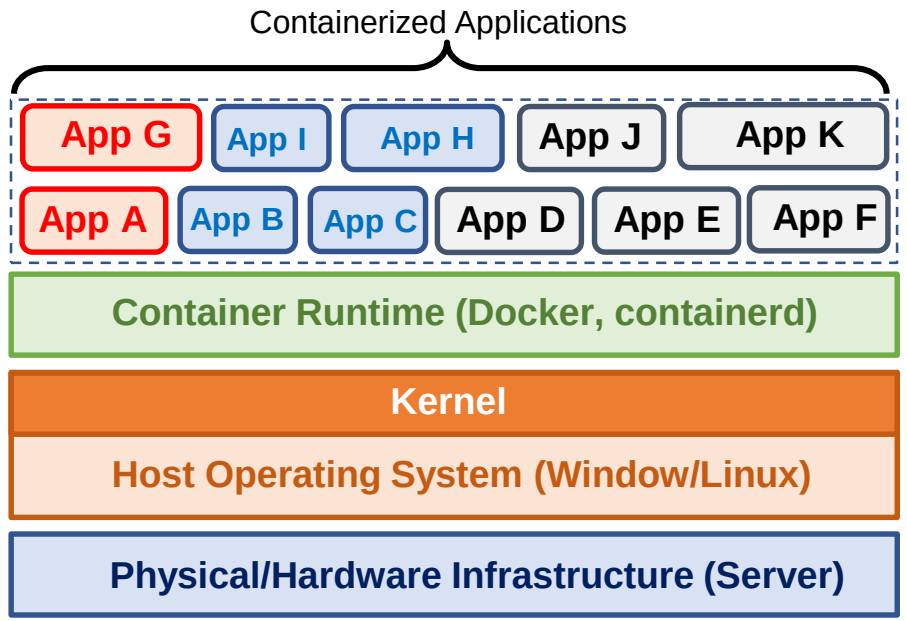
Why use Containers?

1. Containers are best suited to deliver micro-services by providing portable, isolated virtual environments for applications to run without interference from other running applications. Each container has its own file-system, processes, no shared libs across containers

2. Containers run container images, it bundles the application along with its runtime and dependencies.

3. Containers are lightweight, you can have many containers running at once on your system.

4. Containers startup time is on the order of Seconds, but it is Minutes for the VMs (Faster boot-time and creation-time).

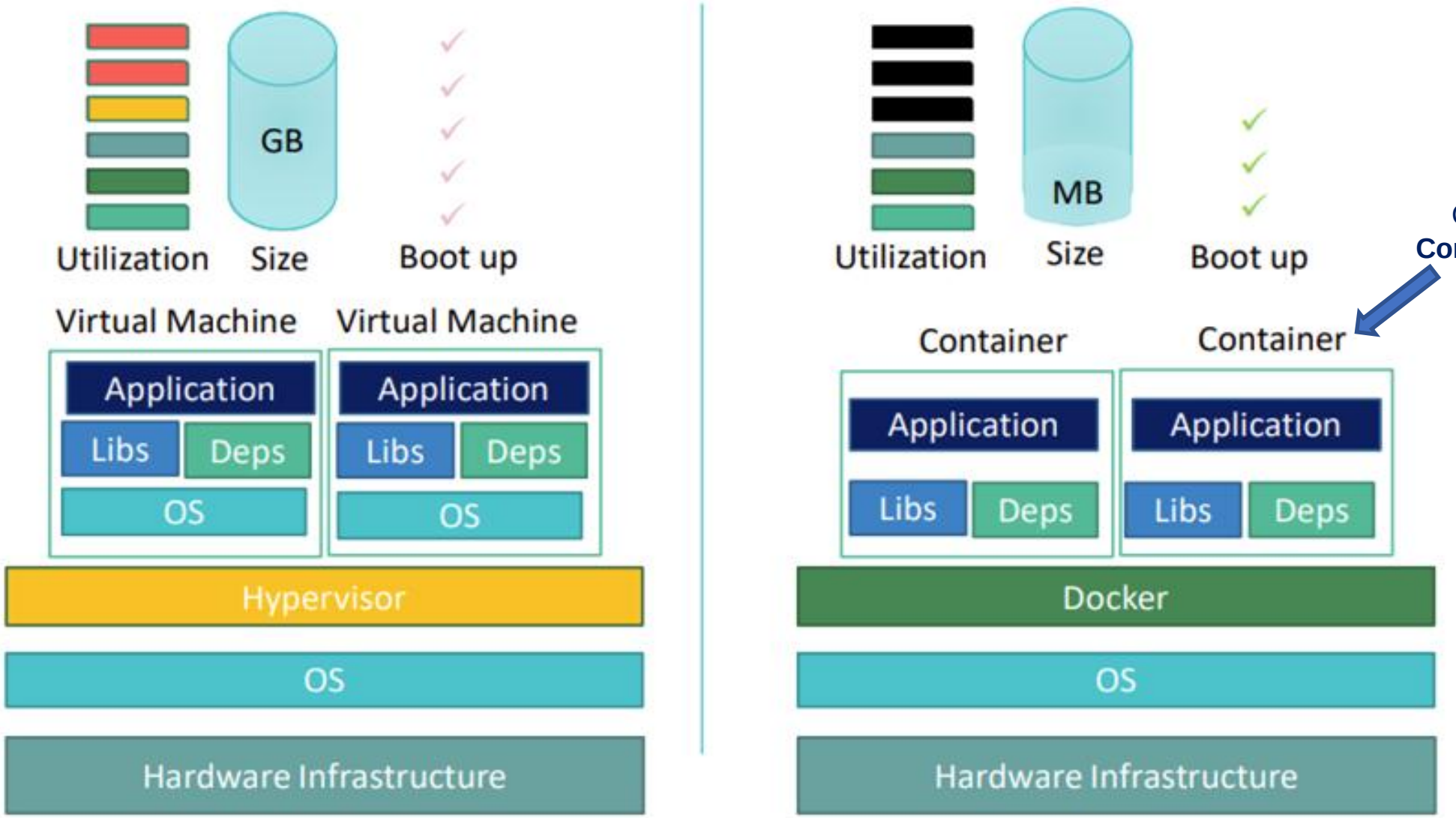


5. Containers are Application-centric methods to deliver high-performing, scalable app on any infrastructure of your choice. You can develop your code on your laptop and Run anywhere, without being worried for dependencies.

BCIT™

6. VMs provide strong isolation between processes sharing (better security) on the same Hardware/OS. Containers run on one Operating System, which has weaker isolation (less security), but lower overhead.

Containers vs Virtual Machines



Created from **Container images**

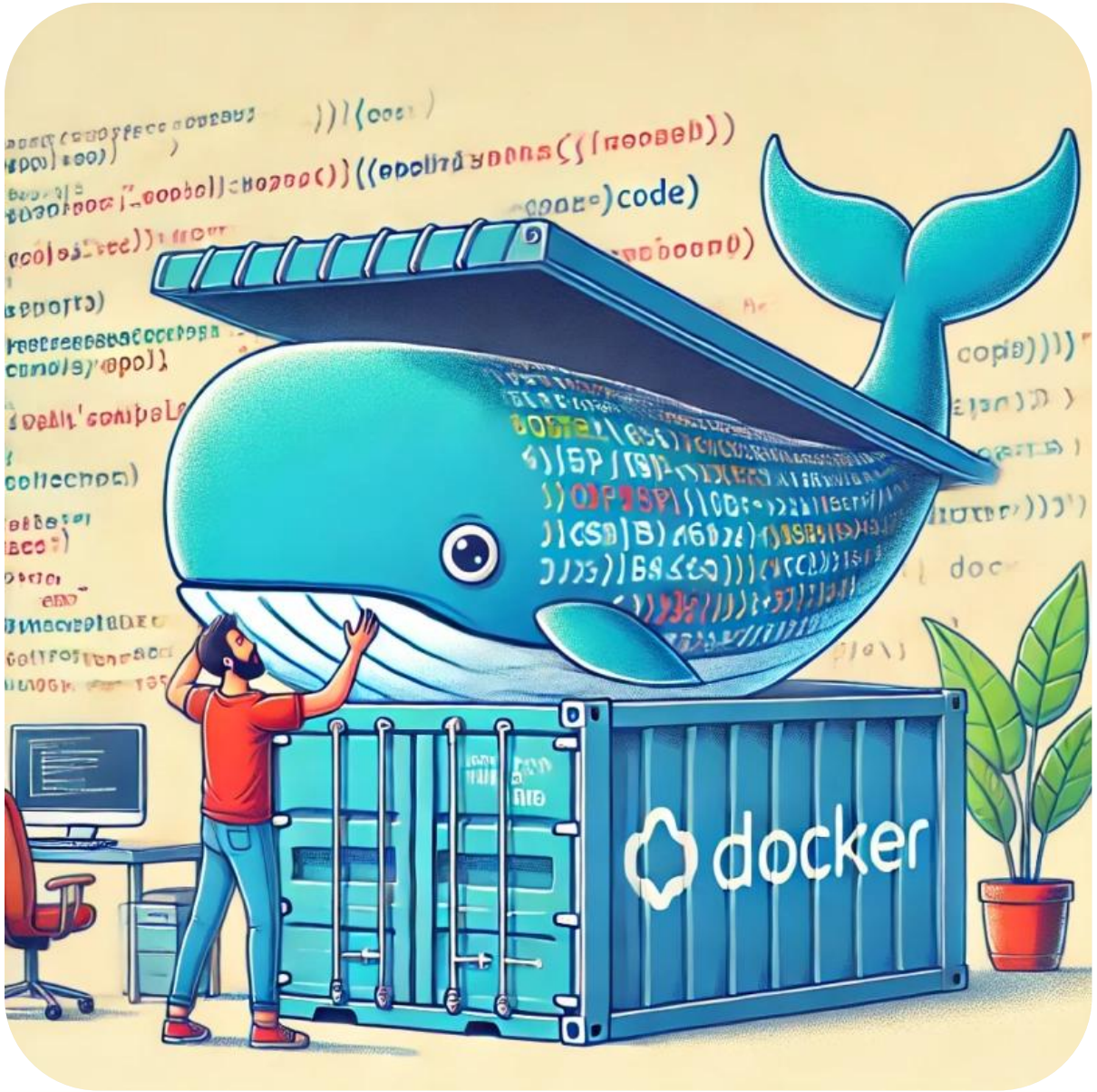
With the same container image, you can reproduce as many containers as you wish.
Think about the image as the recipe, and the container as the cake.
You can make as many cakes as you'd like with a given recipe.

BCIT™

What is Docker?

Docker is an open-source platform that simplifies Application Development by creating lightweight, portable Containers that package code and its dependencies.

Indeed, **Docker** ensures consistency across different platform & environments, enabling quick and reliable testing, deployment, and scaling.



BCIT™

Containers vs Virtual Machines

VMs

- Each VM runs its own OS
- Boot up time is in minutes
- Not version controlled
- Cannot run more than couple of VMs on an average laptop
- Only one VM can be started from one set of VMX and VMDK files

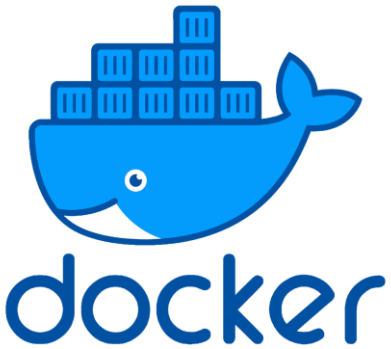
Docker Containers

- Container is just a user space of OS
- Containers instantiate in seconds
- Images are built incrementally on top of another like layers. Lots of images/snapshots
- Images can be diffed and can be version controlled. Docker hub is like Github
- Can run many Dockers in a laptop
- Multiple docker containers can be started from one Docker image

BCIT™

- ✓ The advantage of using containers is that they only virtualize the operating system and do not require dedicated piece of hardware because they share the same kernel of the hosting system.
- ✓ Containers give the impression of a separate OS, however, since they're sharing the kernel, they are much cheaper than a VM.

What is Docker and Why use it?



from DockerHub

Image file

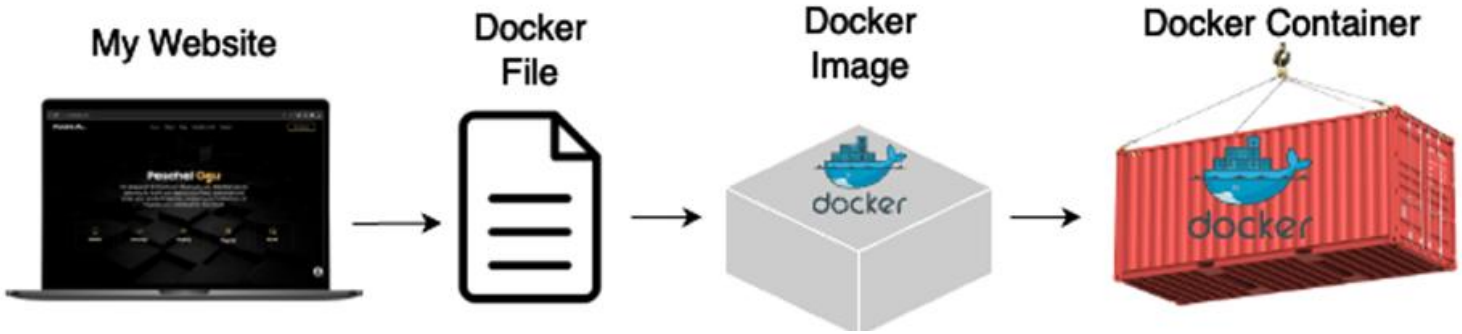
A single file with all the required libraries, package and configuration needed to run a program/app.

Container

Container

Container

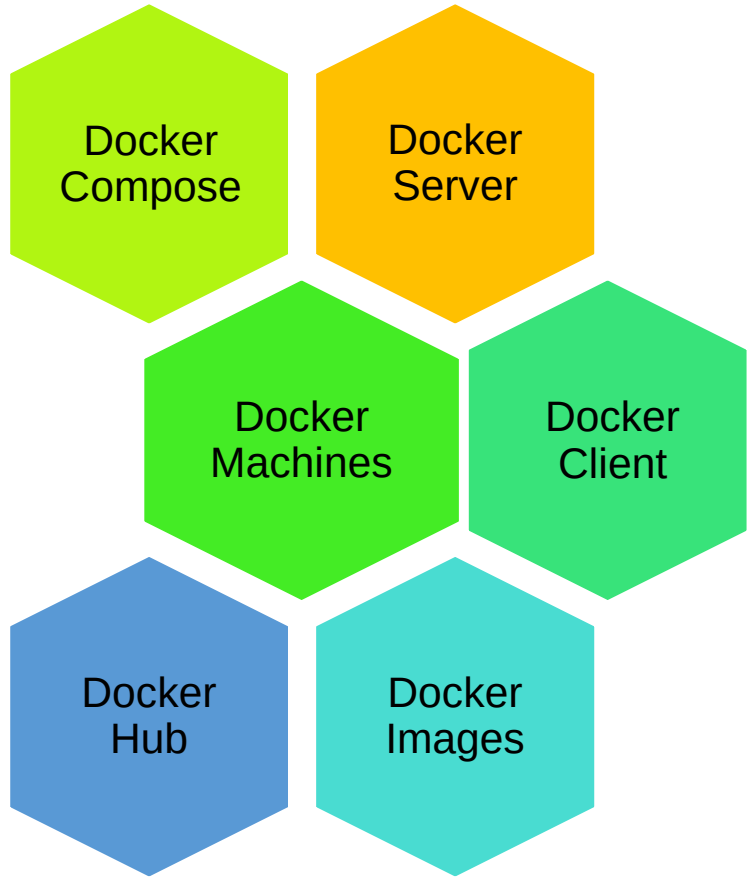
Instance of an image that runs a program (process)



Docker makes it very easy to install and run applications and programs without being worrying about software dependencies.

Docker Ecosystem

A platform for running Containers



What is Docker and Why use it?

define and manage multi-container applications.

Docker Compose

Docker Server (Daemon) ← In charge of creating images and running containers

Docker Runtime

Docker Client ← Command-line too to manage containers

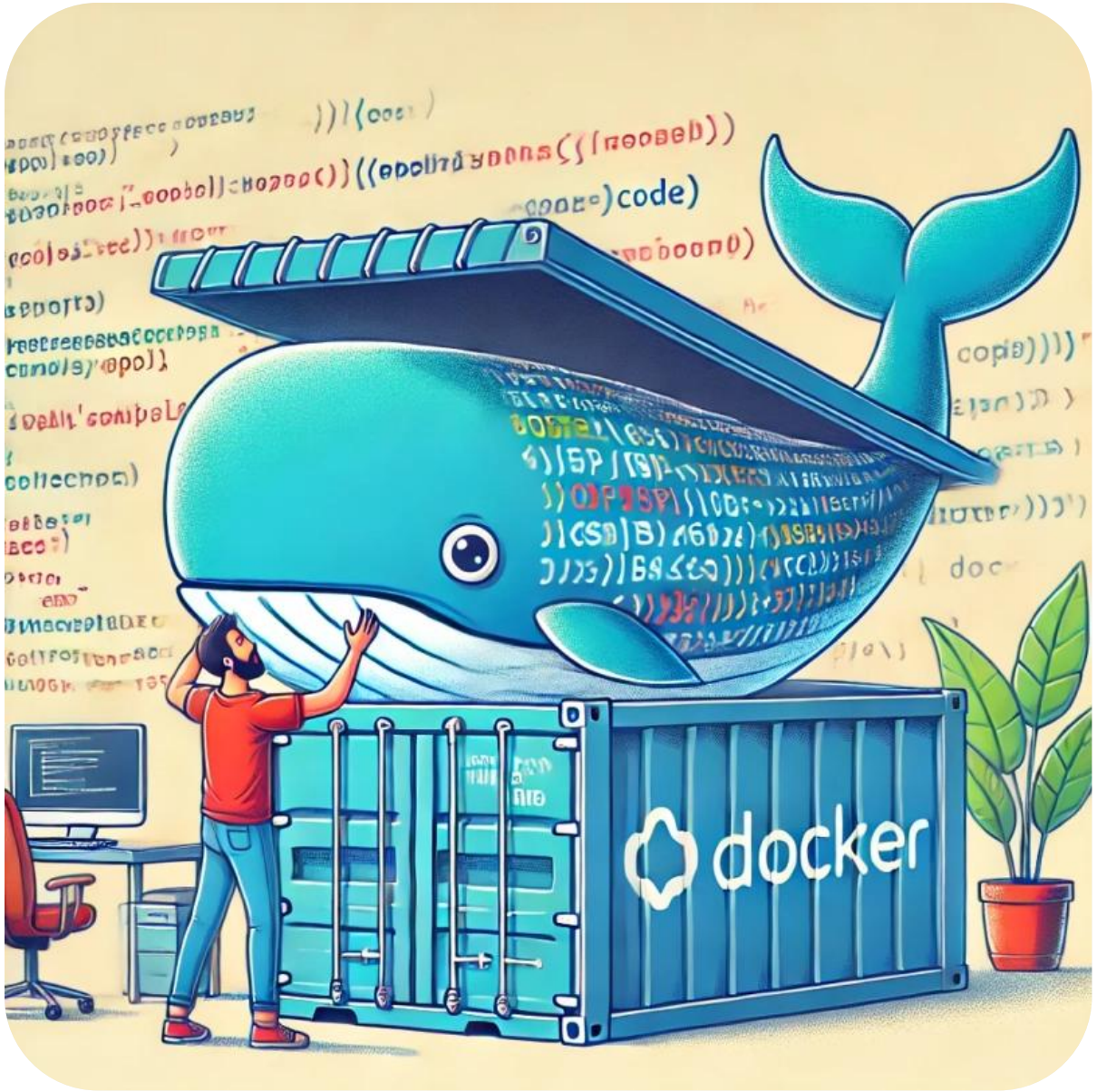
Docker Hub

Docker Images ← A template/blueprint to create a running Docker container. Image is like a **Recipe** Container is like a **Cake**.

responsible for executing containers, managing their lifecycle, and providing isolation between applications

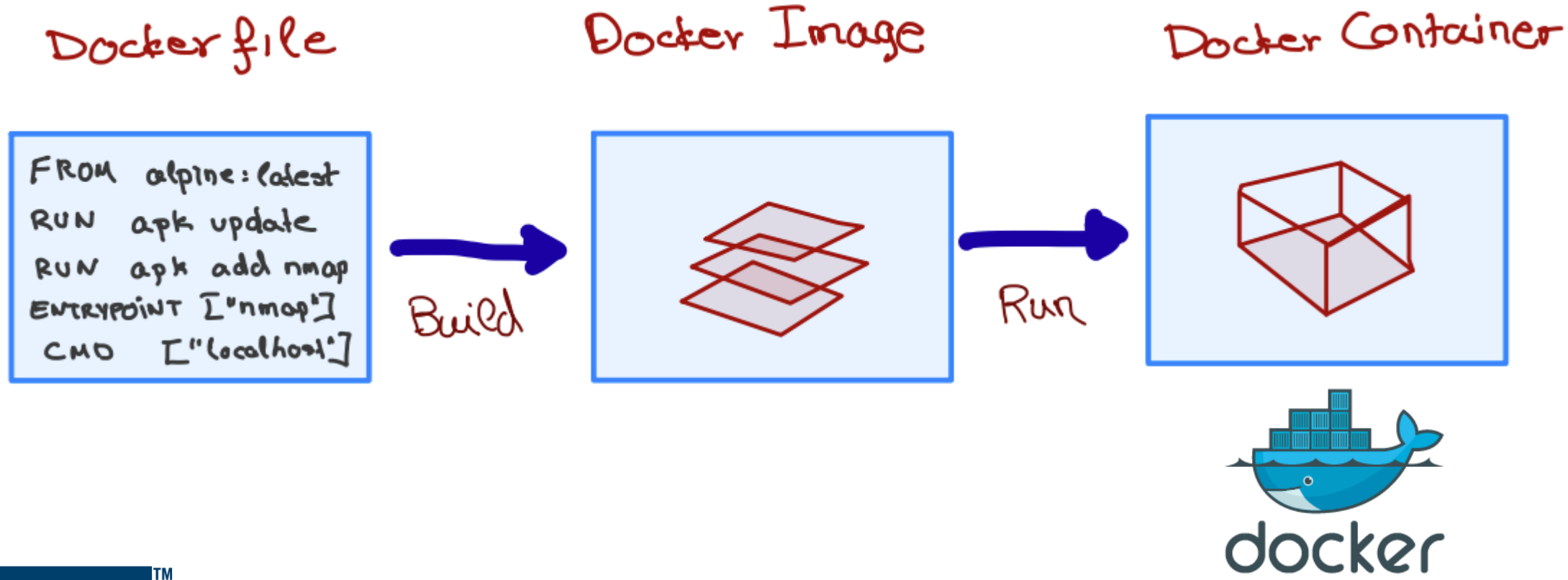
BCIT™

Containers: isolation without performance hits



A closer look into Docker

Dockerfile, as a simple text file allows us to build the Docker Image, which are .iso files and other files. We run the Docker Image to get Docker Container.



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

BCIT™

Docker Registry ➡ storage and distribution service for your images

Centralized Place to Find, Share, and Store Docker Images!

Docker Hub is a Docker Registry, like an App Store for Docker containers.

It's a cloud-based registry where you can find Docker Images as pre-configured software setups that you can use to run containers.

You can think of it as a place to download (pull) ready-made Docker images and also upload (push) your own images.

https://hub.docker.com/

Why Use Docker Hub?

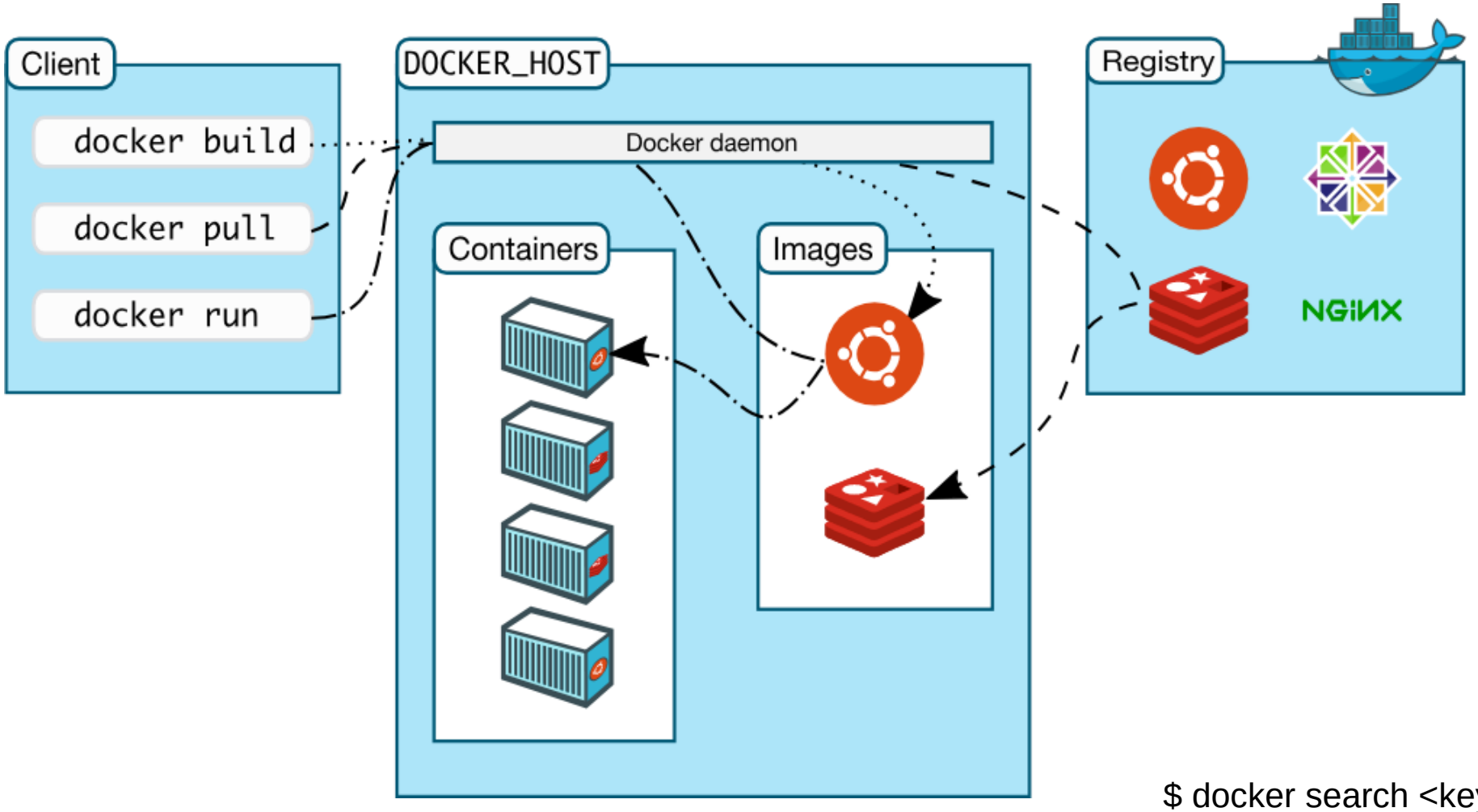
✓ **Ready-made images**: Find thousands of pre-built images, including popular ones like Nginx, MySQL, Ubuntu, and more.

✓ **Easy sharing**: Share your own Docker images with others or access images shared by the community.

✓ **Public and private repositories**: You can store your Docker images either publicly (for free) or privately (for a fee).

BCIT™

Docker Architecture

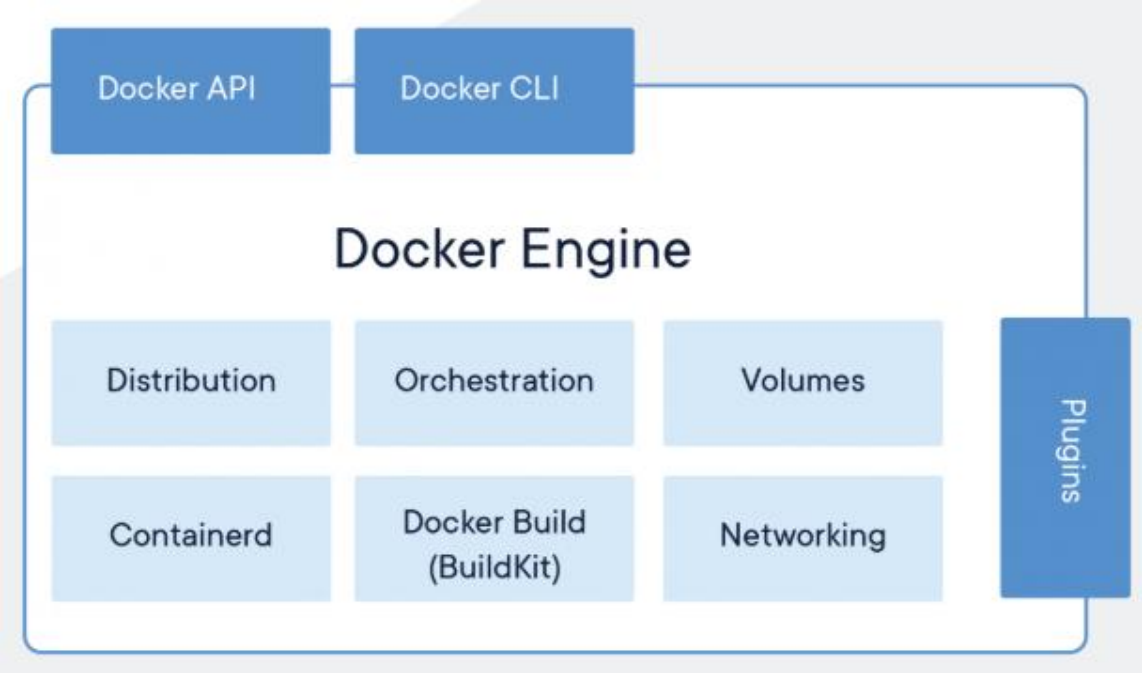


$ docker search <keyword>
$ docker pull <image_name>
$ docker push <image_name>

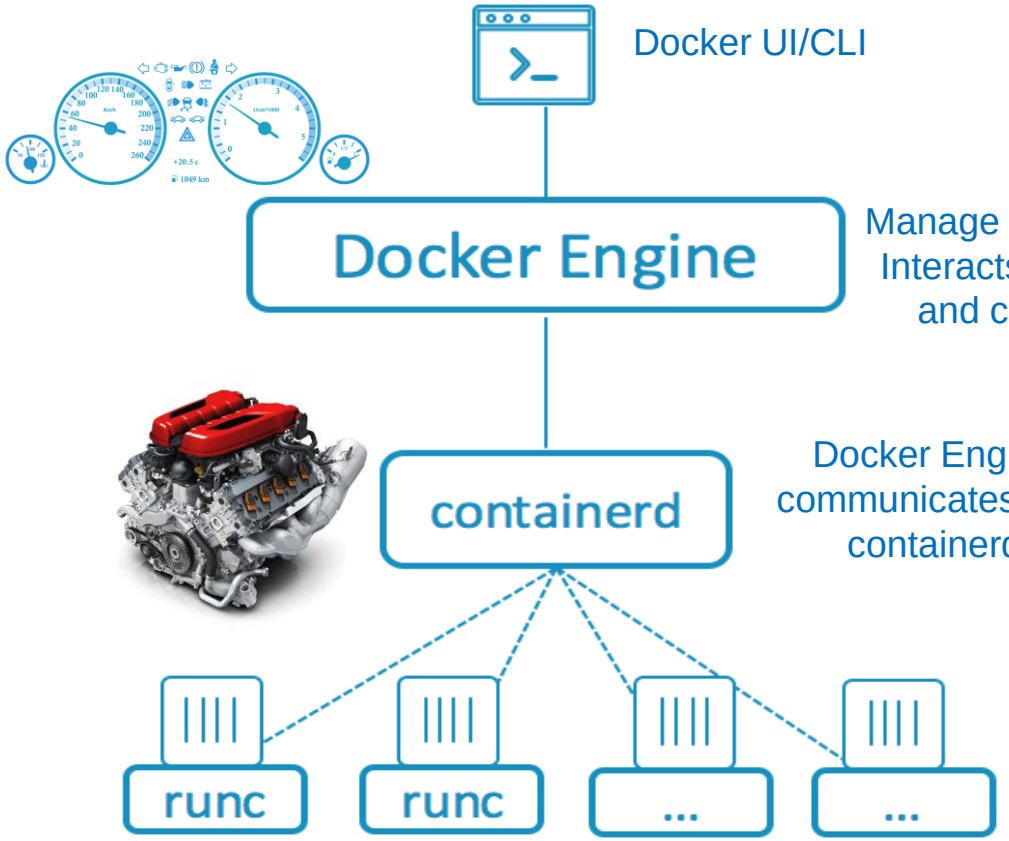
BCIT™

Container Engine vs Container Runtime

Driver's dashboard vs Engine inside the car



https://www.docker.com/products/container-runtime/



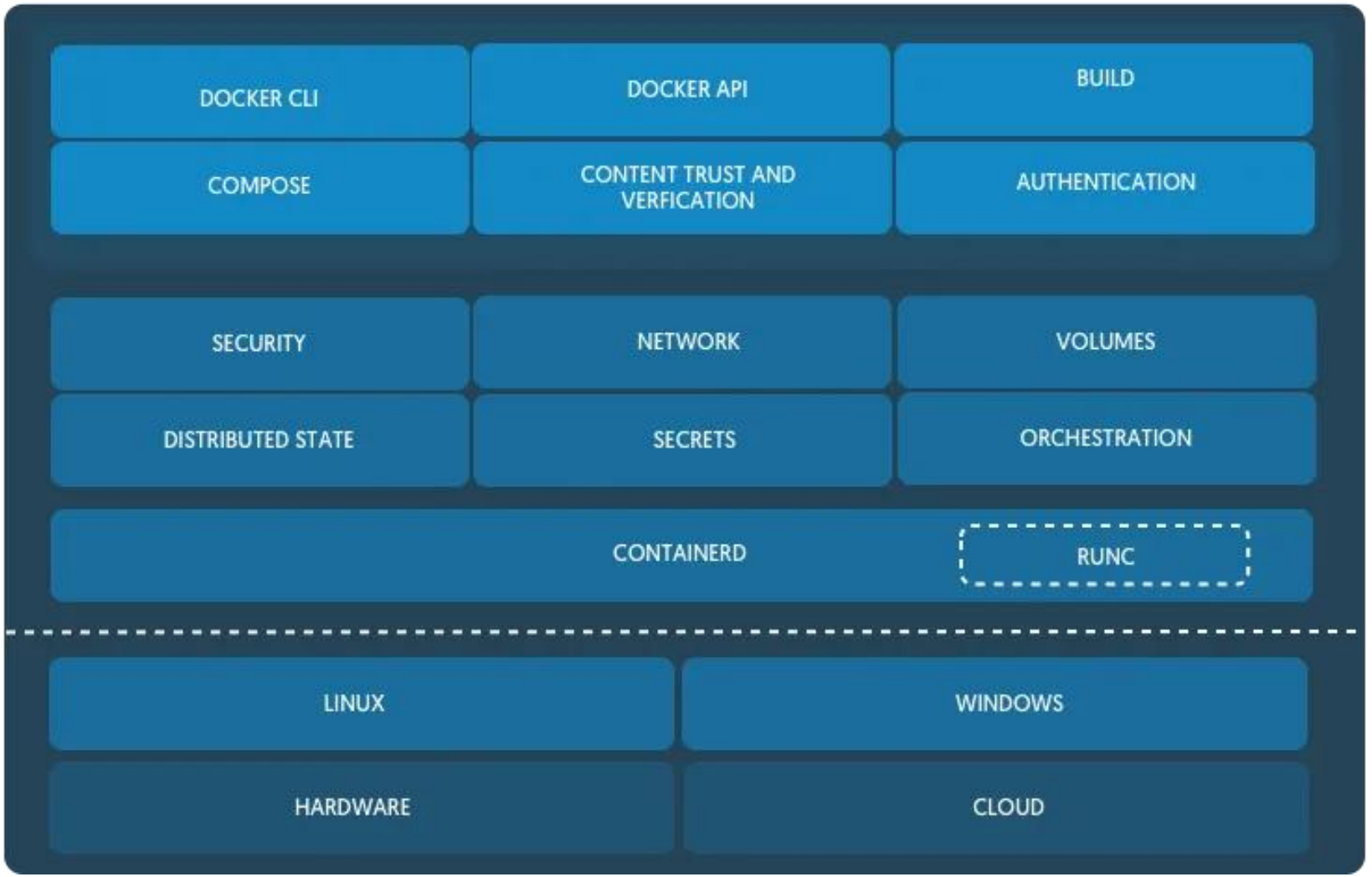
Containerd spins up runc to run containers

BCIT™

Docker itself is a **container engine**, and relies on **container runtimes** under the hood to actually run the containers. In modern Docker setups, **Containerd** is the default runtime for managing the lifecycle of containers.

containerd → Handles container lifecycle.
runc → Low-level runtime to executes containers

Container Engine vs Container Runtime



BCIT™

Container Networking

```
ubuntu@maas:~$ docker network list
NETWORK ID          NAME                DRIVER              SCOPE
8af9761e54f3        bridge              bridge              local
b28ea5ab384d        host                host                local
8e6e052ffe98        none                null                local

ubuntu@maas:~$ docker network inspect bridge | grep bridge.name
            "com.docker.network.bridge.name": "docker0",

ubuntu@maas:~$ brctl show
bridge name         bridge id               STP enabled         interfaces
docker0             8000.0242673cc920       no                      veth0ff5048
virbr0              8000.5254004a6f48       yes                 virbr0-nic
```

BCIT™

Container Networking

```
ubuntu@maas:~$ sudo iptables-save | grep docker
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
-A DOCKER -i docker0 -j RETURN
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
```

```
ubuntu@maas:~$ sysctl net.ipv4.conf.all.forwarding
net.ipv4.conf.all.forwarding = 1
```

BCIT™

Docker Commands

Running a Simple Nginx Web Server Using Docker


```
$ docker --version
$ docker search nginx
$ docker pull nginx
$ docker image ls

$ docker run --name nginx-web-server –d –p 8000:80 nginx
$ docker ps
$ docker ps -s
$ docker logs nginx-web-server
$ docker exec -it nginx-web-server /bin/sh

$ docker stop nginx-web-server
$ docker start nginx-web-server

$ docker rm nginx-web-server
$ docker rmi nginx
```

Search for an image name

Pull an image from Docker Hub

List images pulled to your laptop

Run a container from an image

List running containers

Displays approximate size of containers

Check container logs

Access a shell inside a sunning container

Stop a running container

Start a stopped container

Remove a container

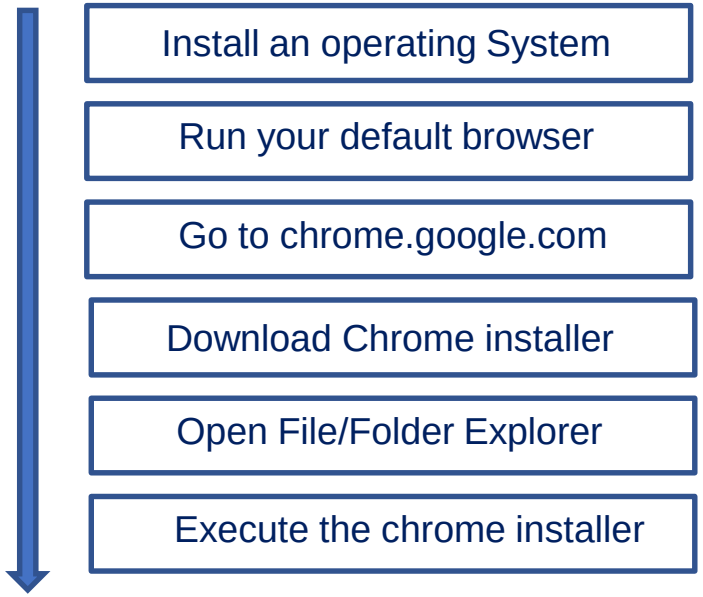
Remove an image

BCIT™

A closer look into Docker

Writing a Docker file is like being given a computer with no OS and being told to install Chrome or other software on it.

How to Install Google Chrome



- ✓ A Docker image is a combination of a filesystem and parameters.
- ✓ Layers built on layers, and Merged together at run time.
- ✓ Each Layer is stored only once no matter how many containers use it.

FROM	**alpine**
RUN	**apk add --update nginx**
CMD	**[“nginx”]**

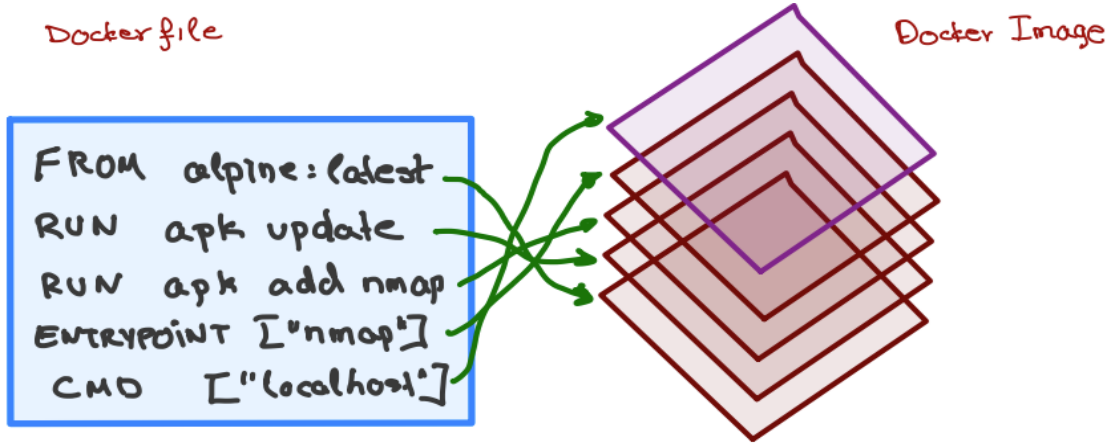
BCIT™

A closer look into Docker

```
# Use Alpine Linux as the base image
FROM alpine:latest

# Set the entry point to echo the message
ENTRYPOINT ["/bin/echo"]

# Set the default argument for the echo command
CMD ["Hello, World!"]
```

```
> docker build -t hello_world_cmd:first -f
Dockerfile_cmd .

> docker run -it hello_world_cmd:first
> Hello world
> docker run -it hello_world_cmd:first Pavlos
> Hello Pavlos
```

- **ENTRYPOINT**
  - sets the command and parameters that will first execute when container is started
  - will override all elements specified with CMD
  - commands or scripts can be passed
- **CMD**
  - executed after ENTRYPOINT
  - can pass arguments to an ENTRYPOINT main command
  - with CMD, commands can be overridden



The daemon reads the Dockerfile and creates a layer for every command.

BCIT™

A closer look into Docker

How to build a customized Docker Image?

$ mkdir nginx-docker
$ cd nginx-docker
$ vim index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Welcome to My Nginx Docker</title>
</head>
<body>
    <h1>Welcome to My Nginx served by a Customized Docker!</h1>
    <p>This page is being served by an Nginx container!</p>
</body>
</html>
```

$ vim Dockerfile

```
# Use the official Nginx image from the Docker Hub
FROM nginx:alpine

# Copy the HTML file into the default Nginx web directory
COPY index.html /usr/share/nginx/html/index.html

# Expose the port that Nginx will run on
EXPOSE 80

# No need to specify CMD or ENTRYPOINT because the official
# Nginx image already comes with the default command to start Nginx.
```

$ docker build -t nginx-docker-app .
$ docker images
$ docker run –d –p 8000:80 nginx-docker-app

→ Check http://localhost:8000 in your browser

BCIT™

A closer look into Docker

How to build a customized Docker Image?

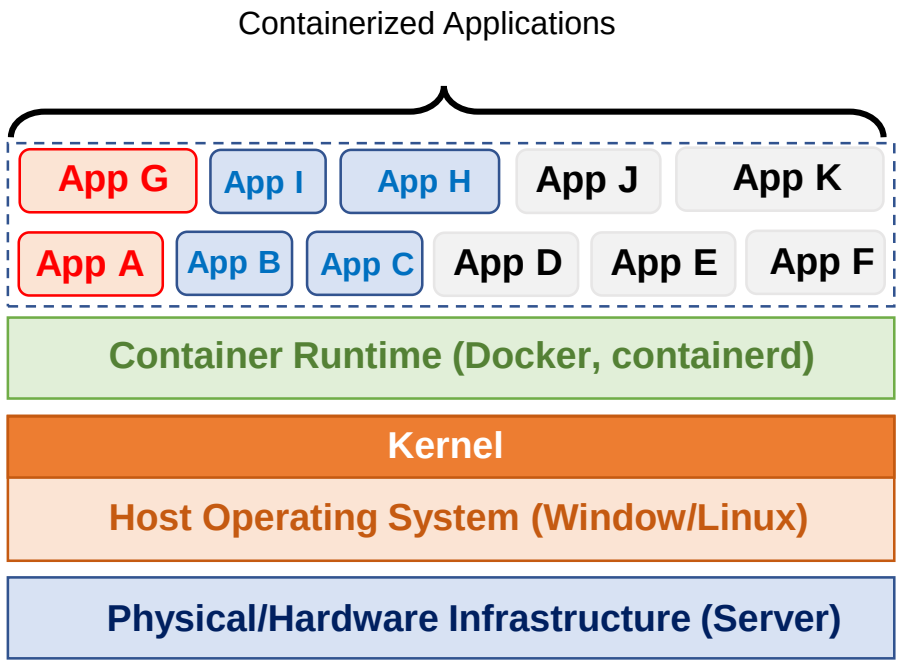
$ vim Dockerfile

More Complex Example

```
FROM ubuntu
LABEL maintainer="docker@example.com”

RUN \
        apt-get update && \
        apt-get install -y nginx && \
        rm -rf /var/lib/apt/lists/* && \
        echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
        chown -R www-data:www-data /var/lib/nginx

COPY index.html /usr/share/nginx/html/index.html
CMD ["nginx"]

EXPOSE 80
EXPOSE 443
```

BCIT™

One more thing about Containers …



Containers are Ephemeral !?!

- ✓ Short-lived by Nature
- ✓ Immutable Behavior
- ✓ Stateless Design
- ✓ Disposable

How to manage Ephemerality?

Despite being ephemeral, containers can work with **Persistent Storage** to handle stateful applications. This is achieved by:

- **Volumes**: Mounting external storage to the container.
- **Databases**: Storing critical data in external databases instead of within the container file-system.
- **Configuration Management**: Using environment variables, secrets, or config maps to manage settings dynamically.

BCIT™

This Ephemeral nature makes containers highly **scalable**, **portable**, and **easy to manage.**

Lecture #2 Summary

1. Virtualization technologies/types
2. Virtual Machine (VM) advantages
3. Containers and Dockers (vs. VMs)
4. Docker as de facto for containerization
5. Understanding Docker components and architecture
6. Creating a simple Docker image

BCIT™

Hands on Docker/Containers Exercise

Review of LAB #1

Exercise 1: Pull, Modify, Push Image.

Exercise 2: Build a Docker Image.

Exercise 3: Try something from NetworkChuck

BCIT™

End of Lecture #2

THANK Y☺U

- Dawood Sajjadi

BCIT™