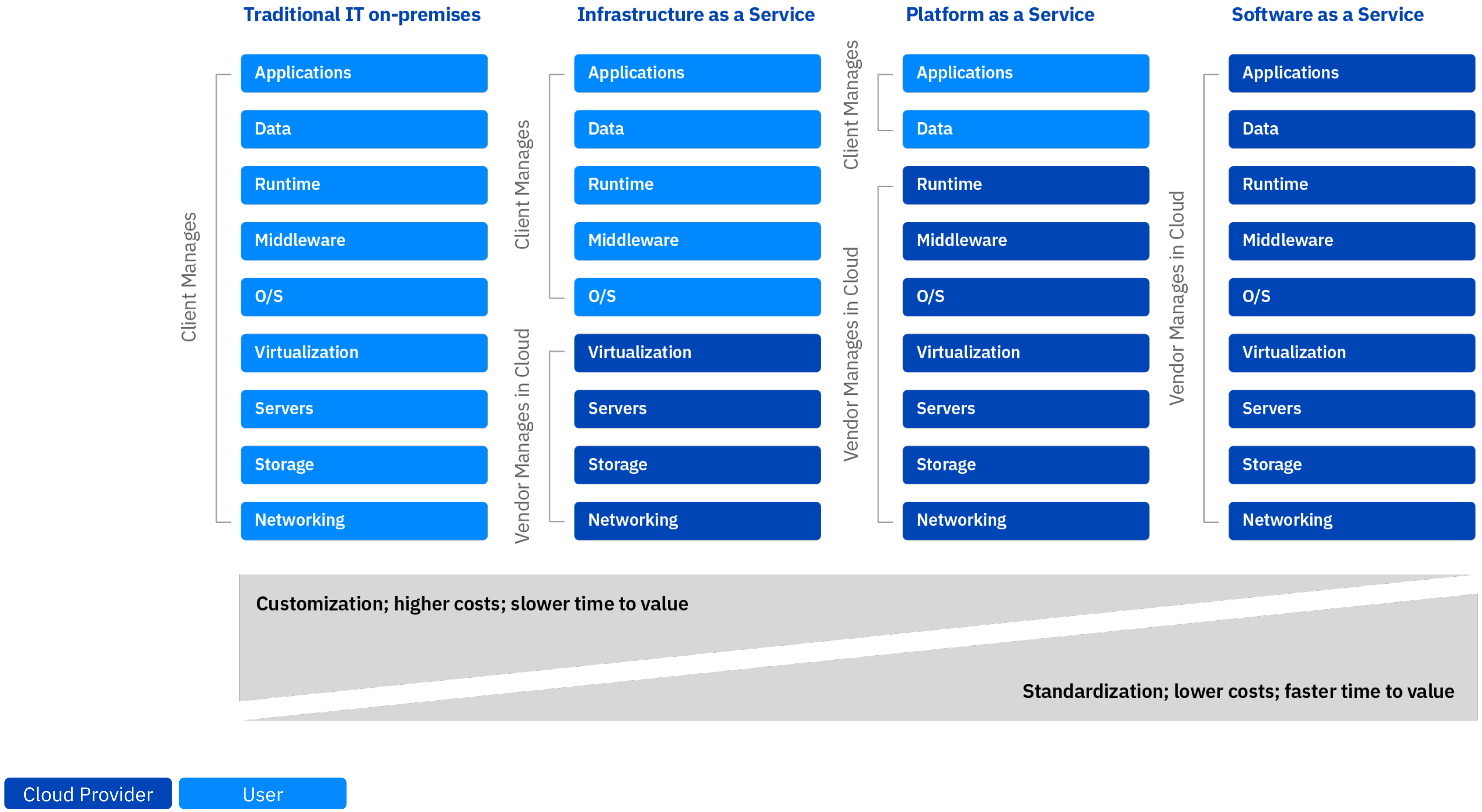
UNIVERSITÀ
CATTOLICA
del Sacro Cuore

Kubernetes Intro

Division of roles



Traditional IT on-premises	Infrastructure as a Service	Platform as a Service	Software as a Service
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

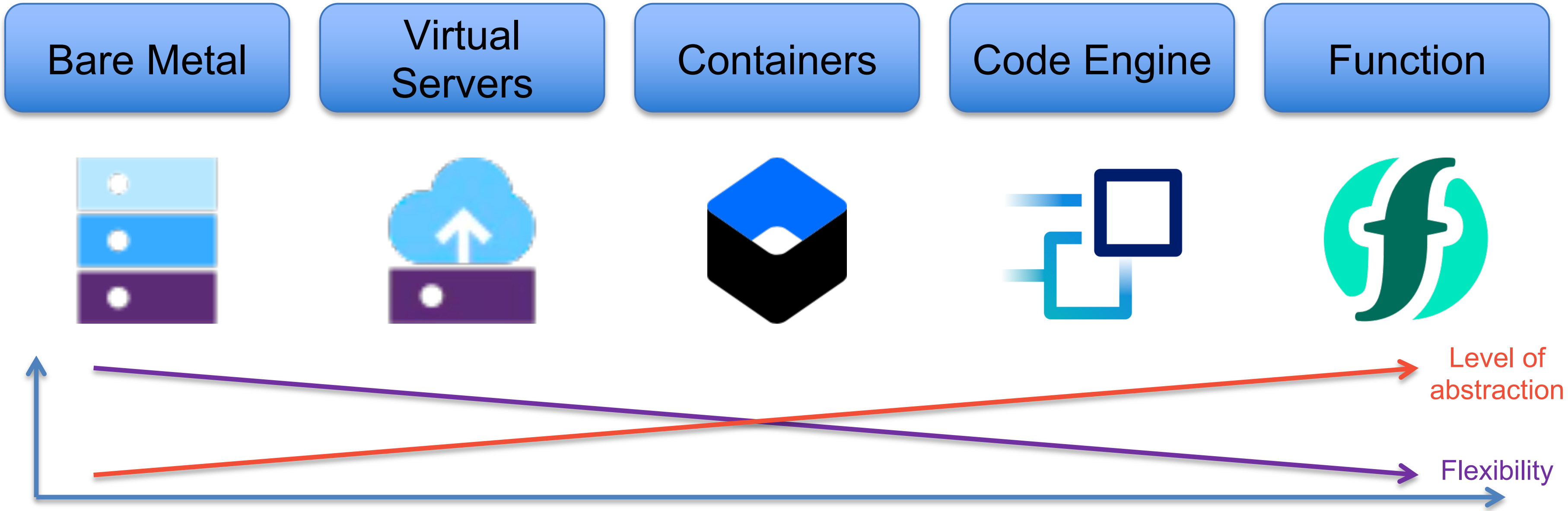
Customization; higher costs; slower time to value

Standardization; lower costs; faster time to value

Cloud Provider

User

Compute Models



Bare Metal

Virtual Servers

Containers

Code Engine

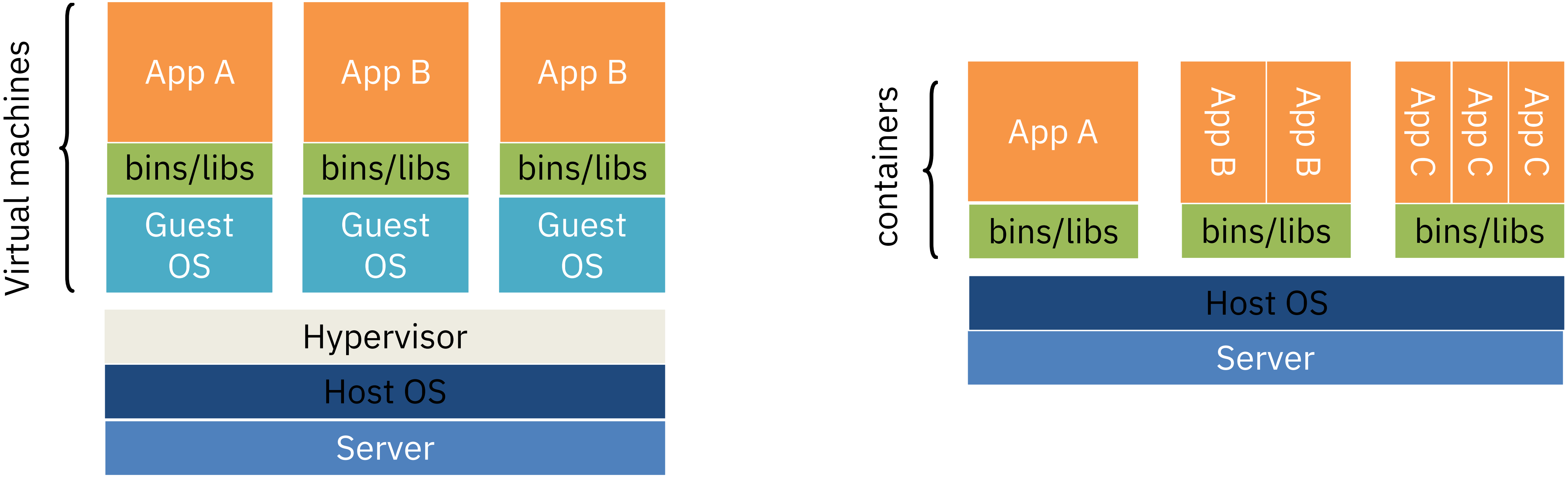
Function

Level of abstraction

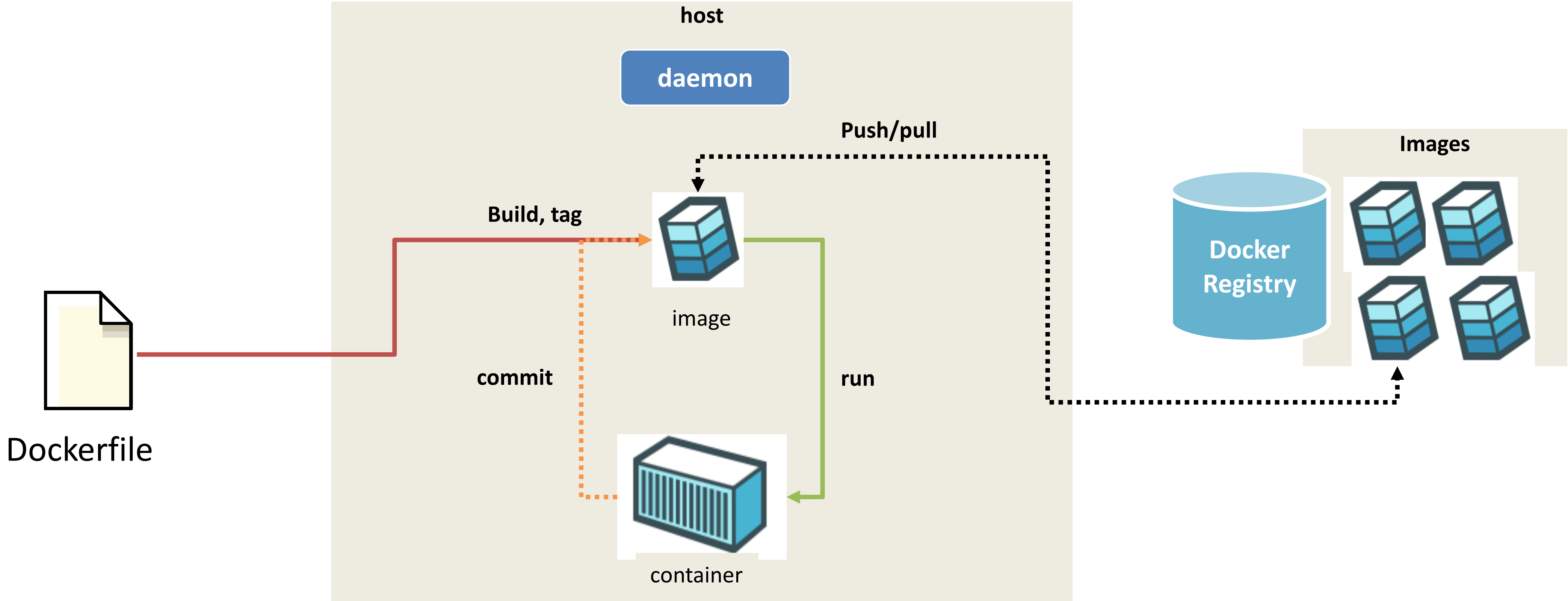
Flexibility

Consistent experience
Common service binding & consumption model
Common user ID & permissions model
Ability to hook into common routing layer

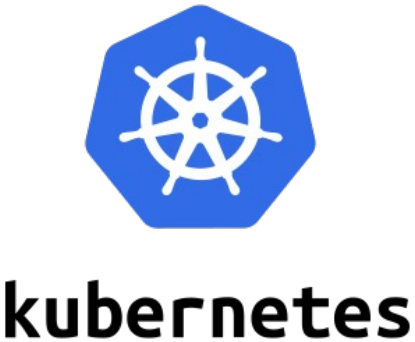
VIRTUAL MACHINES vs CONTAINERS



A typical workflow has you create images, pull them from the repository, build, and run as containers.
Changes to a container may be committed to create a new image. (See docker commit command.)
The more typical and more controlled and documented approach to making image changes is to change the Dockerfile and build a new image.



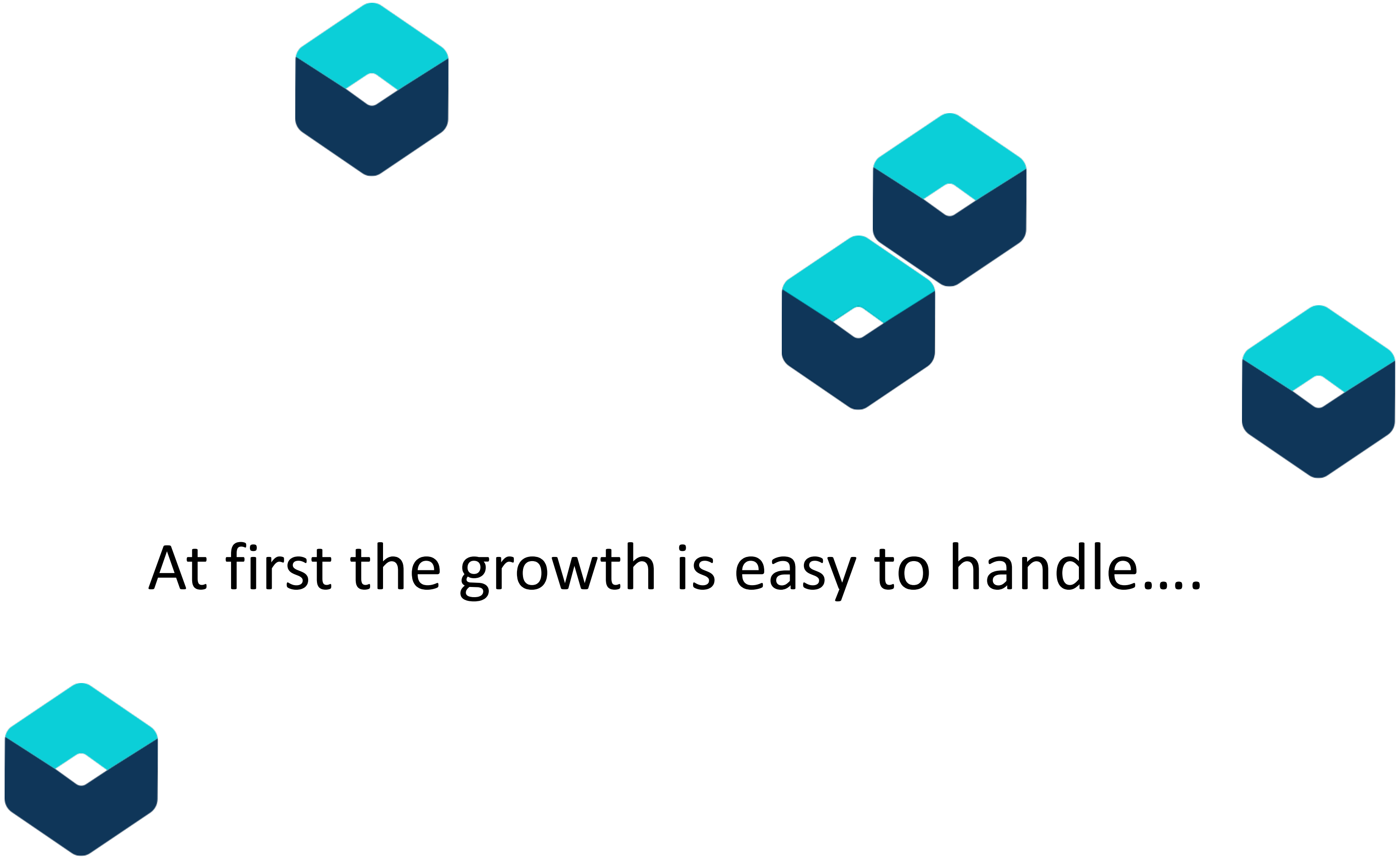
INTRO TO KUBERNETES



INTRO TO KUBERNETES

Everyone’s container journey starts with one container….

INTRO TO KUBERNETES

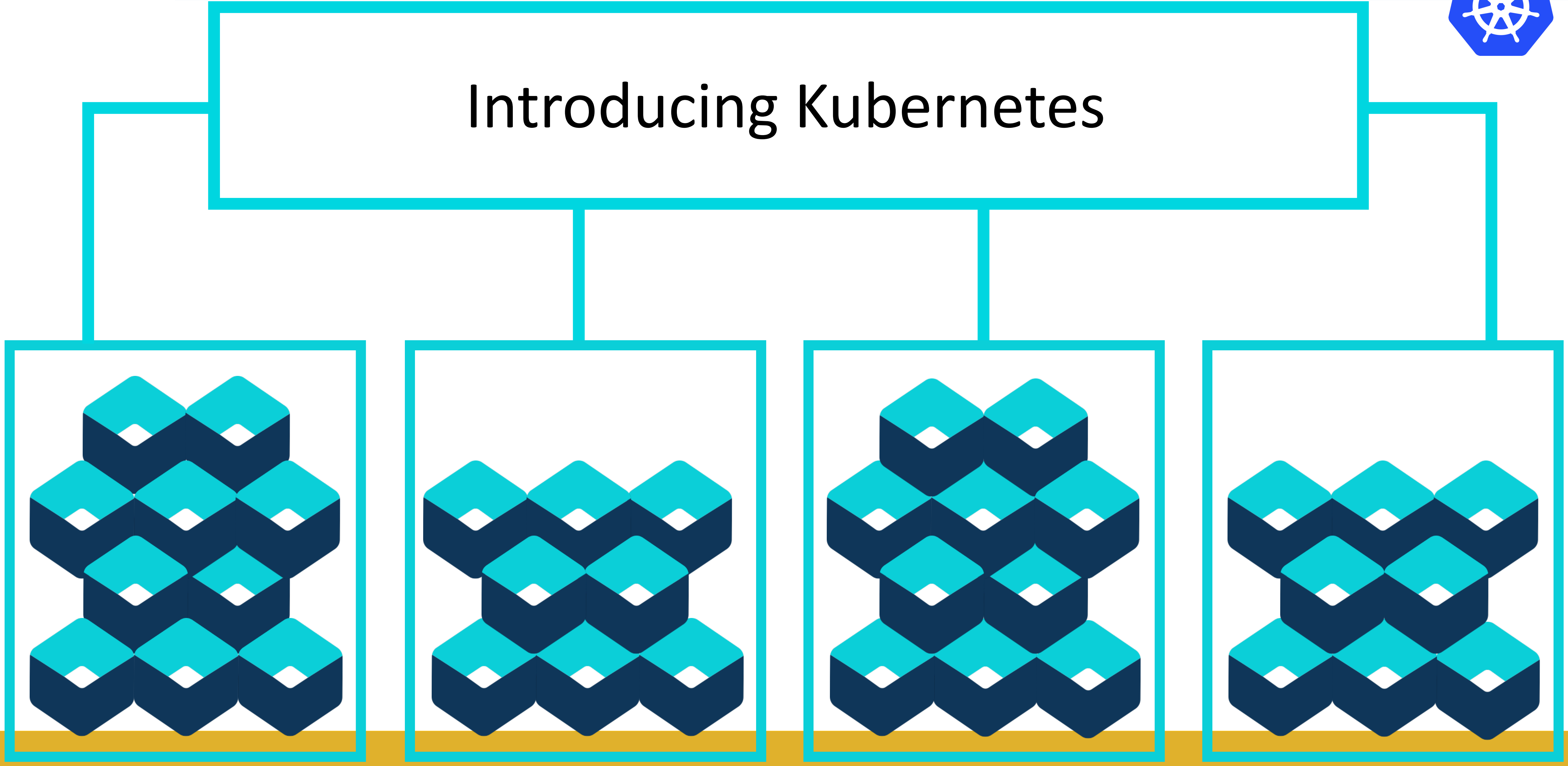


At first the growth is easy to handle....

INTRO TO KUBERNETES

But soon it is overwhelming… chaos reigns

Introducing Kubernetes



WHAT IS CONTAINER ORCHESTRATION?

Container orchestration

- Manages the deployment, placement, and lifecycle of workload containers

Cluster management

- Federates multiple hosts into one target

Scheduling

- Distributes containers across nodes

Service discovery

- Knows where the containers are located
- Distributes client requests across the containers

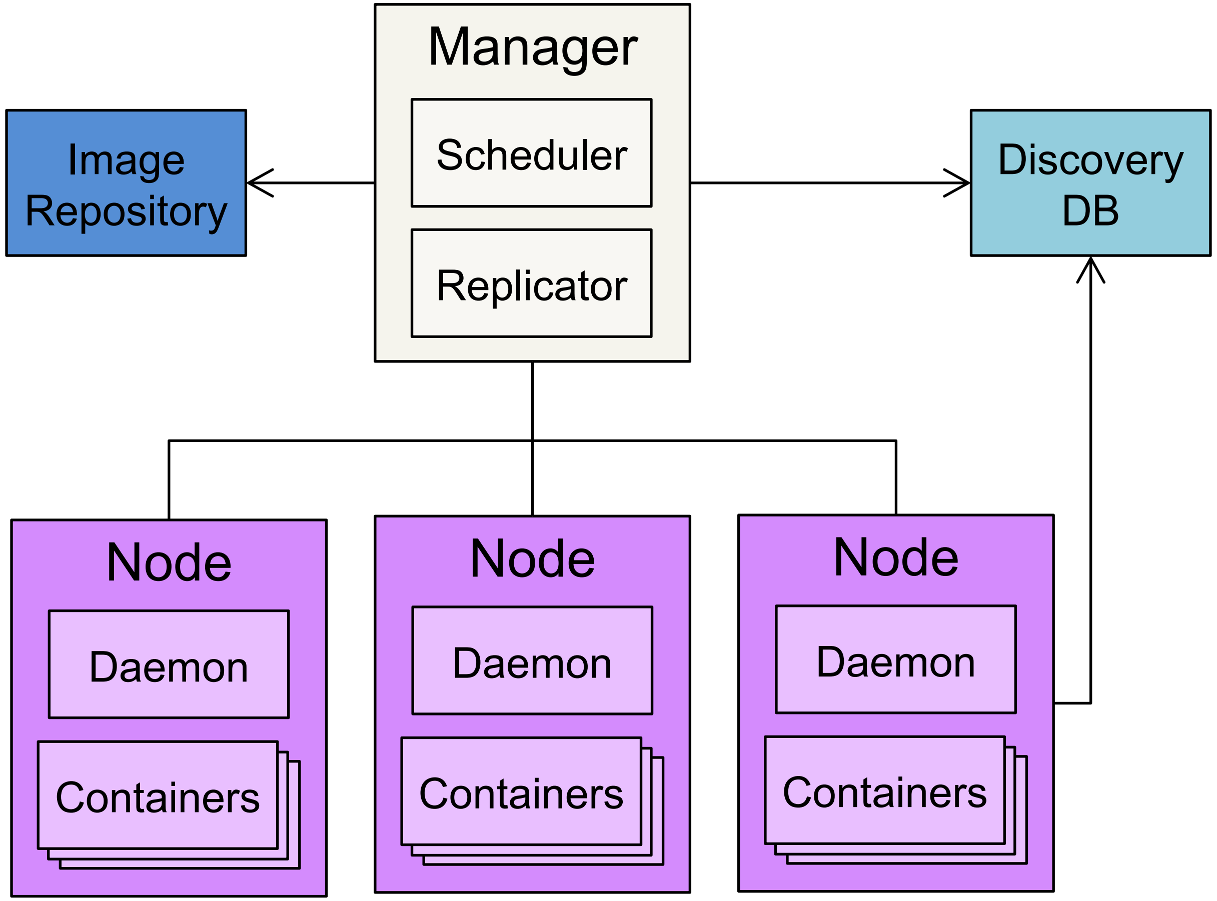
Replication

- Ensures the right number of nodes and containers

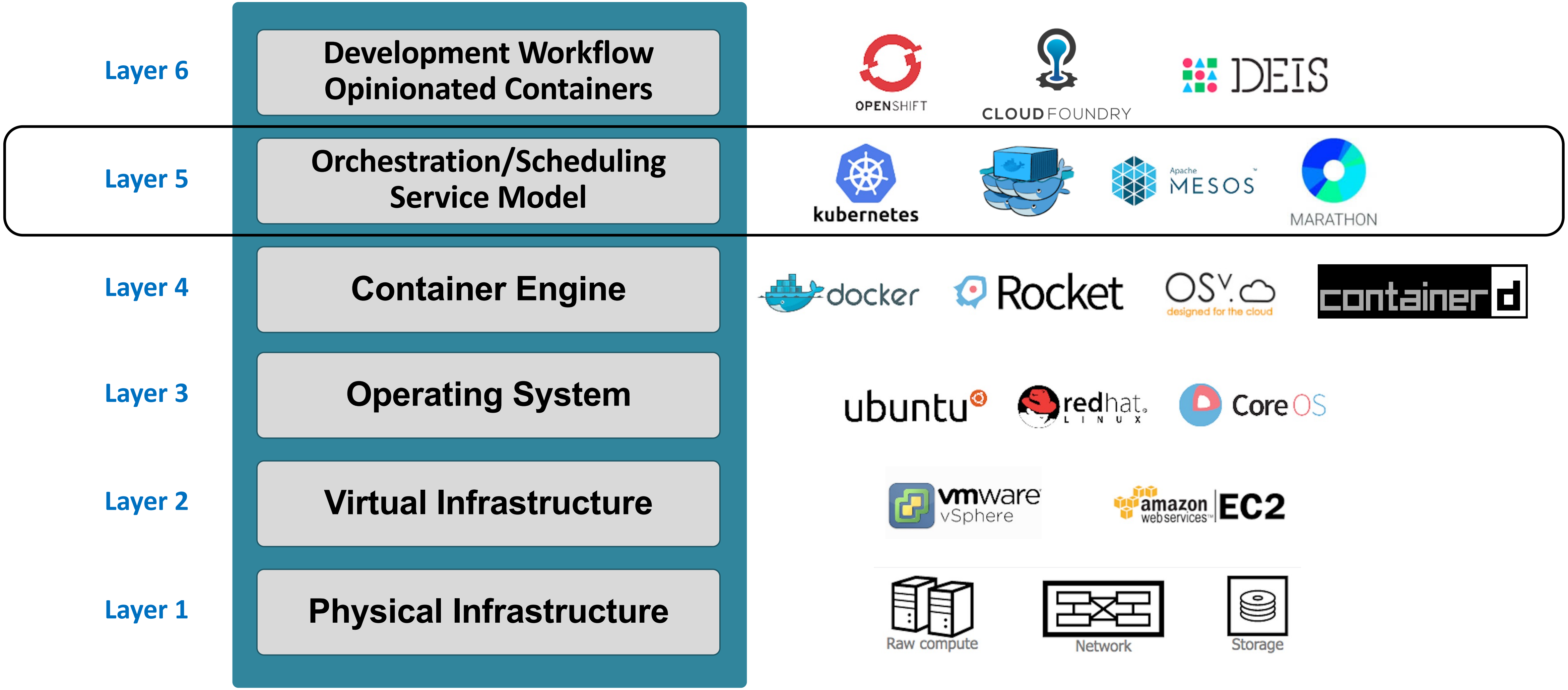
Health management

- Replaces unhealthy containers and nodes

Container Orchestrator



LAYERS



Layer 6 — Development Workflow
Opinionated Containers

Layer 5 — Orchestration/Scheduling
Service Model

Layer 4 — Container Engine

Layer 3 — Operating System

Layer 2 — Virtual Infrastructure

Layer 1 — Physical Infrastructure

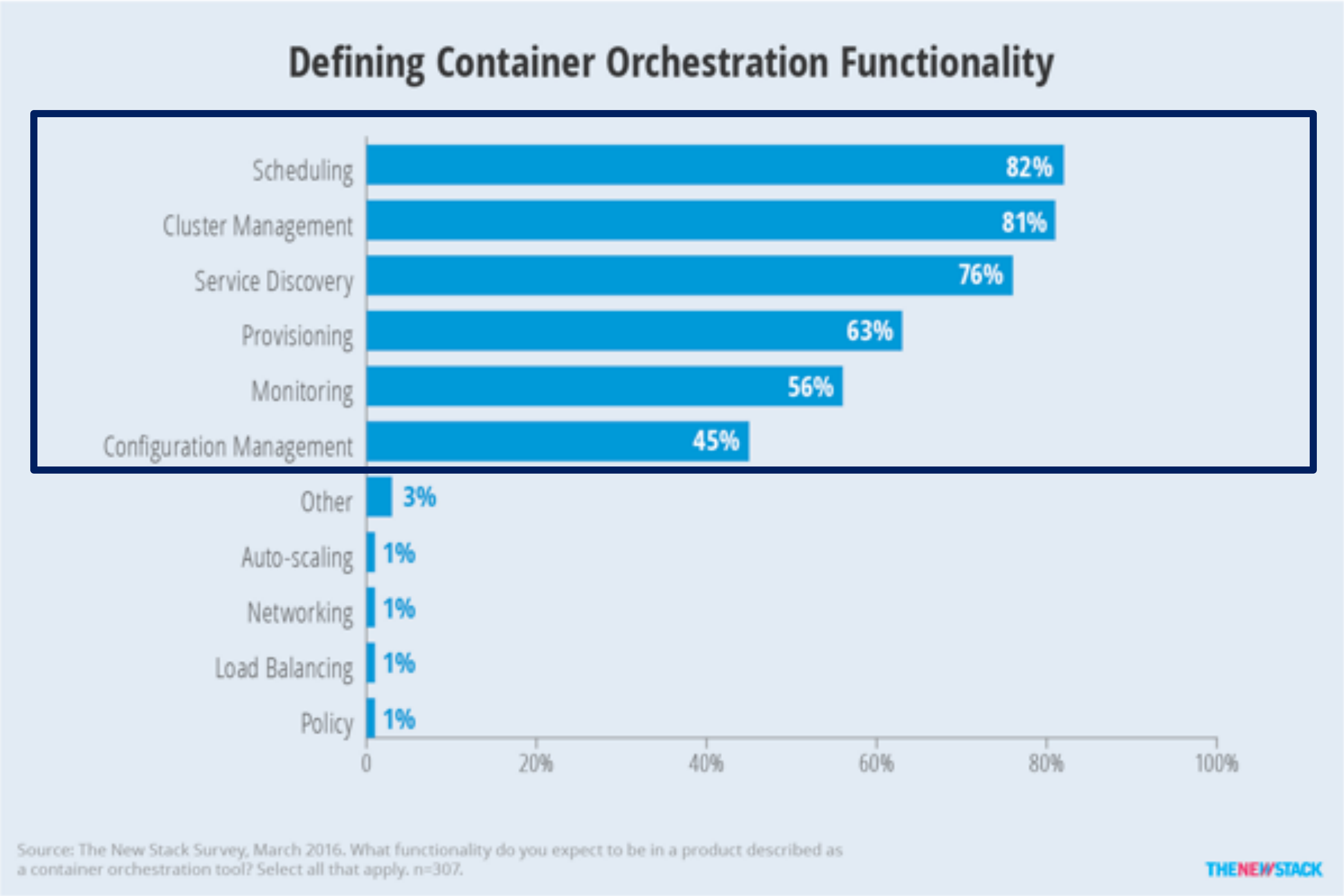
CONTAINER ORCHESTRATION RESPONSIBILITIES

Container orchestration

- Scheduling
- Cluster management
- Service discovery

Related functionality

- Provisioning
- Monitoring
- Configuration management



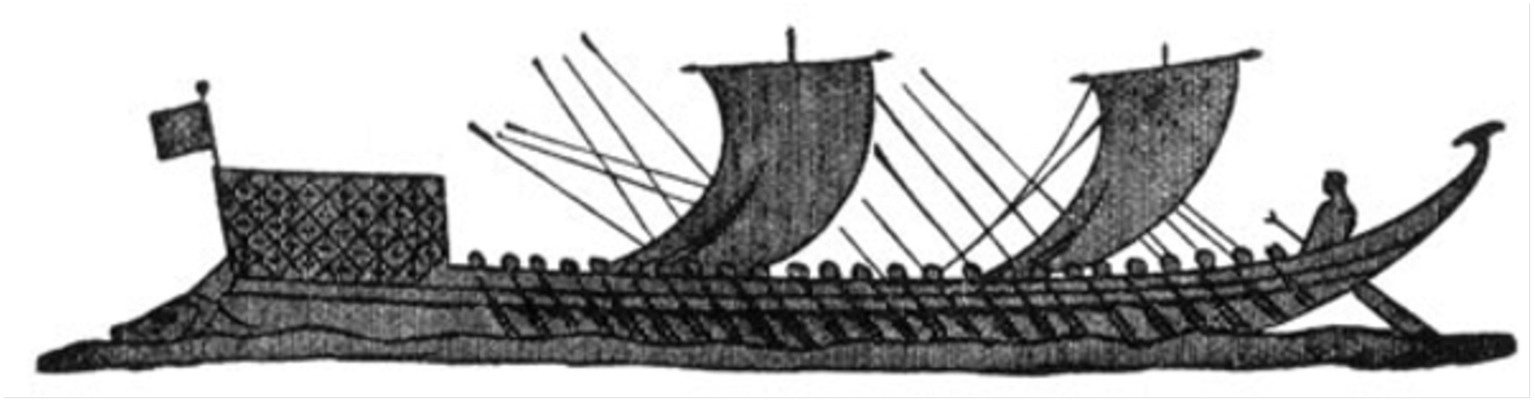
Defining Container Orchestration Functionality

Source: The New Stack Survey, March 2016. What functionality do you expect to be in a product described as a container orchestration tool? Select all that apply. n=307.

KUBERNETES

Greek for “pilot” or

“Helmsman of a ship”



KUBERNETES

Project that was spun out of Google as an open source container orchestration platform.

Built from the lessons learned in the experiences of developing and running Google’s Borg and Omega.

Designed from the ground-up as a **loosely coupled** collection of components (meaning that all the components have little knowledge of each other and function independently) centered around deploying, maintaining and scaling workloads.

INTRO TO KUBERNETES

Known as the **linux kernel of distributed systems**.

Abstracts away the underlying hardware of the nodes and provides a uniform interface for workloads to be both deployed and consume the shared pool of resources.

Works as an engine for resolving state by converging actual and the **desired state** of the system.

INTRO TO KUBERNETES

- linux kernel of distributed systems

- Sort of like a hypervisors, in that it abstracts away the underlying host resources and shares them in a normalized way.

- the underlying substrate or platform to build your applications and tools on top of

- Think of Kubernetes like a higher-level language for your application architecture.
  - you describe what you need and it tries to resolve it
  - with that in mind, it acts as an engine to resolve state using the abstracted resources to deploy and manage your application.

- It is declarative, and not imperative. You tell it or “declare” what you want, and it figures out the rest.
  - imperative you tell something every step you want it to do
  - declarative you tell it what you want it to be, and it figures it out
  - e.g. ‘I want 5 instances of x’ and it just does it, if something dies, it brings it back to get to 5

DECOUPLES INFRASTRUCTURE AND SCALING

All services within Kubernetes are natively Load Balanced.
Can scale up and down dynamically.
Used both to enable self-healing and seamless upgrading or rollback of applications.

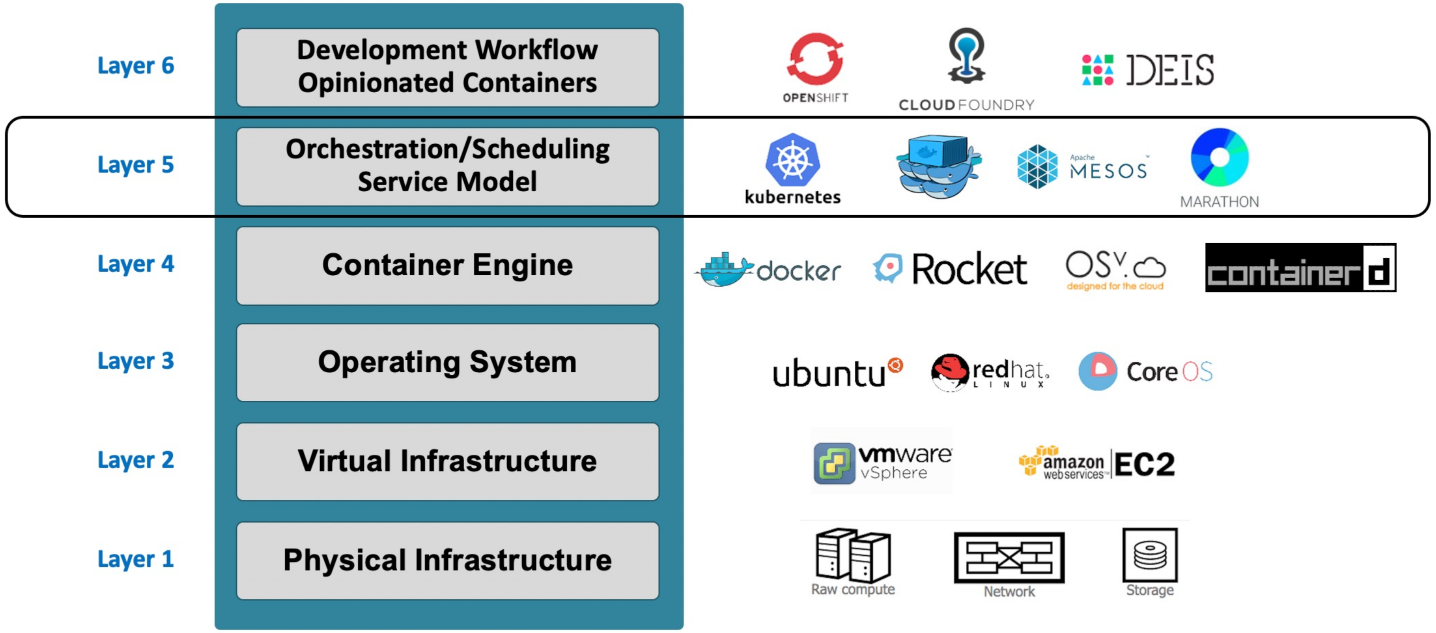
SELF HEALING

Kubernetes will **ALWAYS** try and steer the cluster to its desired state.

Me: “I want 3 healthy instances of redis to always be running.”
Kubernetes: “Okay, I’ll ensure there are always 3 instances up and running.”
Kubernetes: “Oh look, one has died. I’m going to attempt to spin up a new one.”

INTRO TO KUBERNETES

Use the **SAME** API (Commands)
across bare metal and
EVERY cloud provider



Questions?