Kubernetes

Let’s talk about Microservices first

Microservices and Orchestration

IT challenges

What are microservices?

Characteristics of a microservice architecture

Benefits of the microservices approach

Microservices and the cloud

Introducing to Containers

IT Challenges

Globalization and interconnectivity place new demands on organizations and IT departments…

- Applications need to **communicate with many external service providers** over the Internet - the age of silo applications is over
- Customers expect **incremental product updates and feature upgrades**, rather than complete product releases every 12 months
- Architectures must be flexible enough to **scale up** across multiple servers **quickly** when volume spikes
- **Availability and resilience** in the worldwide market are essential

What are Microservices?

According to Wiki:

Microservices is a specialisation of an implementation approach for service-oriented architectures (SOA) used to build flexible, independently deployable software systems.

Services in a microservice architecture (MSA) are processes that communicate with each other over a network in order to fulfil a goal. These services use technology-agnostic protocols.

The microservices approach is a first realisation of SOA that followed the introduction of DevOps and is becoming more popular for building continuously deployed systems.

Characteristics of a Microservice Architecture

Microservices are a move away from traditional monolithic architectures

- Application functionality is broken down into fine-grained distributed components, often as Dockerized containers

Each component is highly cohesive

- Has responsibility for a very specific piece of domain logic
- Has well-defined boundaries
- Completely owns its data
- The implementation technology of a component is irrelevant

Components are loosely coupled

- Each component is deployed independently of other components
- Communicate via technology-neutral protocols, e.g. HTTPS, JSON

Benefits of the Microservices Approach

Scalability

- Microservices can be distributed across multiple nodes
- Makes it easier to scale-out services as needed (easier than scaling out an entire application)

Flexibility

- Microservices offer a finer level of granularity than traditional apps
- Easier to compose and rearrange to deliver new functionality
- This is the oft-cited benefit of OO back in the (19)90s!

Resilience

- Microservices are decoupled, so they degrade/fail in isolation
- Failures can be contained locally, without dragging the whole application down

Microservices and the Cloud

- Microservices are ideally suited for deployment on the cloud
  - Easy to deploy individually, typically as Docker containers
  - Typically small in size, so it's OK to start up a large number of the same service if demand spikes
  - Increases scalability and resilience

Containerized applications

*"Containerization is an approach to software development in which **an application or service, its dependencies, and its configuration** (abstracted as deployment manifest files) are **packaged together** as a container image. You then can test the containerized application as a unit and **deploy it as a container** image instance to the host operating system (OS)."*

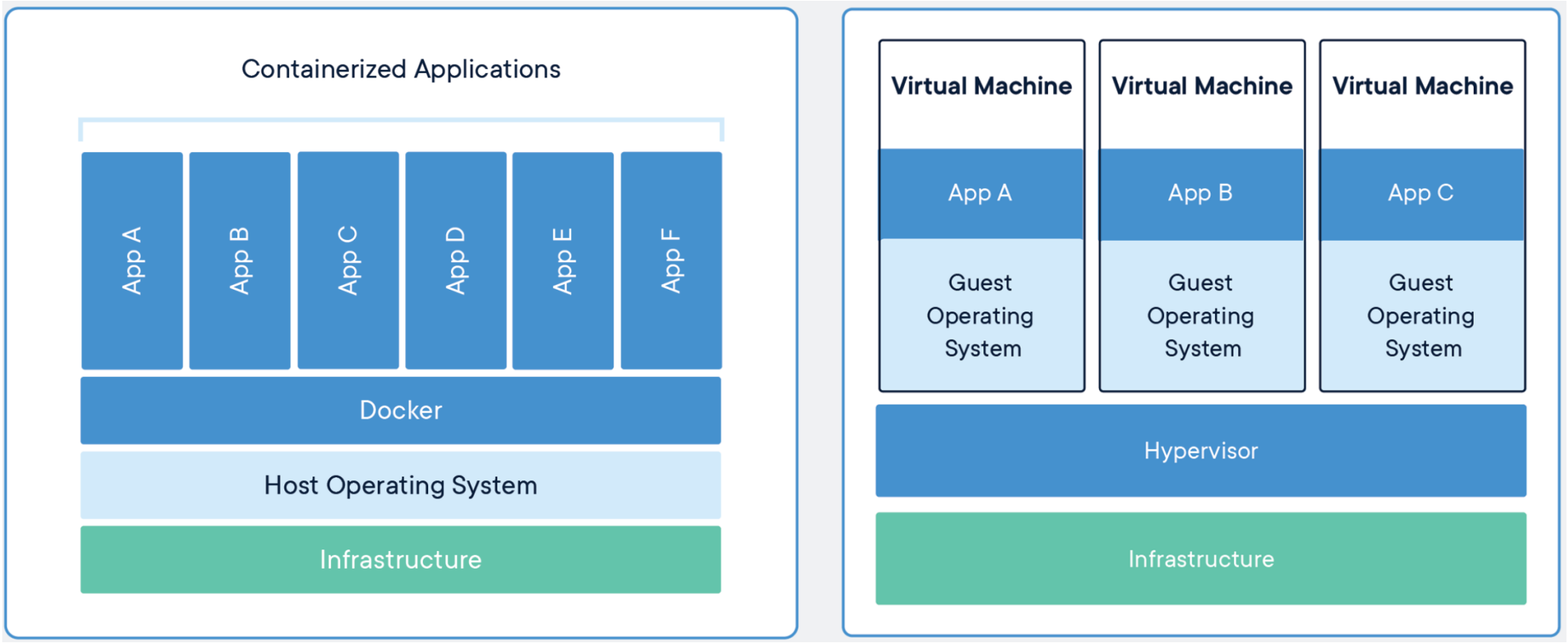
Intro to containers

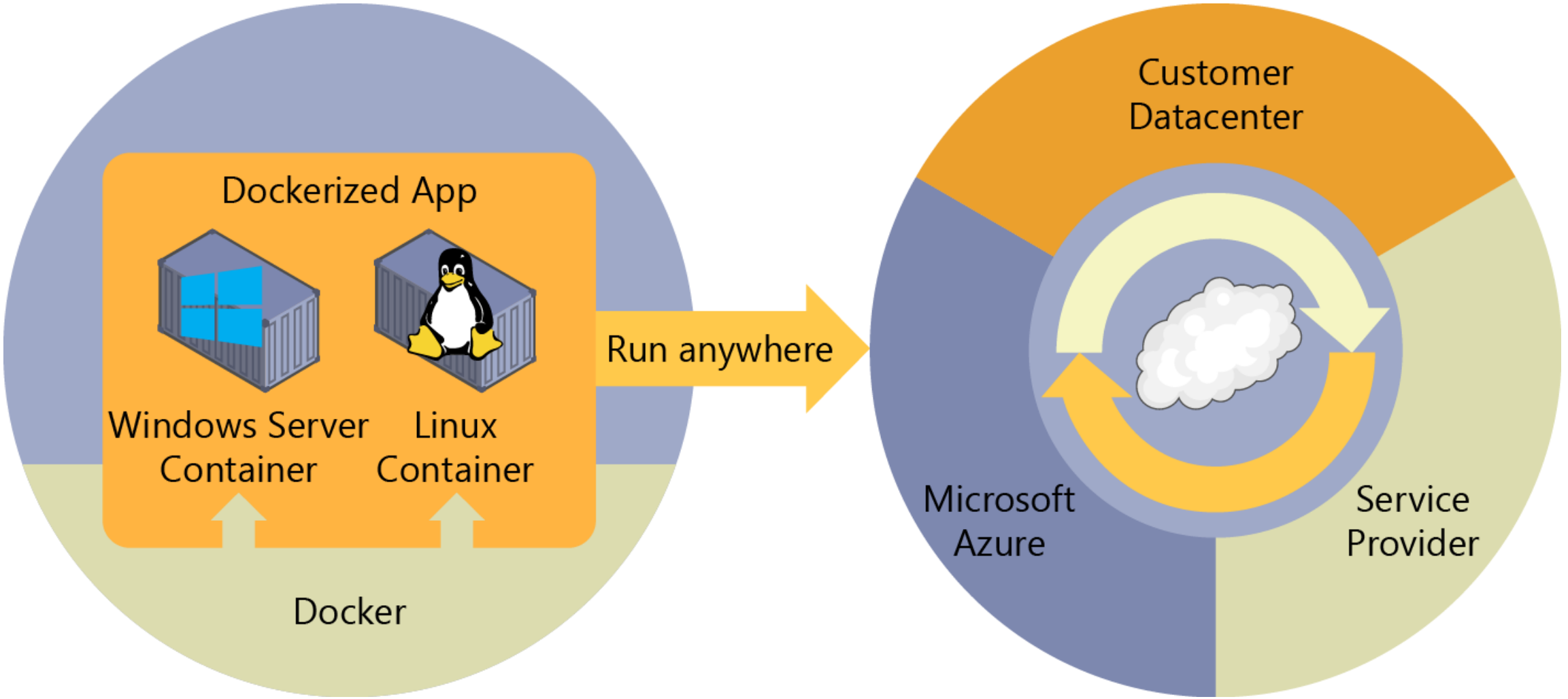
Since 2014

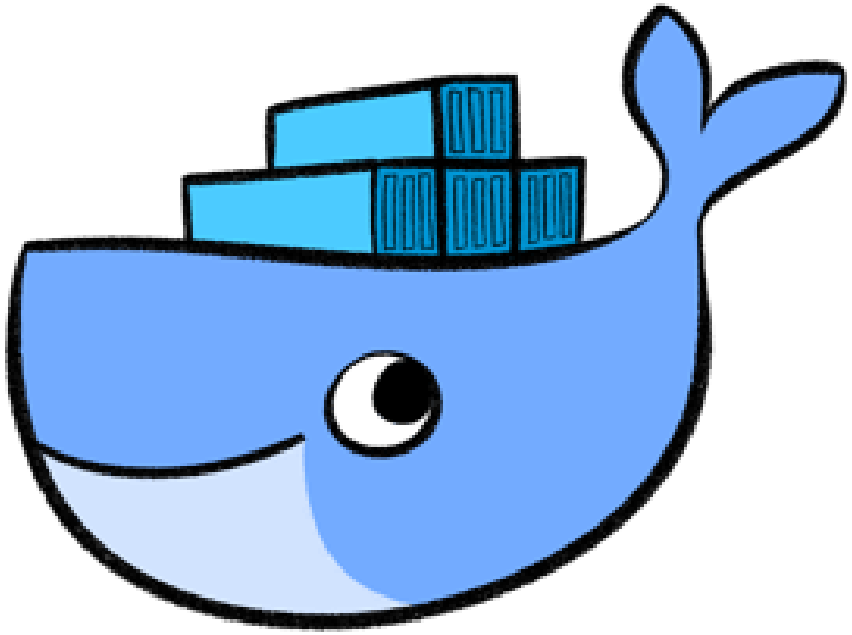
Containers have been around since 2000

Google has lmctfy, running their apps (Gmail, Docs, Search)

Originally for Linux, build on LXC (Linux Containers)

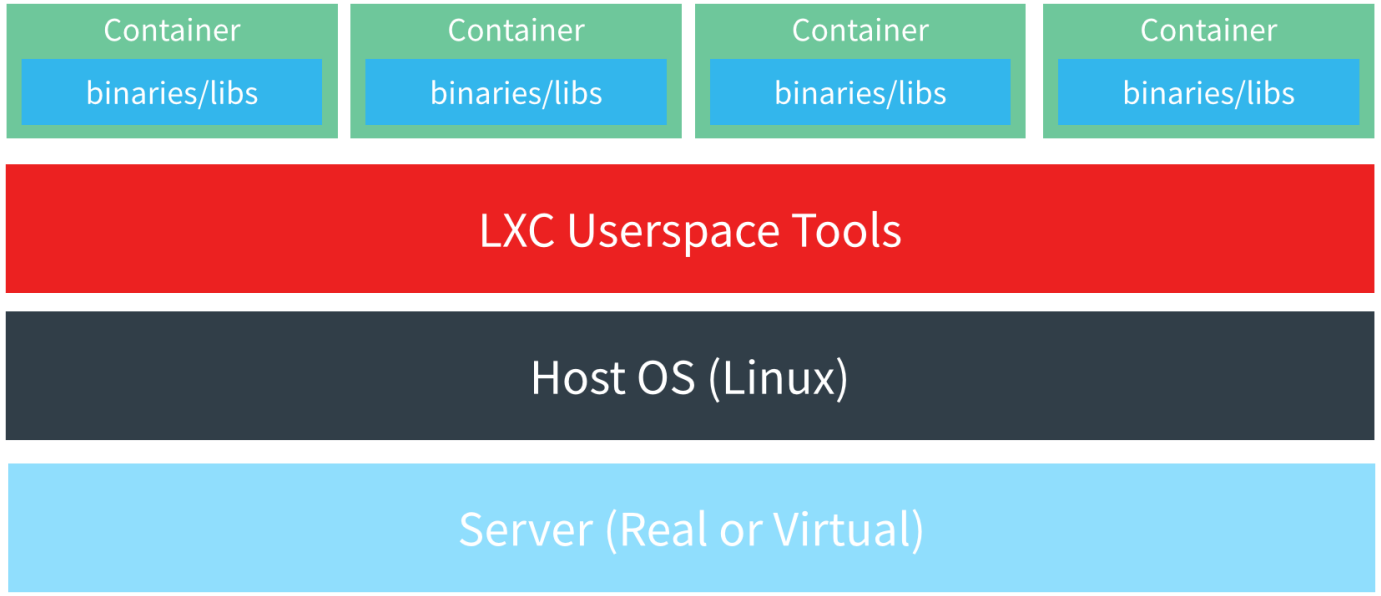

Containerized Applications
App A
App B
App C
App D
App E
App F
Docker
Host Operating System
Infrastructure
Virtual Machine
Virtual Machine
Virtual Machine
App A
App B
App C
Guest Operating System
Guest Operating System
Guest Operating System
Hypervisor
Infrastructure


Dockerized App
Windows Server Container
Linux Container
Docker
Run anywhere
Customer Datacenter
Microsoft Azure
Service Provider



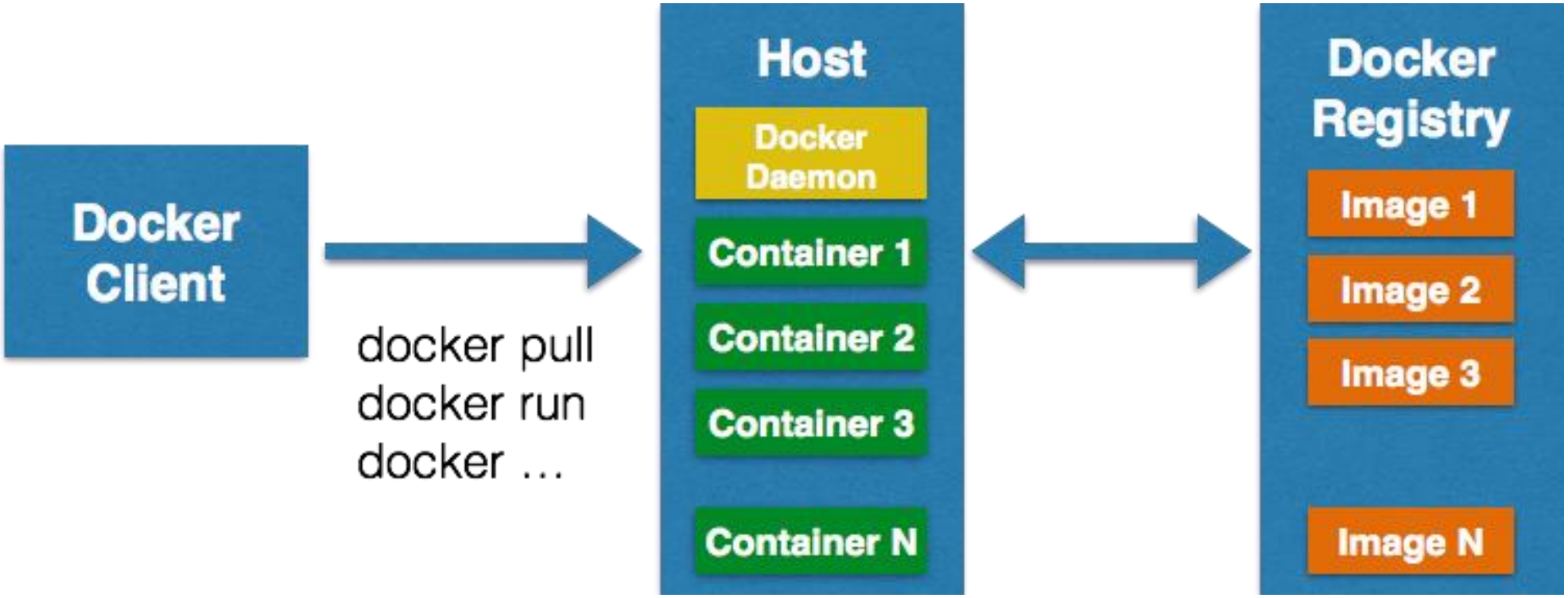
Intro to Containers and Docker

”SHIP MY MACHINE”



How does it work?

The parts


Docker Client
docker pull
docker run
docker …
Host
Docker Daemon
Container 1
Container 2
Container 3
Container N
Docker Registry
Image 1
Image 2
Image 3
Image N

Docker Host

Run as daemon

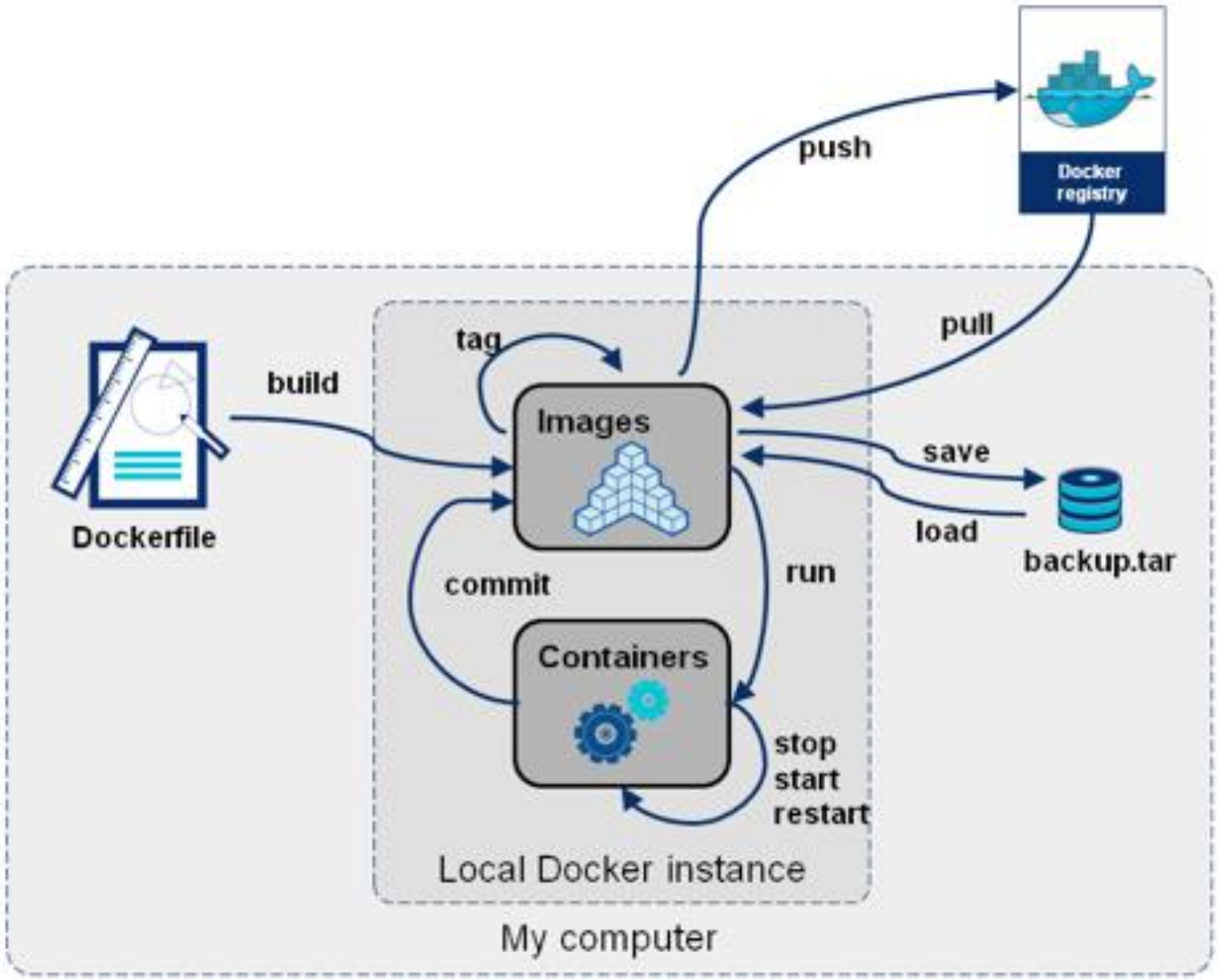
Build images

Download images

Starting & stopping containers

Rest API for remote management

Infrastructure: storage, networking, memory, cpu


Docker registry
push
pull
tag
build
Images
save
load
backup.tar
Dockerfile
commit
run
Containers
stop
start
restart
Local Docker instance
My computer

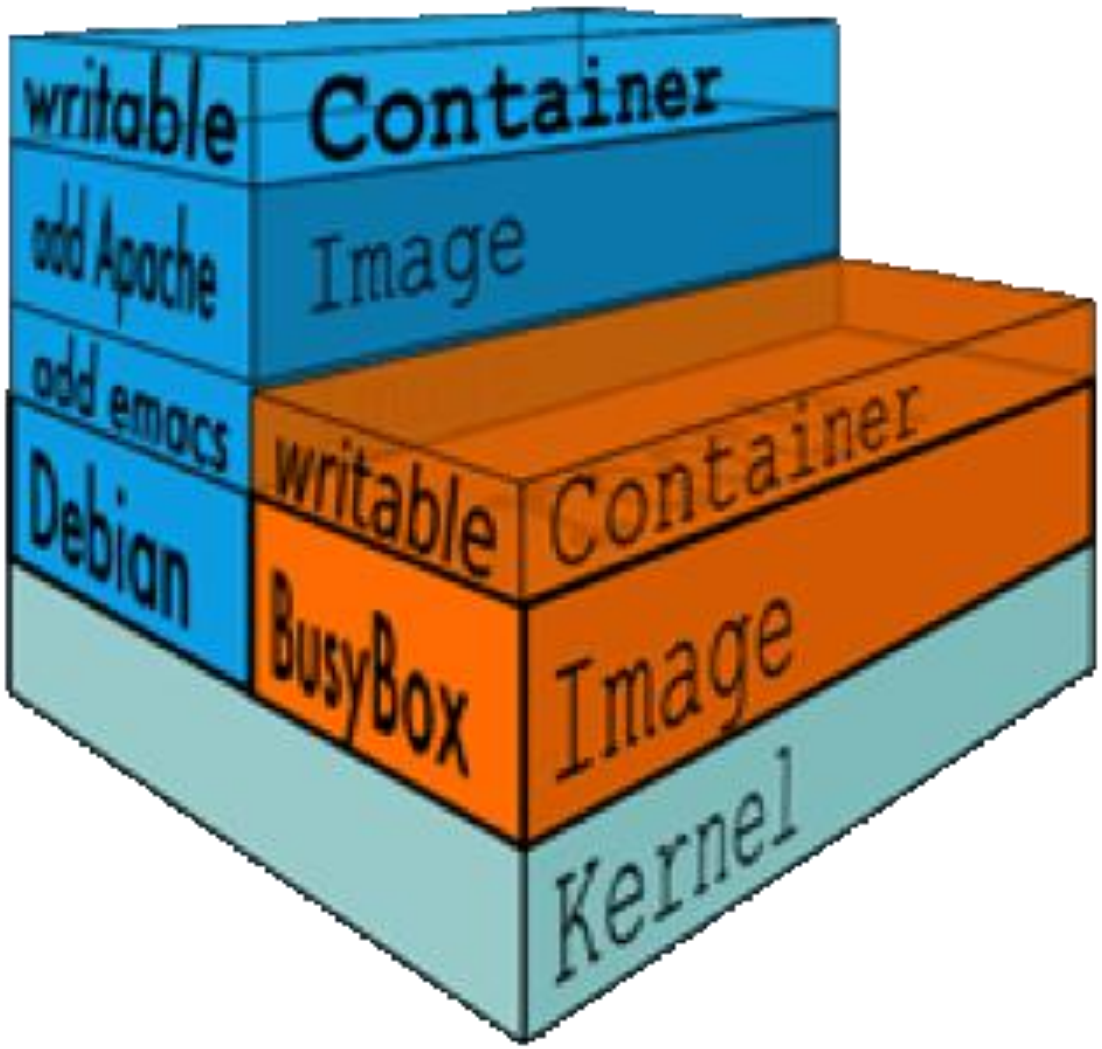
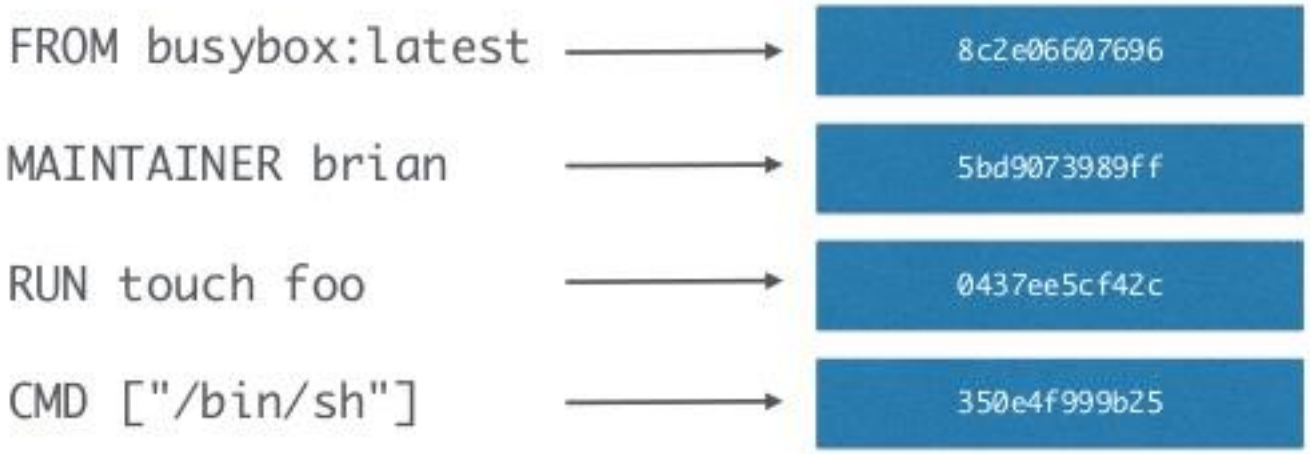

writable
Container
add Apache
Image
add emacs
writable
Container
Debian
BusyBox
Image
Kernel

Image Layers

- Each Dockerfile instruction generates a new layer



Docker terminology

Container image

Dockerfile

Build

Container

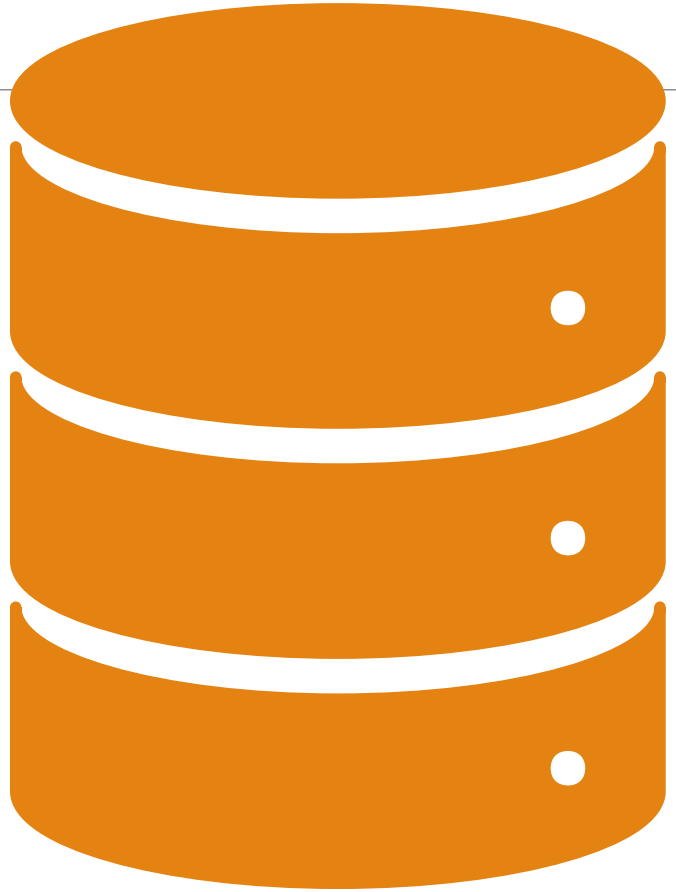
Tag

Registry

Compose

Cluster

Orchestrator

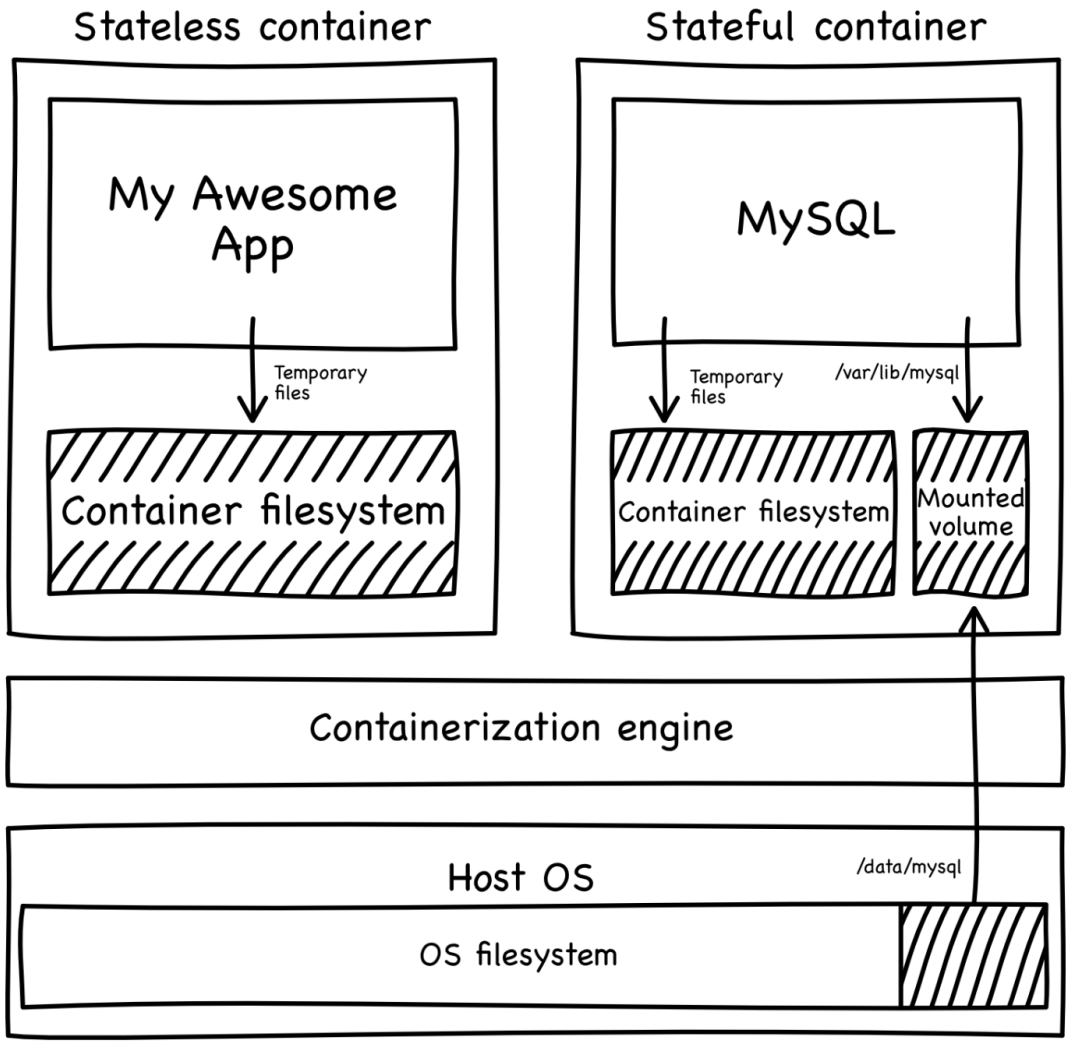


Stateful or Stateless

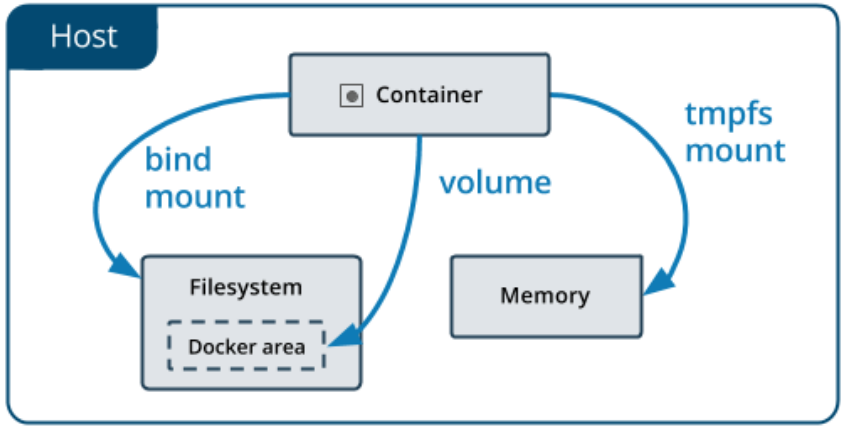
Docker is designed to be stateless

No persistent storage of state!

Workaround: Volumes


Stateless container
My Awesome App
Temporary files
Container filesystem
Stateful container
MySQL
Temporary files
/var/lib/mysql
Container filesystem
Mounted volume
Containerization engine
Host OS
/data/mysql
OS filesystem

Storage



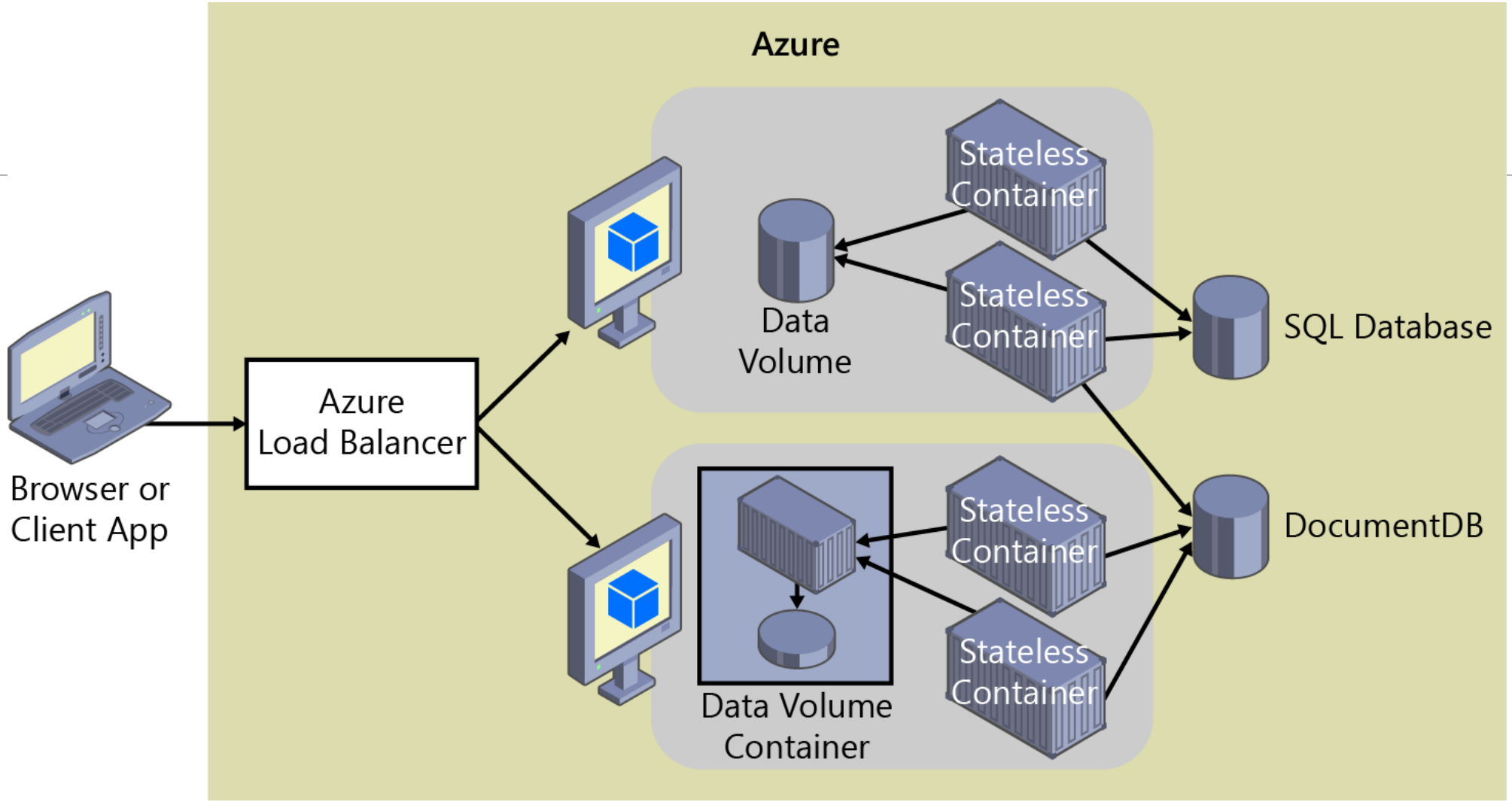
Volume

- Part of host file system
- Managed by Docker

Bind mount

- Anywhere on host
- Inc. system files

Tmpfs


Azure
Stateless Container
Stateless Container
Data Volume
SQL Database
Browser or Client App
Azure Load Balancer
Stateless Container
Stateless Container
Data Volume Container
DocumentDB

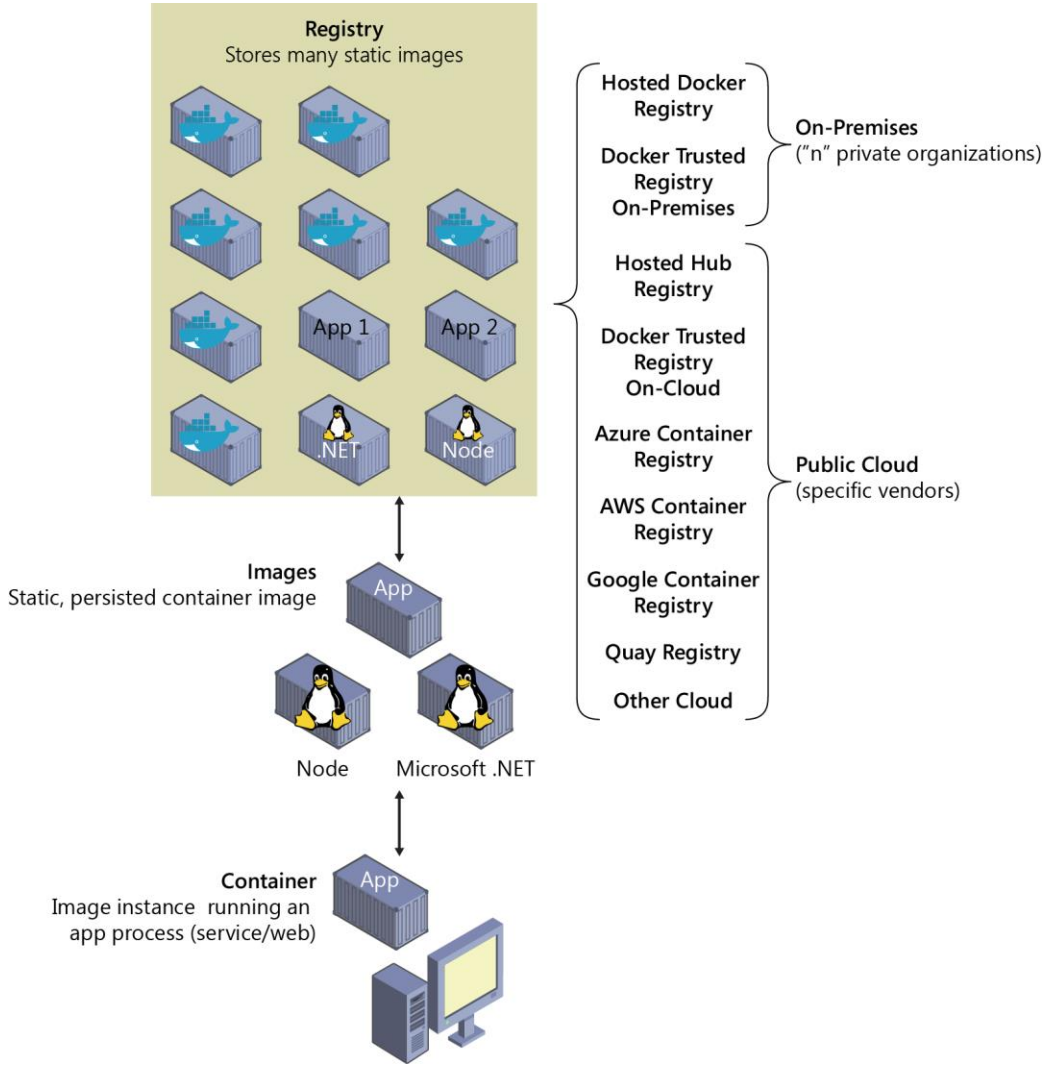
Getting & Building images

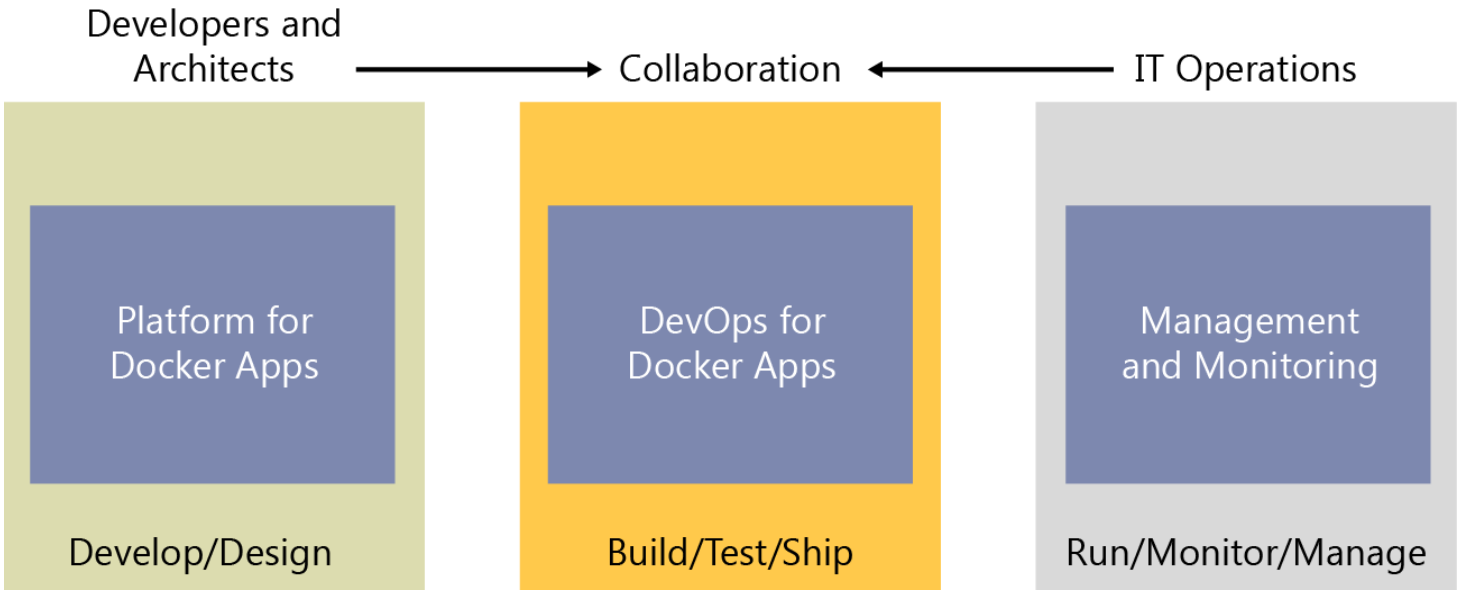
docker pull <image-name>

docker run <image-name>

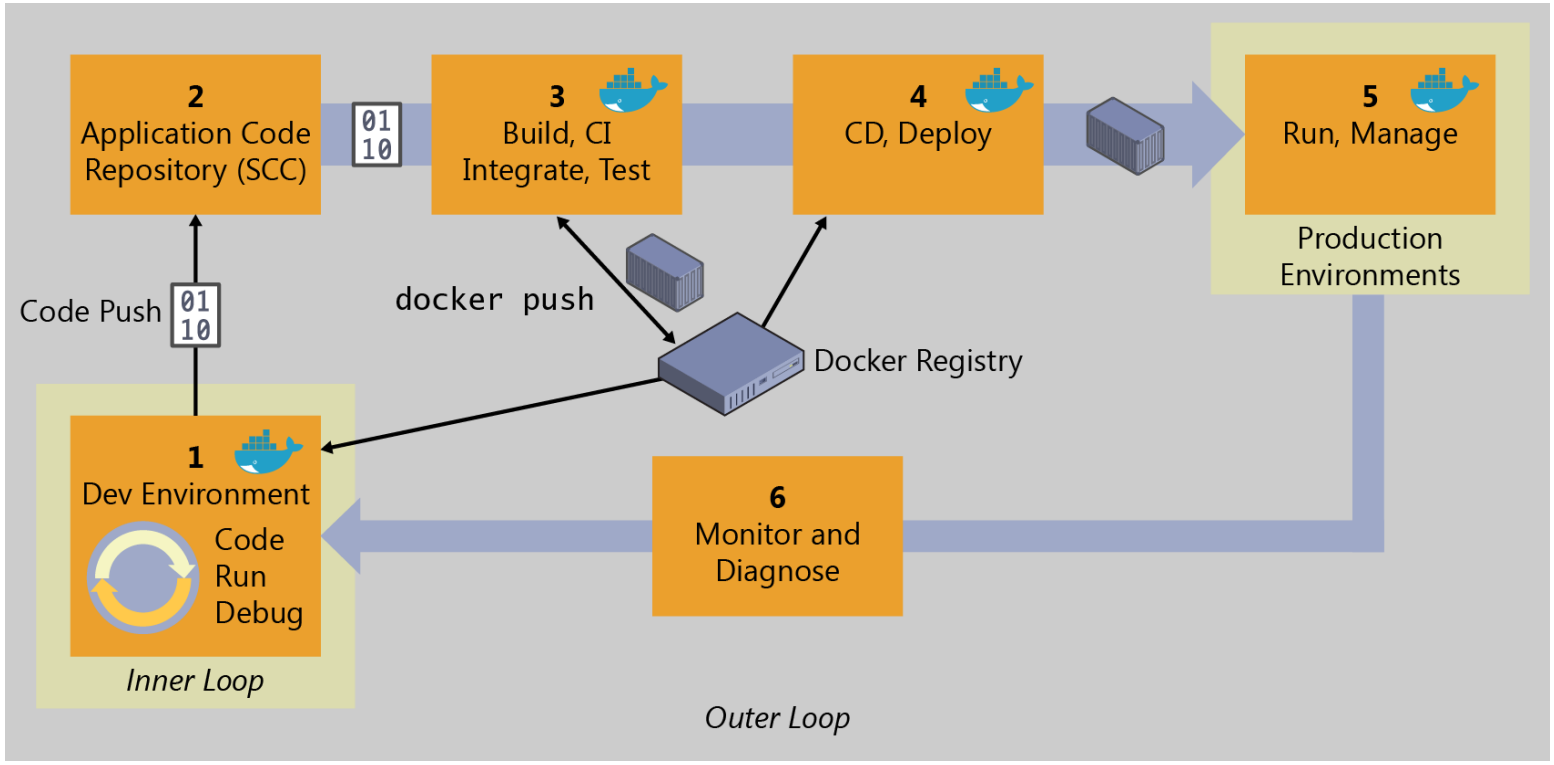
docker build

docker push

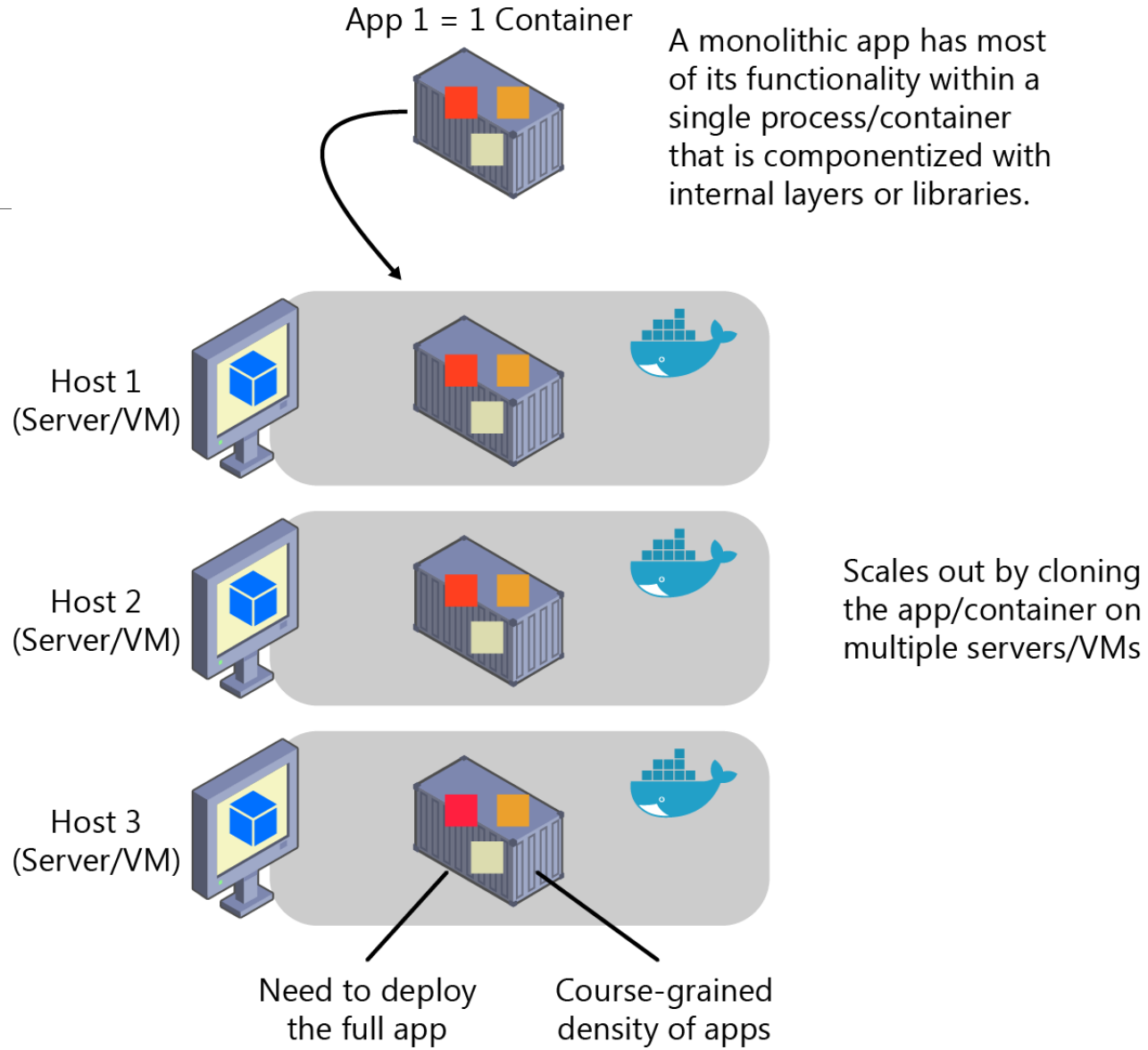

Registry
Stores many static images
App 1
App 2
.NET
Node
Images
Static, persisted container image
App
Node
Microsoft .NET
Container
Image instance running an app process (service/web)
App
Hosted Docker Registry
Docker Trusted Registry On-Premises
On-Premises
("n" private organizations)
Hosted Hub Registry
Docker Trusted Registry On-Cloud
Azure Container Registry
AWS Container Registry
Google Container Registry
Quay Registry
Other Cloud
Public Cloud
(specific vendors)

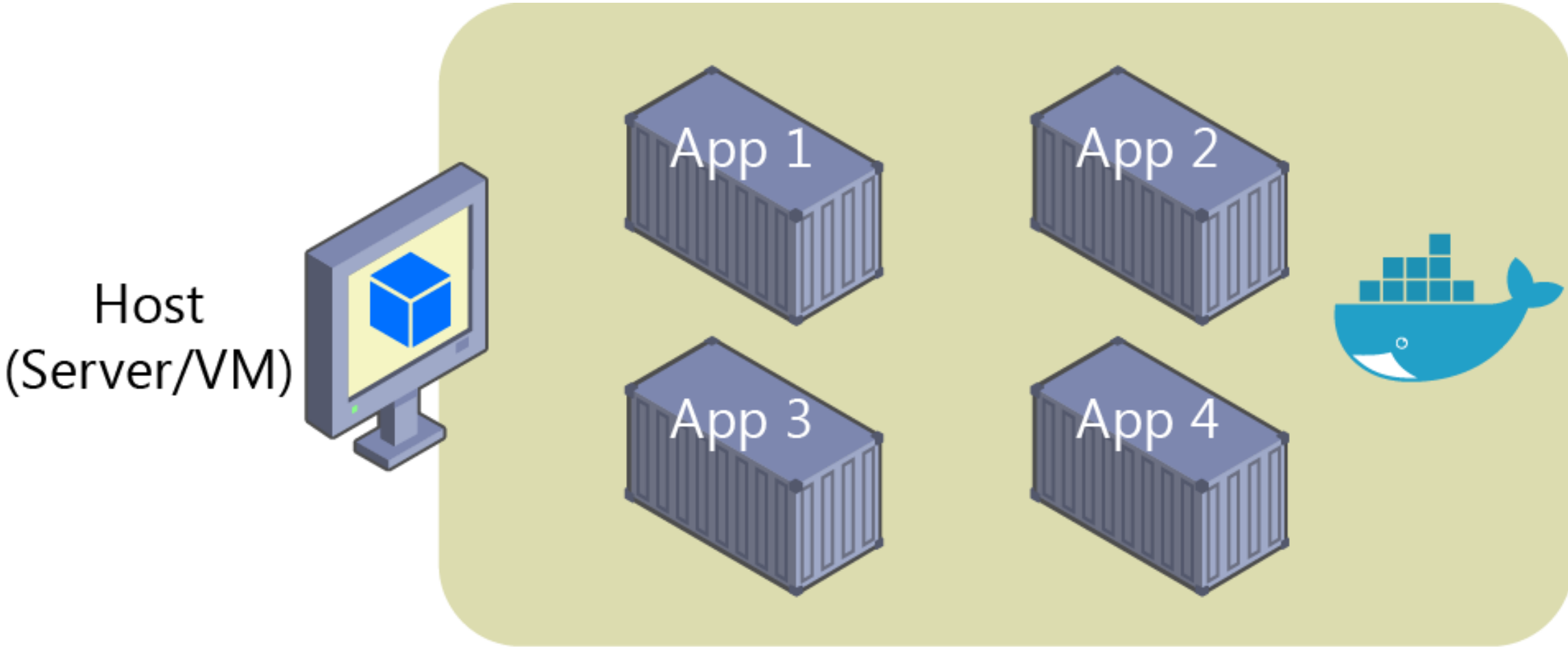

Developers and Architects
Collaboration
IT Operations
Platform for Docker Apps
DevOps for Docker Apps
Management and Monitoring
Develop/Design
Build/Test/Ship
Run/Monitor/Manage

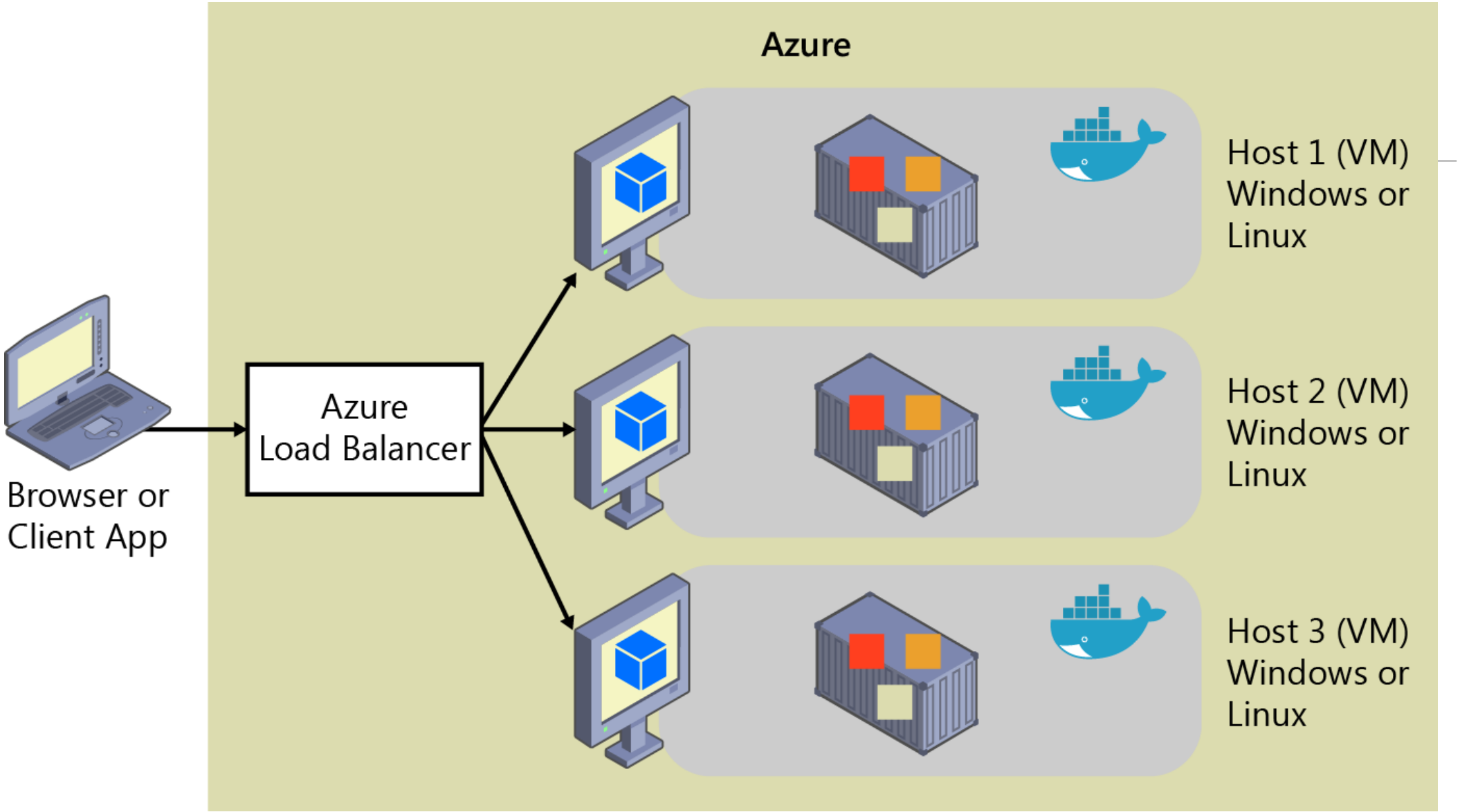
Workflow


2
Application Code
Repository (SCC)
01
10
3
Build, CI
Integrate, Test
4
CD, Deploy
5
Run, Manage
Production
Environments
Code Push
01
10
docker push
Docker Registry
1
Dev Environment
Code
Run
Debug
Inner Loop
6
Monitor and
Diagnose
Outer Loop

Workflow


App 1 = 1 Container
A monolithic app has most of its functionality within a single process/container that is componentized with internal layers or libraries.
Host 1 (Server/VM)
Host 2 (Server/VM)
Host 3 (Server/VM)
Scales out by cloning the app/container on multiple servers/VMs
Need to deploy the full app
Course-grained density of apps


Host
(Server/VM)
App 1
App 2
App 3
App 4


Azure
Azure
Load Balancer
Browser or
Client App
Host 1 (VM)
Windows or
Linux
Host 2 (VM)
Windows or
Linux
Host 3 (VM)
Windows or
Linux

Orchestration

Orchestration

Container orchestration automates the deployment, management, scaling, and networking of containers

Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it.

Microservices in containers make it easier to orchestrate services, including storage, networking, and security.

Orchestration - Why?

Automate the following tasks at scale:

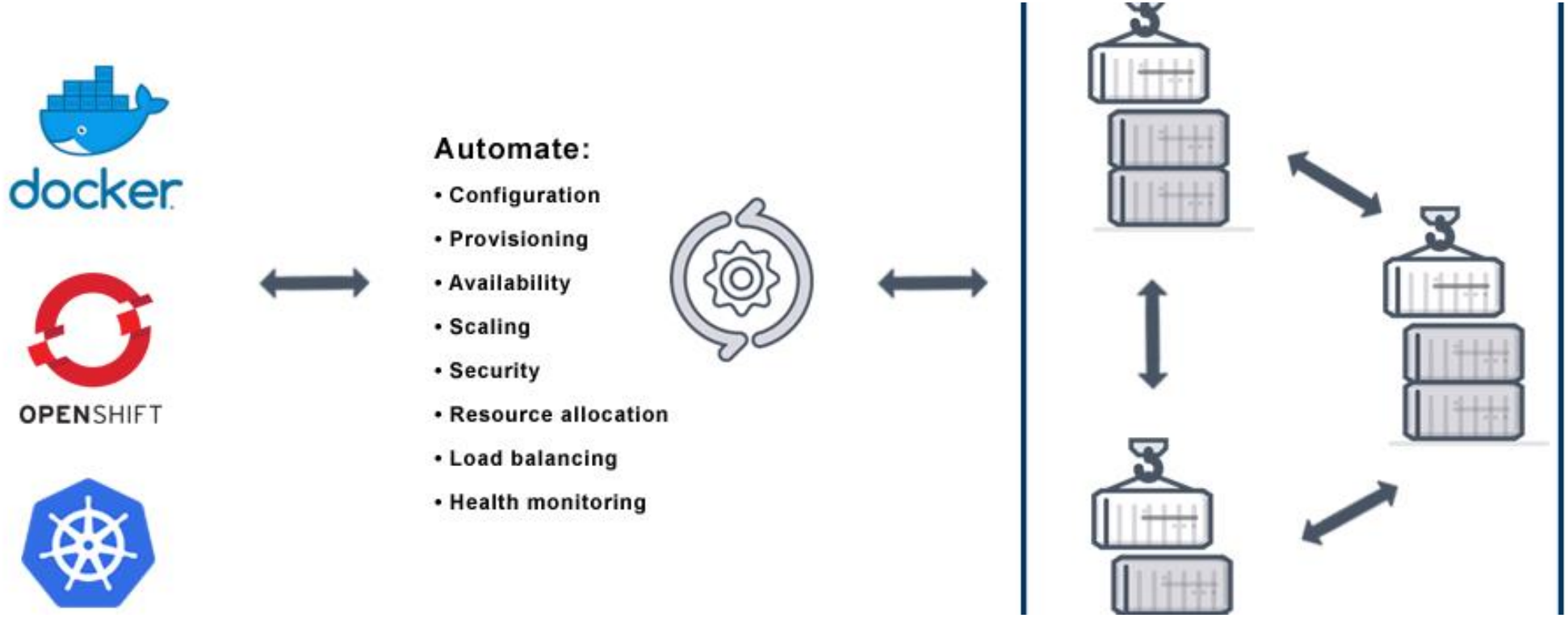
- Configuring and scheduling of containers
- Provisioning and deployments of containers
- Availability of containers
- The configuration of applications in terms of the containers that they run in
- Scaling of containers to equally balance application workloads across infrastructure
- Allocation of resources between containers
- Load balancing, traffic routing and service discovery of containers
- Health monitoring of containers
- Securing the interactions between containers.

Orchestration choices

Docker Swarm
Build into Docker, simple/fast orchestration

Mesos
Very powerful. Containerized apps and vms side-by-side
Paypal, Twitter & Uber use it.

Kubernetes
Extremely portable. Cloud provider integrations.
Originally by Google (Borg project), CNCF main project, backed by IBM,Amazon,Microsoft,Intel,Cisco, RedHat



Orchestration

Docker

Docker Compose

Docker Swarm / Stack

Docker Machine

docker-compose.yml

```yaml
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

Docker Stack

Set of interrelated services

Orchestrated & scaled together

SWARM

1.12.0 and up

Create a cluster (swarm) from Docker CLI

Different roles (set on runtime)

- Managers
- Workers

Scaling

Failover

Multi-host network (vnet over machines)

Service discovery

TLS auth/sec

Docker Swarm - Example components

```
docker swarm init
docker stack deploy -c docker-compose.yml <name>
docker service ls
docker service inspect <name>
docker service scale <name>=replica_count
docker network ls
docker network inspect <name>
```

Local registry in Swarm

```
docker service create --name registry --publish 5000:5000 registry:2
```

Management Tools

Consul

ZooKeeper (SmartStack=+HAProxy, Nerve, Synapse)

Etcd

Serf

Etc etc etc

Kubernetes

Manages container-based applications

- Along with networking and storage requirements
- Focused on application workloads instead of infrastructure components

Makes it easier to orchestrate large solutions using a variety of containers

- Application containers
- Storage containers
- Middleware containers
- Even more…

Applications are described declaratively

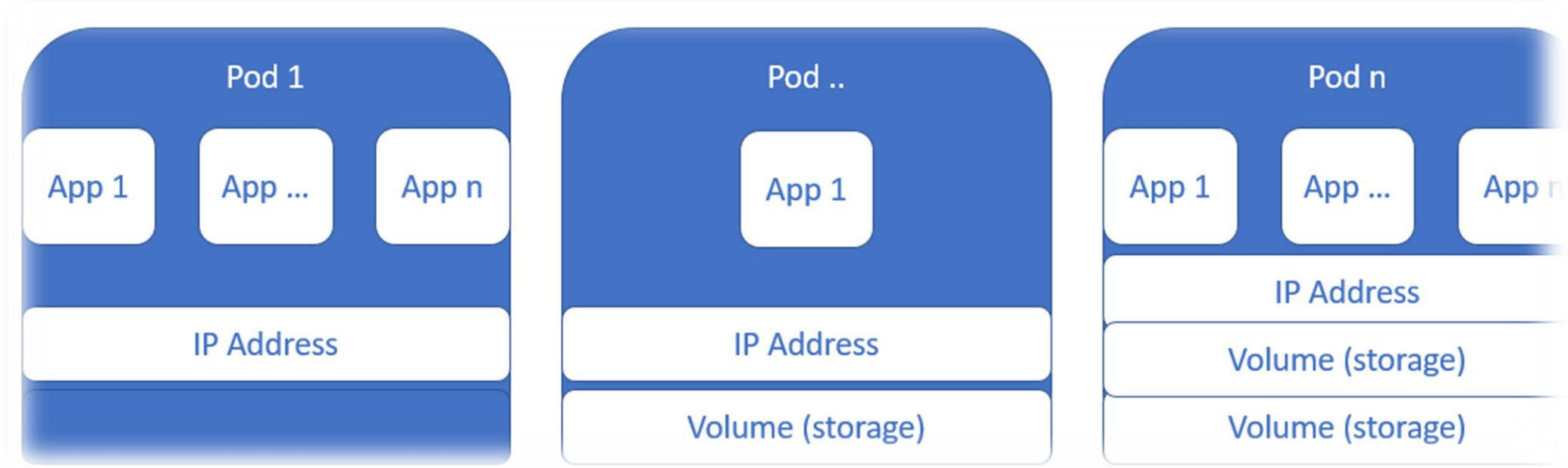
- Use YAML files to describe application
- Kubernetes handles management and deployment

Kubernetes terminology

Nodes	• Individual VM running containerized applications
Pools	• Groups of nodes with identical configurations
Pods	• Single instance of an application • It's possible for a pod to contain multiple containers within the same node
Deployments	• One or more identical pods managed by Kubernetes
Manifests	• YAML file describing a deployment

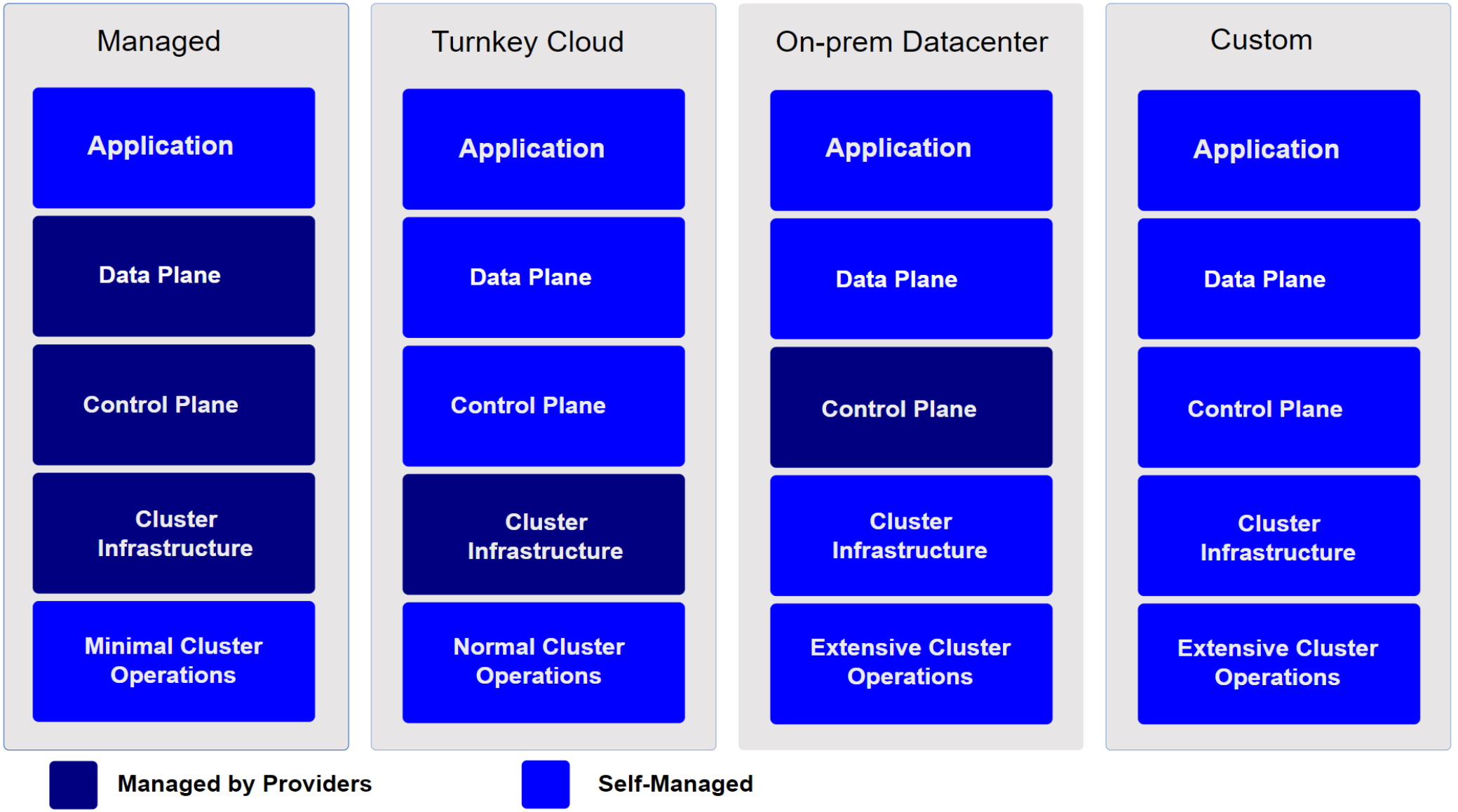
Pods and services

Kubernetes concept of a pod, which is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed

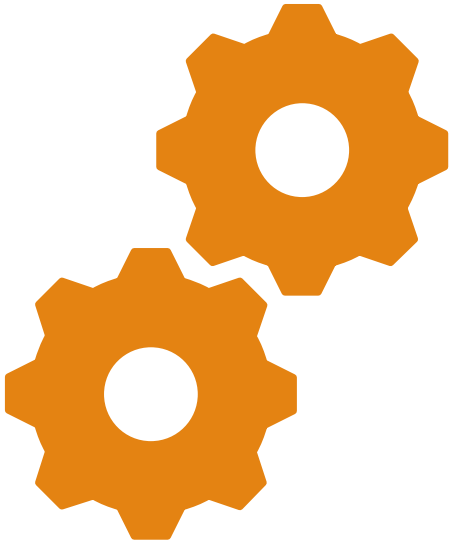


Managed Kubernetes

- Amazon EKS
- Azure Kubernetes Service (AKS)
- Digital Ocean
- Google Kubernetes Engine (GKE)
- IBM Cloud Kubernetes Service


Managed
Application
Data Plane
Control Plane
Cluster Infrastructure
Minimal Cluster Operations
Turnkey Cloud
Application
Data Plane
Control Plane
Cluster Infrastructure
Normal Cluster Operations
On-prem Datacenter
Application
Data Plane
Control Plane
Cluster Infrastructure
Extensive Cluster Operations
Custom
Application
Data Plane
Control Plane
Cluster Infrastructure
Extensive Cluster Operations
Managed by Providers
Self-Managed

Wrap up



Microservices provide agility and scalability, as well as the ability to independently develop and deploy services and easily add new features

Containers are a way to package and deploy microservices

Docker is an industry standard way of building container images and operating containers

Kubernetes is an open-source container orchestration platform used to automate the deployment, scaling, and management of containerized applications