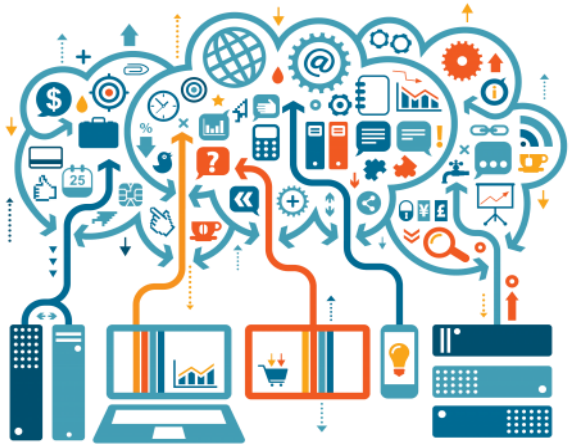
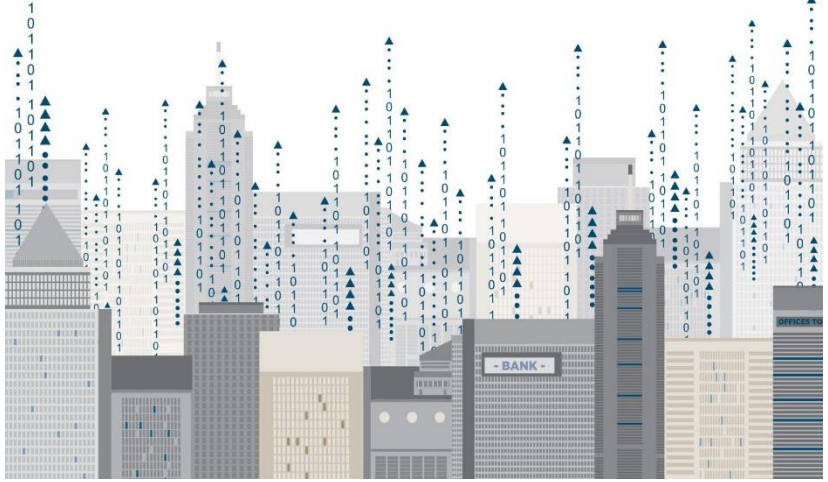




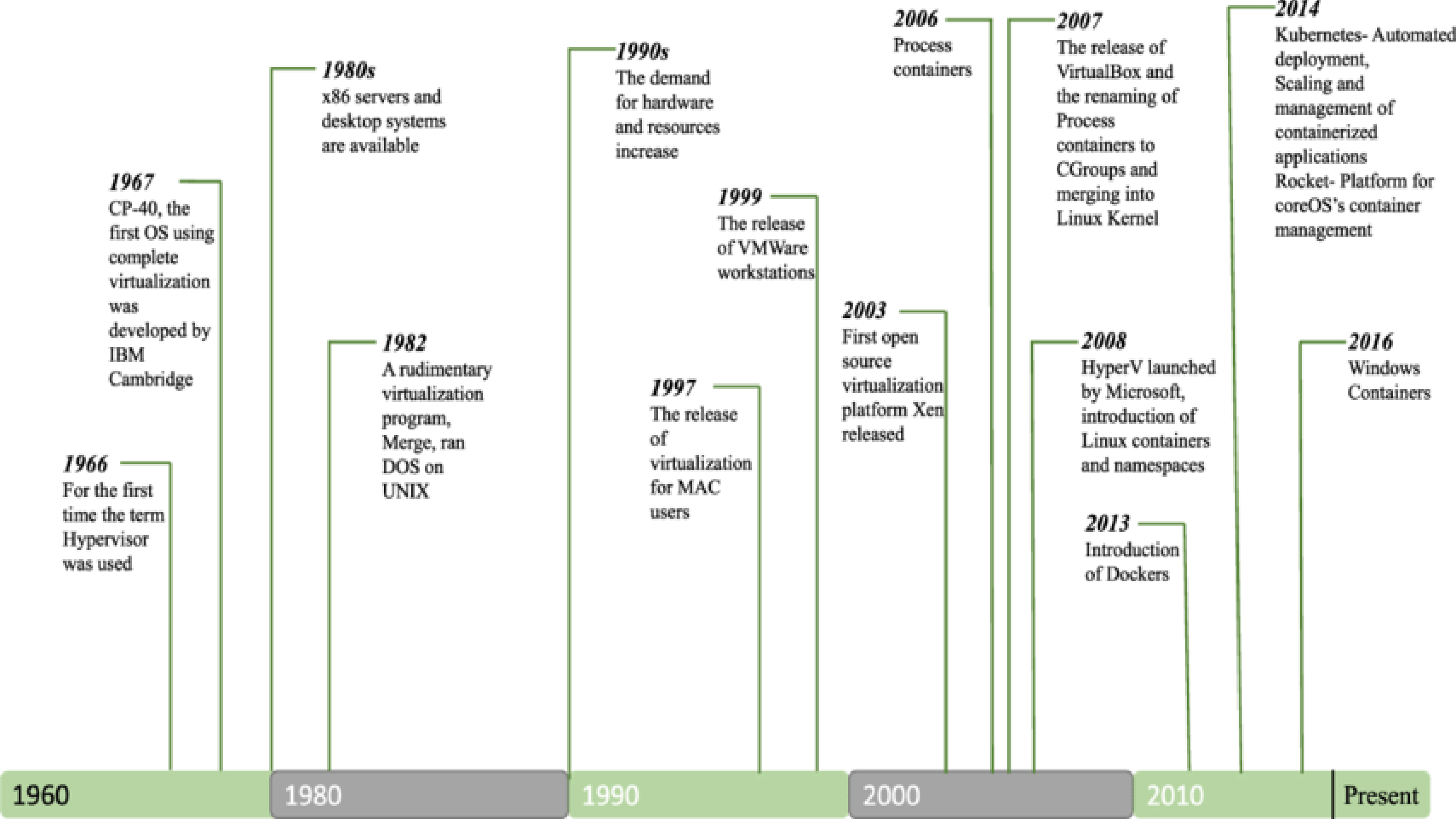
Big Data Analytics

Fall 2025

Lecture 3

Virtualization

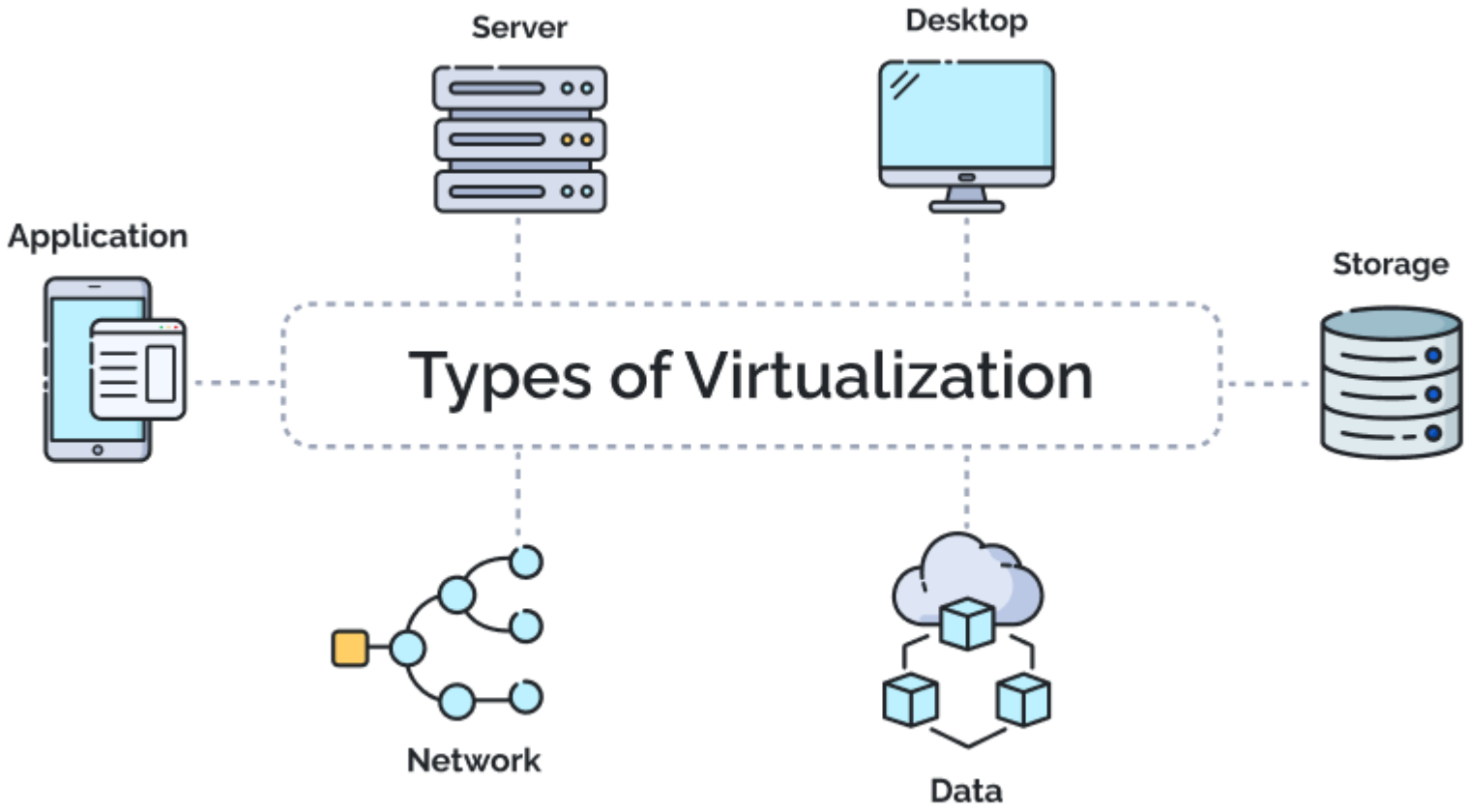
- Start 1960s:
- Logically divide the system resources provided by mainframes between different applications
- The definition has now changed.


1966
For the first time the term Hypervisor was used
1967
CP-40, the first OS using complete virtualization was developed by IBM Cambridge
1980s
x86 servers and desktop systems are available
1982
A rudimentary virtualization program, Merge, ran DOS on UNIX
1990s
The demand for hardware and resources increase
1997
The release of virtualization for MAC users
1999
The release of VMWare workstations
2003
First open source virtualization platform Xen released
2006
Process containers
2007
The release of VirtualBox and the renaming of Process containers to CGroups and merging into Linux Kernel
2008
HyperV launched by Microsoft, introduction of Linux containers and namespaces
2013
Introduction of Dockers
2014
Kubernetes- Automated deployment, Scaling and management of containerized applications
Rocket- Platform for coreOS's container management
2016
Windows Containers
1960
1980
1990
2000
2010
Present

Hardware Virtualization

- Creation of a virtual machine: acts like a real computer with an OS
- Software executed on VMs: separated from the underlying hardware
- A Windows computer may host a VM that looks like a computer with the Ubuntu Linux; Ubuntu-based software can be run on this VM.
- “Host machine”: Software runs on physical machine
- “Guest machine”: software runs on the VM
- The software or firmware that creates a virtual machine on the host hardware is called a “hypervisor” or virtual machine monitor.
- Virtualization: importing a guest OS on the host OS, allowing developers to run multiple OS on different VMs while all of them run on the same host - eliminates the need to provide extra hardware resources.

Types of Virtualization



Application Vir

App virtualization is a technique where an application runs in an **isolated virtual environment**, instead of being directly installed on the underlying operating system (OS).

The app “thinks” it’s running on the host OS normally, but in reality, it’s being redirected through a virtualization layer that manages access to files, registry, libraries, etc.

Benefits

- **No conflicts**: Different versions of the same app can coexist (no DLL Hell).
- **Easy deployment**: Just stream or drop the app package to users.
- **Security**: App changes don’t affect the host OS.
- **Portability**: Run legacy apps on newer systems without compatibility issues.

1.Isolation Layer

1. A virtualization layer intercepts an app’s calls to the OS (like file, registry, or API calls).
2. Instead of touching the host OS directly, those calls are redirected to a **virtualized container**.

2.No Full Installation Needed

1. The app doesn’t need to be installed in the traditional way.
2. Its binaries, DLLs, and configuration files live in a packaged container.

3.Controlled Integration

1. The app can still interact with the host OS (e.g., display UI, use network).
2. But its files/registry changes don’t pollute the host system.

Server Vir

Server virtualization is the process of dividing one **physical server** into multiple **virtual servers (VMs)** using a **hypervisor**.

Each VM behaves like a complete independent machine with its own:

- **Operating System (guest OS)**
- **CPU** (vCPUs mapped to host cores)
- **RAM** (allocated from host memory)
- **Disk & Network** (virtualized, mapped to physical resources)

So instead of dedicating one physical server per workload, you run many workloads on **one physical server**.

1.Hypervisor Layer

1. The hypervisor sits between the hardware and the VMs.
2. It controls CPU scheduling, memory allocation, I/O access, and device virtualization.

2.VM Creation

1. You can spin up multiple VMs, each with its own OS (Windows, Linux, etc.).
2. They share the **same physical hardware**, but don't interfere with each other.

3.Resource Management

1. Hypervisor allocates CPU cycles, RAM, disk, and NICs dynamically.
2. It can also overcommit resources (e.g., 16 VMs each with 2 vCPUs on an 8-core host).

Benefits of Server Virtualization

- **Better Hardware Utilization** – Instead of one workload per server, run many.
- **Isolation** – Problems in one VM don't crash others.
- **Flexibility** – Run Linux and Windows on the same physical server.
- **Easier Backups & Recovery** – VMs can be snapshotted, migrated, cloned.
- **Scalability** – Quickly deploy new servers as VMs without buying new hardware.

Imagine you have a **physical server with 32 cores and 128 GB RAM**.

- Without virtualization → You install Linux and run a single app.
- With virtualization → You can create:
  - 10 VMs (each 4 cores, 12 GB RAM) for web servers
  - 5 VMs (each 2 cores, 4 GB RAM) for databases
  - 2 VMs (each 8 cores, 32 GB RAM) for analytics

All on the same hardware, isolated from each other.

Desktop Vir

Desktop virtualization is a technology where a user's **desktop environment** (the whole OS interface, apps, files, and settings) is separated from the physical computer and delivered **remotely** or from a **virtual machine**.

Instead of running directly on your laptop/PC, your desktop runs in a **virtualized environment** (usually on a data center server or in the cloud).

- A **virtual machine (VM)** is created on a server.
- This VM runs a **desktop OS** (e.g., Windows 10/11, Linux).
- Users connect to that desktop over the network using a **client device** (thin client, laptop, tablet, etc.).
- The user sees the desktop as if it's local, but all the computation and storage actually happen on the server.

Benefits

- **Anywhere Access** – Users can log in to their desktop from any device.
- **Security** – Data stays in the data center/cloud, not on the endpoint.
- **Centralized Management** – IT can patch, update, and control desktops in one place.
- **Cost-Efficiency** – Use thin clients instead of powerful PCs.
- **Disaster Recovery** – Easy to restore desktops after a crash.

Real-World Example

- A university gives students **virtual Windows desktops** hosted on servers.
- Students connect using their laptops, Chromebooks, or tablets.
- All apps (MS Office, MATLAB, etc.) run in the data center, not on the student's device.

Storage Vir

Storage virtualization is the process of pooling multiple physical storage devices (like HDDs, SSDs, SAN/NAS systems) into a **single logical storage resource** that applications and users can access as if it were one big disk.

You may have dozens of physical disks across servers, but storage virtualization makes them look like **one unified storage system**.

1.Abstraction Layer

1. A virtualization layer sits between physical storage (the hardware) and the host systems.
2. This layer maps **logical storage** to **physical storage**.

2.Unified View

1. To the operating system or application, it looks like one big pool of storage.
2. Behind the scenes, the hypervisor/storage manager spreads the data across multiple devices.

3.Dynamic Management

1. Capacity, performance, and redundancy can be managed centrally.
2. Data can be moved between devices without disrupting applications.

- **Simplified Management** – Manage many storage devices from a single console.
- **Scalability** – Add new disks or arrays without reconfiguring apps.
- **High Availability** – Data can be mirrored across devices; failures are hidden from users.
- **Better Utilization** – Storage resources are pooled and allocated as needed.
- **Flexibility** – Move workloads without worrying about physical disk location.

Real-World Example

- A data center has 50 physical disks across 10 servers.
- Without virtualization → Each server uses its own disks separately.
- With storage virtualization → All disks are combined into a **single storage pool**, and virtual volumes are allocated to VMs, databases, or apps as needed.

Network Vir

Network virtualization is the process of combining hardware (switches, routers, NICs, firewalls) and software network resources into a **single, software-defined virtual network**.

It makes the **network behave like software**, independent of the physical cabling and devices.

Just like server virtualization abstracts CPUs/RAM and storage virtualization abstracts disks,
network virtualization abstracts physical networking into logical, flexible networks.

- **Abstraction Layer**
- A virtualization layer sits on top of the physical network.
- It creates **virtual networks** (vLANs, VXLANs, overlays, tunnels) on top of physical infrastructure.

- **Virtual Network Functions (VNFs)**
- Traditional hardware functions (firewall, load balancer, router) become **software appliances**.
- Example: Instead of a physical firewall box, you spin up a virtual firewall VM.

- **Software-Defined Networking (SDN)**
- Centralized controllers manage traffic dynamically.
- The physical network just provides connectivity; the **logic lives in software**.

Data Vir

Data virtualization is a technology that lets you access and use data from multiple different sources **as if it were a single database**, **without physically moving or copying the data**.

Instead of ETL (extract–transform–load) pipelines that replicate data into one warehouse, data virtualization provides a **real-time, unified view** of all your data.

1. **Connectors** link to various sources (databases, files, APIs, cloud apps, data lakes).

2. A **virtual data layer** maps those sources into a single logical schema.

3. When a user/application queries the data:
    1. The virtualization engine translates the query.
    2. It fetches the required data from the sources in real time.
    3. It presents results as if they came from **one unified system**.

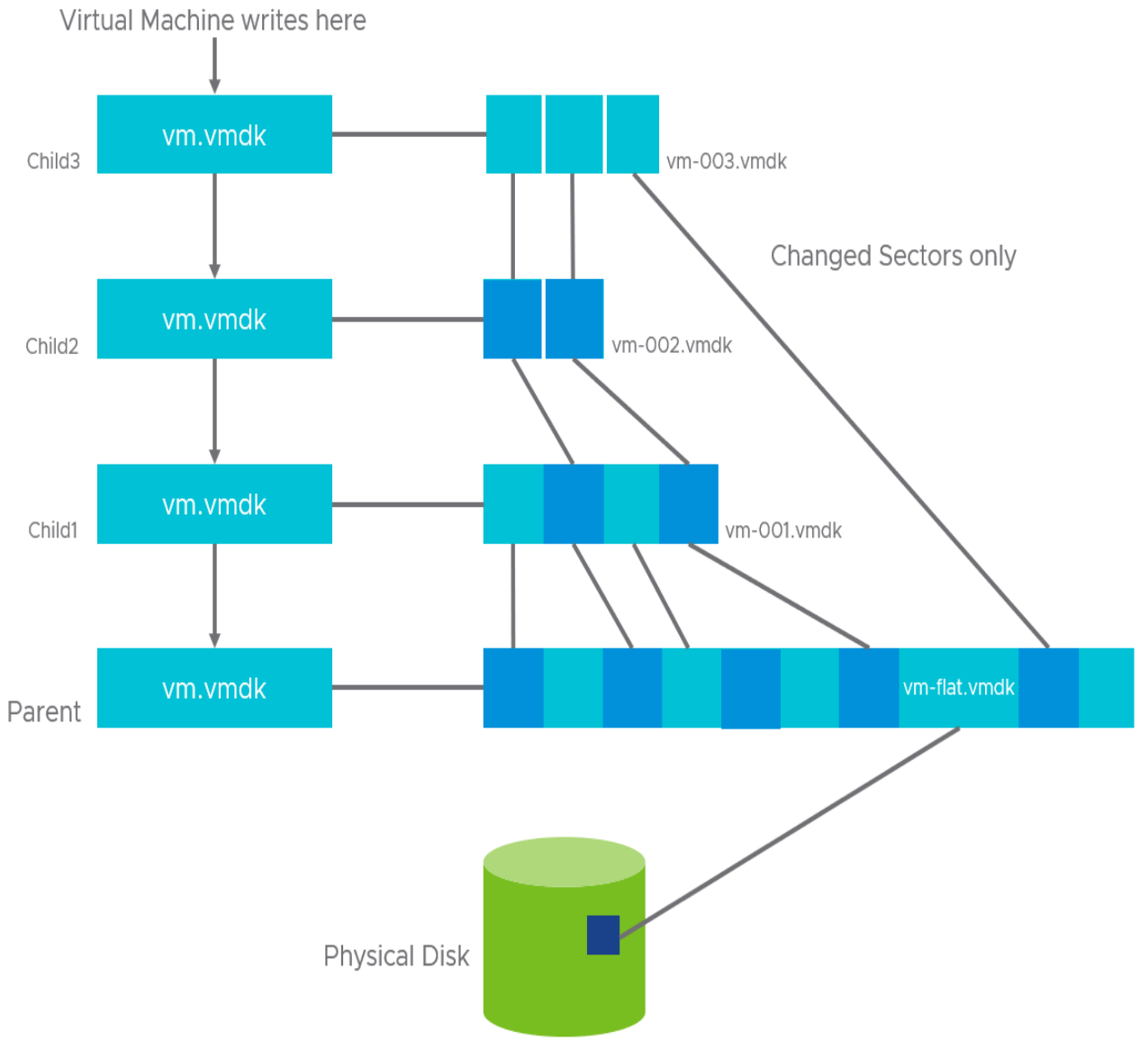
No need to copy or duplicate — it's all **on-demand integration**.

- A bank wants a **360° view of the customer**:
  - Transactions in Oracle DB
  - Credit reports from an external API
  - Support tickets in Salesforce
  - Data virtualization presents all of it as **one virtual customer profile**, without moving data.

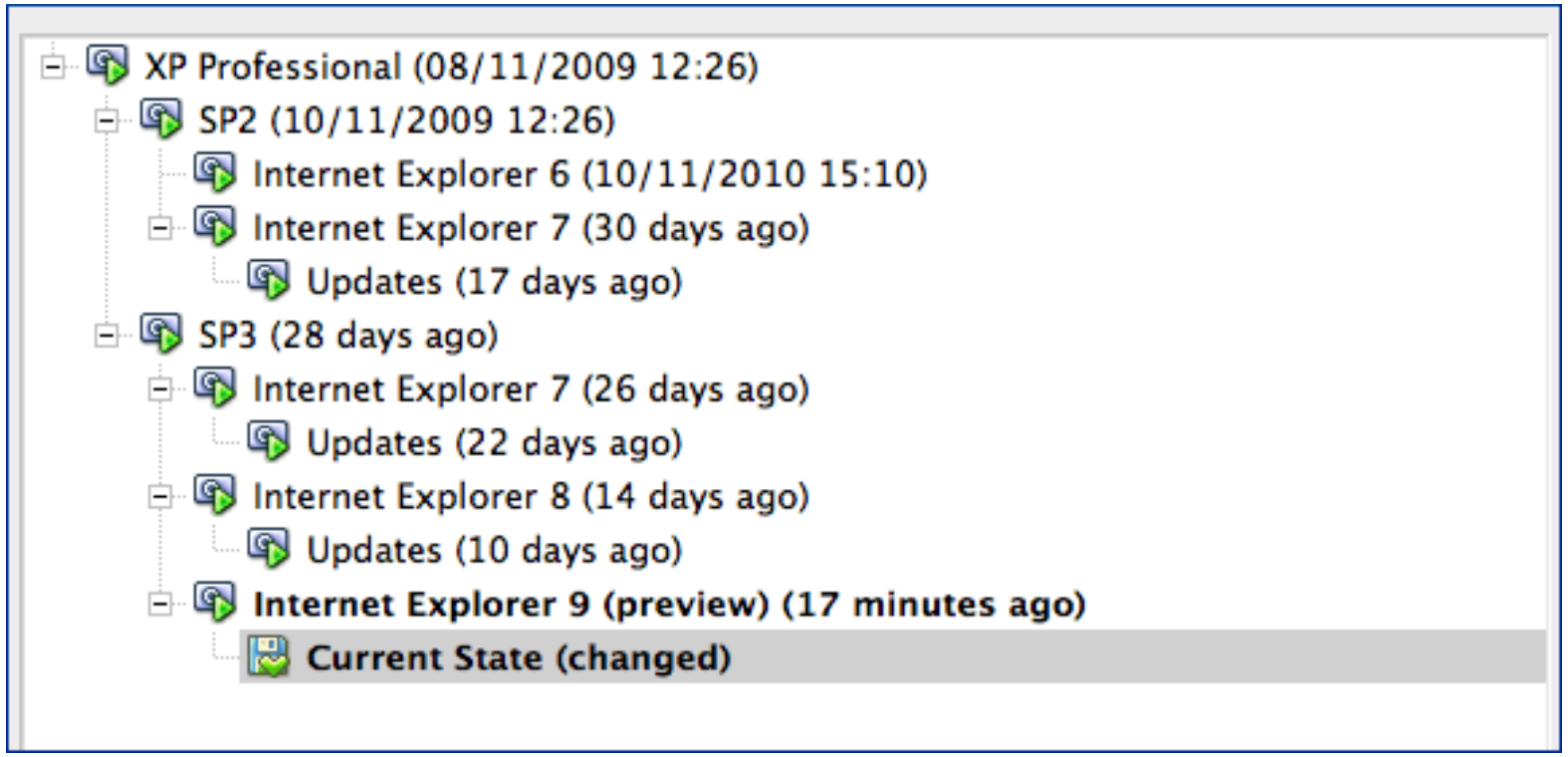
In analytics: Instead of building one huge data warehouse, analysts can query across **Snowflake, BigQuery, and on-prem SQL Server** via one virtual layer.

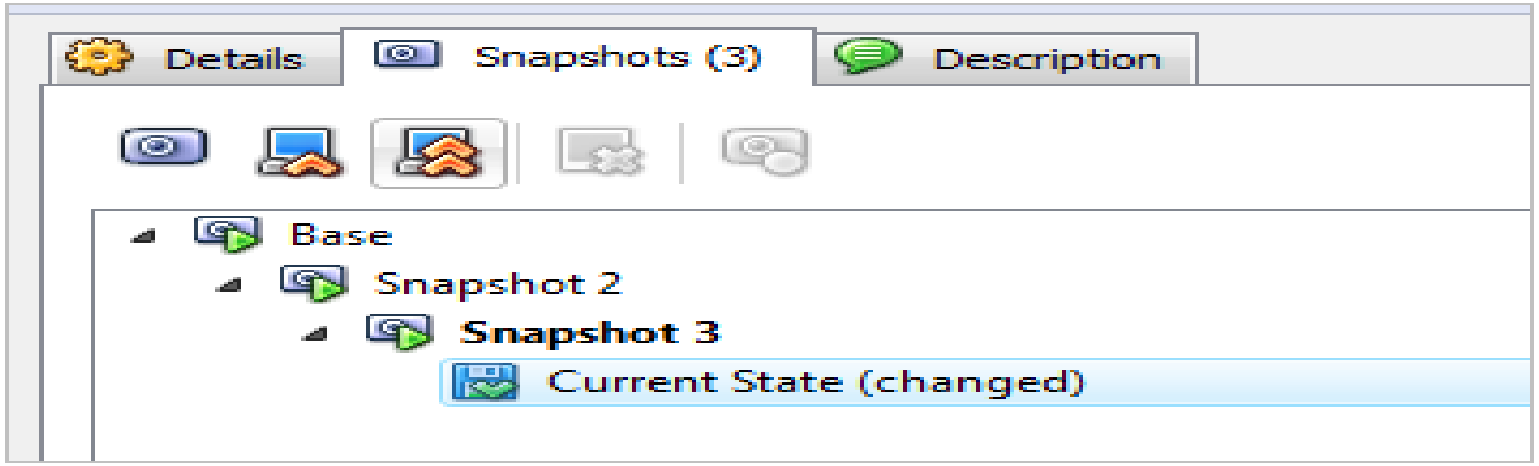
Snapshot

- A state of a virtual machine, and generally its storage devices, at an exact point in time.
- Enables the VMs state at the time of the snapshot to be restored later, effectively undoing any changes that occurred afterwards.
- Useful as a backup technique, for example, prior to performing a risky operation.



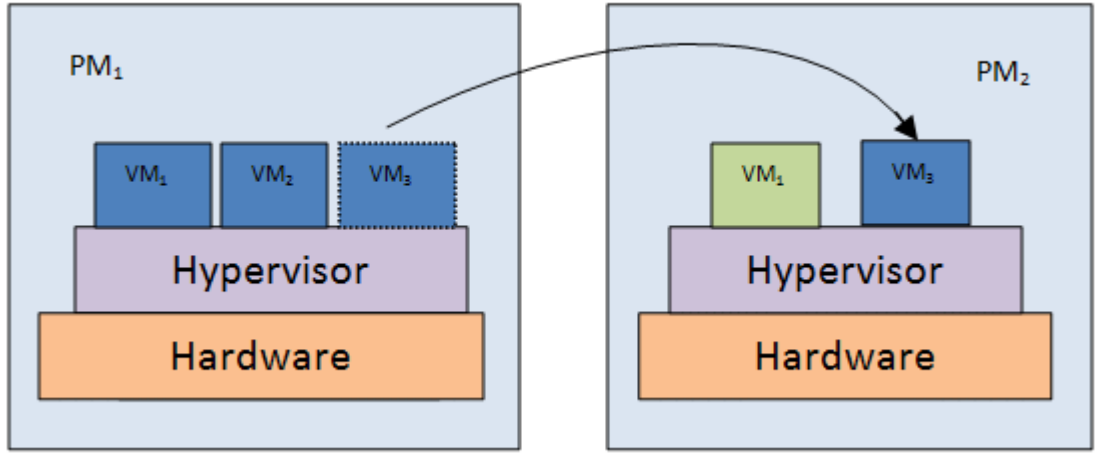
Snapshot

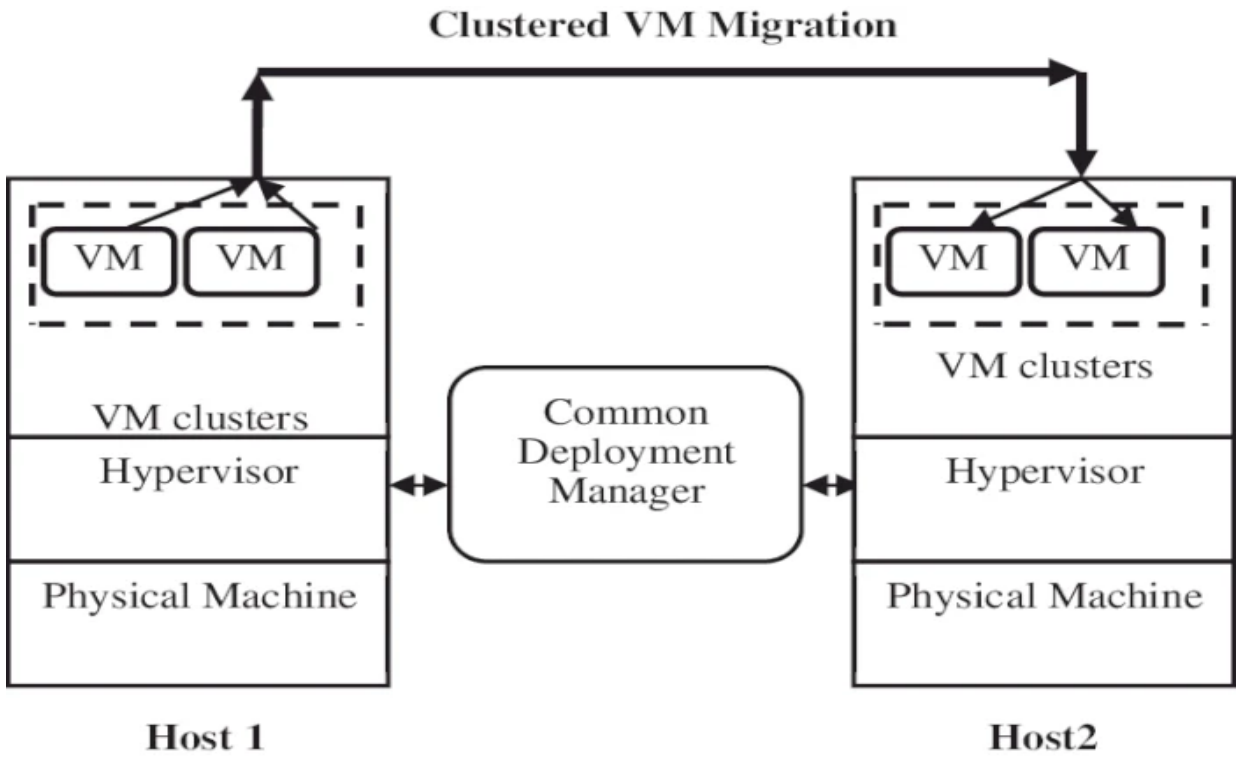

XP Professional (08/11/2009 12:26)
SP2 (10/11/2009 12:26)
Internet Explorer 6 (10/11/2010 15:10)
Internet Explorer 7 (30 days ago)
Updates (17 days ago)
SP3 (28 days ago)
Internet Explorer 7 (26 days ago)
Updates (22 days ago)
Internet Explorer 8 (14 days ago)
Updates (10 days ago)
Internet Explorer 9 (preview) (17 minutes ago)
Current State (changed)

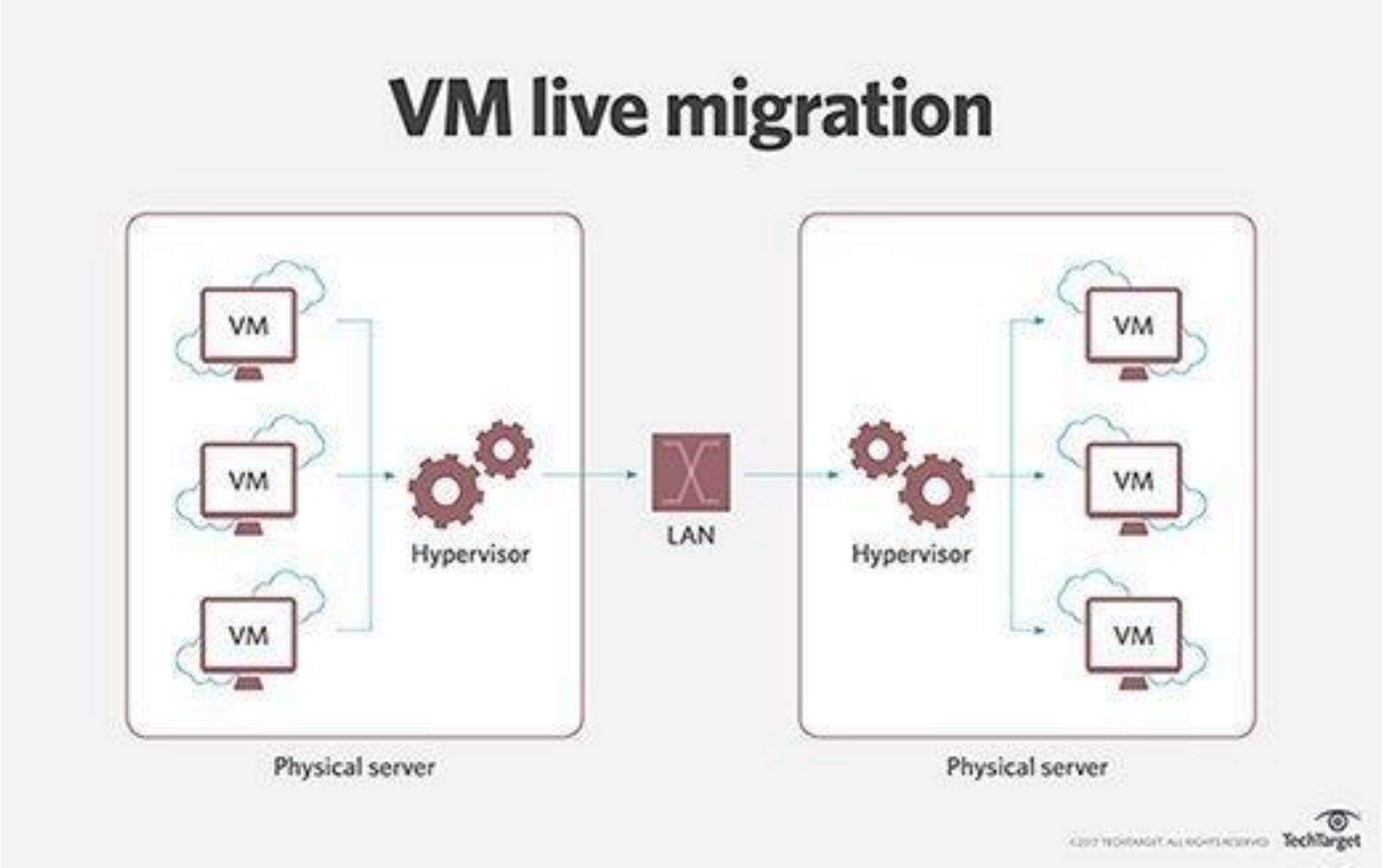

Details
Snapshots (3)
Description
Base
Snapshot 2
Snapshot 3
Current State (changed)

Migration and Failure

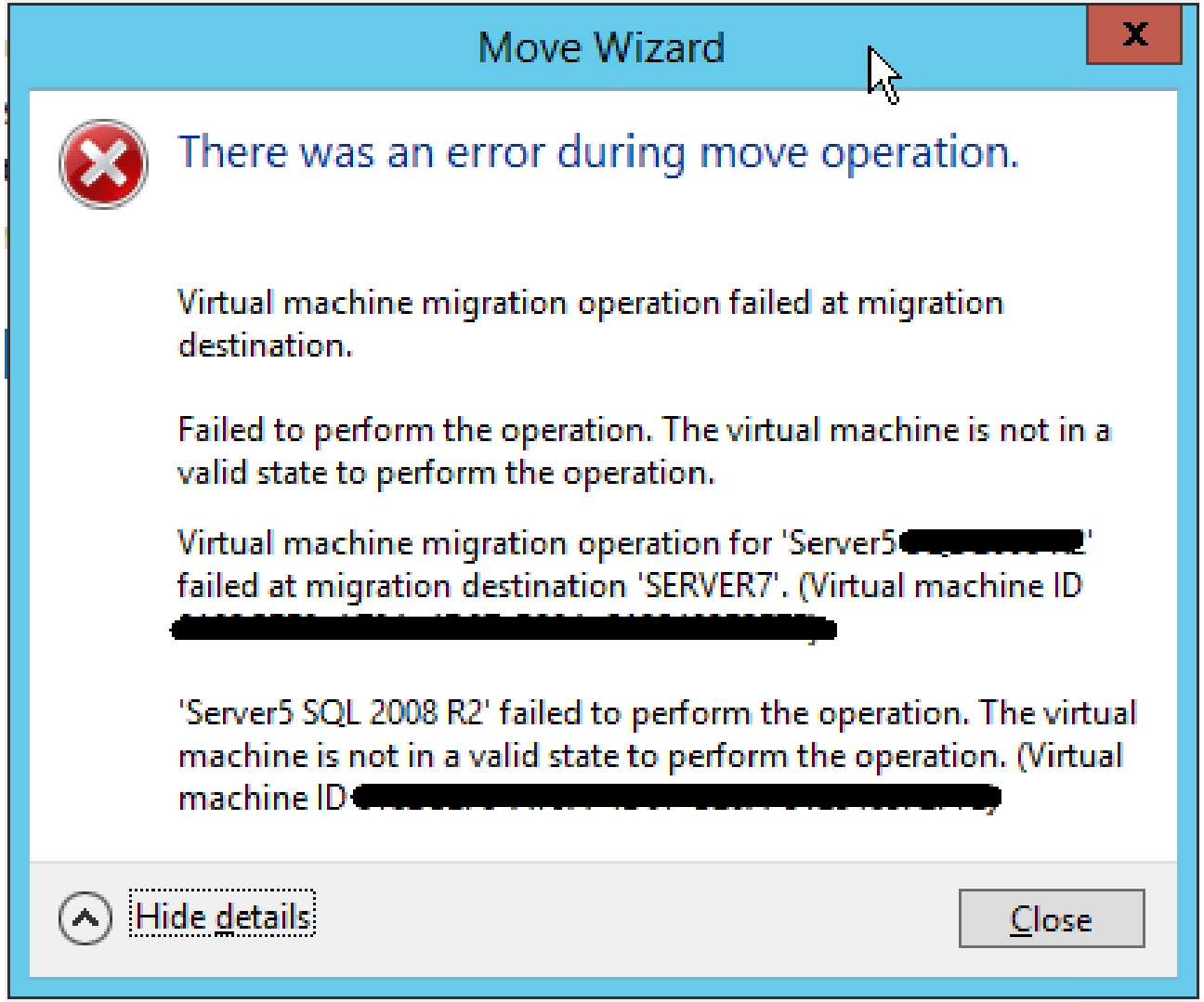
- The snapshots can be moved to another host machine with its own hypervisor; when the VM is temporarily stopped, snapshotted, moved, and then resumed on the new host, this is known as migration.
- Like migration, failover allows the VM to continue operations if the host fails.
- Generally, it occurs if the migration has stopped working.



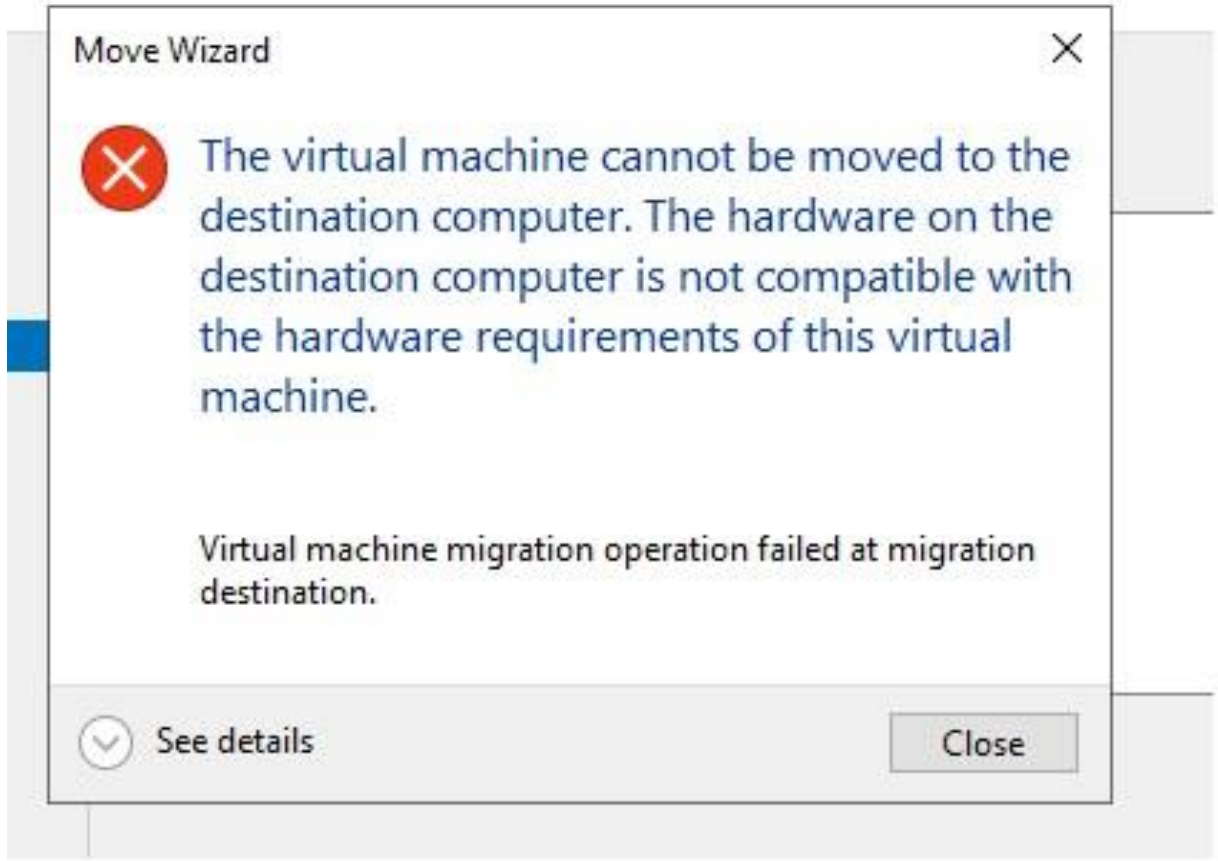



VM live migration
VM
VM
VM
Hypervisor
LAN
Hypervisor
VM
VM
VM
Physical server
Physical server
TechTarget

Migration and Failure

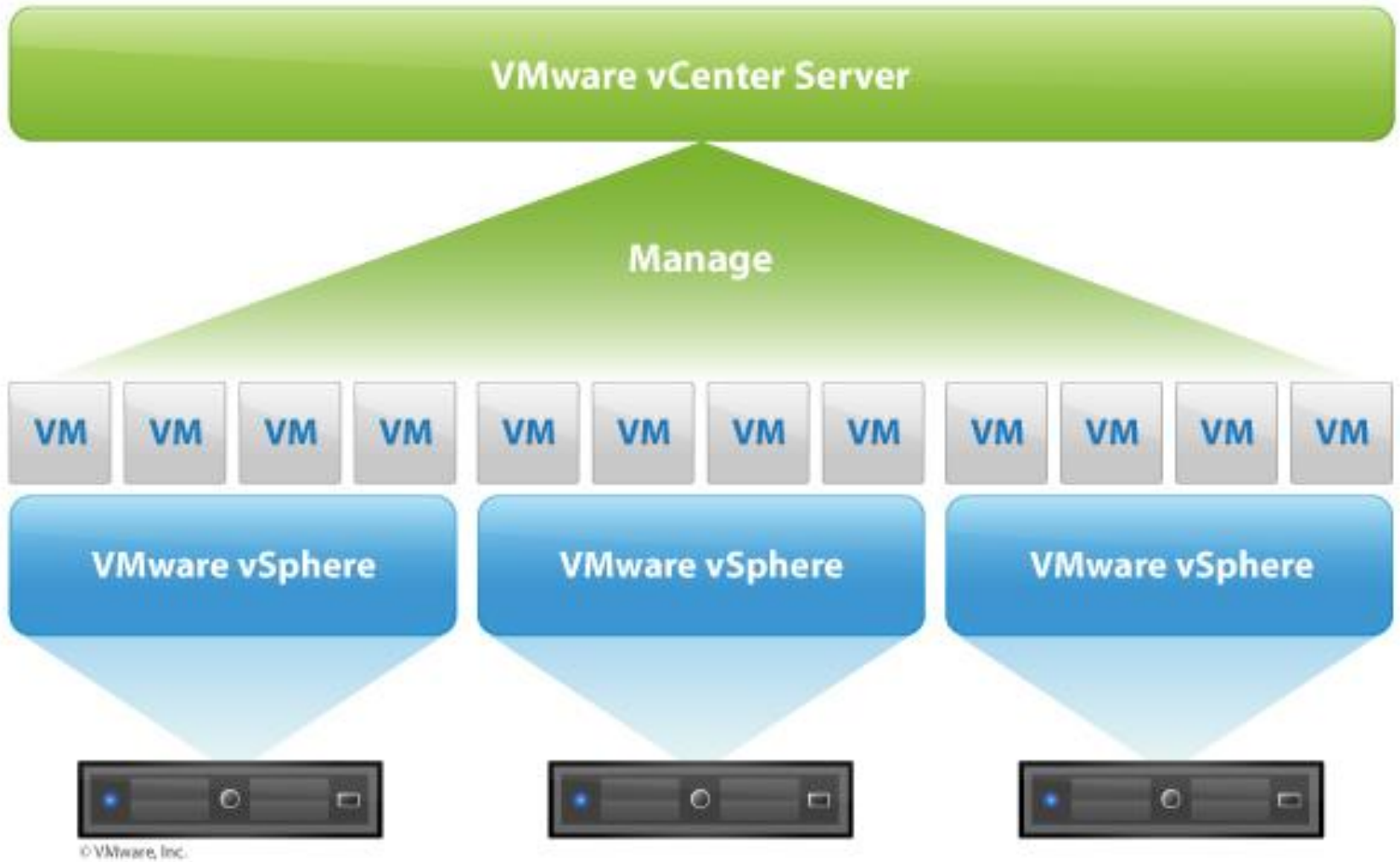


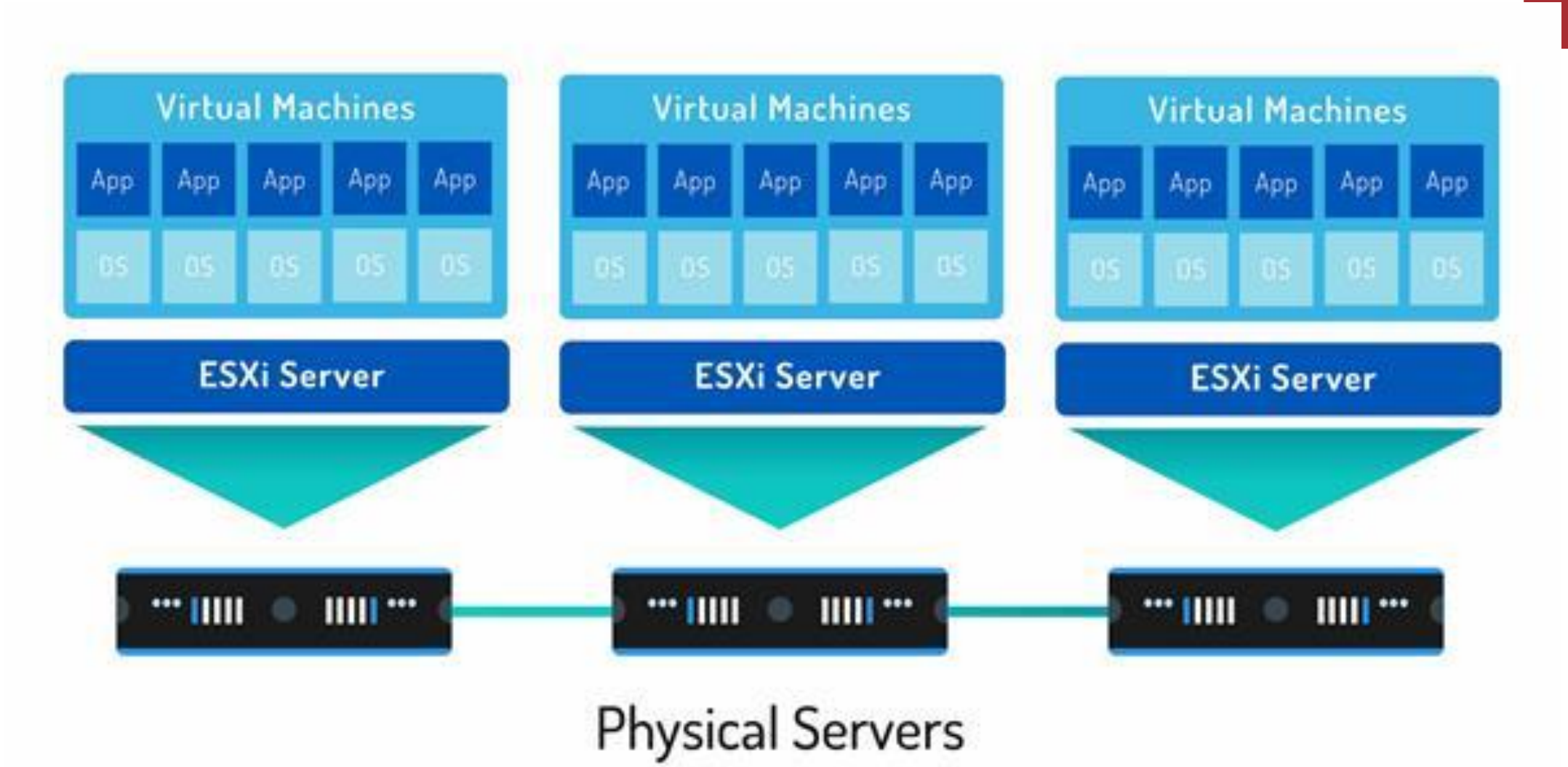
Migration and Failure

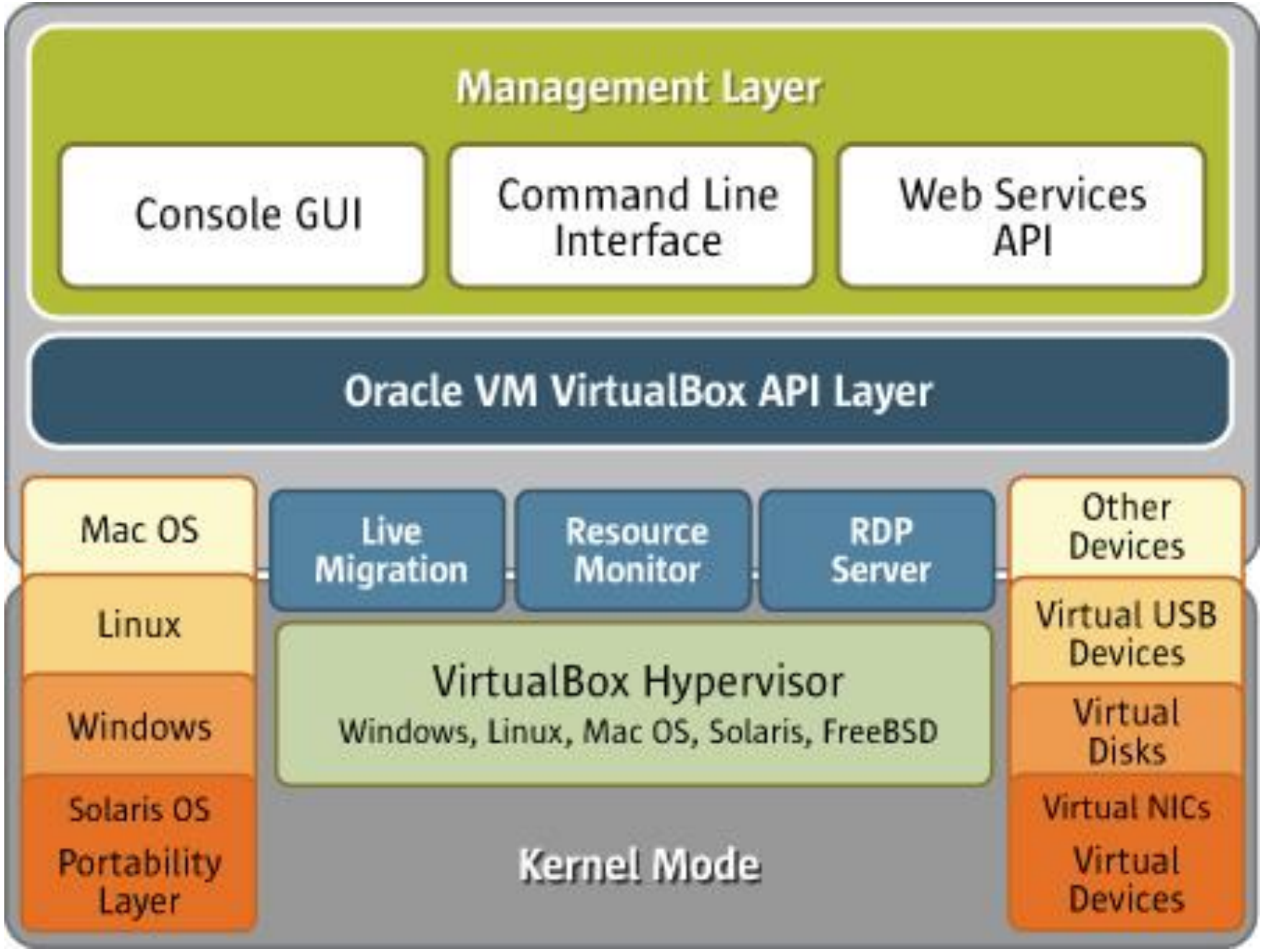


Advantages of Virtualization

- Enable multiple OS on the same machine
- Cheaper due to less/compact infrastructure setup
- Easy to recover in case of failure
- Easy for maintenance
- Faster provisioning of applications and resources required for tasks
- Increase in IT productivity


VMware vCenter Server
Manage
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VMware vSphere
VMware vSphere
VMware vSphere
© VMware, Inc.


Virtual Machines
App
App
App
App
App
OS
OS
OS
OS
OS
ESXi Server
Virtual Machines
App
App
App
App
App
OS
OS
OS
OS
OS
ESXi Server
Virtual Machines
App
App
App
App
App
OS
OS
OS
OS
OS
ESXi Server
Physical Servers


Management Layer
Console GUI
Command Line Interface
Web Services API
Oracle VM VirtualBox API Layer
Mac OS
Linux
Windows
Solaris OS
Portability Layer
Live Migration
Resource Monitor
RDP Server
VirtualBox Hypervisor
Windows, Linux, Mac OS, Solaris, FreeBSD
Kernel Mode
Other Devices
Virtual USB Devices
Virtual Disks
Virtual NICs
Virtual Devices

Virtualization Host

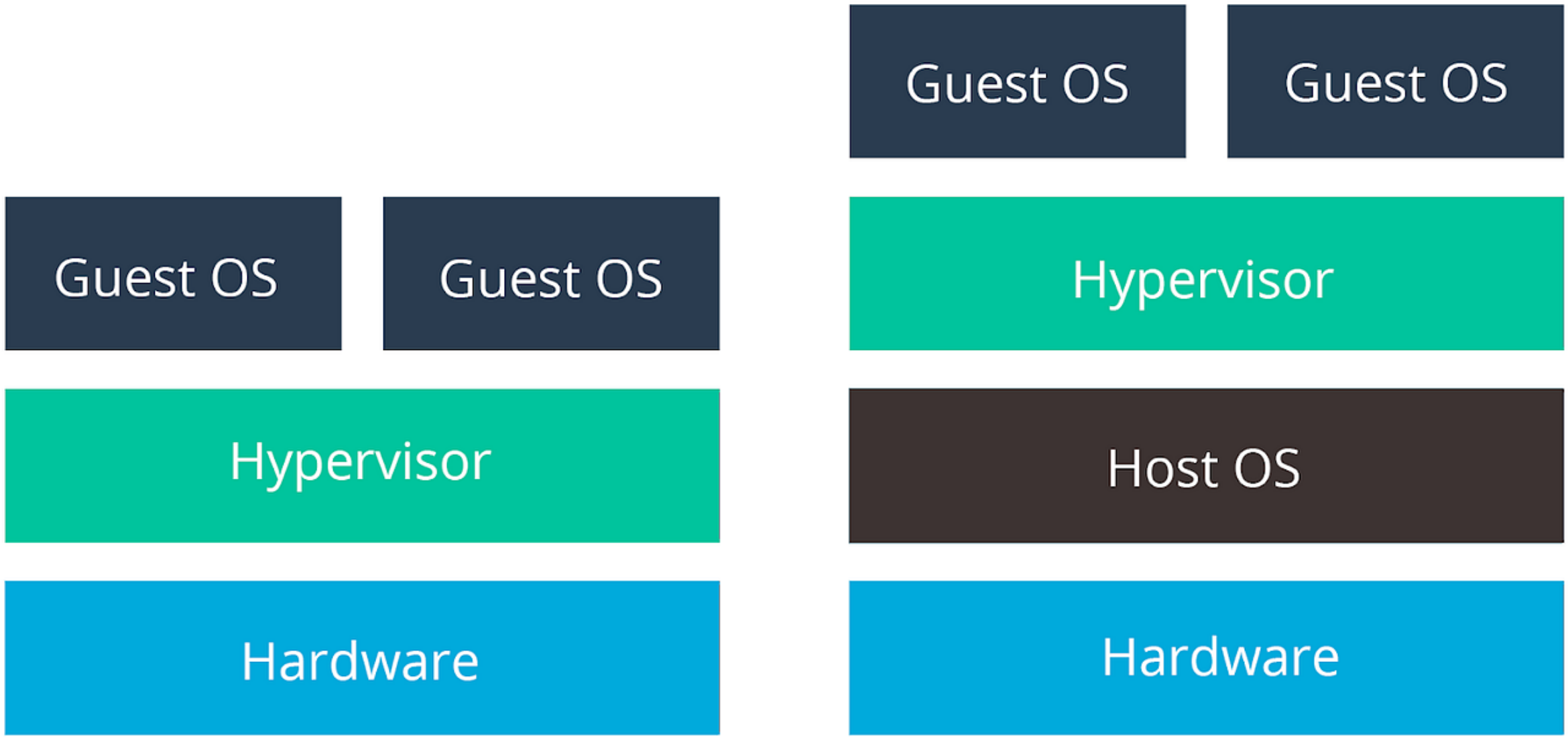
- Virtualization lets us run our applications on fewer physical servers.
- Each app and OS live in a separate software container (VM).
- All resources like CPUs, storage, are pooled together, and they are delivered dynamically to each VM by a software called a hypervisor.
- Running multiple VMs over the same host leads to degradation in performance.
- Guest OS have their own kernel, libraries, and dependencies - large occupation of processor, hard disk and, RAM.
- Bootup process takes a long time

Virtualization Host

- Type 1 hypervisor is a hypervisor that runs directly on the host's hardware to control the hardware and to manage guest operating systems while Type 2 hypervisors run on a conventional operating system just as other computer programs do.
- Hyper-V (Type 1)
- Virtual box (Type 2)
- VMWare (Type 2)
- Xen by AWS (Type 1)

Disadvantages of Virtualization

- Running multiple VMs leads to unstable performance
- Hypervisors are not as efficient as the host operating system
- Boot up process is long and takes time


Guest OS
Guest OS
Hypervisor
Hardware
TYPE 1 HYPERVISOR
Guest OS
Guest OS
Hypervisor
Host OS
Hardware
TYPE 2 HYPERVISOR

Criteria	Type 1 hypervisor	Type 2 hypervisor
AKA	Bare-metal or Native	Hosted
Definition	Runs directly on the system with VMs running on them	Runs on a conventional Operating System
Virtualization	Hardware Virtualization	OS Virtualization
Operation	Guest OS and applications run on the hypervisor	Runs as an application on the host OS
Scalability	Better Scalability	Not so much, because of its reliance on the underlying OS.
Setup/Installation	Simple, as long as you have the necessary hardware support	Lot simpler setup, as you already have an Operating System.
System Independence	Has direct access to hardware along with virtual machines it hosts	Are not allowed to directly access the host hardware and its resources
Speed	Faster	Slower because of the system's dependency
Performance	Higher-performance as there's no middle layer	Comparatively has reduced performance rate as it runs with extra overhead
Security	More Secure	Less Secure, as any problem in the base operating system affects the entire system including the protected Hypervisor
Examples	• VMware ESXi • Microsoft Hyper-V • Citrix XenServer	• VMware Workstation Player • Microsoft Virtual PC • Sun’s VirtualBox

Criteria	Type 1 hypervisor	Type 2 hypervisor
AKA	Bare-metal or Native	Hosted
Definition	Runs directly on the system with VMs running on them	Runs on a conventional Operating System
Virtualization	Hardware Virtualization	OS Virtualization
Operation	Guest OS and applications run on the hypervisor	Runs as an application on the host OS
Scalability	Better Scalability	Not so much, because of its reliance on the underlying OS.
Setup/Installation	Simple, as long as you have the necessary hardware support	Lot simpler setup, as you already have an Operating System.
System Independence	Has direct access to hardware along with virtual machines it hosts	Are not allowed to directly access the host hardware and its resources
Speed	Faster	Slower because of the system's dependency
Performance	Higher-performance as there's no middle layer	Comparatively has reduced performance rate as it runs with extra overhead
Security	More Secure	Less Secure, as any problem in the base operating system affects the entire system including the protected Hypervisor
Examples	• VMware ESXi • Microsoft Hyper-V • Citrix XenServer	• VMware Workstation Player • Microsoft Virtual PC • Sun’s VirtualBox

The Need for Virtualization

- Resource Utilization:
- Underutilization of Hardware: In traditional IT, servers were often dedicated to a single app, leading to low utilization rates (10-15%)
- **Virtualization** allows multiple virtual machines (VMs) to run on a single physical server, significantly improving resource utilization.
- Each VM can run a different operating system and application, optimizing the use of CPU, memory, and storage.

The Need for Virtualization

- Cost Efficiency:
- **Hardware Costs**: Before virtualization, organizations needed to purchase and maintain separate physical servers for different applications, leading to high costs.
- **Consolidation**: Virtualization reduces the number of physical servers required by consolidating workloads onto fewer machines. This results in lower hardware, energy, and maintenance costs

The Need for Virtualization

- Scalability and Flexibility:
- **Dynamic Workloads**: Traditional physical servers are not flexible enough to handle dynamic workloads. Scaling up or down required purchasing and configuring new hardware, which was time-consuming and expensive
- **Virtualization** allows for easy scalability. VMs can be quickly created, modified, or moved between physical servers, enabling rapid response to changing demands

The Need for Virtualization

- Isolation and Security:
- **Application Isolation**: Running multiple applications on a single server without isolation could lead to conflicts and security risks
- **VM Isolation**: Virtualization provides a layer of isolation between VMs, so that one application's failure or security breach doesn't affect others running on the same physical hardware

Containerization

- Virtualization is brought to an operating system level.
- We virtualize the OS resources.
- It is more efficient as there is no guest OS consuming the host resources;
- Containers utilize only the host OS and share relevant libraries and resources only when they are required.
- The required binaries and libraries of containers run on the host kernel leading to faster processing and execution.
- Containerization is lightweight virtualization acting as an alternative to hypervisor virtualization - Any app can be bundled in a container and run without caring about dependencies, libraries, and binaries.

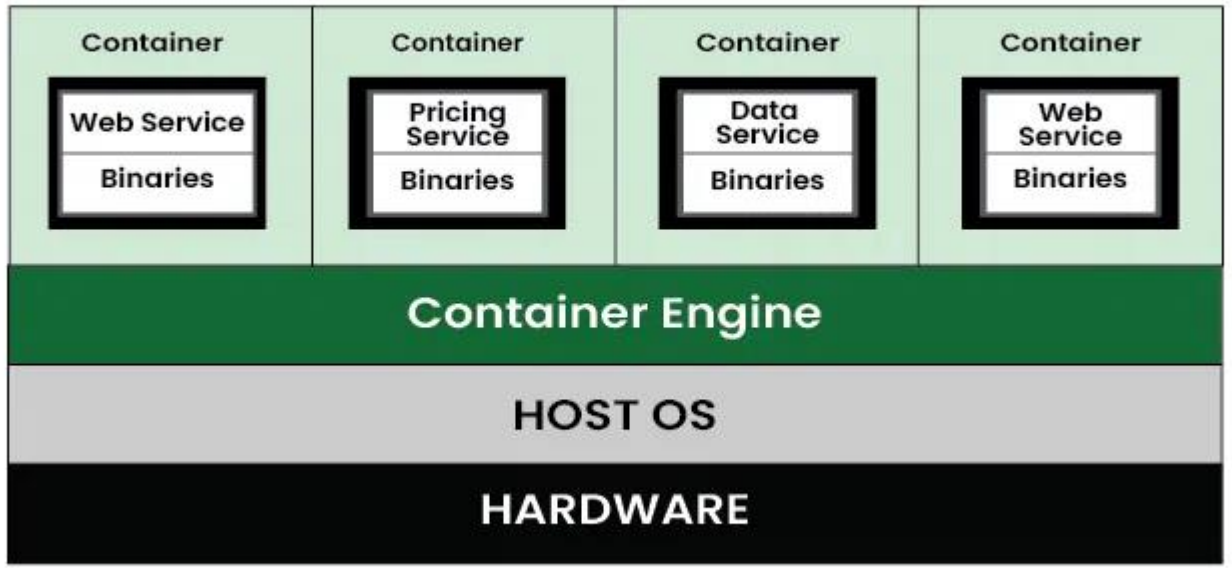
Containerization

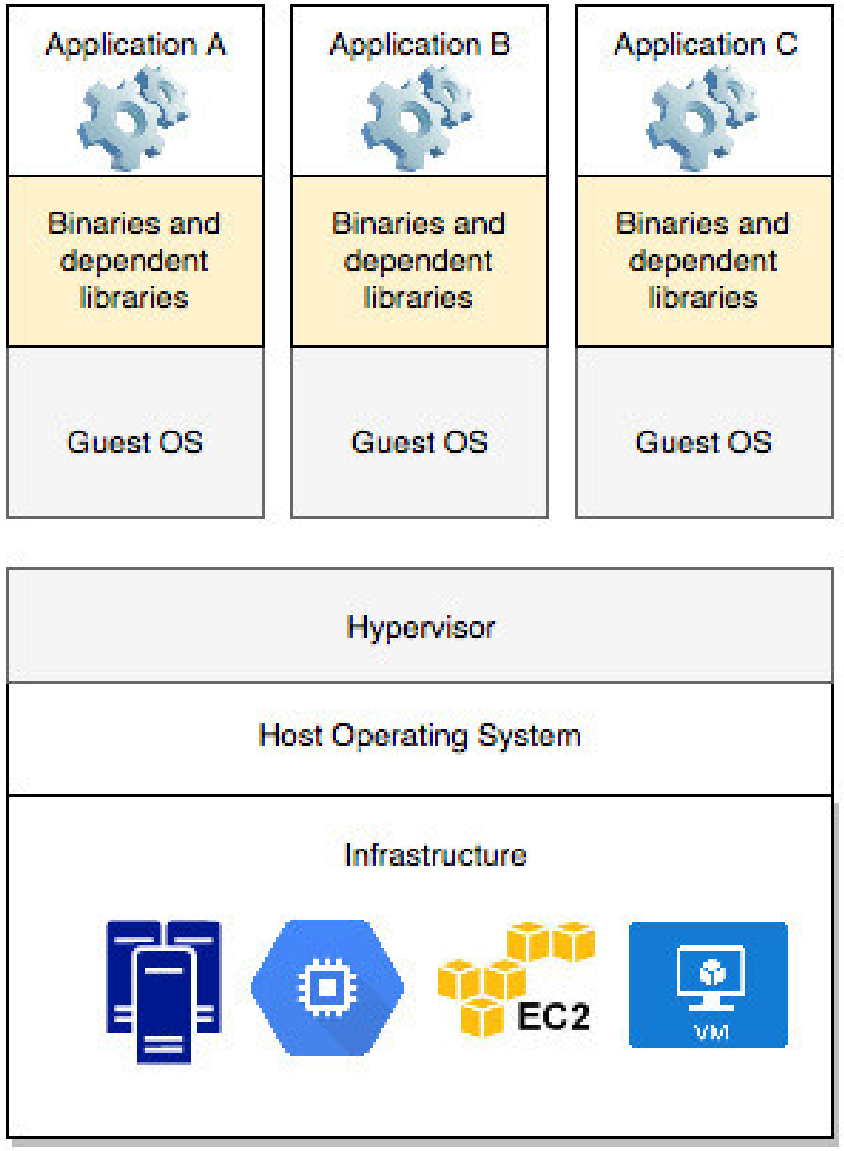
- In the case of containerization, all containers share the same host operating system.
- Multiple containers get created for every type of application making them faster but without wasting the resources, unlike virtualization where a kernel is required for every OS and lots of resources from the host OS are utilized.
- Containers are small and lightweight as they share the same OS kernel.
- They do not take much time to boot-up (only seconds).
- They exhibit high performance with lower resource utilization

Containerization

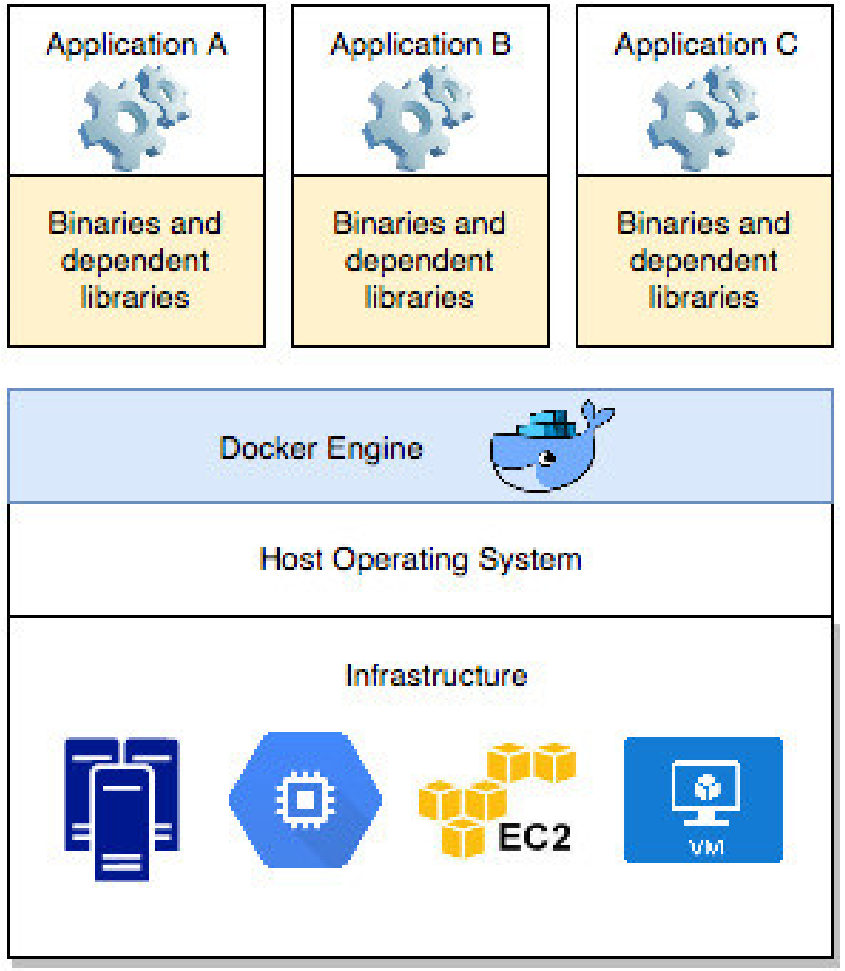
- In the case of containerization, all containers share the same host operating system.
- Multiple containers get created for every type of application making them faster but without wasting the resources, unlike virtualization where a kernel is required for every OS and lots of resources from the host OS are utilized.
- Containers are small and lightweight as they share the same OS kernel.
- They do not take much time to boot-up (only seconds).
- They exhibit high performance with lower resource utilization

Containerization Architecture



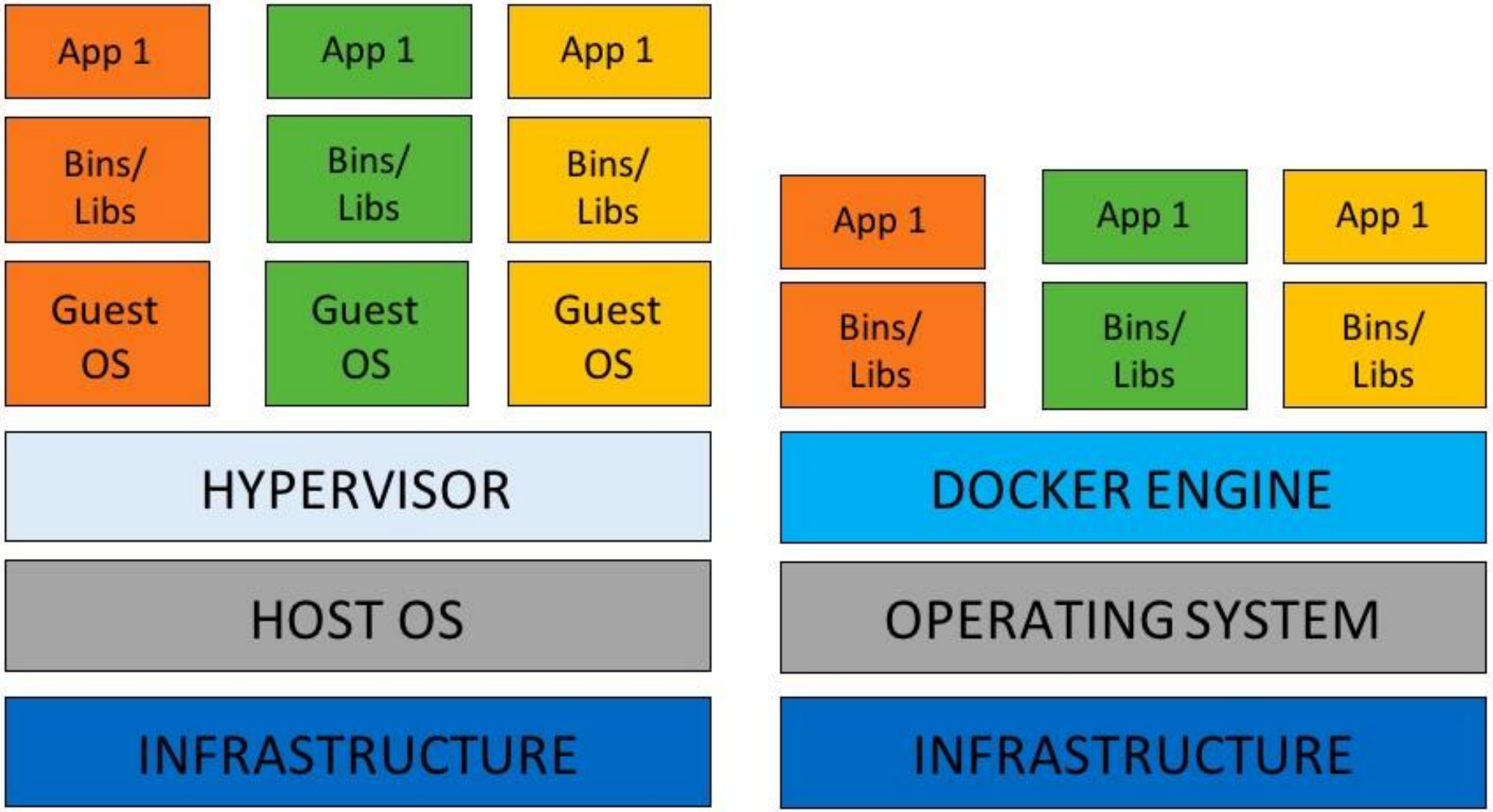


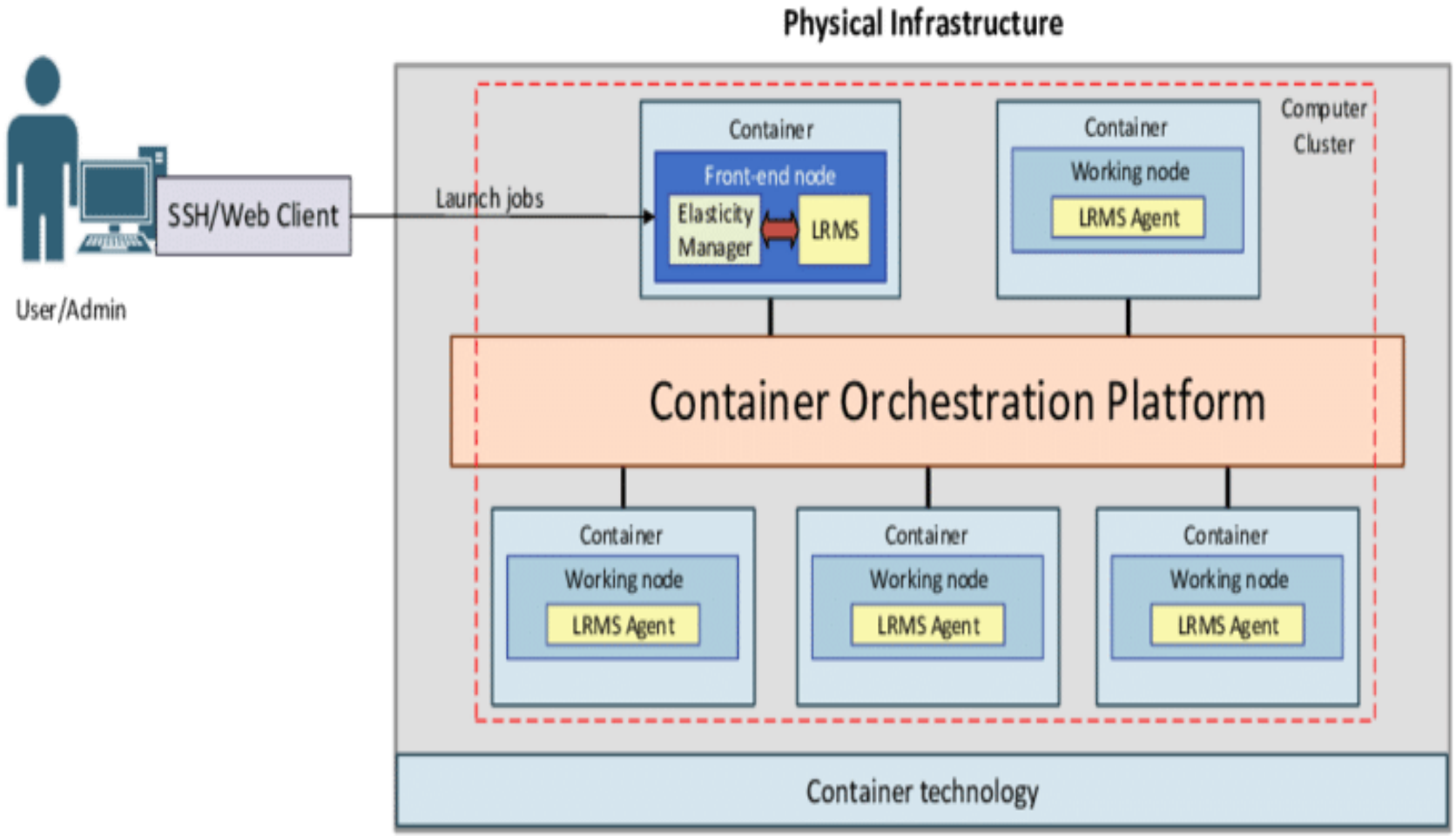
Virtual Machines based architecture



Container based architecture

Pablo Iorio


App 1
App 1
App 1
Bins/ Libs
Bins/ Libs
Bins/ Libs
Guest OS
Guest OS
Guest OS
HYPERVISOR
HOST OS
INFRASTRUCTURE
App 1
App 1
App 1
Bins/ Libs
Bins/ Libs
Bins/ Libs
DOCKER ENGINE
OPERATING SYSTEM
INFRASTRUCTURE


Physical Infrastructure
SSH/Web Client
User/Admin
Launch jobs
Container
Front-end node
Elasticity Manager
LRMS
Container
Working node
LRMS Agent
Computer Cluster
Container Orchestration Platform
Container
Working node
LRMS Agent
Container
Working node
LRMS Agent
Container
Working node
LRMS Agent
Container technology

Key differences between Containerization & Virtualization in 2020

Basis	Virtualization (VMware)	Containerization (Dockers)
Framework	Virtualization on the hardware level. Each VM equips its copy of the OS	Virtualization at the software level. Dockers don't have its OS, & run on the host operating system.
Segregation	Fully segregate	Process or application-level segregation
Portability	In this department, it is less portable because of hardware & operating system burden. And it also requires manual setup on storage, RAM & network.	It is highly portable because of its lightweight feature & no dependence on the hardware.
Scalability and code-reusability	It needs vmware administrative support while spinning up. Running a new machine requires extra effort in the process.	It simply means spinning up new containers on pre-built images inside the host OS. The dockers are also configured on-the-fly passing basis & run-time.
Up gradation	The vmware virtual machines get a release patch for updates.	In this containers get built on a single image & get rebuilt & distributed amongst multiple platforms
Security	On comparability with docker, vmware is more secure which doesn't affect its host OS.	Docker is less secure on the comparison chart because if any container were compromised then it affects other containers & the whole OS too.

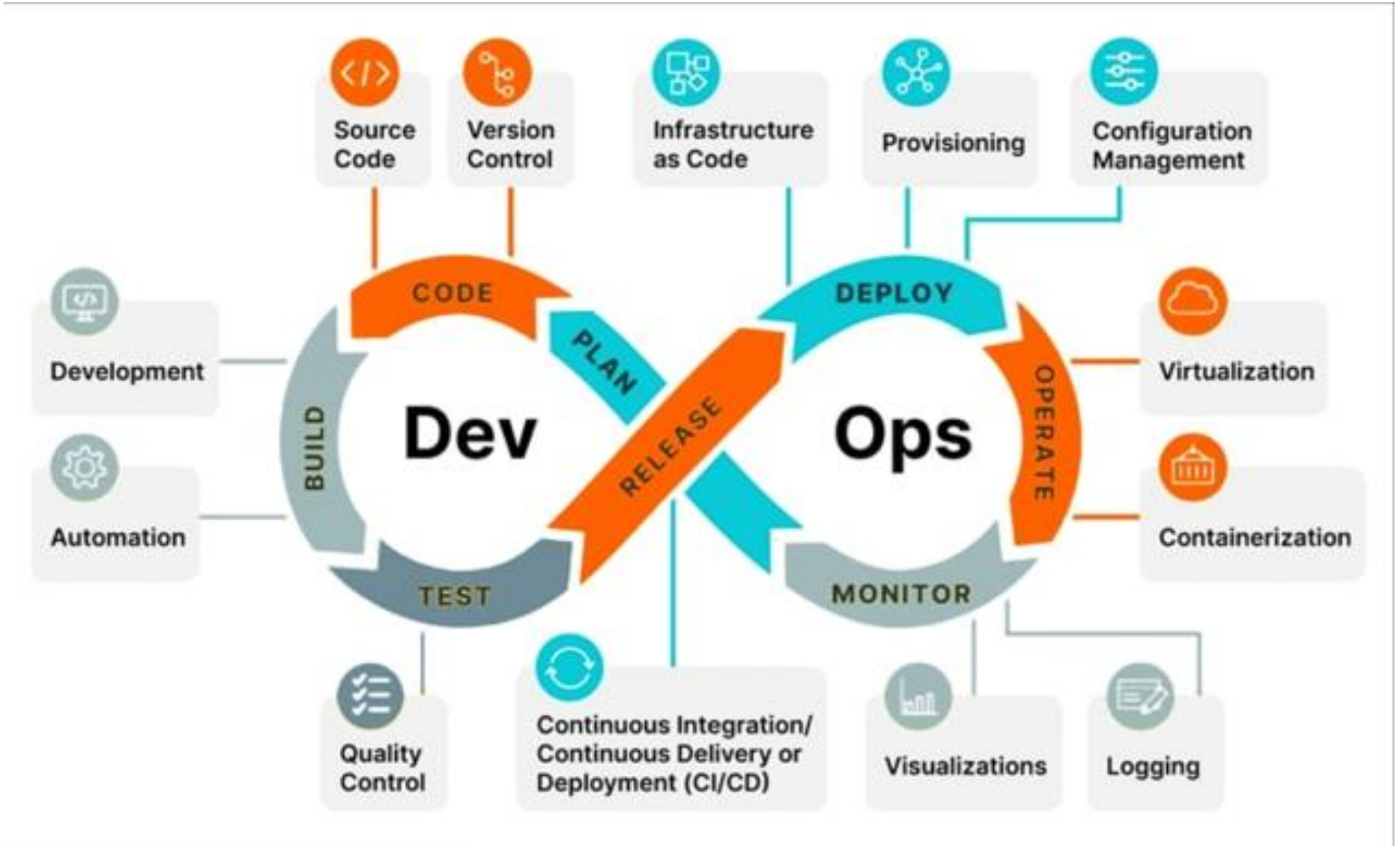
Feature	**App Virtualization**	**Containerization (e.g., Docker)**
What is virtualized?	Only the **application layer** (files, registry, dependencies).	The **entire runtime environment** (app + dependencies + libraries + OS user-space).
Host OS dependency	App is packaged and redirected via a virtualization layer, but still runs on the **host OS APIs**.	Containers share the **host OS kernel**, but isolate the app + environment completely.
Isolation	Limited — app is sandboxed from host OS, but still more "tied" to host.	Stronger isolation — containers behave like lightweight VMs.
Deployment	Typically used in enterprises for Windows apps (e.g., MS App-V, VMware ThinApp, Citrix).	Used for cloud-native / microservices apps (e.g., Docker, Kubernetes).
Performance	Almost native speed, since only calls are redirected.	Near-native speed, but with small overhead for isolation (namespaces, cgroups).
Use case	Run legacy or conflicting apps side by side, deliver apps without traditional install.	Build once, run anywhere — portable deployments across dev, test, prod.
Example	Run **MS Word 2010** and **MS Word 2021** on the same PC without conflict.	Package a **Python web service** with Flask + libraries + OS deps in one container.

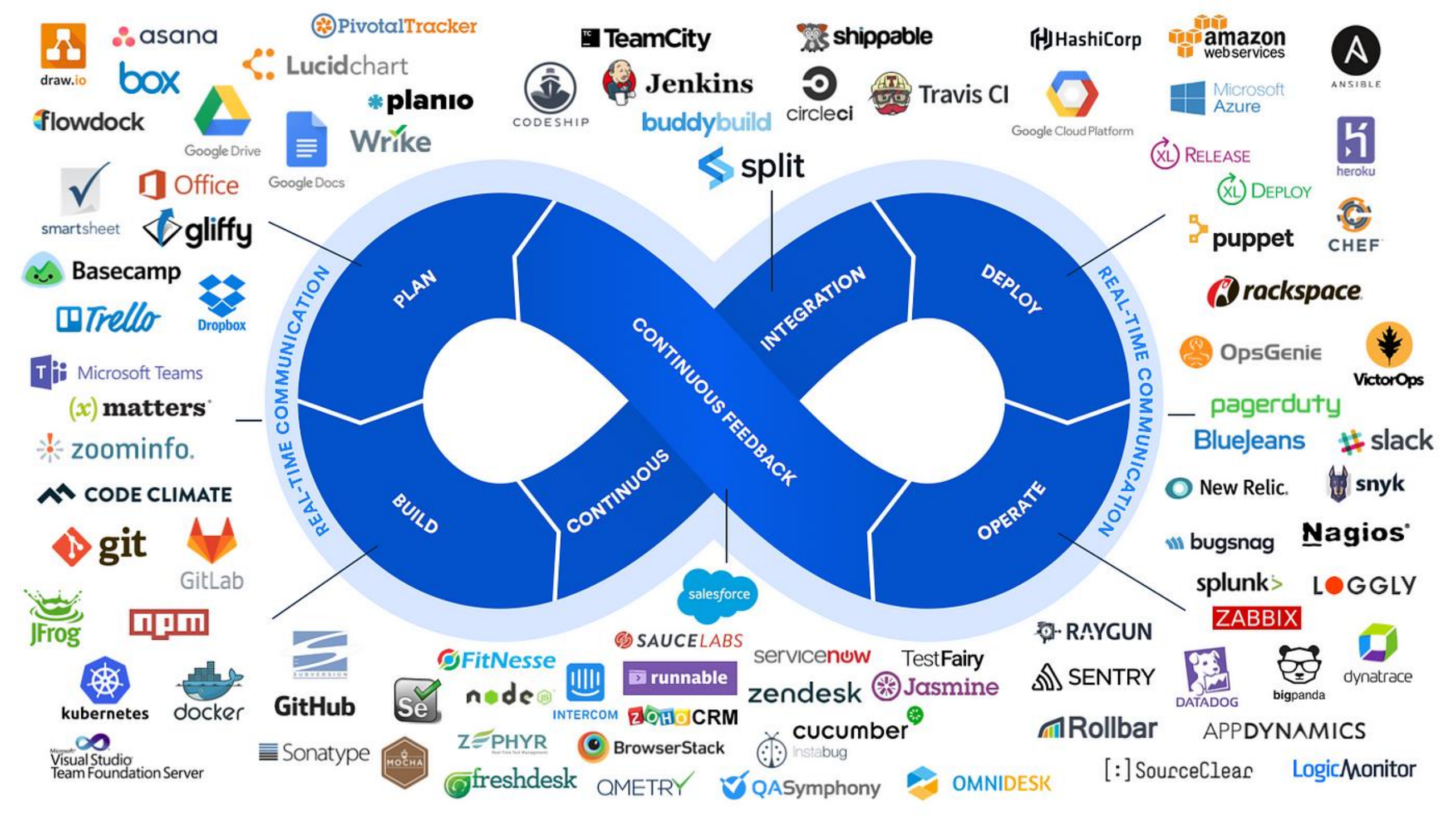
Docker

- Docker is a containerization platform that packages app and all its dependencies together in the form of Containers to ensure seamless work
- Each app will run on a separate container and will have its own set of libraries and dependencies.
- This also ensures that there is process level isolation: each app is independent of other apps, giving developers surety that they can build applications that will not interfere with one another.
- As a developer, one can build a container which has different applications installed on it and give it to QA team who will only need to run the container to replicate the developer environment.

Docker

- With docker, the QA team need not install all the dependent software and applications to test the code and this helps them save lots of time and energy.
- This also ensures that the working environment is consistent across all the individuals involved in the process, starting from development to deployment.
- The number of systems can be scaled up easily and the code can be deployed on them effortlessly.


Source Code
Version Control
Infrastructure as Code
Provisioning
Configuration Management
Development
Automation
Virtualization
Containerization
CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR
Dev
Ops
Quality Control
Continuous Integration/ Continuous Delivery or Deployment (CI/CD)
Visualizations
Logging


PLAN
CONTINUOUS
BUILD
CONTINUOUS
INTEGRATION
DEPLOY
OPERATE
CONTINUOUS FEEDBACK
REAL-TIME COMMUNICATION
REAL-TIME COMMUNICATION
draw.io
asana
PivotalTracker
box
Lucidchart
planio
flowdock
Google Drive
Google Docs
Wrike
smartsheet
Office
gliffy
Basecamp
Trello
Dropbox
Microsoft Teams
(x) matters
zoominfo.
CODE CLIMATE
git
GitLab
JFrog
npm
kubernetes
docker
Visual Studio Team Foundation Server
GitHub
Sonatype
TeamCity
CODESHIP
Jenkins
buddybuild
shippable
circleci
Travis CI
split
salesforce
SAUCELABS
FitNesse
node
runnable
INTERCOM
ZOHO CRM
ZEPHYR
BrowserStack
freshdesk
QMETRY
servicenow
TestFairy
zendesk
Jasmine
cucumber
Instabug
QASymphony
OMNIDESK
HashiCorp
amazon web services
Google Cloud Platform
Microsoft Azure
ANSIBLE
RELEASE
heroku
DEPLOY
puppet
CHEF
rackspace
OpsGenie
VictorOps
pagerduty
BlueJeans
slack
New Relic.
snyk
bugsnag
Nagios
splunk>
LOGGLY
ZABBIX
RAYGUN
SENTRY
DATADOG
bigpanda
dynatrace
Rollbar
APPDYNAMICS
[:] SourceClear
LogicMonitor