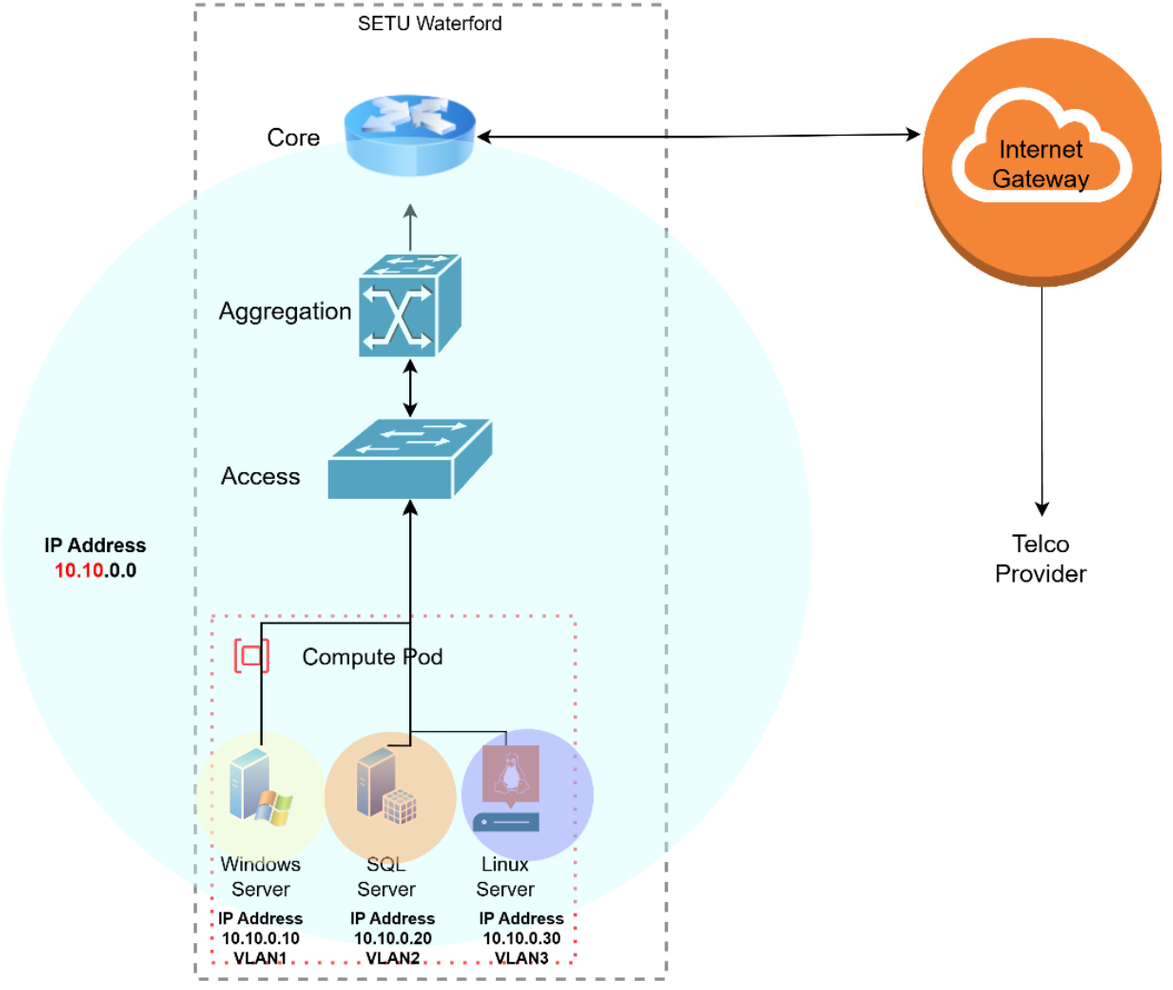
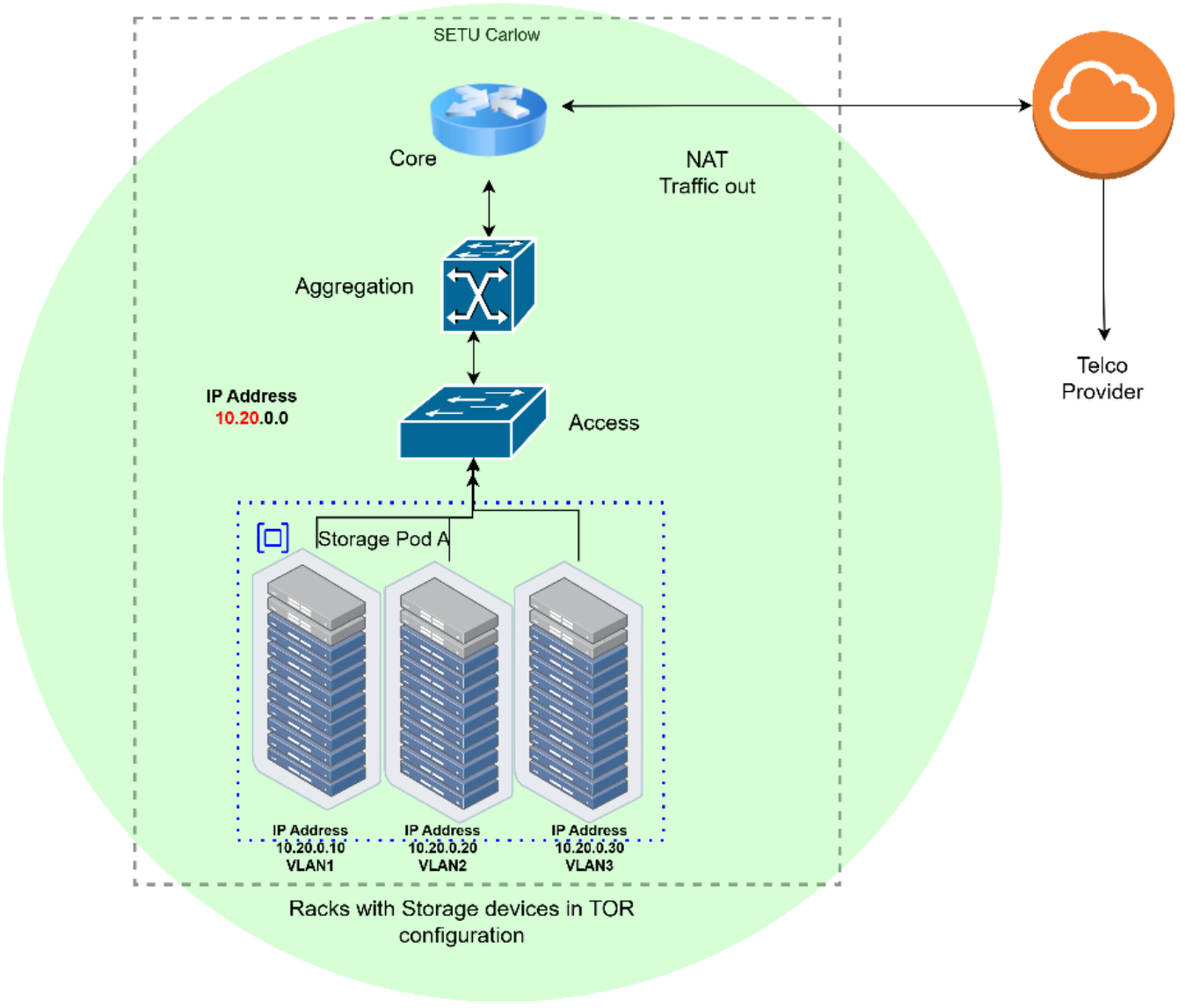
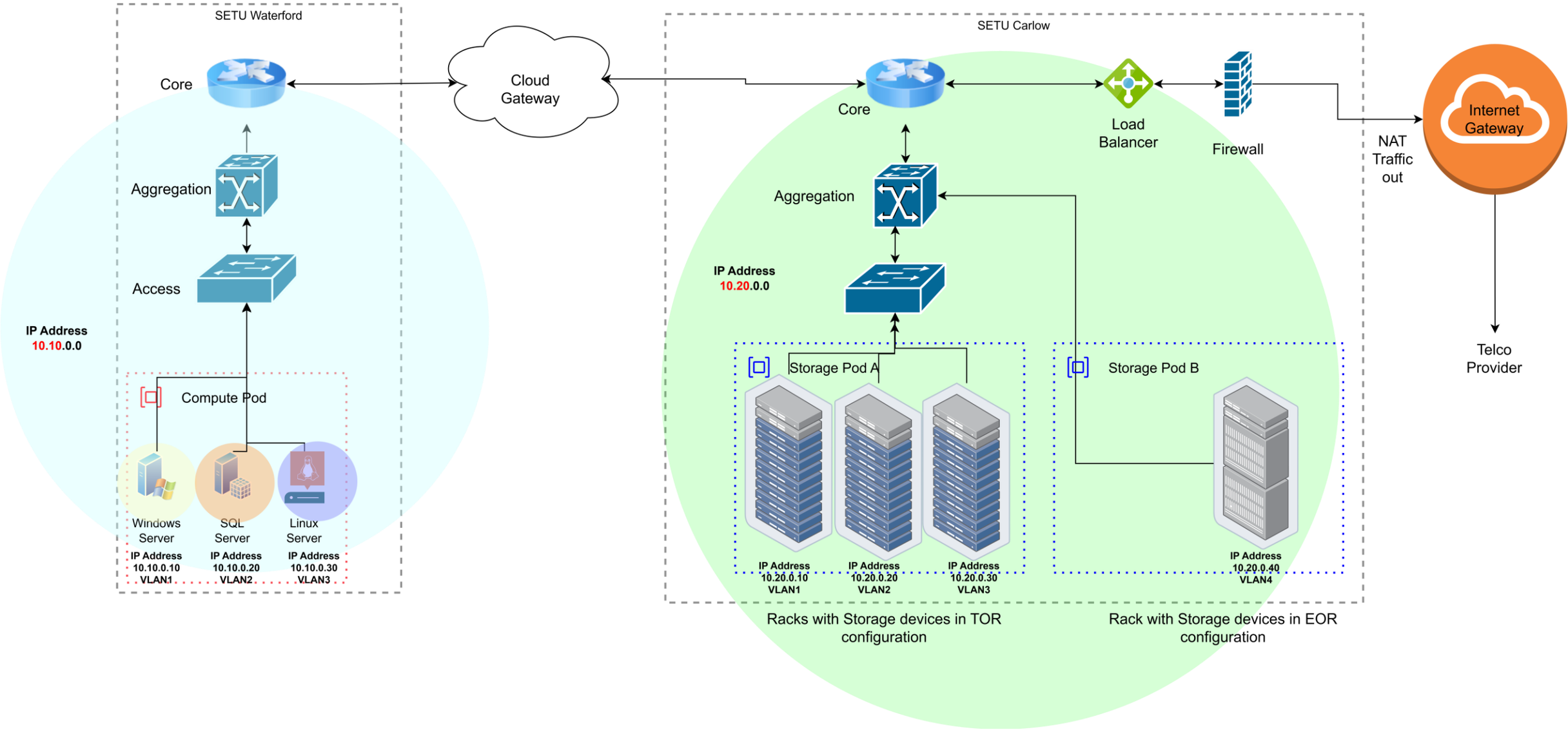
Designing Cloud Data Center

What we have so far?


SETU Waterford
Core
Internet Gateway
Aggregation
Access
IP Address
10.10.0.0
Telco Provider
Compute Pod
Windows Server
SQL Server
Linux Server
IP Address
10.10.0.10
VLAN1
IP Address
10.10.0.20
VLAN2
IP Address
10.10.0.30
VLAN3

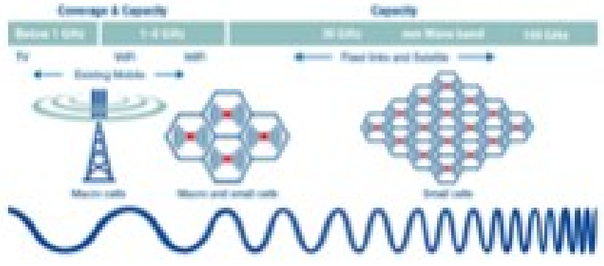

SETU Carlow
Core
NAT
Traffic out
Aggregation
IP Address
10.20.0.0
Access
Storage Pod A
IP Address
10.20.0.10
VLAN1
IP Address
10.20.0.20
VLAN2
IP Address
10.20.0.30
VLAN3
Racks with Storage devices in TOR configuration
Telco
Provider


SETU Waterford
Core
Aggregation
Access
IP Address
10.10.0.0
Compute Pod
Windows Server
SQL Server
Linux Server
IP Address 10.10.0.10 VLAN1
IP Address 10.10.0.20 VLAN2
IP Address 10.10.0.30 VLAN3
Cloud Gateway
SETU Carlow
Core
Load Balancer
Firewall
NAT Traffic out
Internet Gateway
Telco Provider
Aggregation
IP Address
10.20.0.0
Storage Pod A
Storage Pod B
IP Address 10.20.0.10 VLAN1
IP Address 10.20.0.20 VLAN2
IP Address 10.20.0.30 VLAN3
IP Address 10.20.0.40 VLAN4
Racks with Storage devices in TOR configuration
Rack with Storage devices in EOR configuration

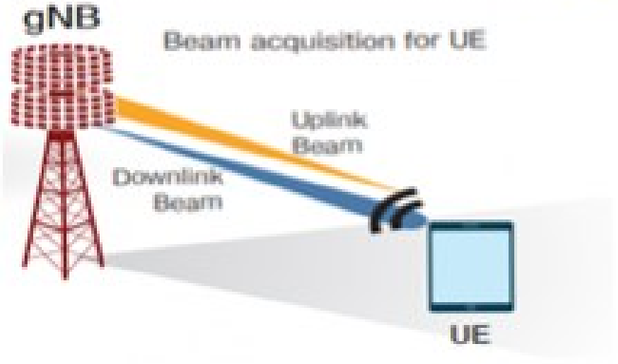
Enabling technologies

5G Key Technologies

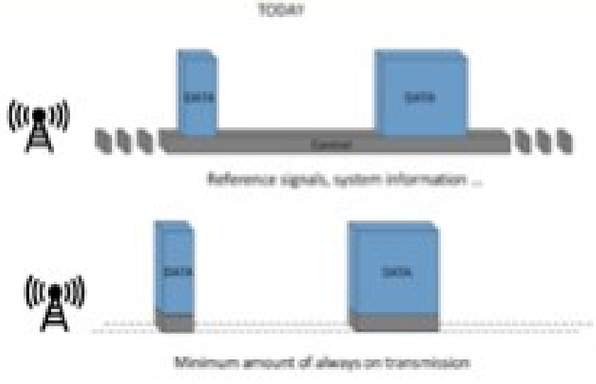
New Spectrum and Bandwidth



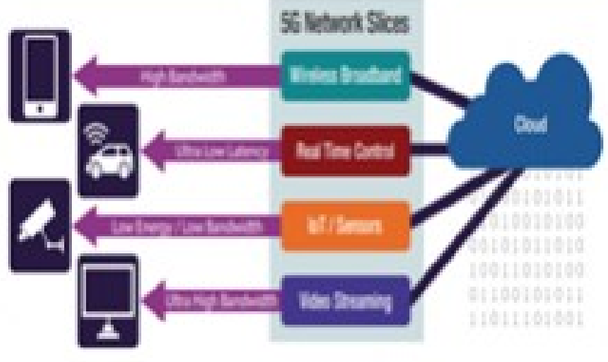
mMIMO and Beamforming



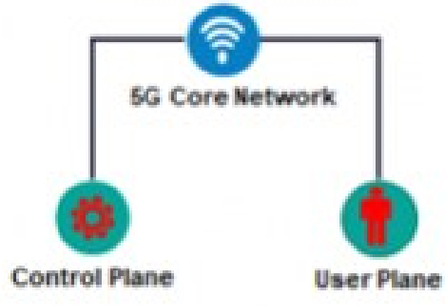
Lean Design



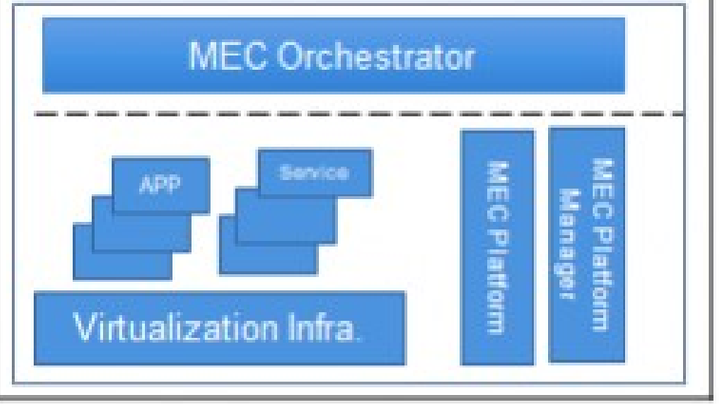
Network Slicing

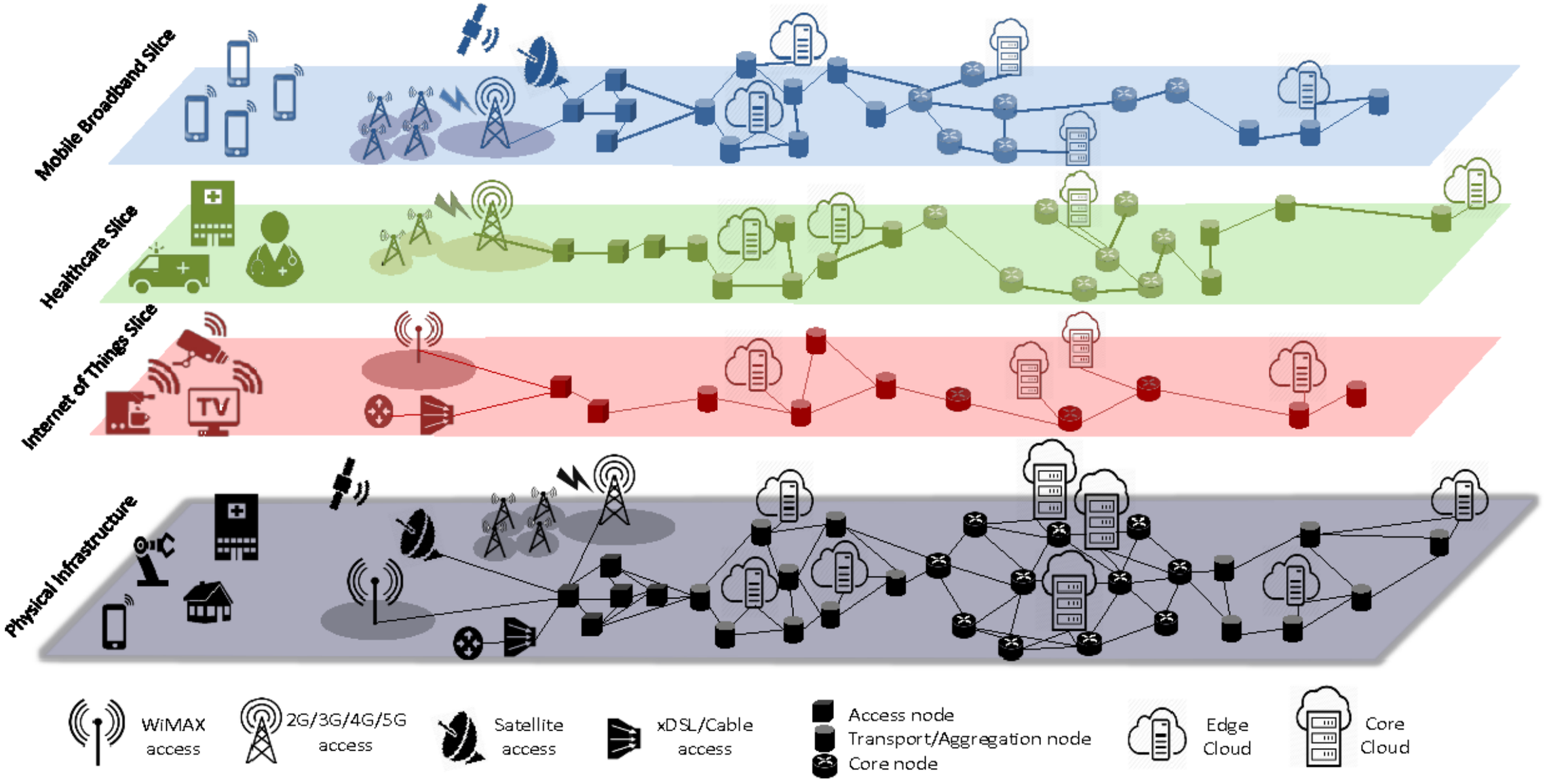


CUPS



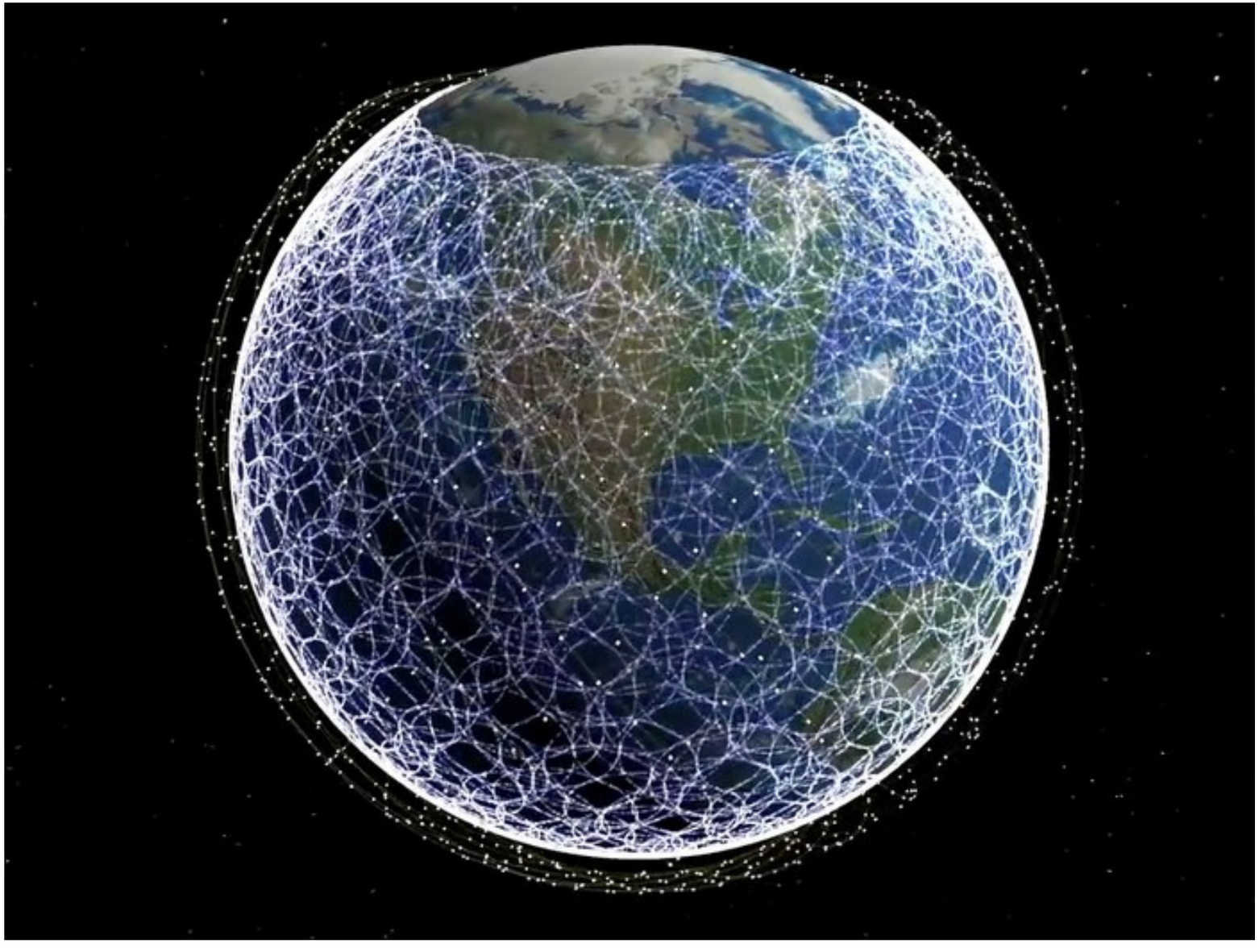
Multi-access Edge Computing



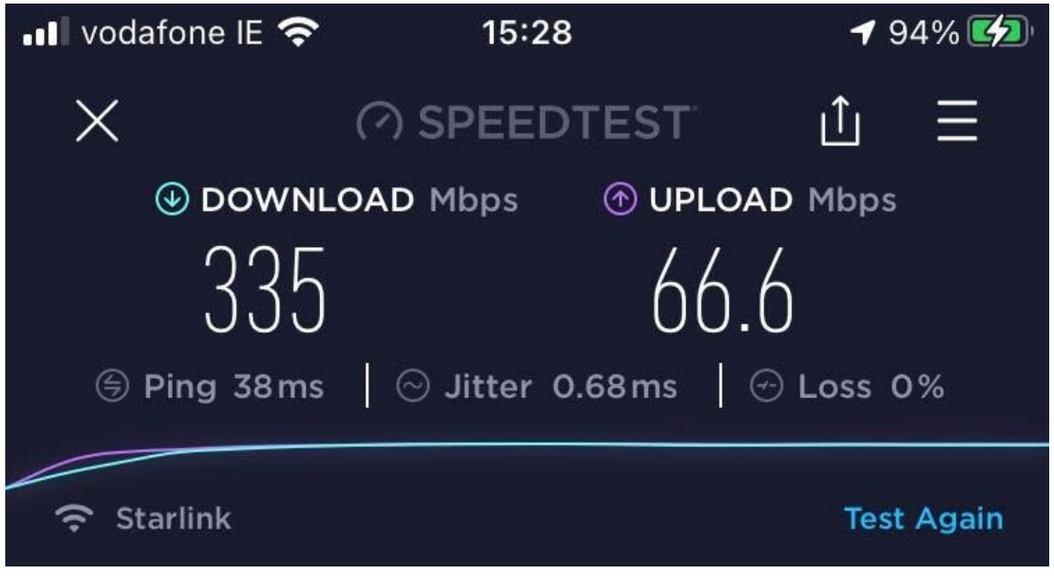

Mobile Broadband Slice
Healthcare Slice
Internet of Things Slice
TV
Physical Infrastructure
WiMAX access
2G/3G/4G/5G access
Satellite access
xDSL/Cable access
Access node
Transport/Aggregation node
Core node
Edge Cloud
Core Cloud

Google Balloon

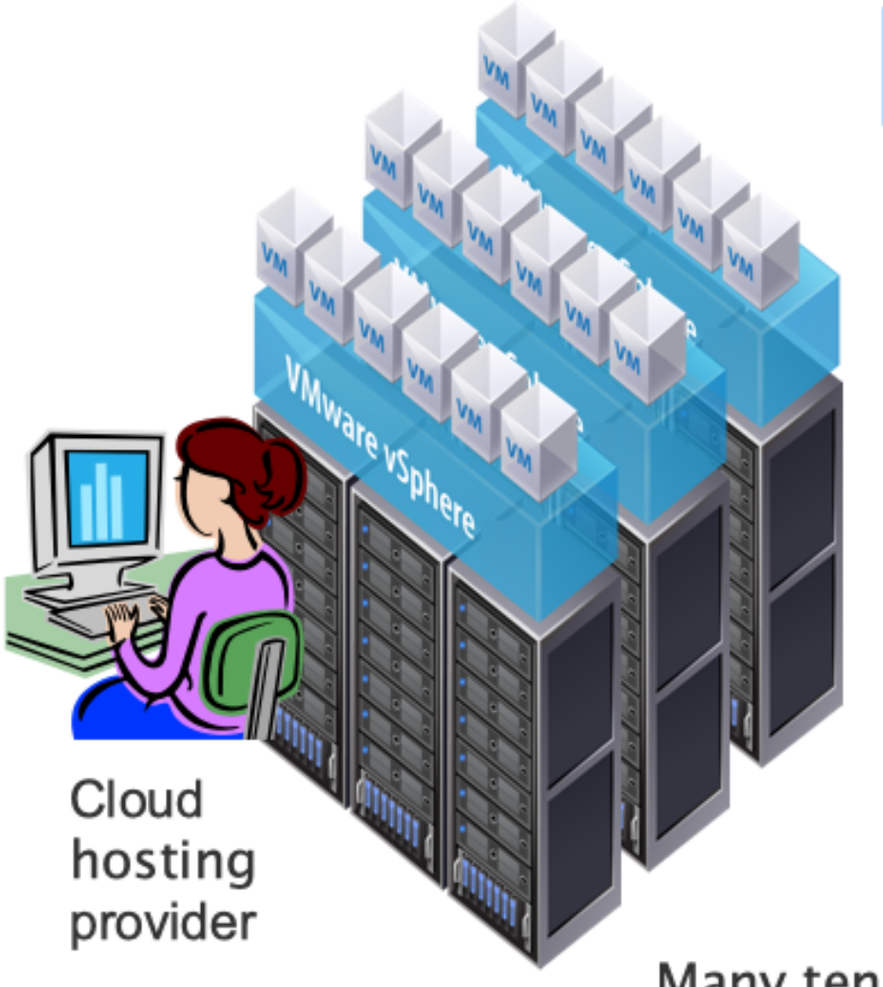
Tesla StarLink



StartLink@Countryside

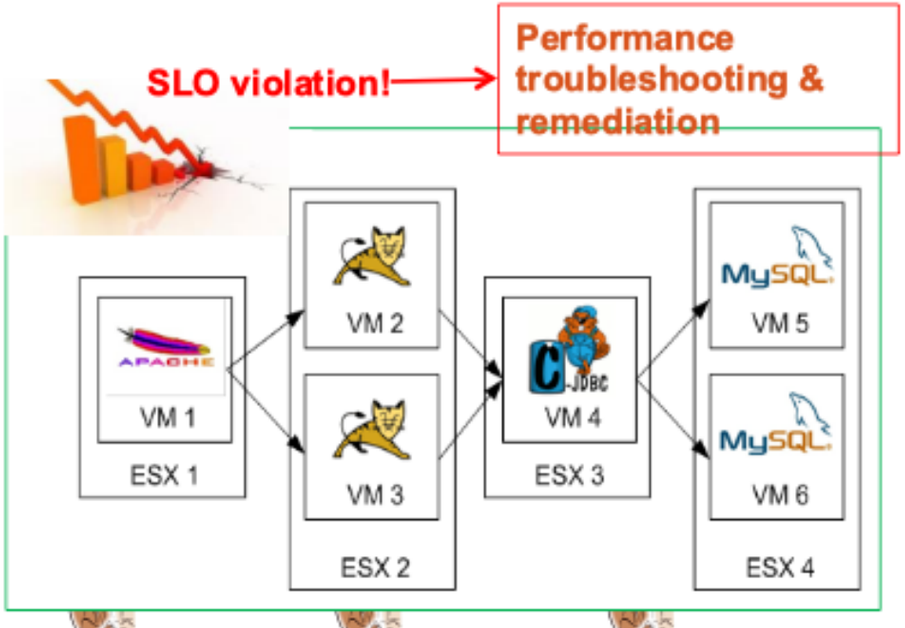


Application performance management



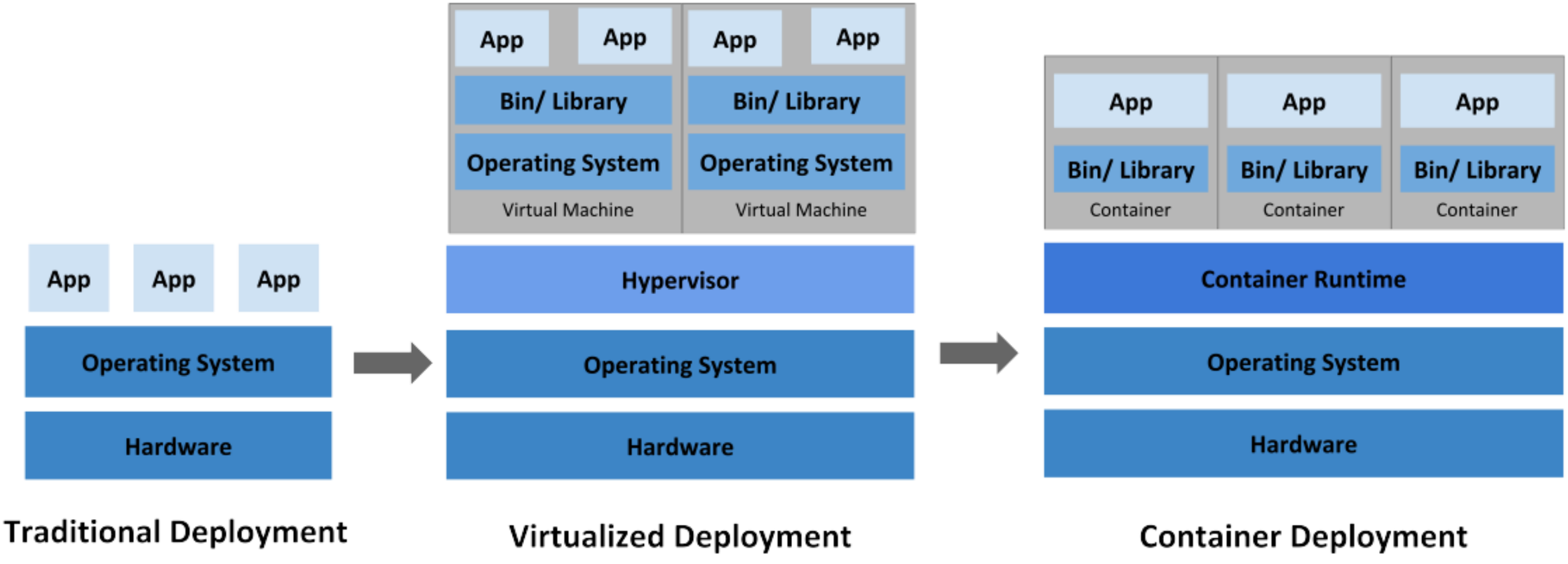
Cloud hosting provider

Service Level Objective: 95% of all transactions should be completed within 500ms



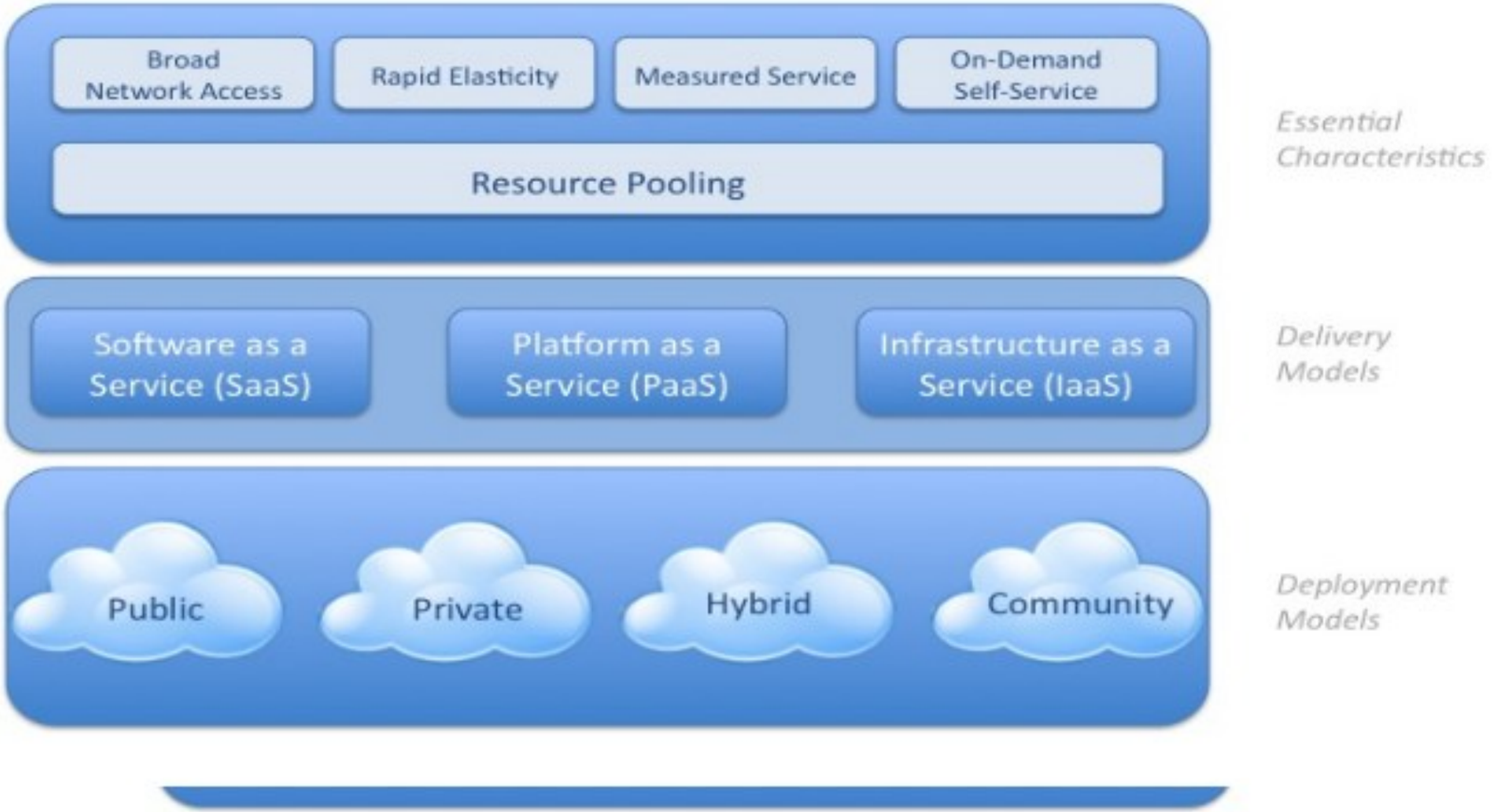
Many tenant applications

Virtualization Technology



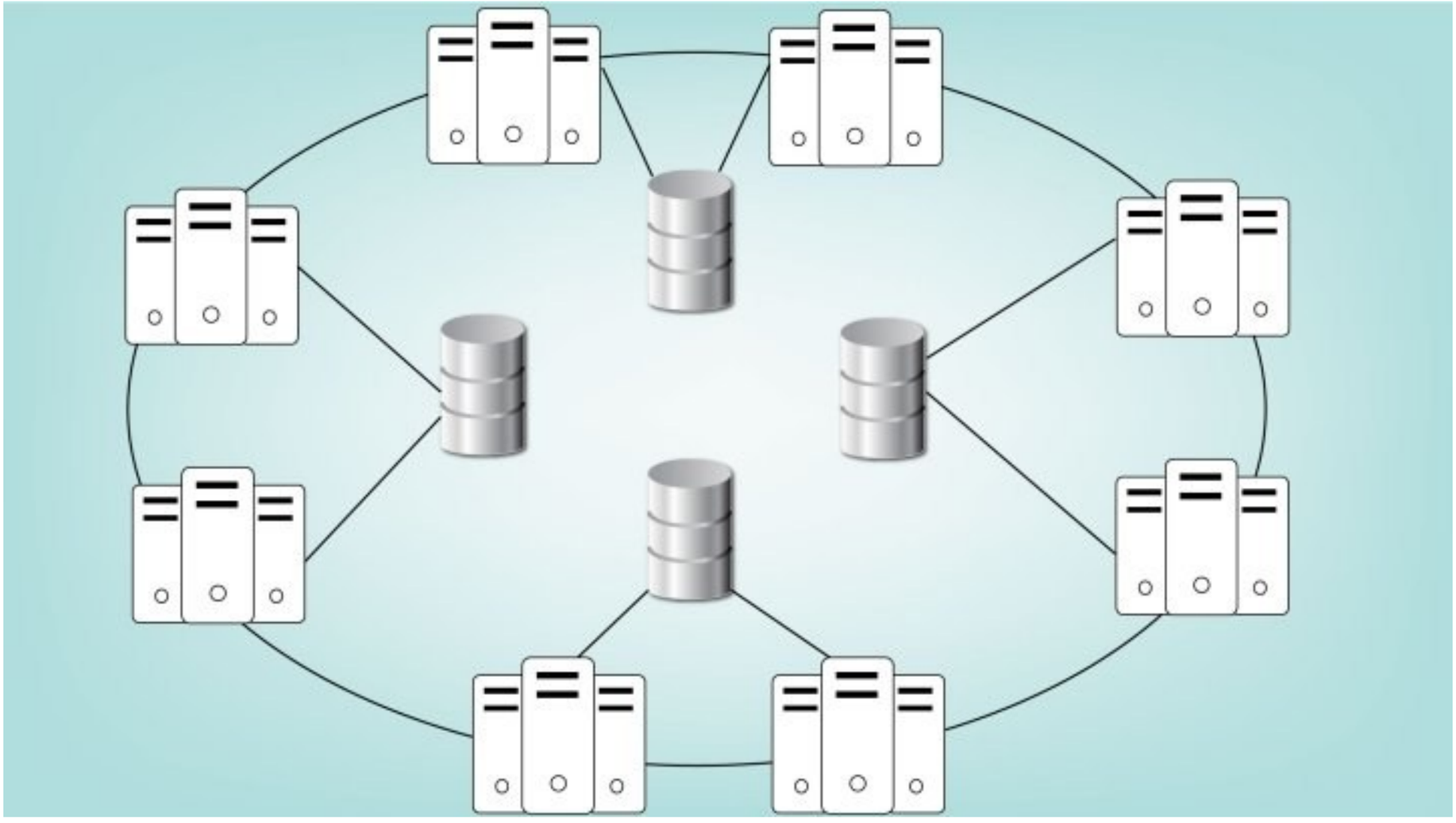
What is cloud computing?

Visual Model Of NIST Working Definition Of Cloud Computing
http://www.csrc.nist.gov/groups/SNS/cloud-computing/index.html

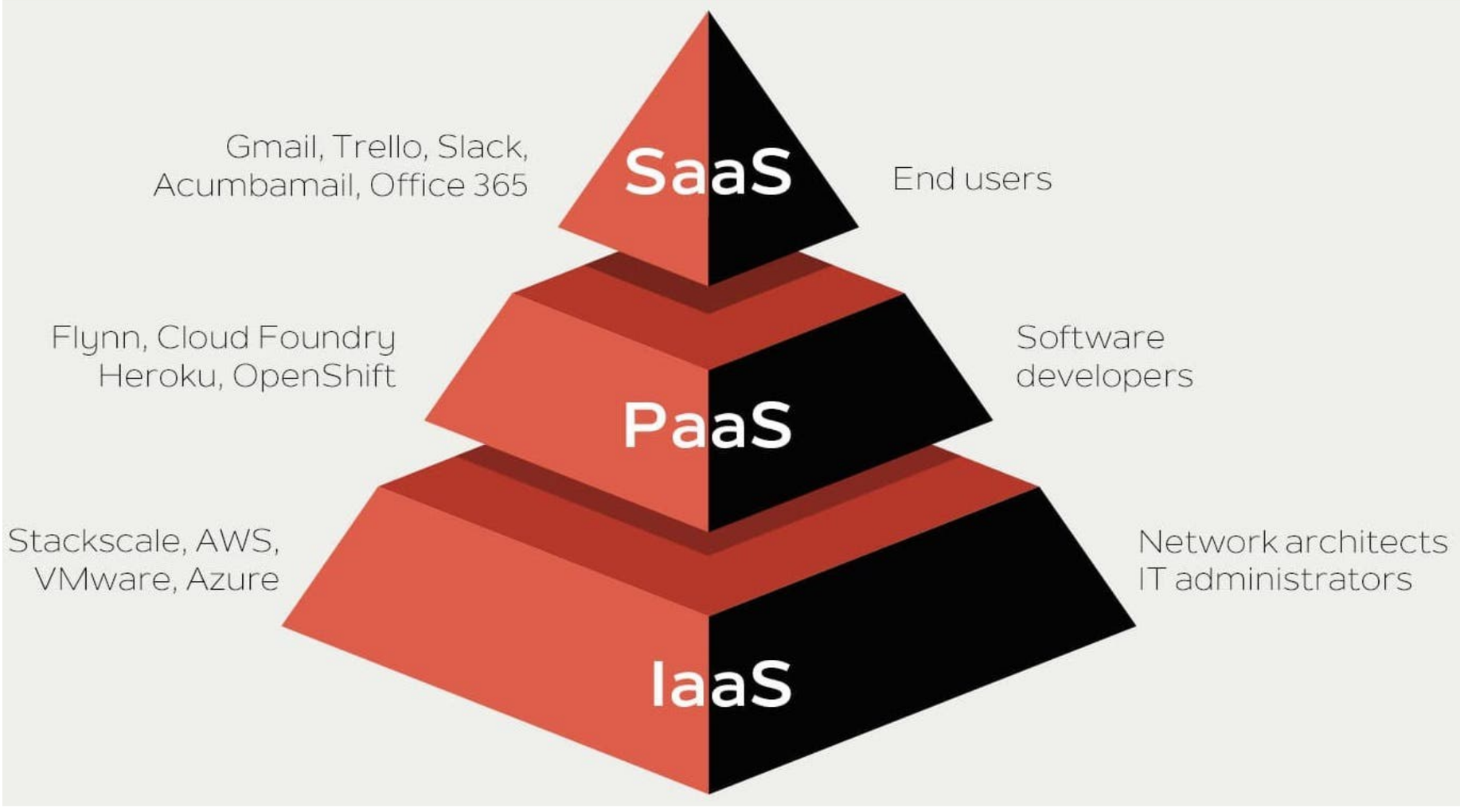


- *Source: NIST definition of cloud computing. NIST special publication 800-145, Sep. 2011.*

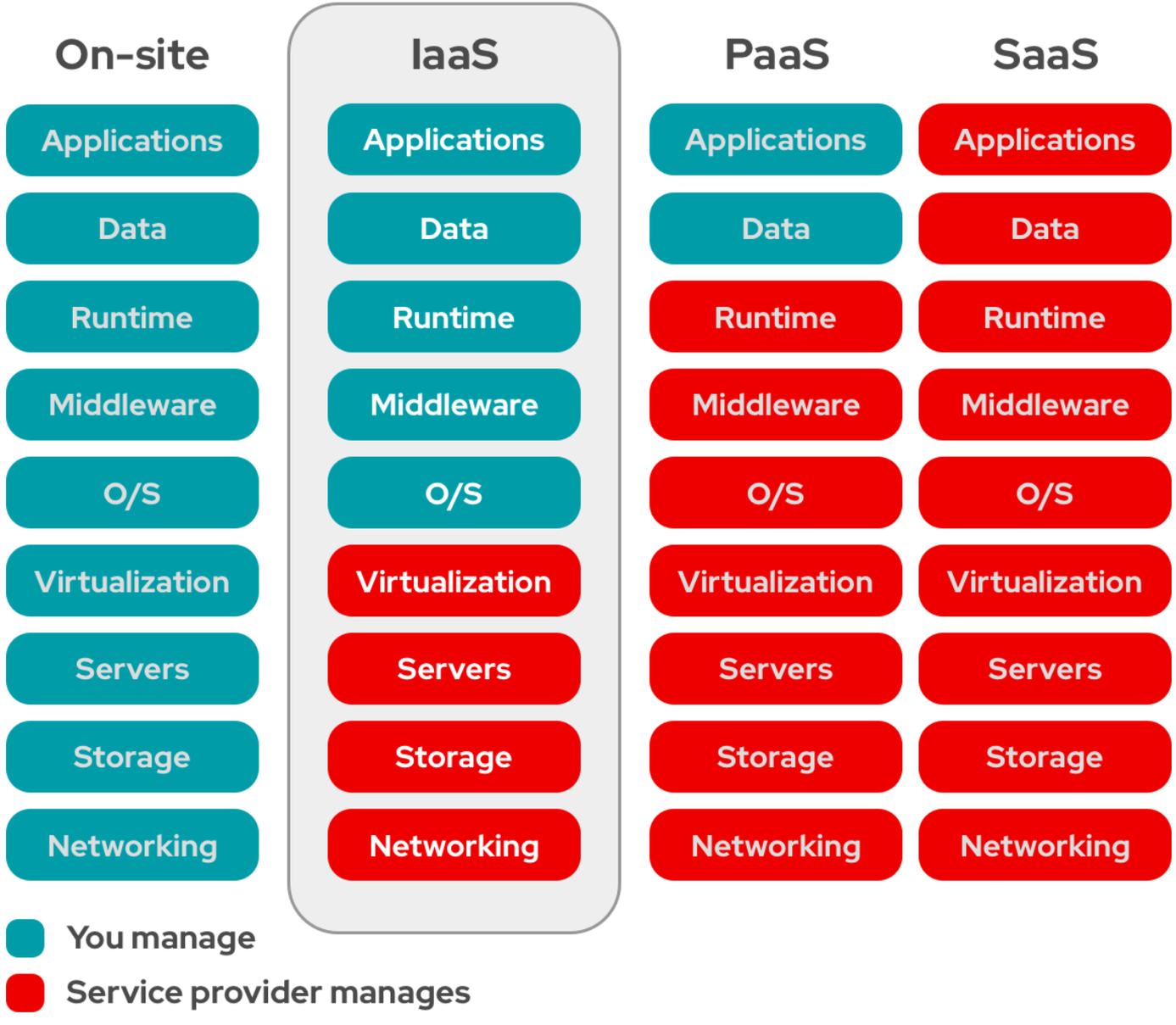
Distributed Systems



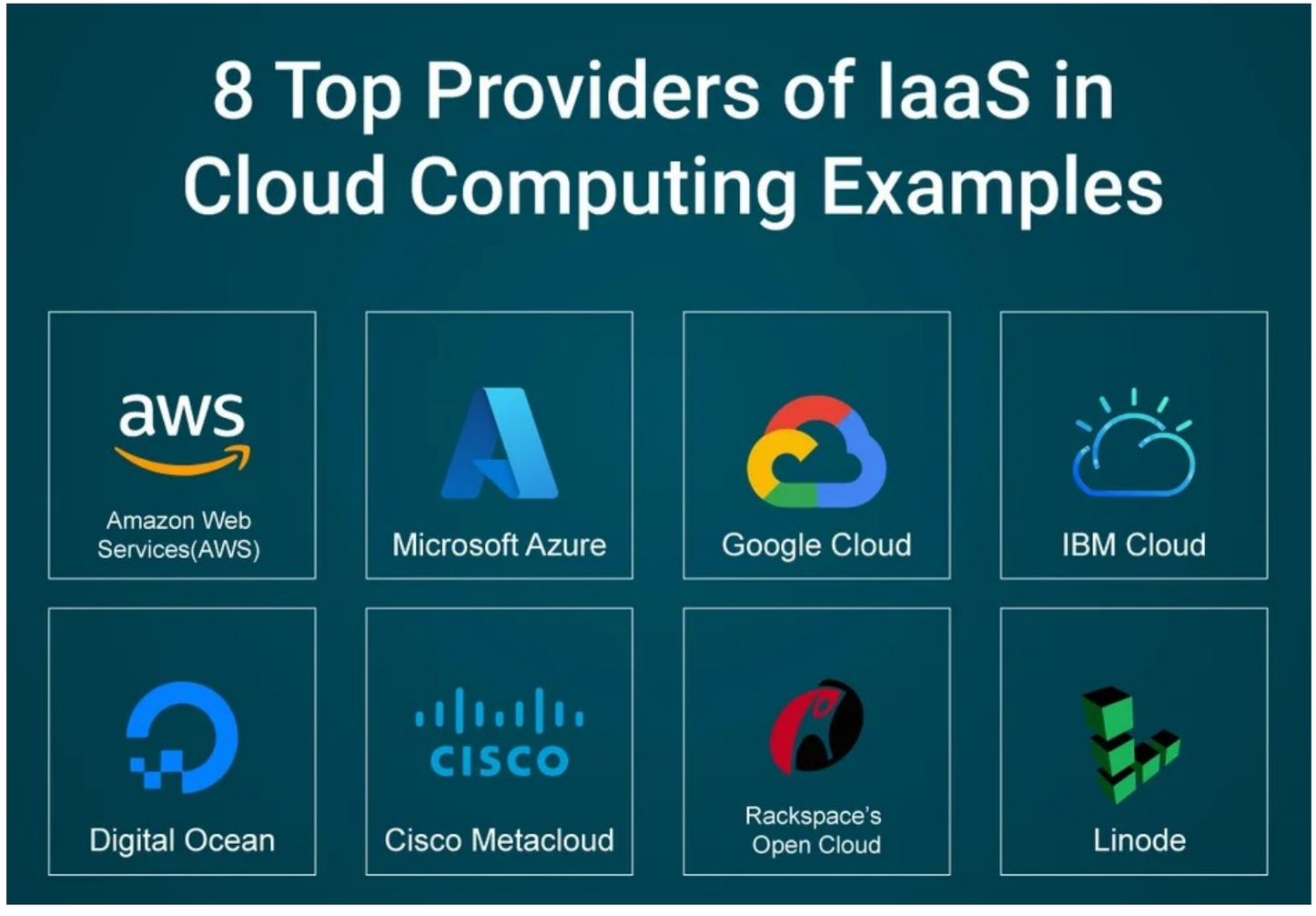
Cloud Service Models



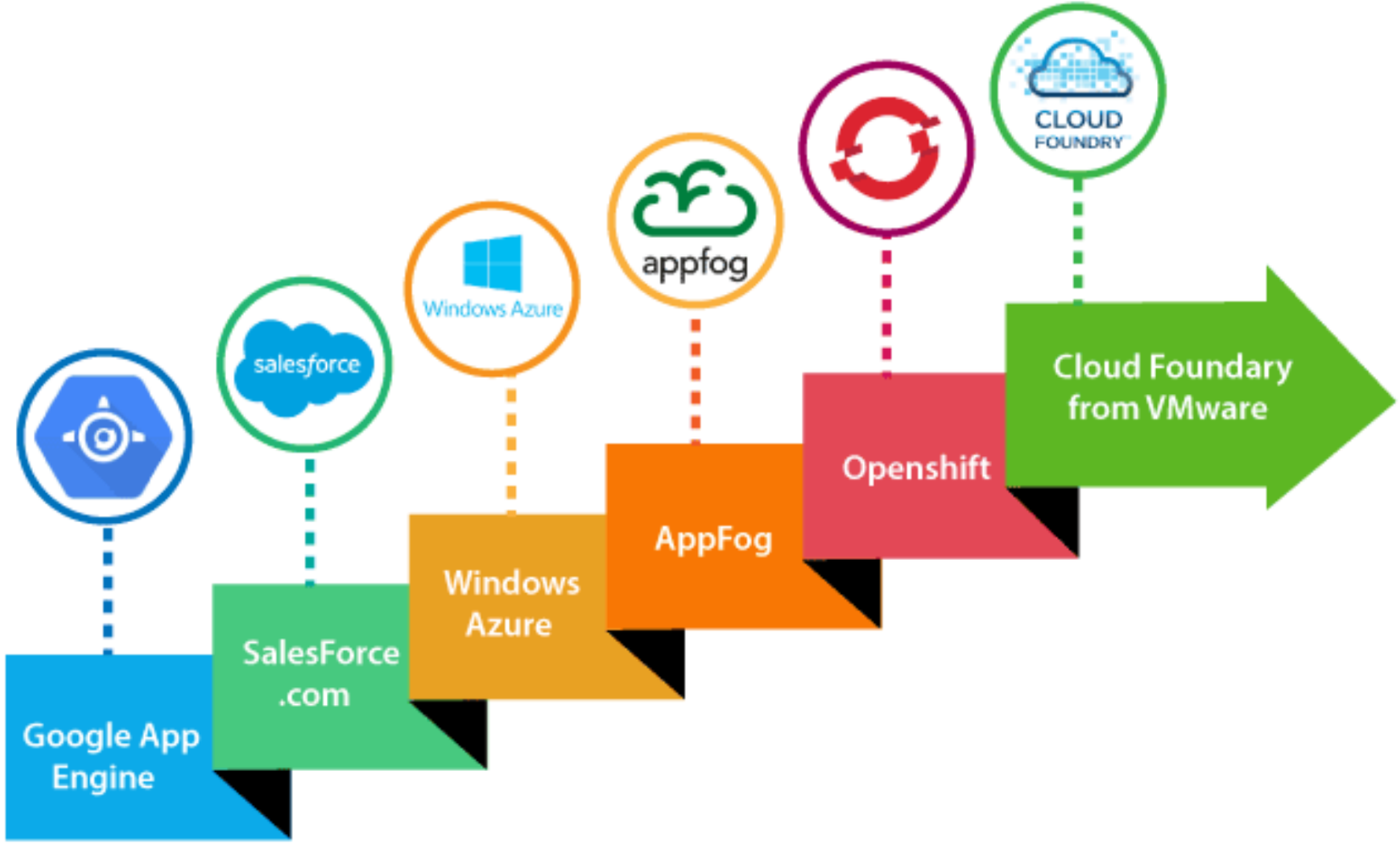
IaaS, PaaS and SaaS



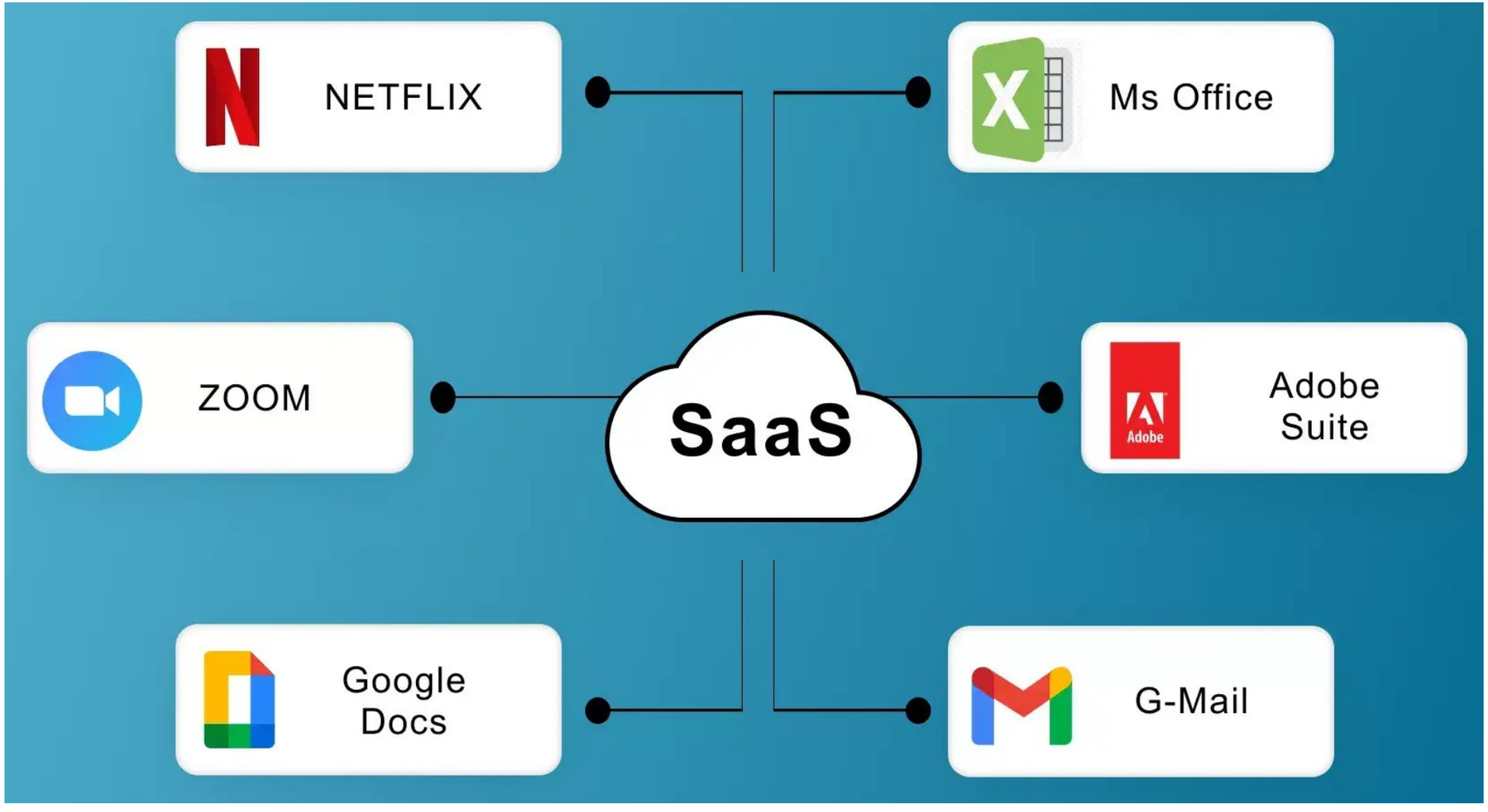
Infrastructure as a Service (IaaS)



Platform as a Service (PaaS)



Software as a Service (SaaS)



Cloud Computing in the Wild

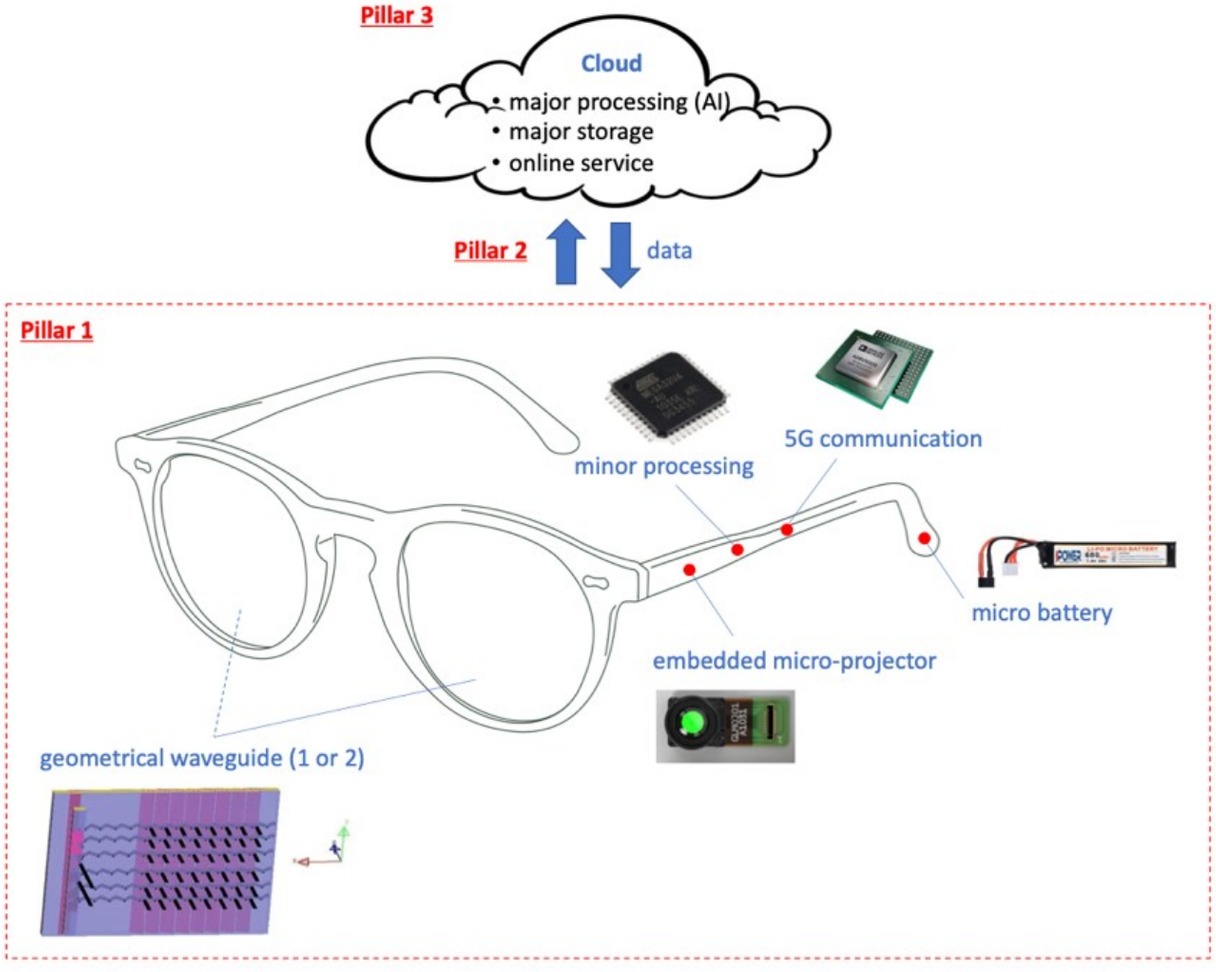
Cloud Gaming

Bitcoin Cloud Mining

Mobile Cloud AR Glass

Mobile Cloud AR Glass

"Extremely Simple Design" AR Glass



Pillars

- Advances in geometrical waveguide technology for AR display
- Low latency 5G communication
- Mobile Cloud Computing

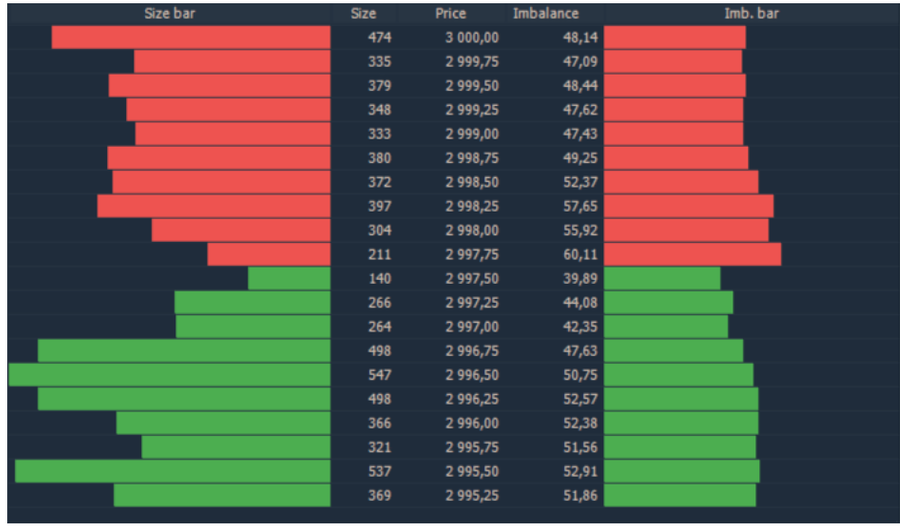
Applications

- Simultaneous Interpretation
- Augmented Reality Gaming
- Augmented Reality Driving Experience

Data Analytics in Cryptocurrency

Price Prediction – A Deep Reinforcement Learning Approach

- Limit Order Book from Coinbase
- Sentiment Analysis (Twitter)
- RL Algorithms: A2C, PPO2, SAC
- Policy Networks: LSTM, MLP



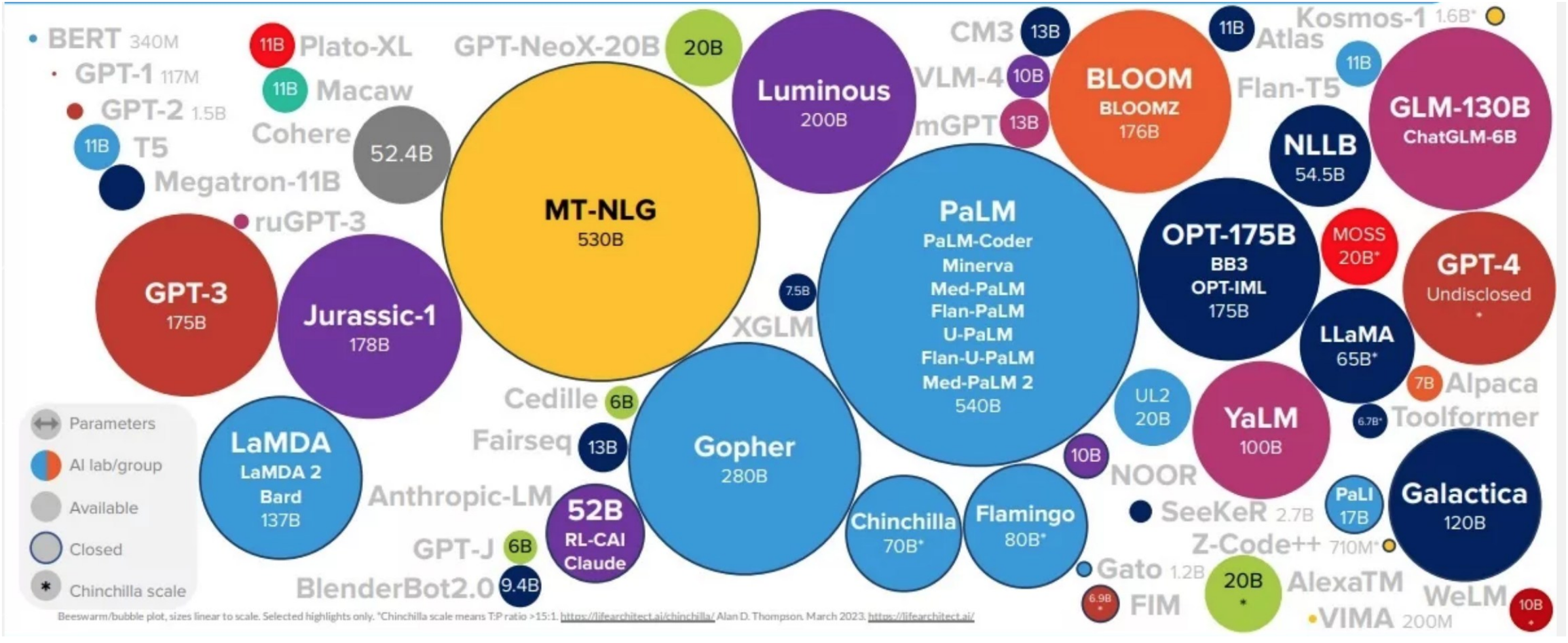
OpenAI

OPTUNA

TensorFlow

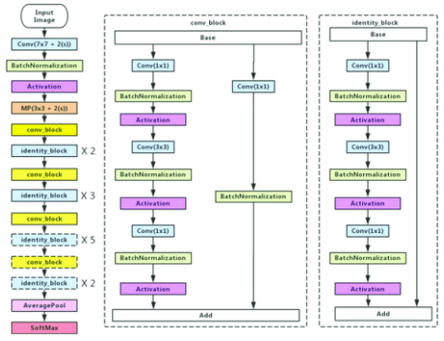
Stable Baselines

Large Language Models

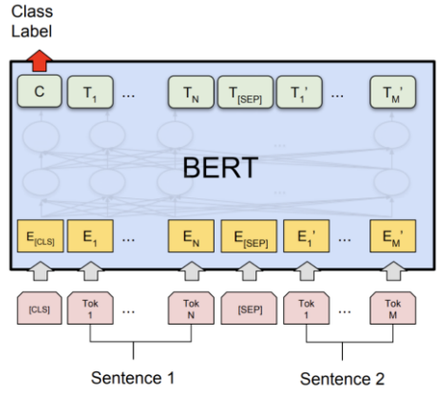


Deep Neural Network (DNN)

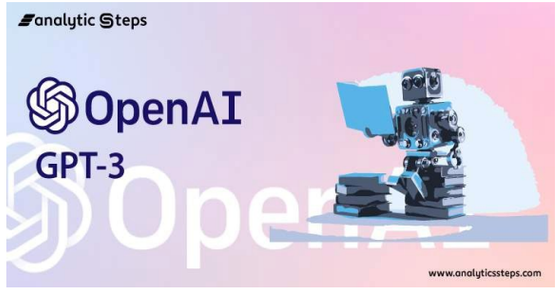
DNN Models



ResNet-50
(26M params)



BERT-Large
(387M params)



GPT-3
(175B params)

DNN Training

DNN model	Device	Computation time (FP + BP)	Estimated #iters. to converge	Estimated training time
ResNet-50	1x Tesla V100	32ms + 64ms (batch size 32)	3.6M	96 hours
BERT-Large	1x Tesla V100	339ms + 508ms (batch size 35)	8M	78.4 days

Need distributed training to scale out!

Virtualization

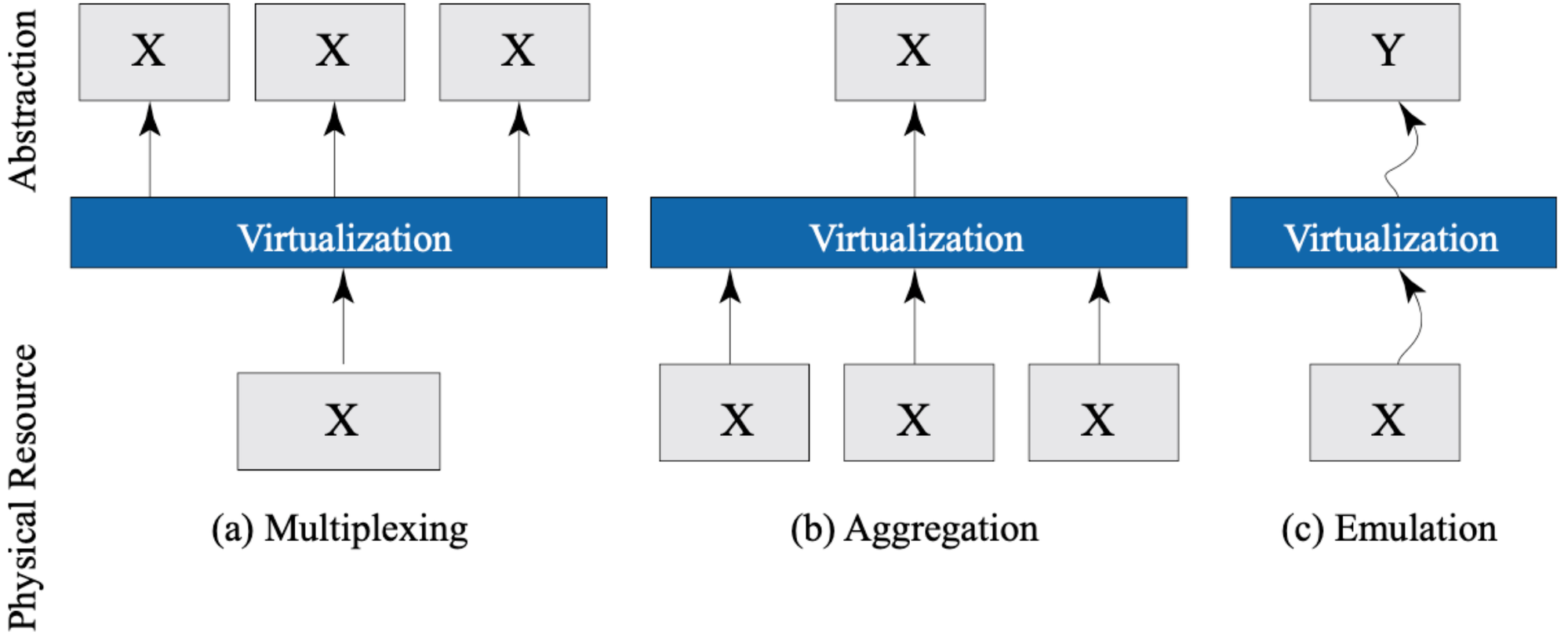
Virtualization – a new computing paradigm

Key benefits

- Higher hardware utilization
- Easier deployment
- Elastic capacity
- Better agility via live migration
- Higher availability
- Fault tolerance
- Lower energy cost

vmware

Virtualization



Three basic implementation techniques of virtualization.
X represents both the physical resource and the virtualized abstraction.

Multiplexing

Multiplexing exposes a resource among multiple virtual entities.

space multiplexing ==> OS multiplexes different pages of physical memory across different address spaces (virtual-to-physical mappings)

time multiplexing ==> schedules threads on the physical cores transparently, thereby multiplexing in software the limited physical CPU resource

Aggregation

Aggregation takes multiple physical resources and makes them appear as a single abstraction

- A RAID [Redundant array of independent disks] controller aggregates multiple disks into a single volume [a single logic unit], ensuring all read/write operations to the volume are appropriately reflected onto the various disks of the RAID group.
- A processor's memory controller aggregates the capacity of multiple DIMM [dual in-line memory module] into a single physical address space.

Emulation

Emulation relies on a level of indirection in SW to expose a virtual resource or device that corresponds to a physical device, even if it is not present in the current computer system.

Apple Rosetta emulates a PowerPC process on an X86 computer

Memory and disks can emulate each other

- The paging process of virtual memory uses disk sectors to emulate virtual memory
- A RAM disk emulates the function of a disk using DRAM as backing store.

Virtual Machine

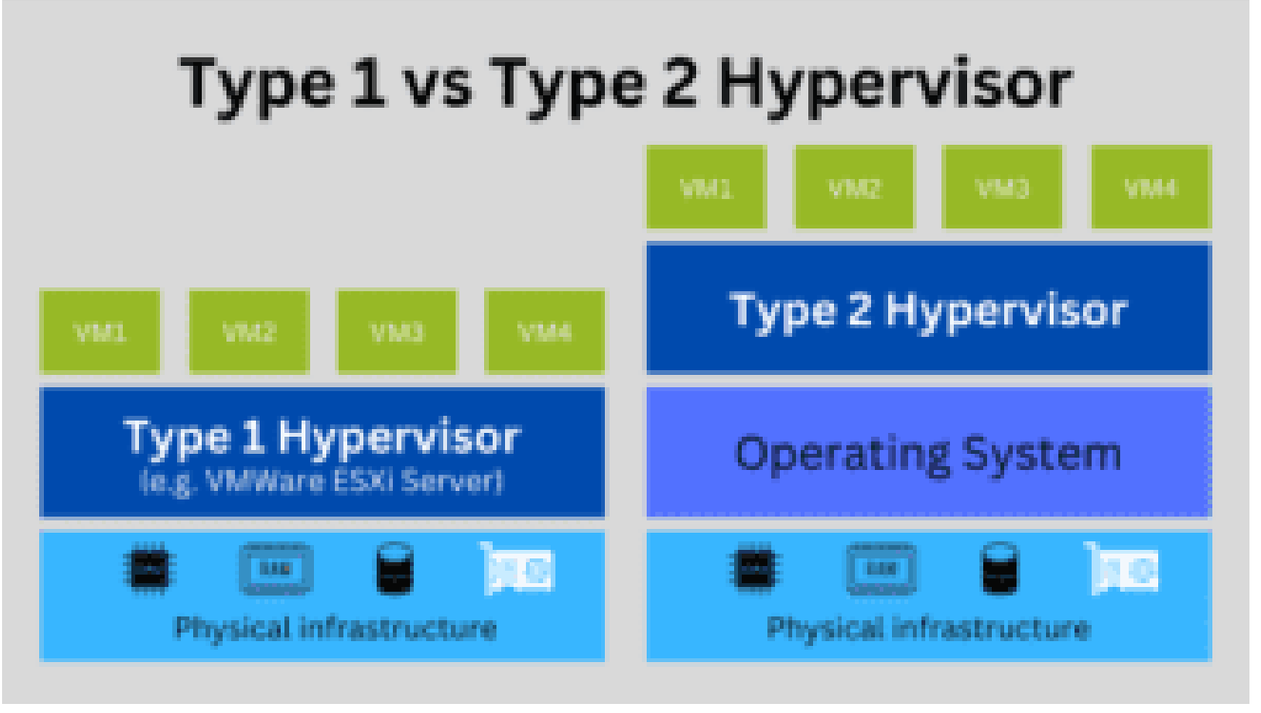
A virtual machine is an abstraction of a complete compute environment through the combined virtualization of the processor, memory and I/O components of a computer.
A virtual machine is a complete compute environment with its own isolated processing capabilities, memory, and communication channels.

Hypervisor

The hypervisor is a specialized piece of system software that manages and runs virtual machines.

- Type I hypervisor (bare-metal hypervisor, native hypervisor) is a virtualization platform that runs directly on the hardware of a physical computer
- Type II hypervisor (hosted hypervisors, desktop hypervisors) is a virtualization platform that runs on top of host OS (operates as applications within a standard OS)

Type 1 vs Type Hypervisor



Type I hypervisor

Operates without the need for a host OS
Provides a virtualization layer that allows multiple guest OS to run on the same HW simultaneously.
Examples:

- Vmware ESXi Server
- Xen
- KVM (Kernel-based virtual machine)
- Microsoft Hyper-V (when installed on bare metal, not as a role on the Windows server)

Type II hypervisor

Share resources with the host (e.g., CPU, memory and storage)

Examples:

- Oracle VirtualBox
- VMware Workstation
- VMware player
- MS Hyper-V (when installed as a feature on a Windows Desktop OS)
- Parallels Desktop and Vmware fusion @MAC

Light Weight Virtualization: Container

Why Linux Containers?

What are we trying to solve?

docker

Many payloads

- backend services (API)
- databases
- webapps

docker

Many payloads

- Go
- Java
- Node.js
- PHP
- Python
- Ruby
- …

docker

Many targets

- your local development environment
- your coworkers' developement environment
- your Q&A team's test environment
- some random demo/test server
- the staging server(s)
- the production server(s)
- bare metal
- virtual machines
- shared hosting

docker

Many targets

- BSD
- Linux
- OS X
- Windows

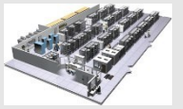
docker

The Matrix From Hell

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?







Real-world analogy: containers

docker

Many products

- clothes
- electronics
- raw materials
- produce
- …

docker

Many transportation methods

- ships
- trains
- trucks
- ...

docker

Problem: shipping code

	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
						

Solution:
the *Linux* container

SCaLE 12x

docker

Solved!

SCALE 12x

docker

Separation of concerns: Dave the Developer

- inside my container:
  - my code
  - my libraries
  - my package manager
  - my app
  - my data

Separation of concerns: Oscar the Ops guy

- outside the container:
  - logging
  - remote access
  - network configuration
  - monitoring

Linux containers...

- run everywhere
  - regardless of kernel version
  - regardless of host distro
  - (but container and host architecture must match)
- run anything
  - if it can run on the host, it can run in the container
  - i.e., if it can run on a Linux kernel, it can run

High level approach: it's a lightweight VM

- own process space
- own network interface
- can run stuff as root
- can have its own /sbin/init (different from the host)

docker

Docker

HOW POPULAR IS DOCKER?

Accelerate how you build, share, and run modern applications.

13 million +
developers

7 million +
applications

13 billion +
monthly image downloads

What is docker

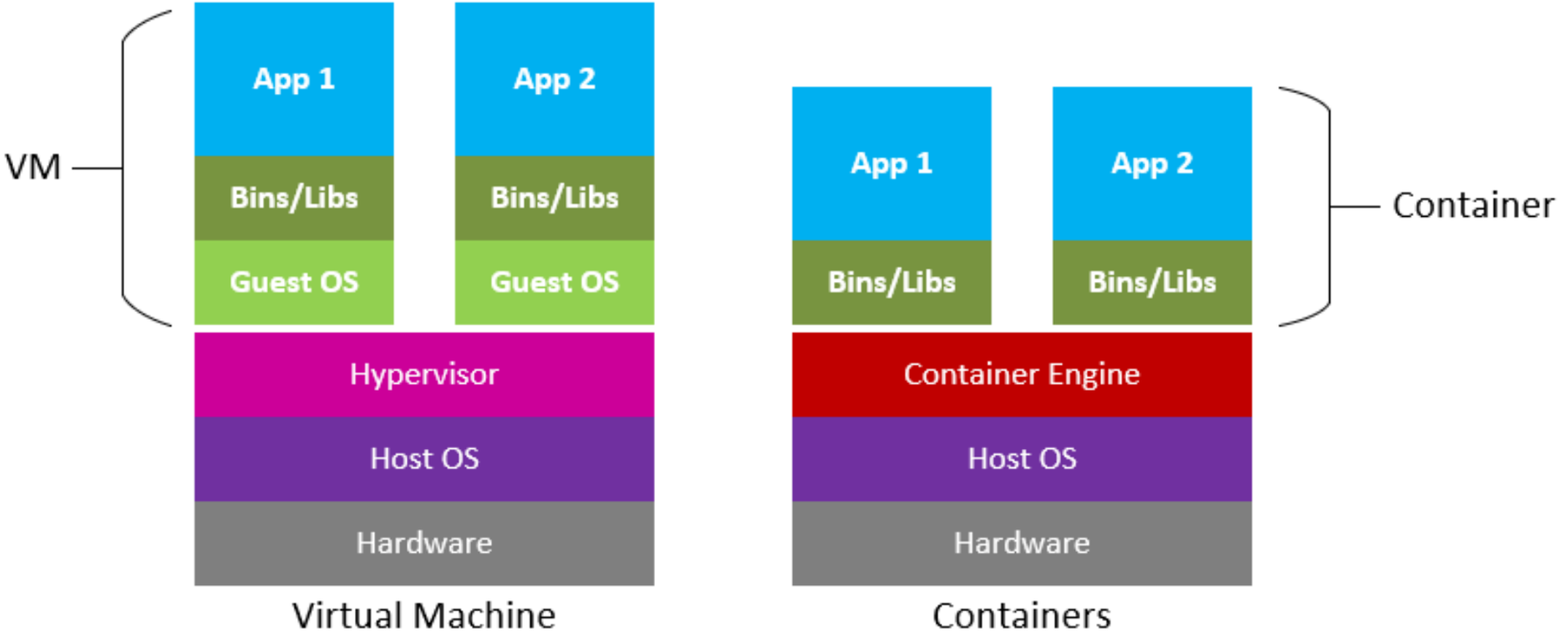
- Docker is an open-source project designed for creating, deploying, and running applications by using containers. These containers are portable, self-sufficient and can be run on any infrastructure in the cloud or on-premises on which Docker is installed/supported
- It automates the deployment of software applications inside **containers** by providing an additional layer of abstraction and automation of **OS-level virtualization** on Linux.

Resource isolation

Implemented by a number of Linux APIs:

- **cgroups:** Restrict resources a process can consume
  - CPU, memory, disk IO, ...
- **namespaces:** Change a process´s view of the system
  - Network interfaces, PIDs, users, mounts, ...
- **capabilities:** Limits what a user can do
  - mount, kill, chown, ...
- **chroots:** Determines what parts of the filesystem a user can see

Container vs. VM



Container Image

- An **image** is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files.

Container

- A **container** is a runtime instance of an image—what the image becomes in memory when actually executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.
- Containers run apps natively on the host machine's kernel.
  - Containers can get native access to host resources, each one running in a discrete process.
  - Better performance characteristics than virtual machines that only get virtual access through a hypervisor.

Container Engine

- **Container Engine** effectively virtualizes the host operating system (or kernel) and isolates an application's dependencies from other containers running on the same machine.

How does Docker work ?

- You can build Docker images that hold your applications
- You can create Docker containers from those Docker images to run your applications.
- You can share those Docker images via Docker Hub or your own registry

Process for building Docker Images

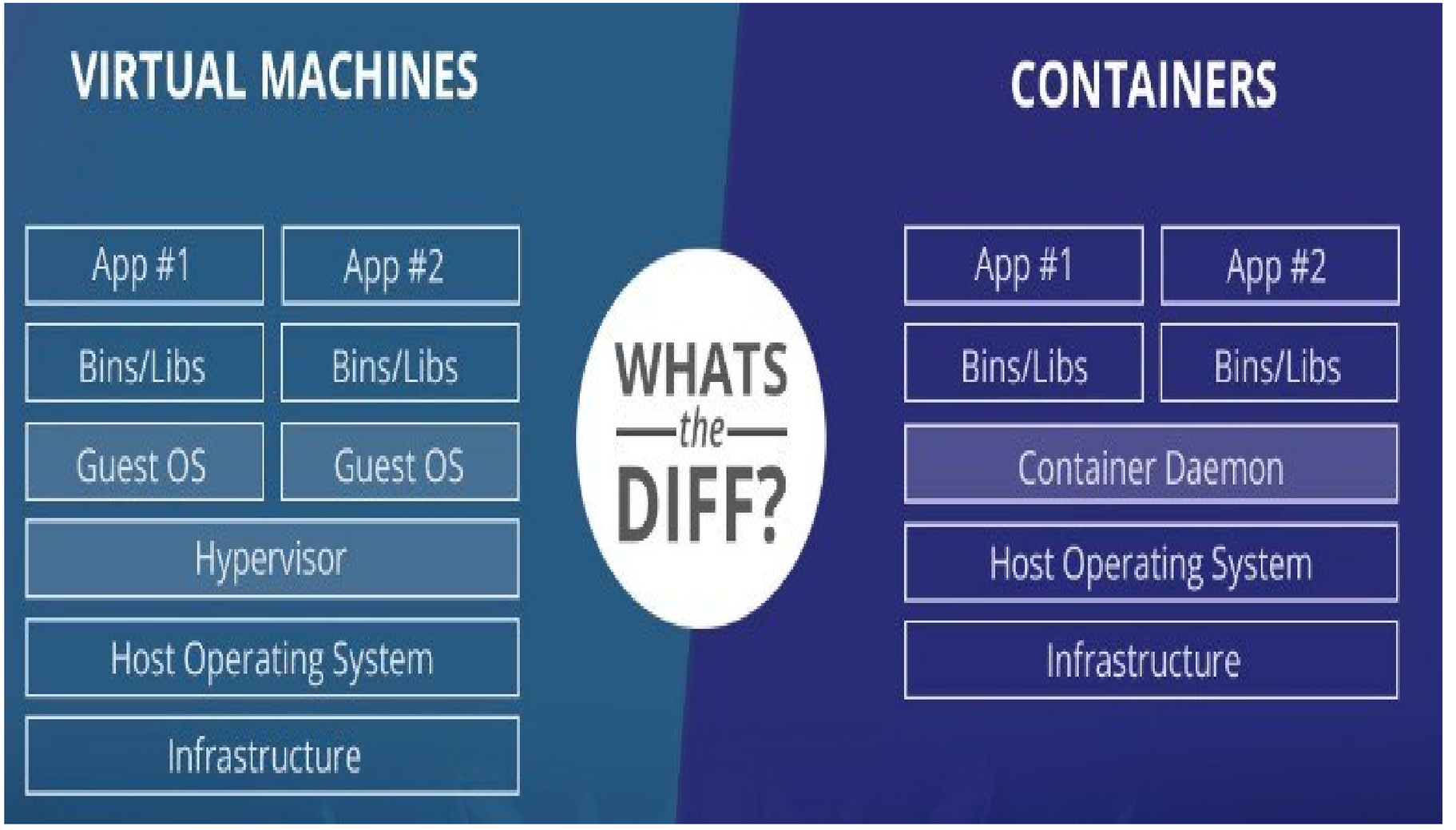
1. Install Docker
2. Write a Dockerfile
    + **Dockerfile** is a text file that contains instructions, using which, Docker can build images automatically
    + Name of the file: Dockerfile
3. Build the Docker image and tag it
    + A file containing the snapshot of a container is known as a **Docker image**
    + It is created using the build command, and produces a container when it starts running
    + You can add tags to the images during the build step or while saving it to a repository - the default tag is “latest”
4. Run the image that you built
5. Register on DockerHub - use the credentials to push the image
    + Images are stored in a Docker registry such as registry.hub.docker.com
    + Pull Images later on any system that you want to run the application on

How can containers be created from docker images

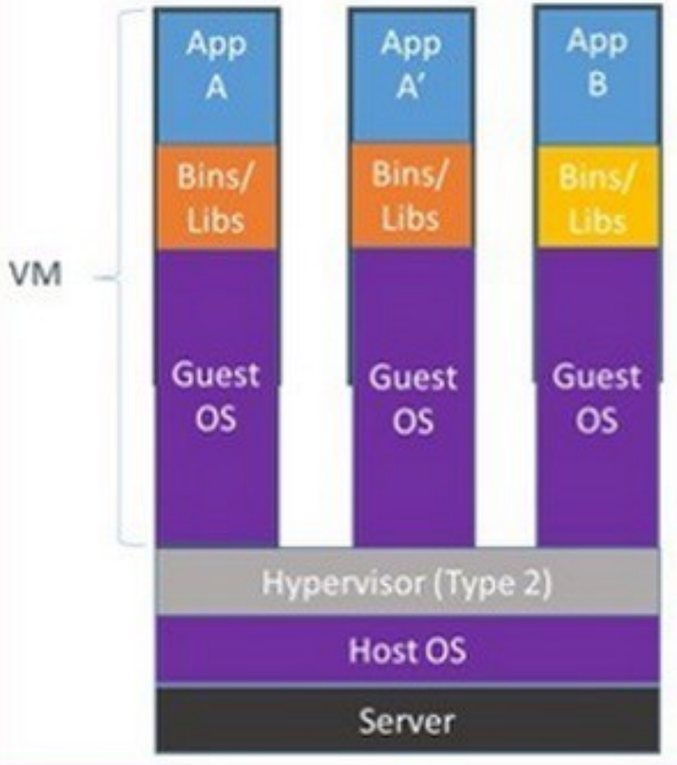
- You can docker commands such as “docker pull” or “docker run”
  - “Pull” will always fetch the latest version of the image from Docker Hub before running
  - “Run” will search for an image locally and run it, and if there is nothing available locally, it will go to Docker Hub.

VM vs. Container

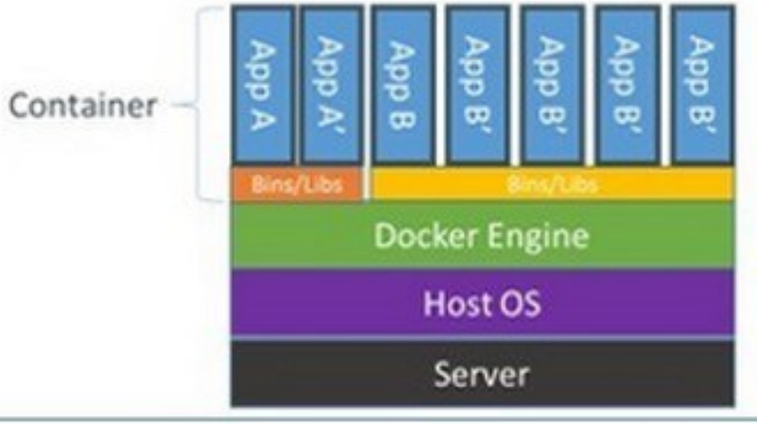
- Virtual machines (VMs) emulate server hardware, on which you can install a “guest” operating system and your applications.
- Containers virtualize the OS [not the full machine] and allow you to run many images using a common OS.
- **Container Images** package individual applications and their dependencies into portable artifacts that can be run on any compatible operating system.


VIRTUAL MACHINES
App #1
App #2
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Hypervisor
Host Operating System
Infrastructure
WHATS
the
DIFF?
CONTAINERS
App #1
App #2
Bins/Libs
Bins/Libs
Container Daemon
Host Operating System
Infrastructure

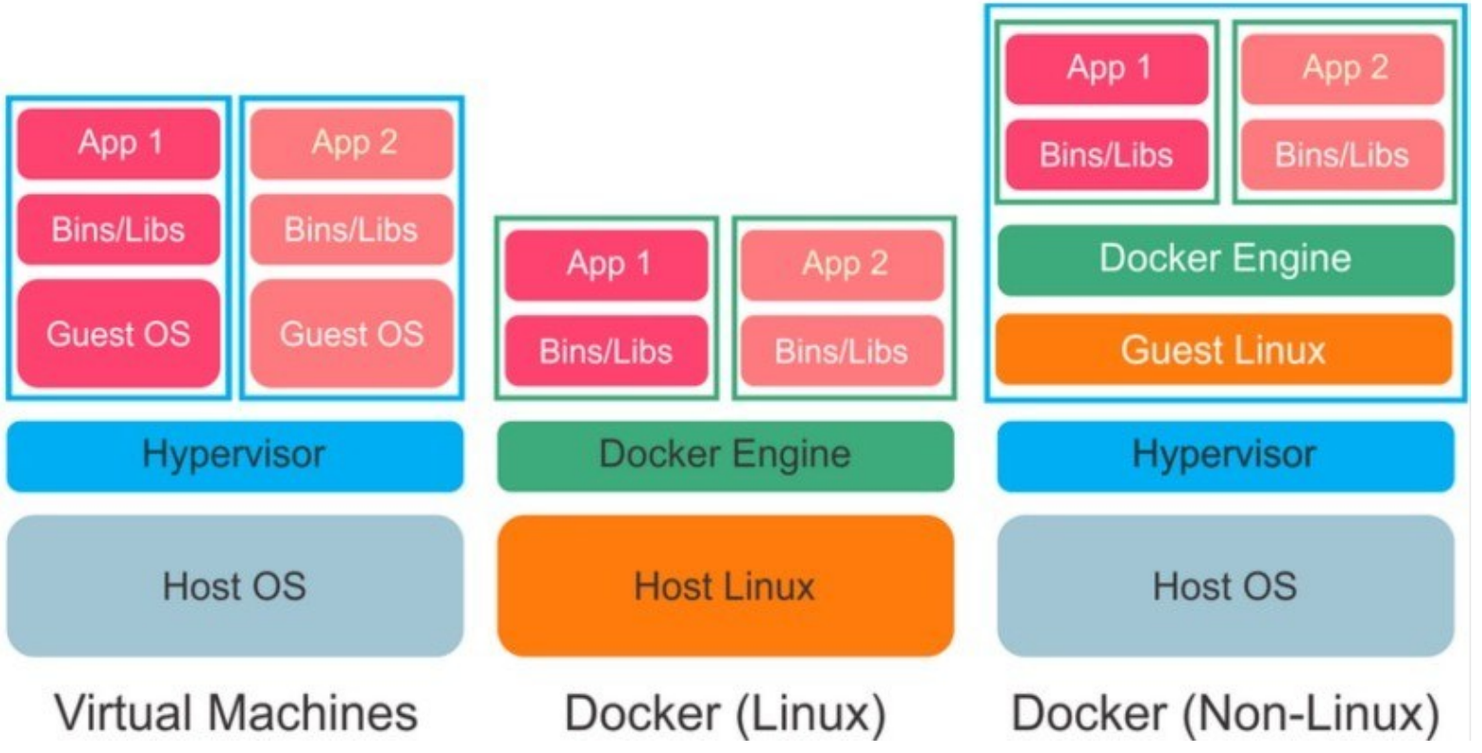
Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries



Container vs. VM

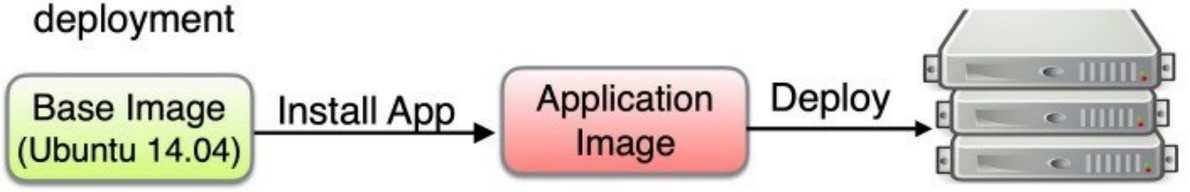


What's the Diff: VMs vs Containers

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

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)

- Docker effectively provides isolation of one group of processes from other files and system processes without the expense of running another operating system.
- Docker allows a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Summary

- Many systems are being built using docker container technology.
- It provides light weight virtualization and resultant portability and performance.
- It makes deployment of applications very easy to manage.

Understanding docker through examples

Docker commands

- Ensure docker has been successfully installed
  - docker run –it hello-world
- See locally available images
  - docker images
  - docker image ls
- List all docker containers (running and stopped)
  - docker container ls -a
  - docker ps -a
- start a container using name/[docker ID]
  - docker container start -i “unique random name”
  - docker container start -i ”docker ID”