VM, Container, Docker, Kubernetes, LXC

Everything is damn agile right now.

What is Virtual Machine?

Is an emulation of a computer system.

A Virtual of machine inside an actual machine.

Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination.

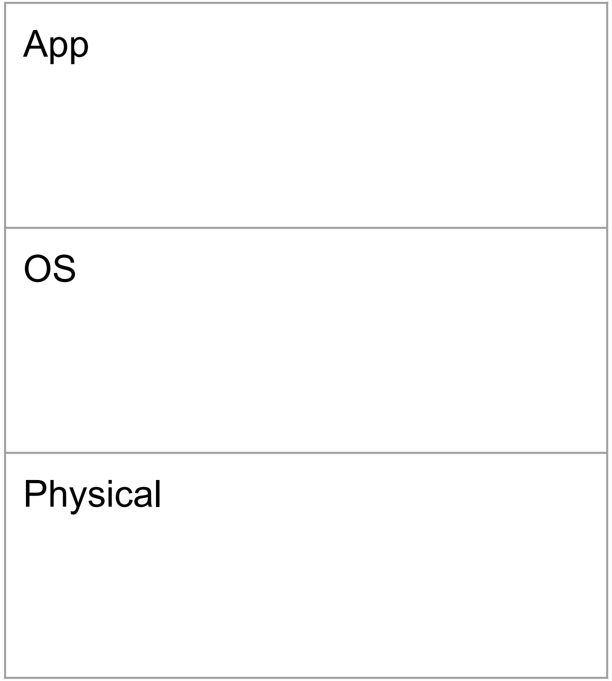
What is Container?

A virtual environment of an Operating system or application inside a host.

It’s isolation levels (the containers) can be used to become a sandbox of specific apps or to emulate an entire new multiple fresh hosts inside that host.

The containers depend on the actual physical hardware with our host. (shared operating system)

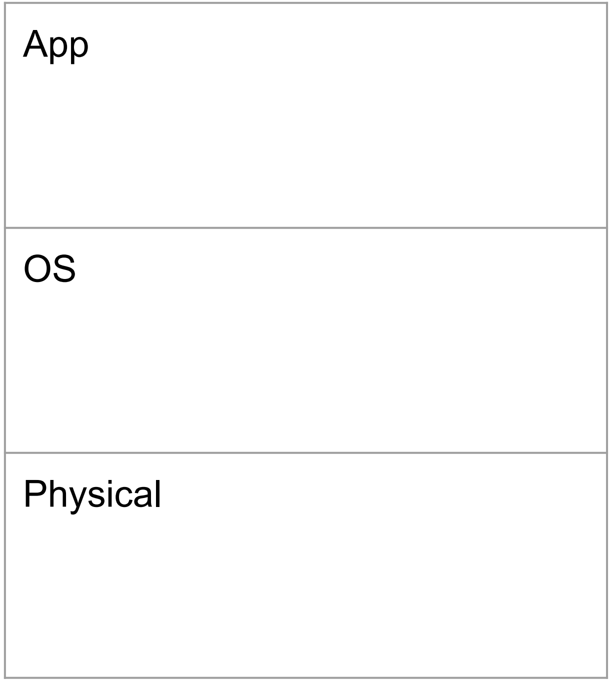
VM vs VE



App

OS

Physical

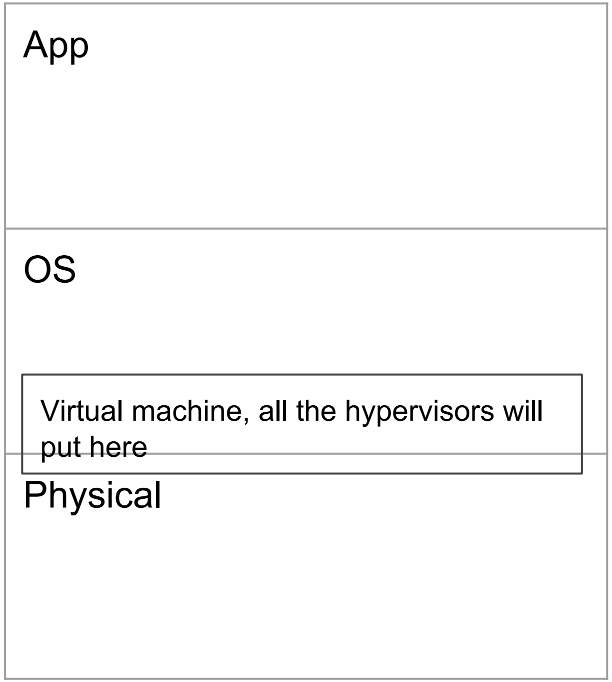


App

OS

Physical

VM vs VE

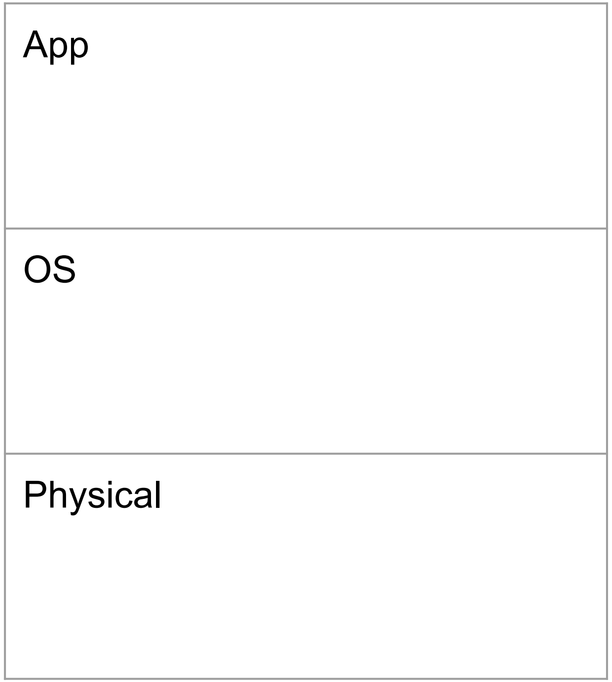


App

OS

Virtual machine, all the hypervisors will put here

Physical

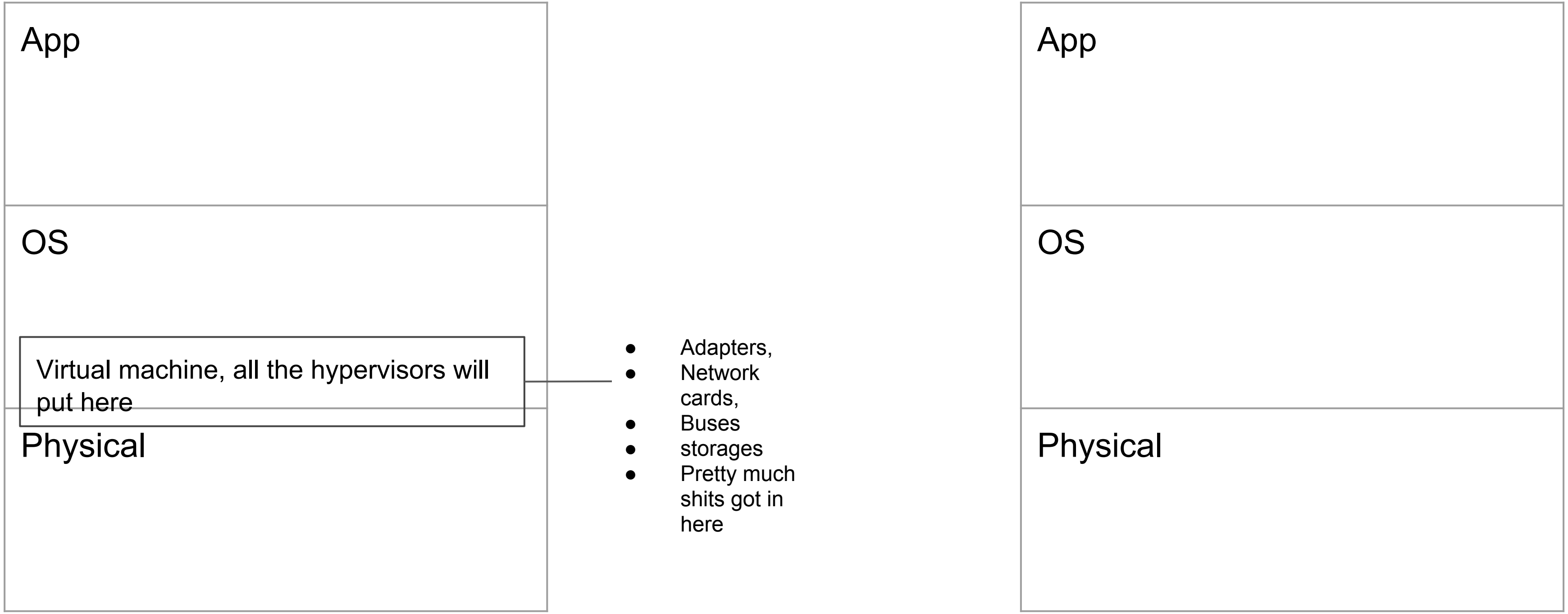


App

OS

Physical

VM vs VE



App

OS

Virtual machine, all the hypervisors will put here

Physical

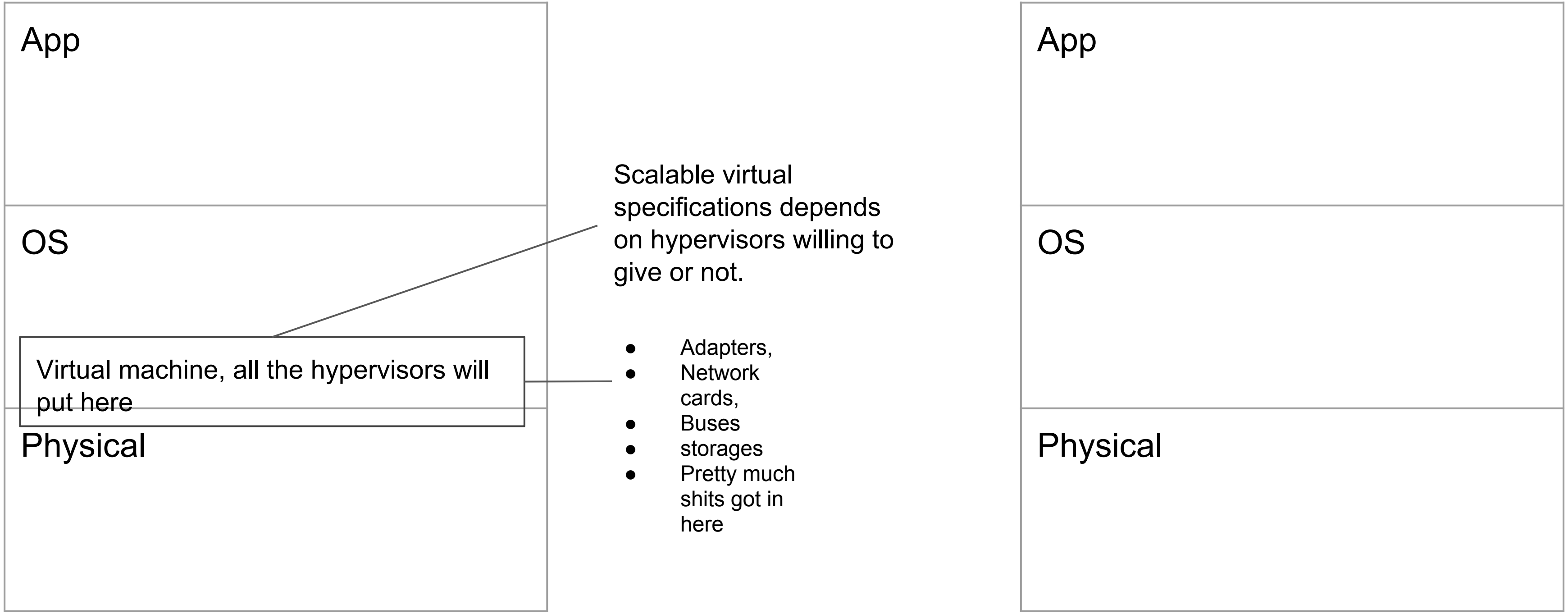
- Adapters,
- Network cards,
- Buses
- storages
- Pretty much shits got in here

App

OS

Physical

VM vs VE



App

OS

Virtual machine, all the hypervisors will put here

Physical

Scalable virtual specifications depends on hypervisors willing to give or not.

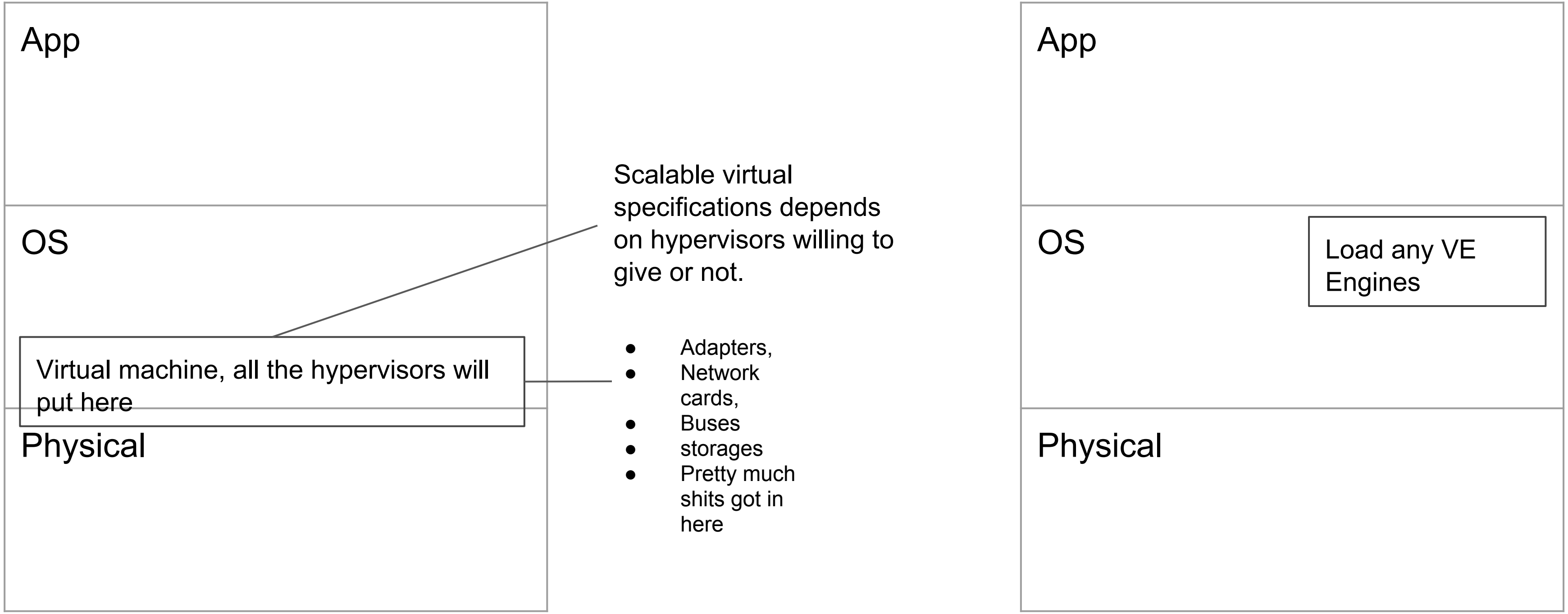
- Adapters,
- Network cards,
- Buses
- storages
- Pretty much shits got in here

App

OS

Physical

VM vs VE



App

OS

Virtual machine, all the hypervisors will put here

Physical

Scalable virtual specifications depends on hypervisors willing to give or not.

- Adapters,
- Network cards,
- Buses
- storages
- Pretty much shits got in here

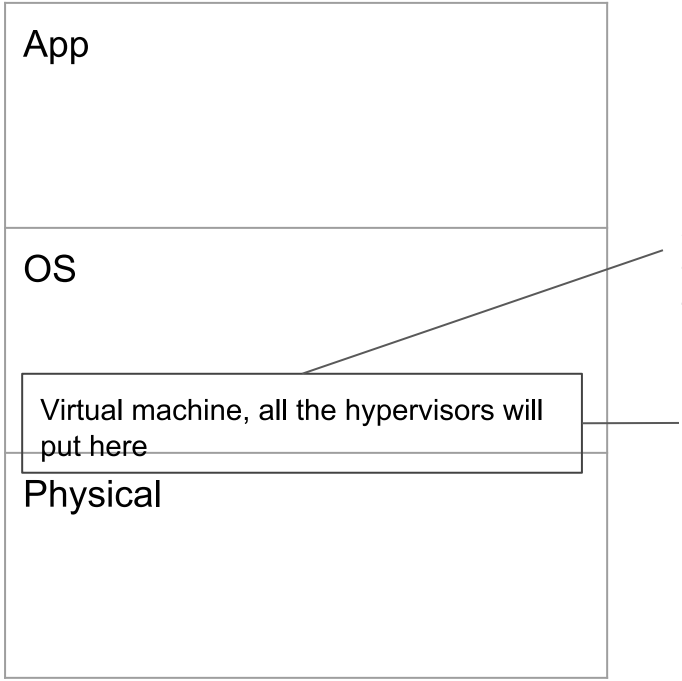
App

OS

Load any VE Engines

Physical

VM vs VE



App

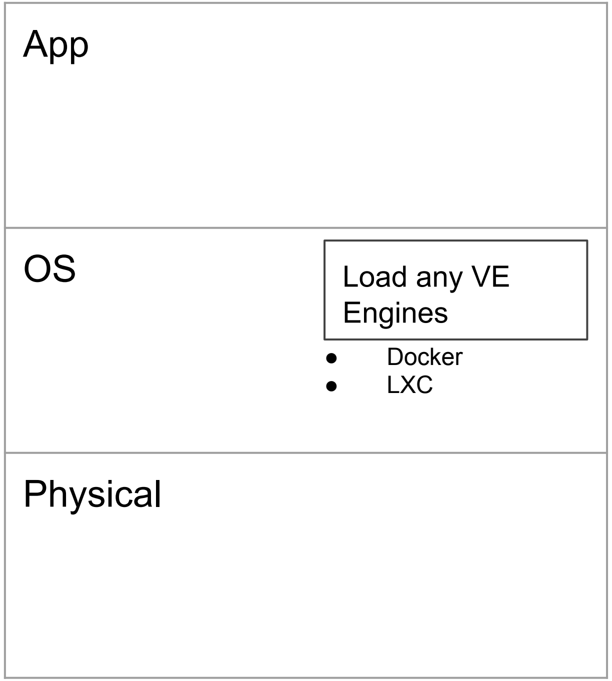
OS

Virtual machine, all the hypervisors will put here

Physical

Scalable virtual specifications depends on hypervisors willing to give or not.

- Adapters,
- Network cards,
- Buses
- storages
- Pretty much shits got in here



App

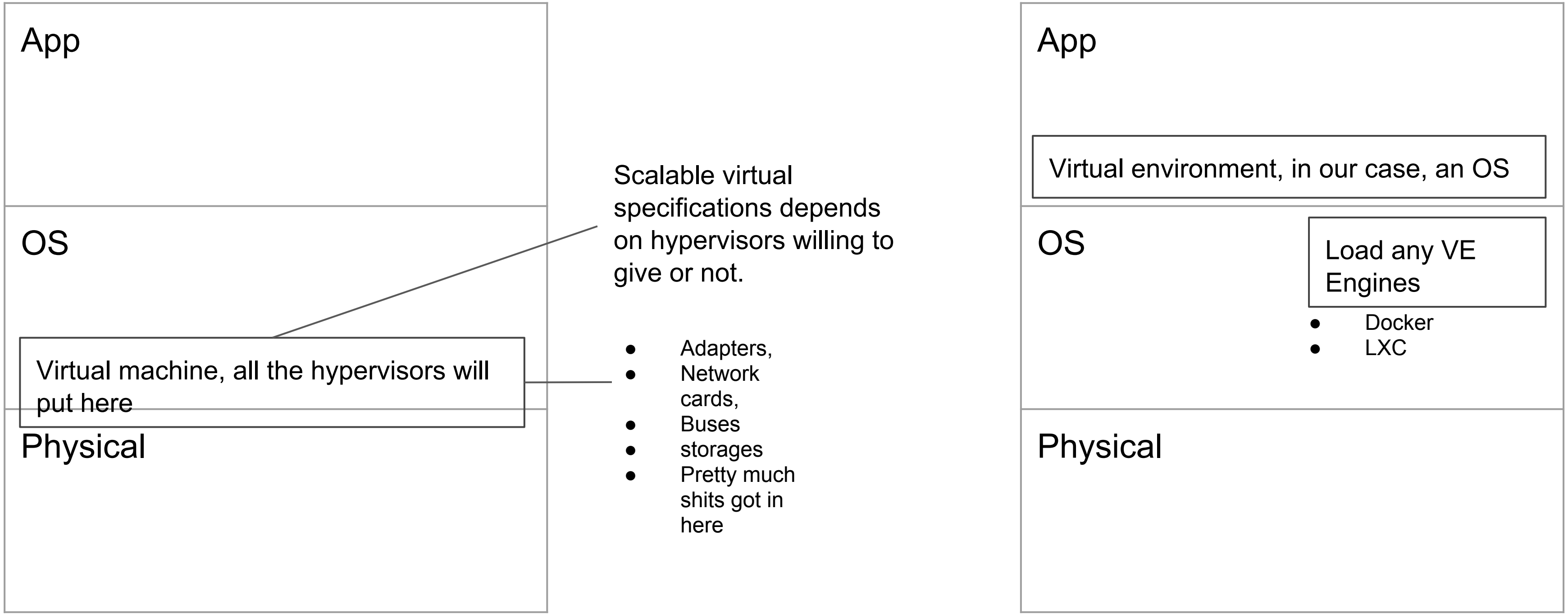
OS

Load any VE Engines

- Docker
- LXC

Physical

VM vs VE



App

OS

Virtual machine, all the hypervisors will put here

Physical

Scalable virtual specifications depends on hypervisors willing to give or not.

- Adapters,
- Network cards,
- Buses
- storages
- Pretty much shits got in here

App

Virtual environment, in our case, an OS

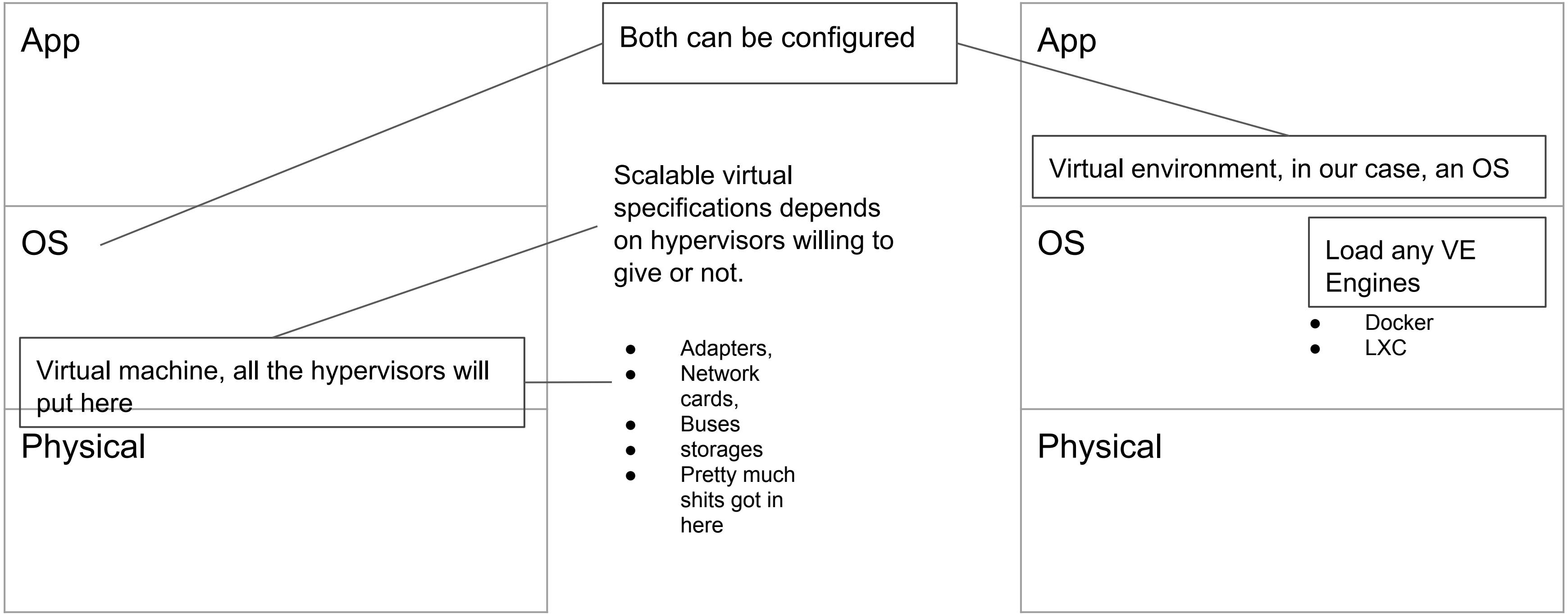
OS

Load any VE Engines

- Docker
- LXC

Physical

VM vs VE



App

OS

Physical

Both can be configured

Virtual machine, all the hypervisors will put here

Scalable virtual specifications depends on hypervisors willing to give or not.

- Adapters,
- Network cards,
- Buses
- storages
- Pretty much shits got in here

App

Virtual environment, in our case, an OS

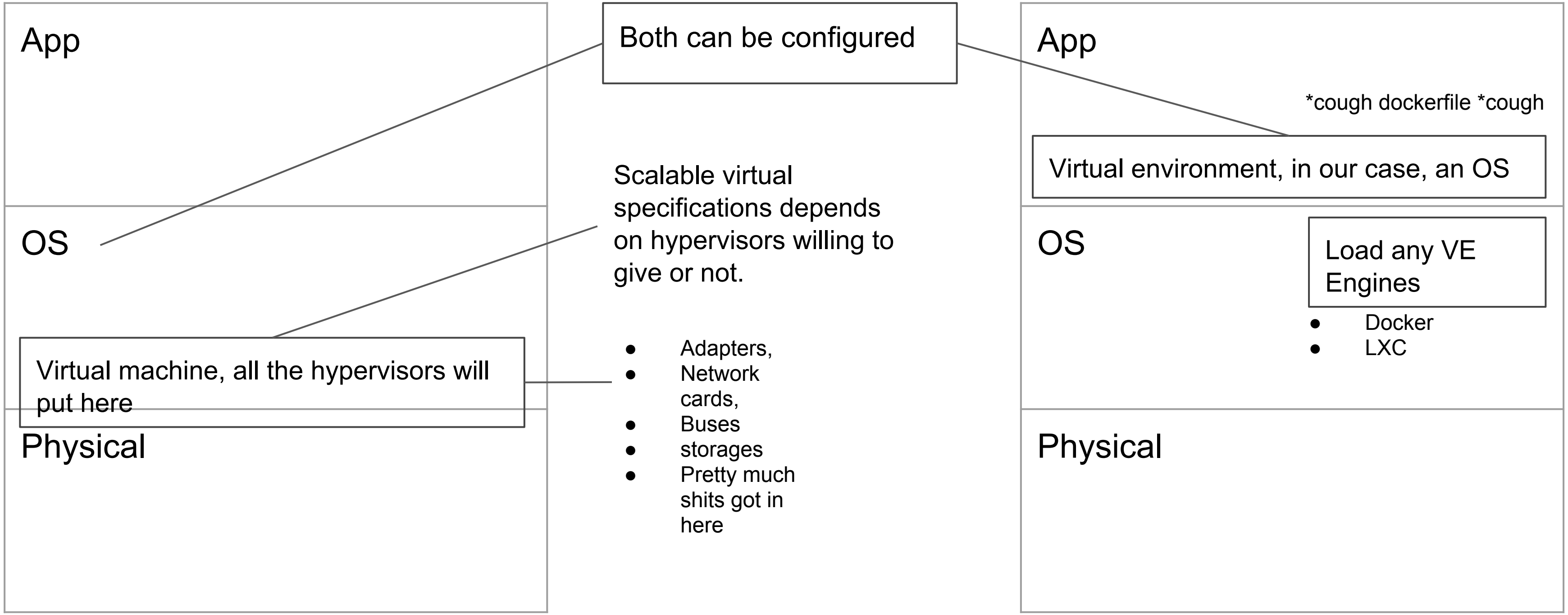
OS

Load any VE Engines

- Docker
- LXC

Physical

VM vs VE



App

OS

Virtual machine, all the hypervisors will put here

Physical

Both can be configured

Scalable virtual specifications depends on hypervisors willing to give or not.

- Adapters,
- Network cards,
- Buses
- storages
- Pretty much shits got in here

App

*cough dockerfile *cough

Virtual environment, in our case, an OS

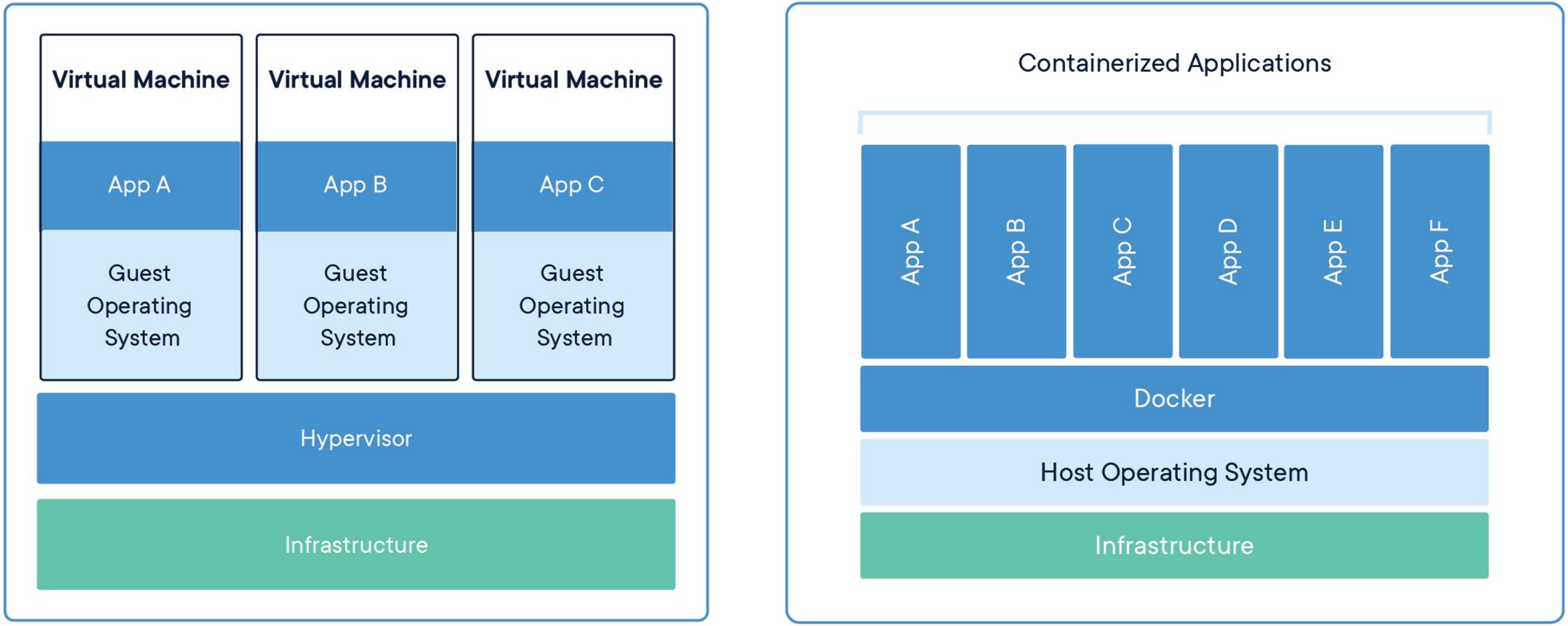
OS

Load any VE Engines

- Docker
- LXC

Physical

Summary of it with beautifully,



LXC and Docker

Docker, previously called dotCloud, was started as a side project and only open-sourced in 2013.

really an extension of LXC’s capabilities. This it achieves using a high-level API that provides a lightweight virtualization solution to run processes in isolation.

developed in the Go language and utilizes LXC, cgroups, and the Linux kernel itself.

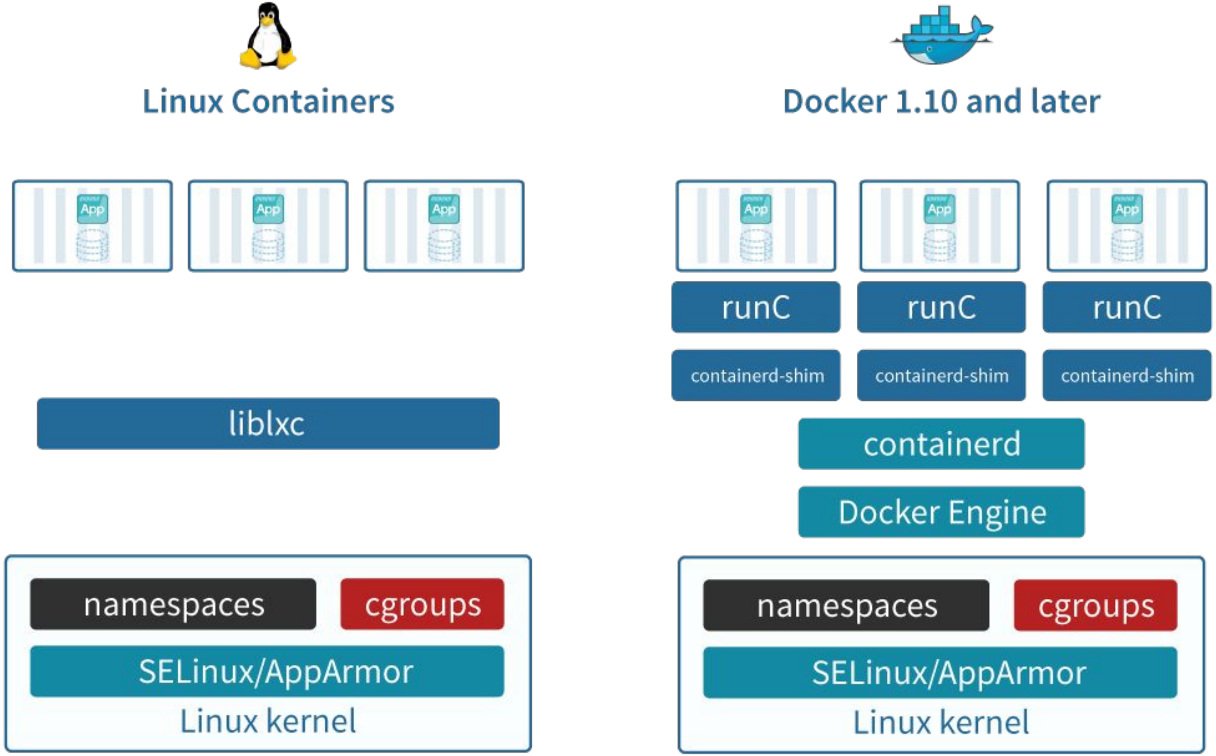
LXC vs Docker

Docker created their own libraries, containerd and runc

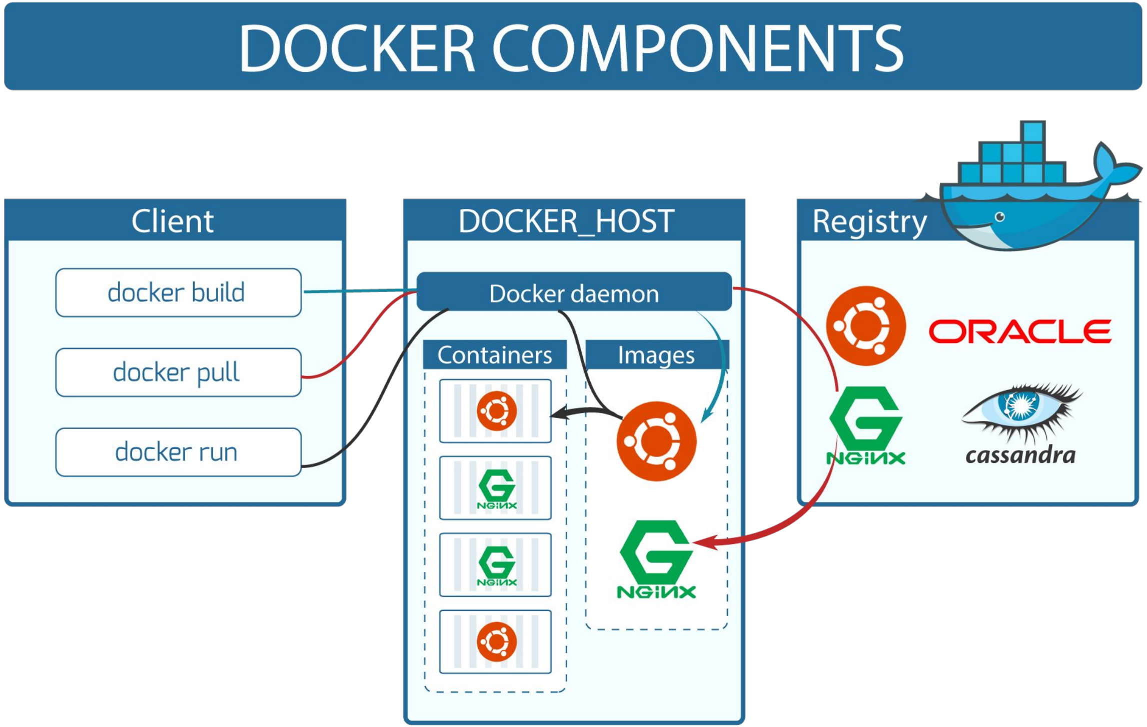
https://containerd.io/

https://github.com/opencontainers/runc

Better for large digital ecosystems.

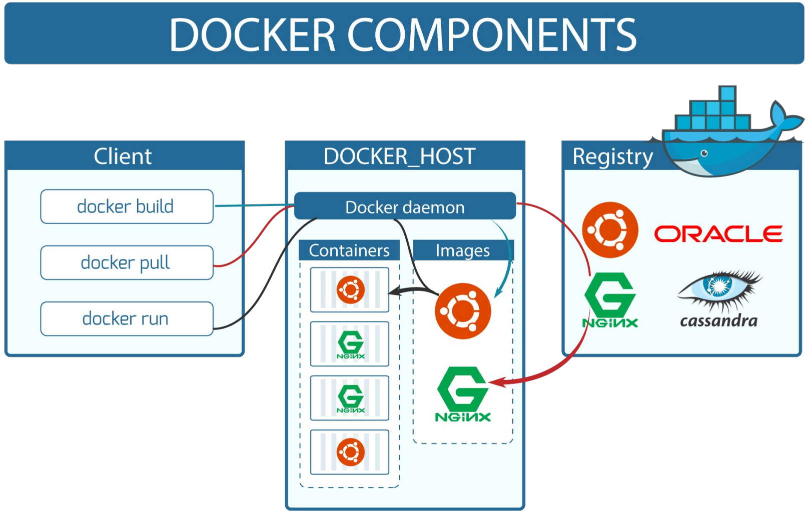


Docker, Docker, Dockah

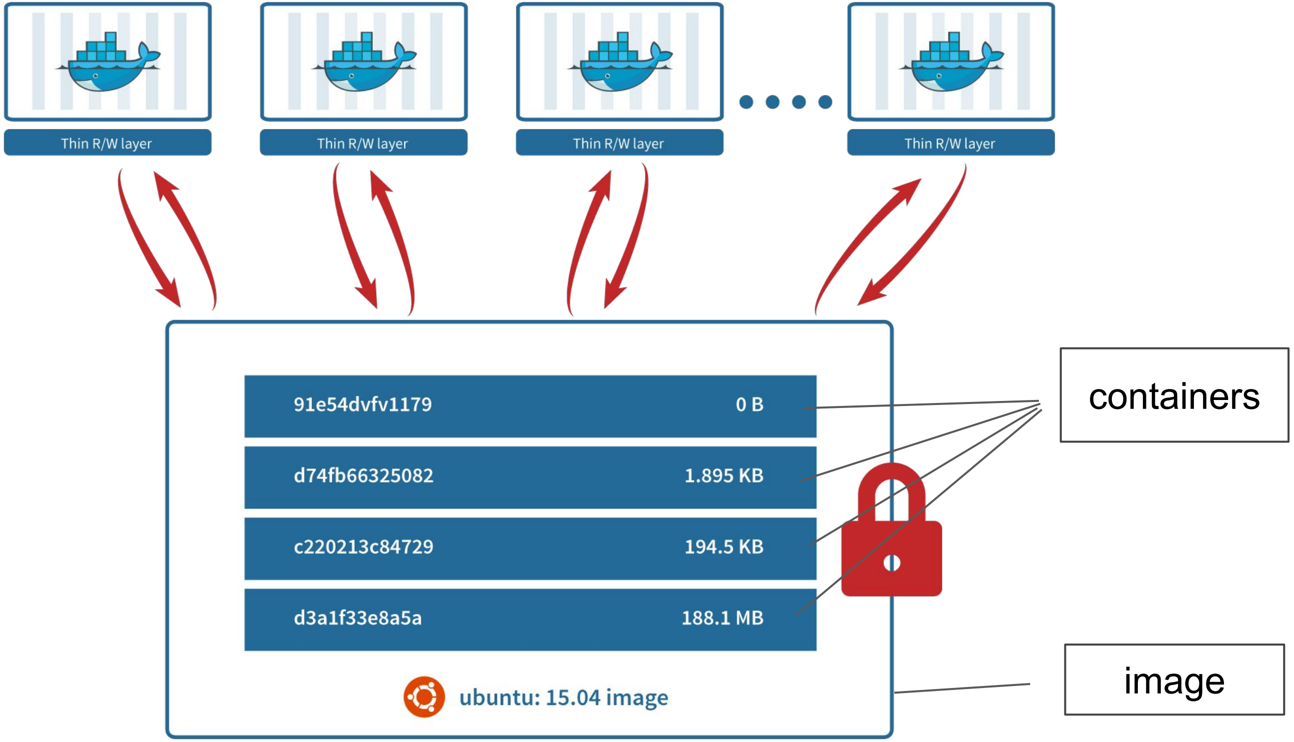


Docker, Docker, Dockah

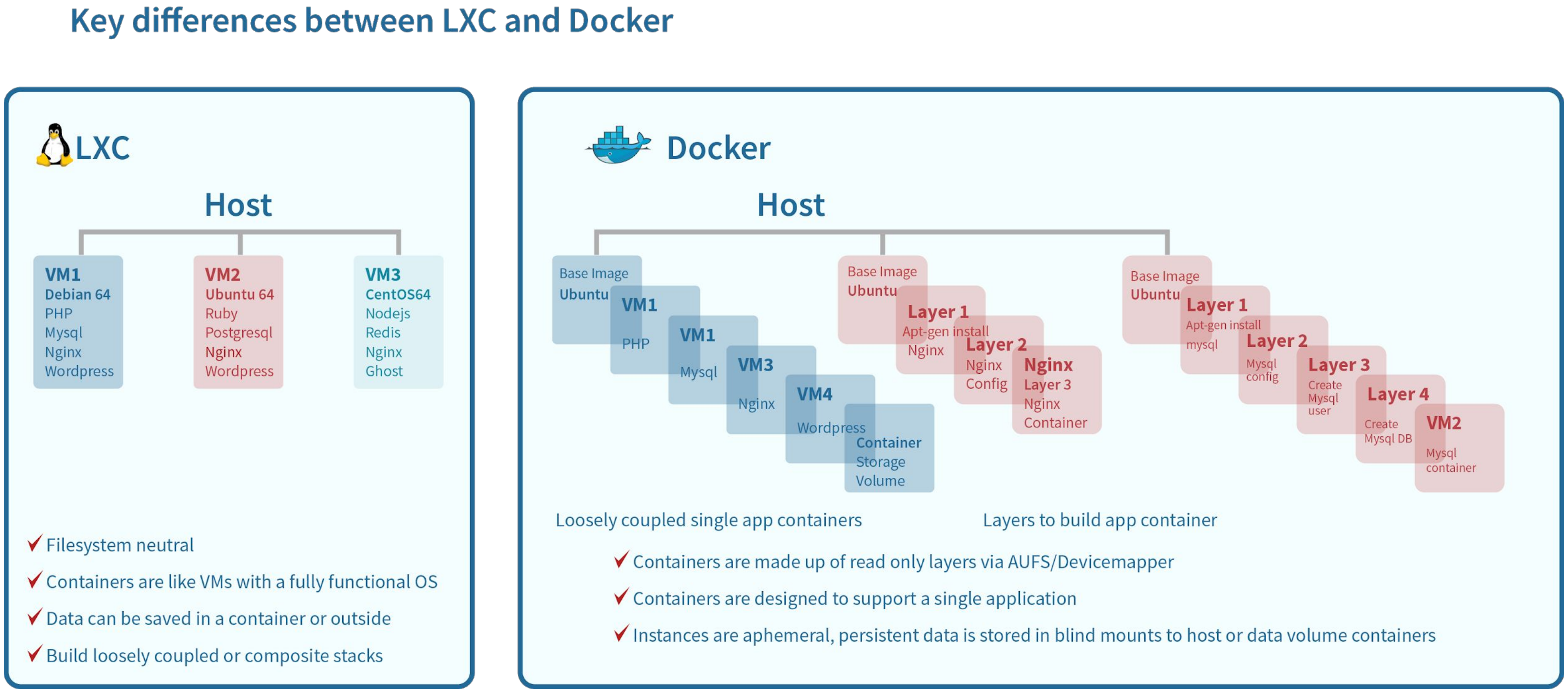
- Docker daemon: runs on a host
- Client: connects to the daemon, and is the primary user interface
- Images: read-only template used to create containers
- Containers: runnable instance of a Docker image
- Registry: private or public registry of Docker images



Docker images and containers



Key differences



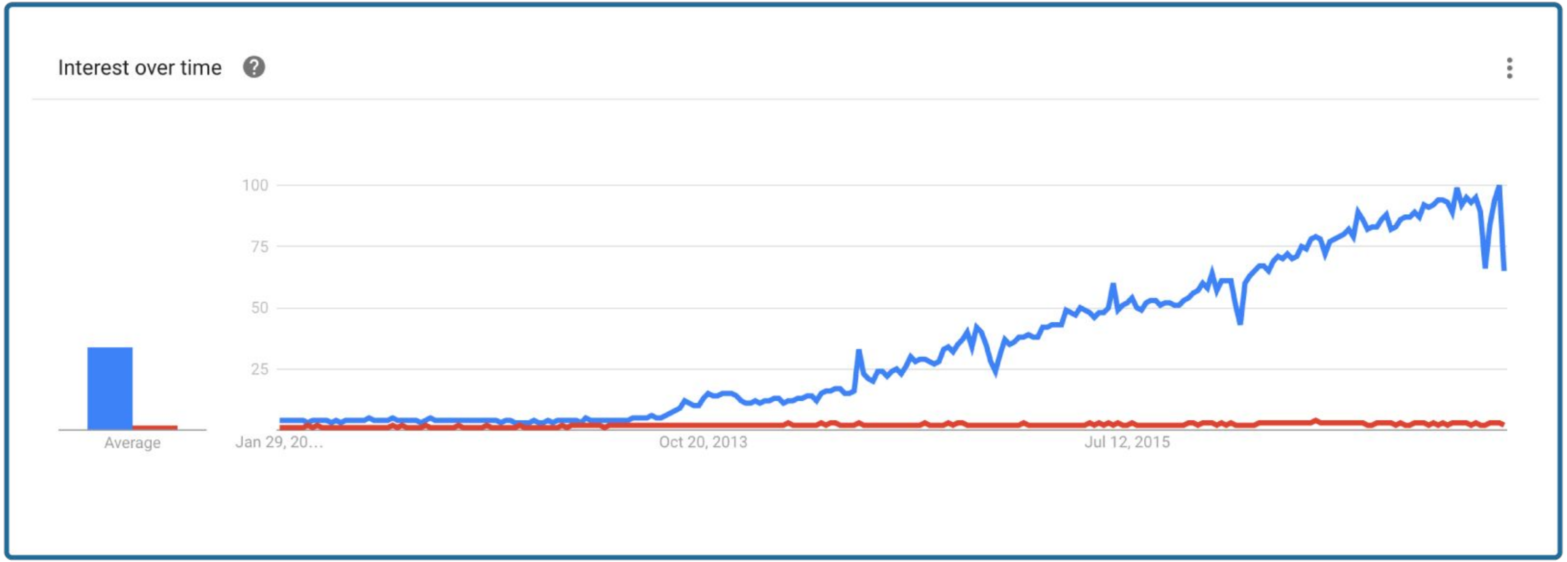
Key differences between LXC and Docker

- ✓ Filesystem neutral
- ✓ Containers are like VMs with a fully functional OS
- ✓ Data can be saved in a container or outside
- ✓ Build loosely coupled or composite stacks

- ✓ Containers are made up of read only layers via AUFS/Devicemapper
- ✓ Containers are designed to support a single application
- ✓ Instances are aphemeral, persistent data is stored in blind mounts to host or data volume containers

Interest over time

Compare: ● Docker vs ● LXC



Container orchestration

As always, everything need to be monitored and controlled using a centralized system. Or else, you might be shock if one of your container become a terminator.

Nobody wants a headache!

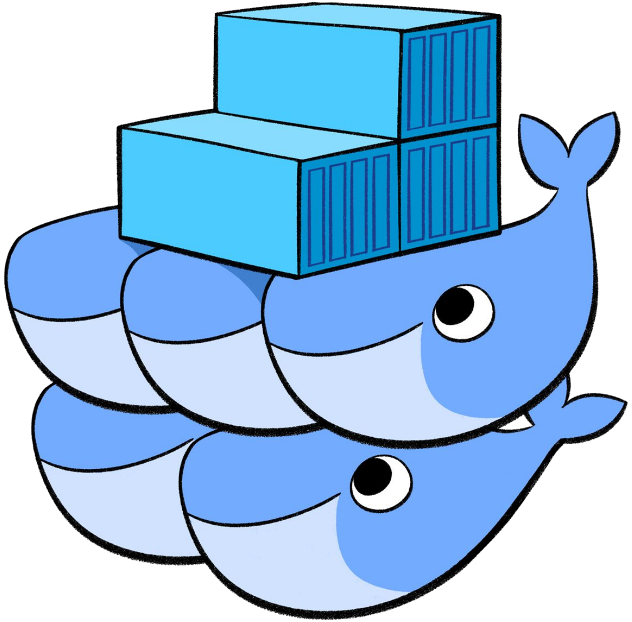
Plus, it such a lame if the terminators exist from a container, LOL.

Container orchestration (cont)

What if sometime you need to scale up your specifications, or you want to create a cluster with nodes talking each others? With ease and fast.

That is why you need a container orchestrator.

Usually we heard about Docker Swarm and Kubernetes.

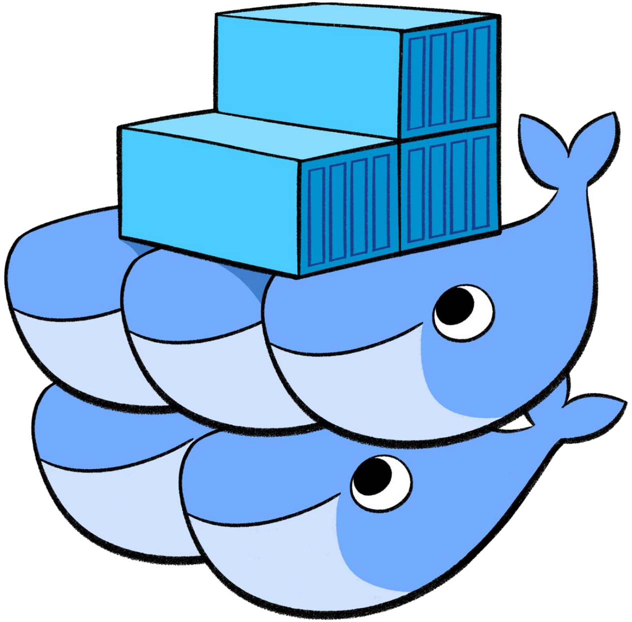


Docker Swarm

Docker's own container orchestration tool

It uses the standard Docker API and networking, easy when we already familiar with Docker.

- A lot easier to install and reason about
- Built into the official Docker CLI
- More lightweight and has less moving parts
- Compatibility with docker-compose.yml files out of the box
- Less sophisticated web UIs vs. Kubernetes for the open source version



Kubernetes

open-source container manager that was originally developed at Google

it’s been ported to Azure, DC/OS, and pretty much on clouds we found.

- Self-healing
- Automated rollback
- Storage orchestration
- Hard check in system

kubernetes

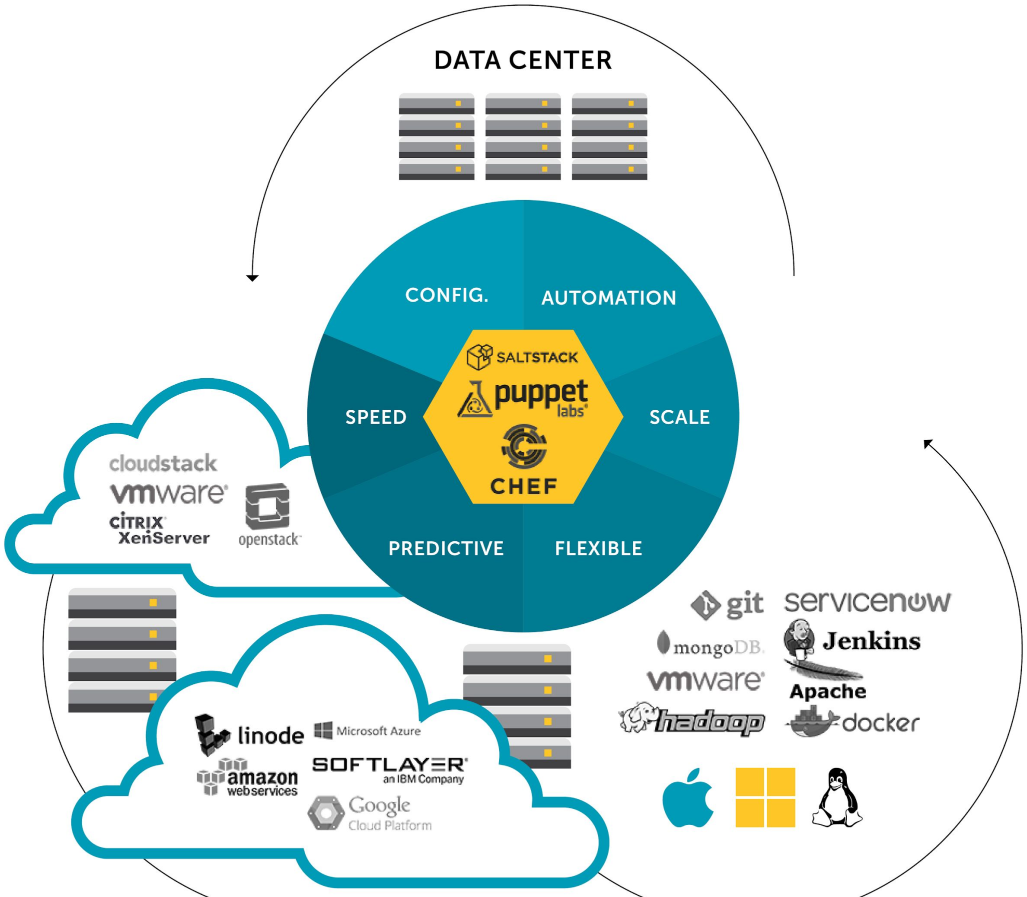
So we need to use Swarm and Kubernetes?

When doing small scale of system, go to Docker Swarm.

Suitable for experimenting, less fault tolerance.

When doing large scale of system plus big data, go to Kubernetes.

High fault tolerance, better for deployment, ported with storage orchestration. Not good for our money or pitih.



ITOps

DATA CENTER

CONFIG.

AUTOMATION

SPEED

SCALE

PREDICTIVE

FLEXIBLE

CloudOps

DevOps

References

https://robinsystems.com/blog/containers-deep-dive-lxc-vs-docker-comparison/

https://www.upguard.com/articles/docker-vs-lxc

https://www.youtube.com/watch?v=L1ie8negCjc

https://www.youtube.com/channel/UCdkGV51Nu0unDNT58bHt9bg

https://www.youtube.com/channel/UCtxCXg-UvSnTKPOzLH4wJaQ