Microsoft

# Project Phoenix

App modernization using K8s on Azure
github.com/CSA-OCP-GER/phoenix

# Implementation challenges

Cost

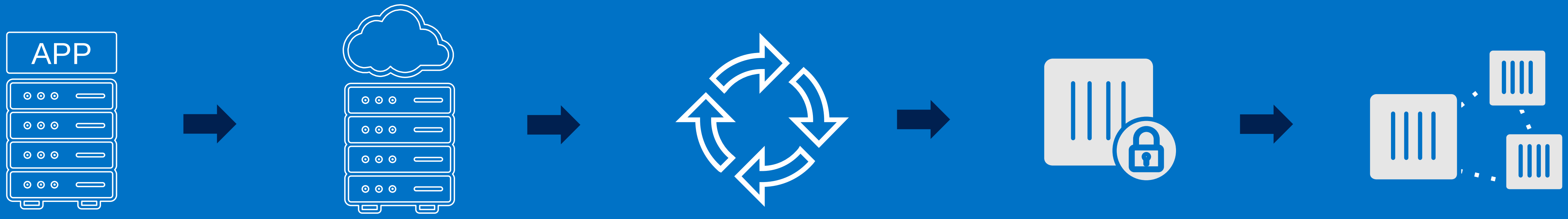
Scale

Complexity

Agility

Security

# From traditional app to modern app



**Existing Application**

**Modern Infrastructure**

Move to the cloud as VMs or Containers or refresh HW.

**Modern Methodologies**

Implement CI/CD and automation.

**Containerize Applications**

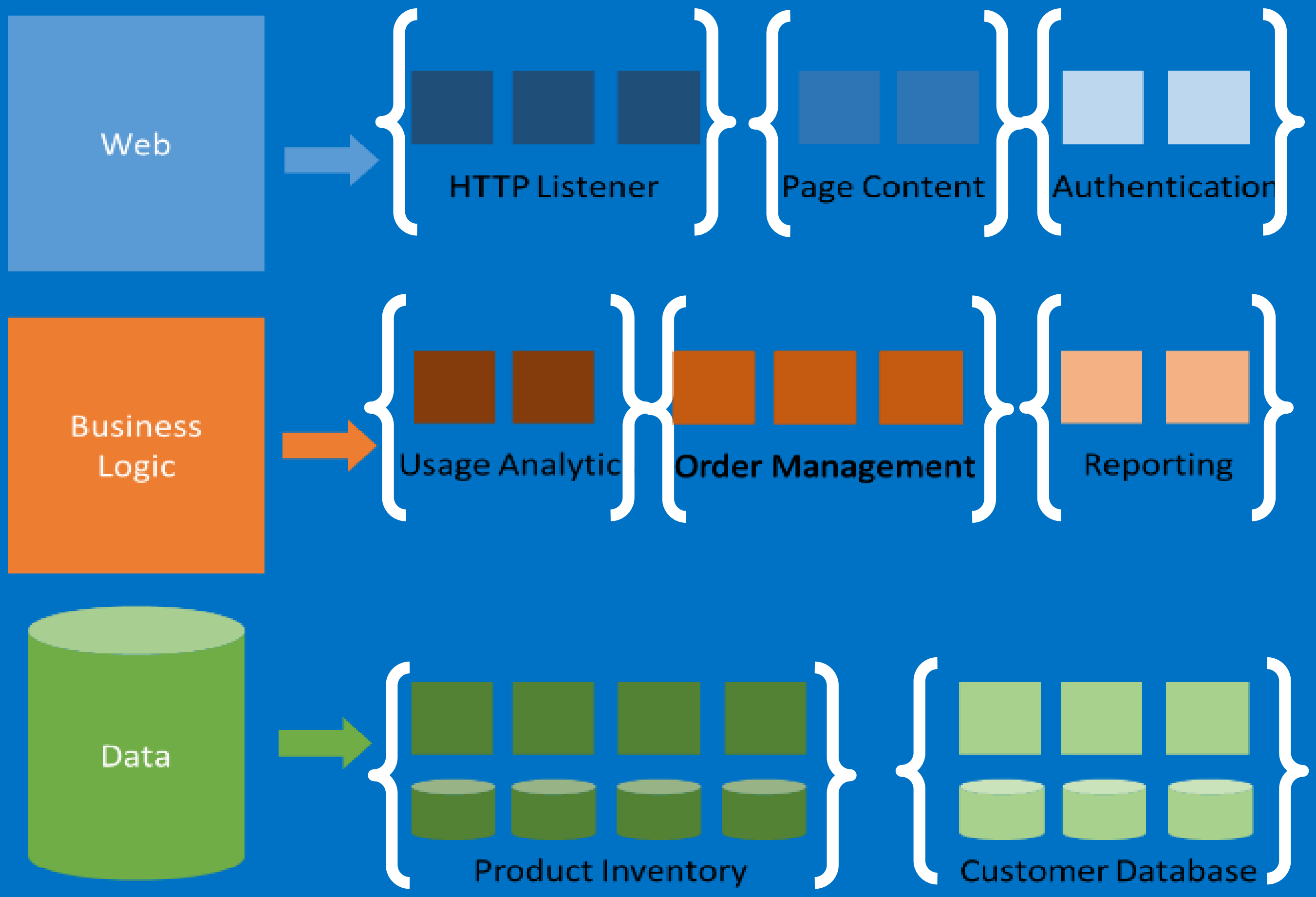
Re-architect apps for scale with containers.

**Modern Microservices**

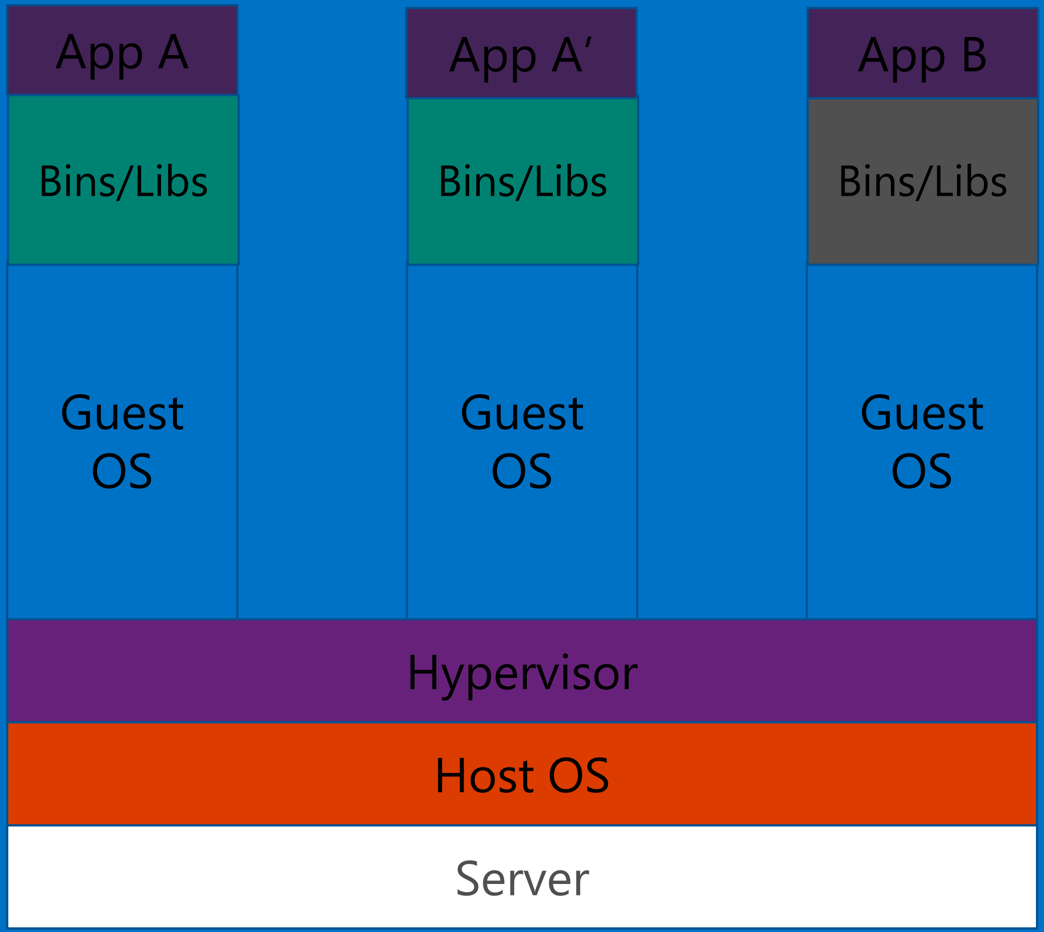
Add new services or start peeling off services from monolithic code.

# Microservices as an architecture pattern

- Individually built and deployed
- Small, independently services
- Integrate using published API
- Fine-grained, loosely coupled

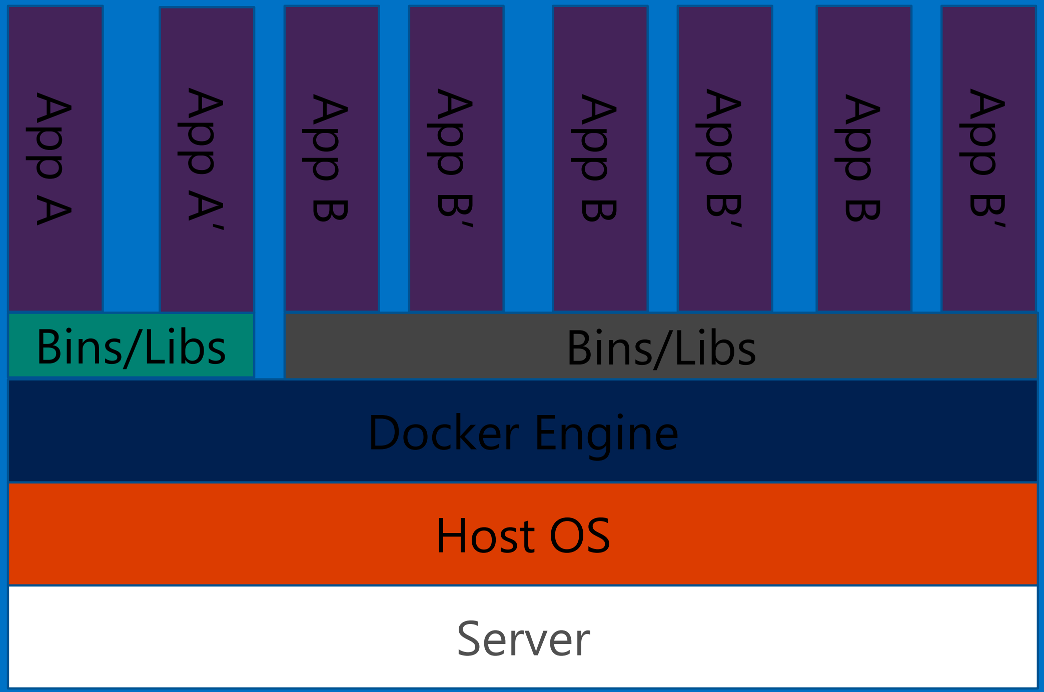


# Container 101



App A
App A’
App B
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host OS
Server

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



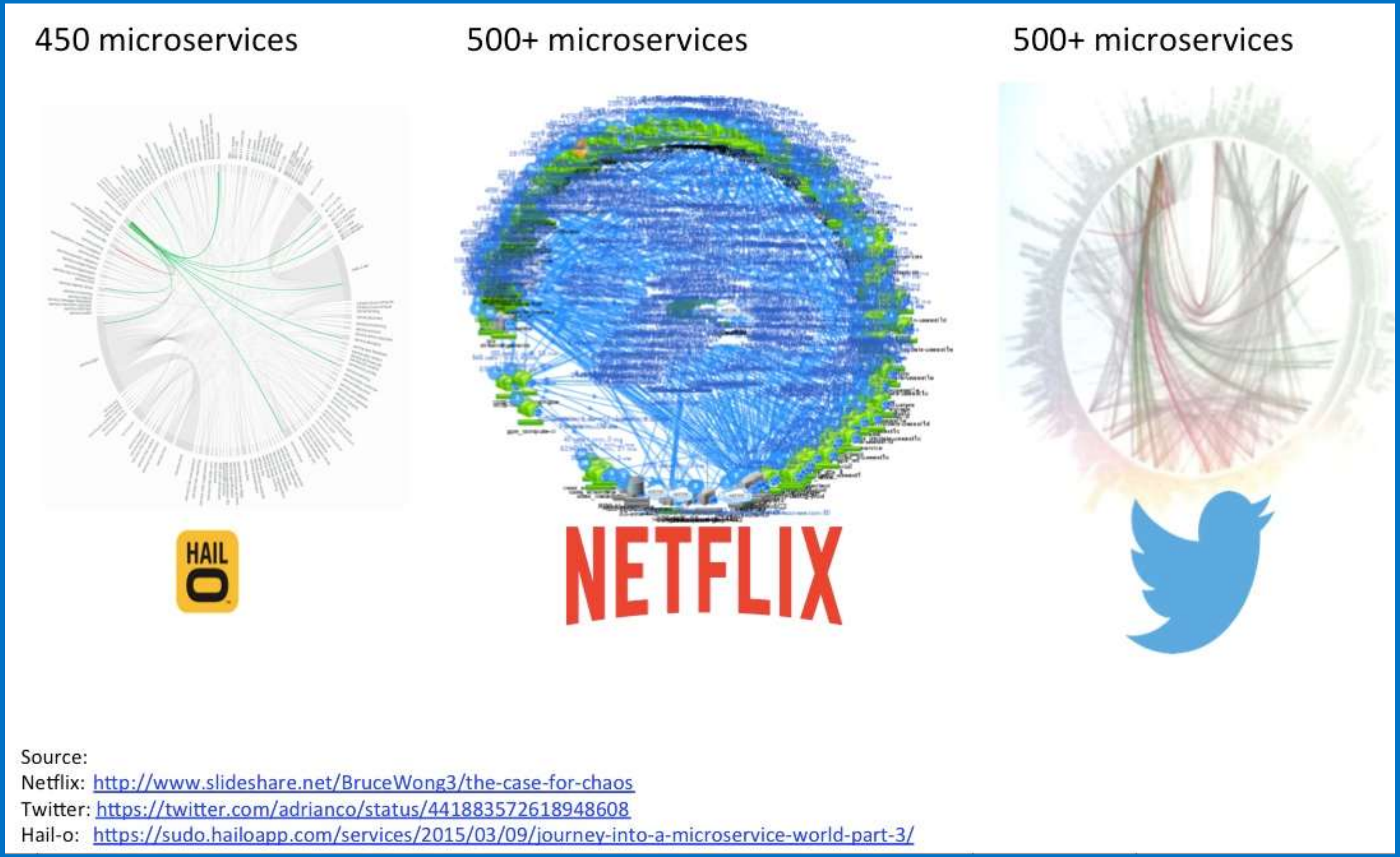
App A
App A’
App B
App B’
App B
App B’
App B
App B’
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Server

# microservices ≠ containers

microservices is an architectural design approach

containers are an implementation detail that often helps

# Scaling microservices is hard



450 microservices

500+ microservices

500+ microservices

Source:
Netflix: http://www.slideshare.net/BruceWong3/the-case-for-chaos
Twitter: https://twitter.com/adrianco/status/441883572618948608
Hail-o: https://sudo.hailoapp.com/services/2015/03/09/journey-into-a-microservice-world-part-3/

# Orchestrators = containers at scale

| | | | | |
|---|---|---|---|---|
| **Cluster Management**<br>Deploy and manage cluster resources | **Scheduling**<br>When containers run | **Lifecycle & Health**<br>Keep containers running despite failure | **Naming & Discovery**<br>Where are my containers | **Load Balancing**<br>Distribute traffic evenly |
| **Scaling**<br>Make container sets elastic in number | **Image Repository**<br>Centralized, secure container images | **Continuous Delivery**<br>CI/CD pipeline and DevOps workflow | **Logging & Monitoring**<br>Track events in containers and cluster | **Storage Volumes**<br>Persistent data for containers |

# 12-Factor Apps (1-5)

1. Single root repo; don’t share code with another app
2. Deploy dependent libs with app
3. No config in code; read from environment vars
4. Handle unresponsive app dependencies robustly
5. Strictly separate build, release, & run steps
   - Build: Builds a version of the code repo & gathers dependencies
   - Release: Combines build with config ReleaseId (immutable)
   - Run: Runs app in execution environment

# 12-Factor Apps (6-12)

6. App executes as 1+ stateless process & shares nothing
7. App listens on ports; avoid using (web) host
8. Use processes for isolation; multiple for concurrency
9. Processes can crash/be killed quickly & start fast
10. Keep dev, staging, & prod environments similar
11. Log to stdout (dev=console; prod=file & archived)
12. Deploy & run admin tasks (scripts) as processes

# Azure Container support

Pivotal Cloud Foundry

RED HAT OPENSHIFT Container Platform

Mesos DC/OS

kubernetes

docker

---

Azure Container Instance

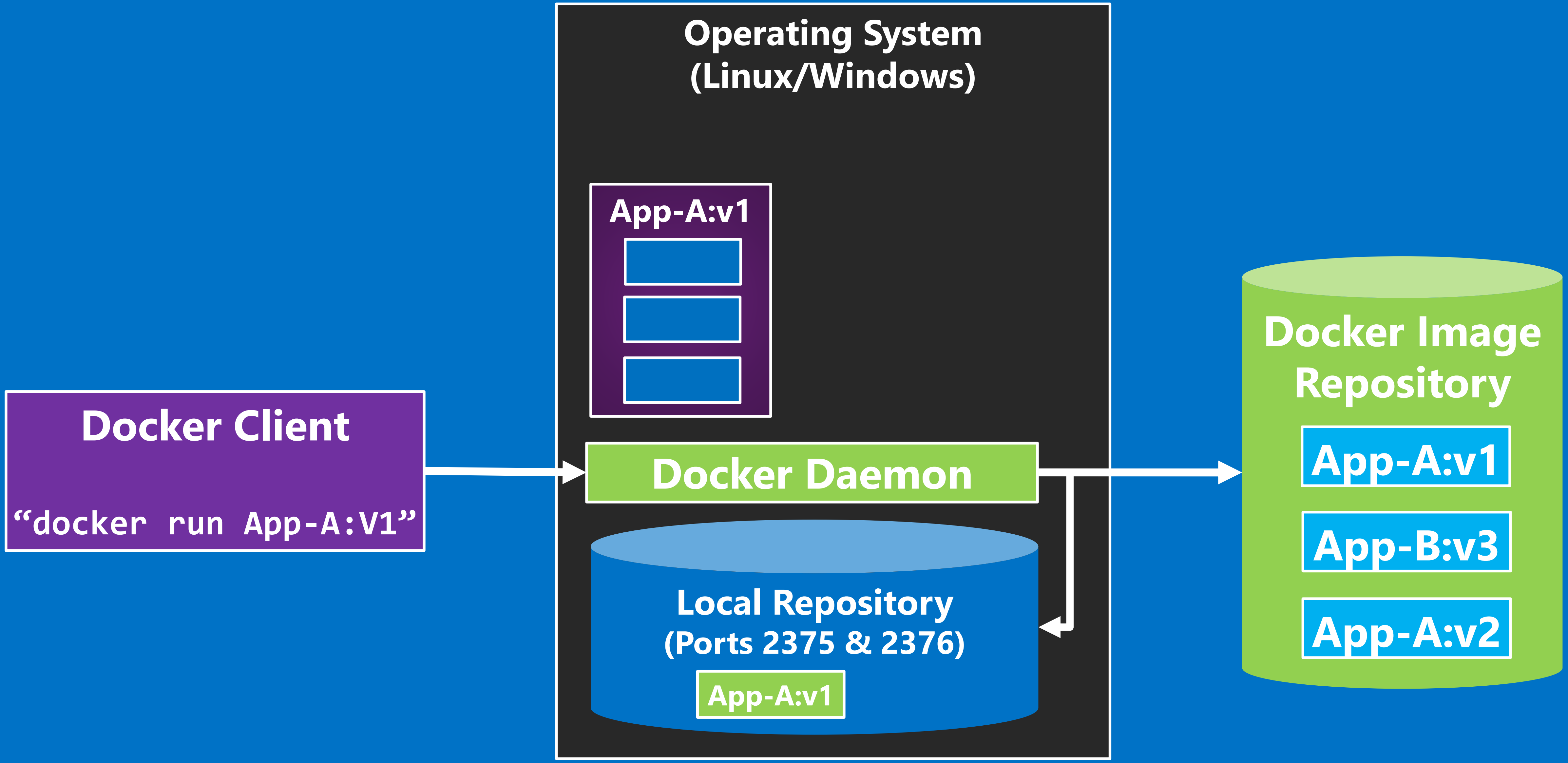
Container Service

Service Fabric

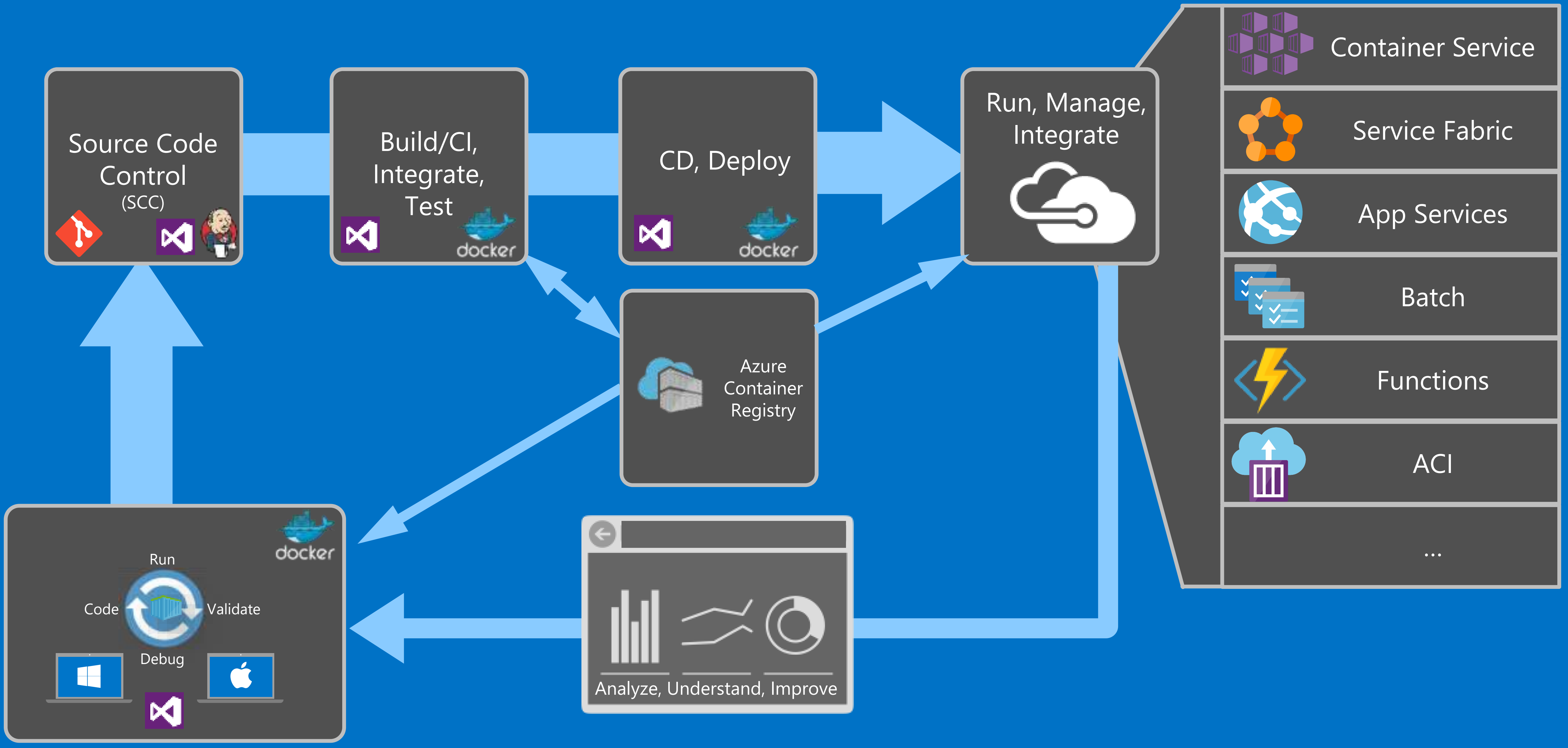
Web Apps

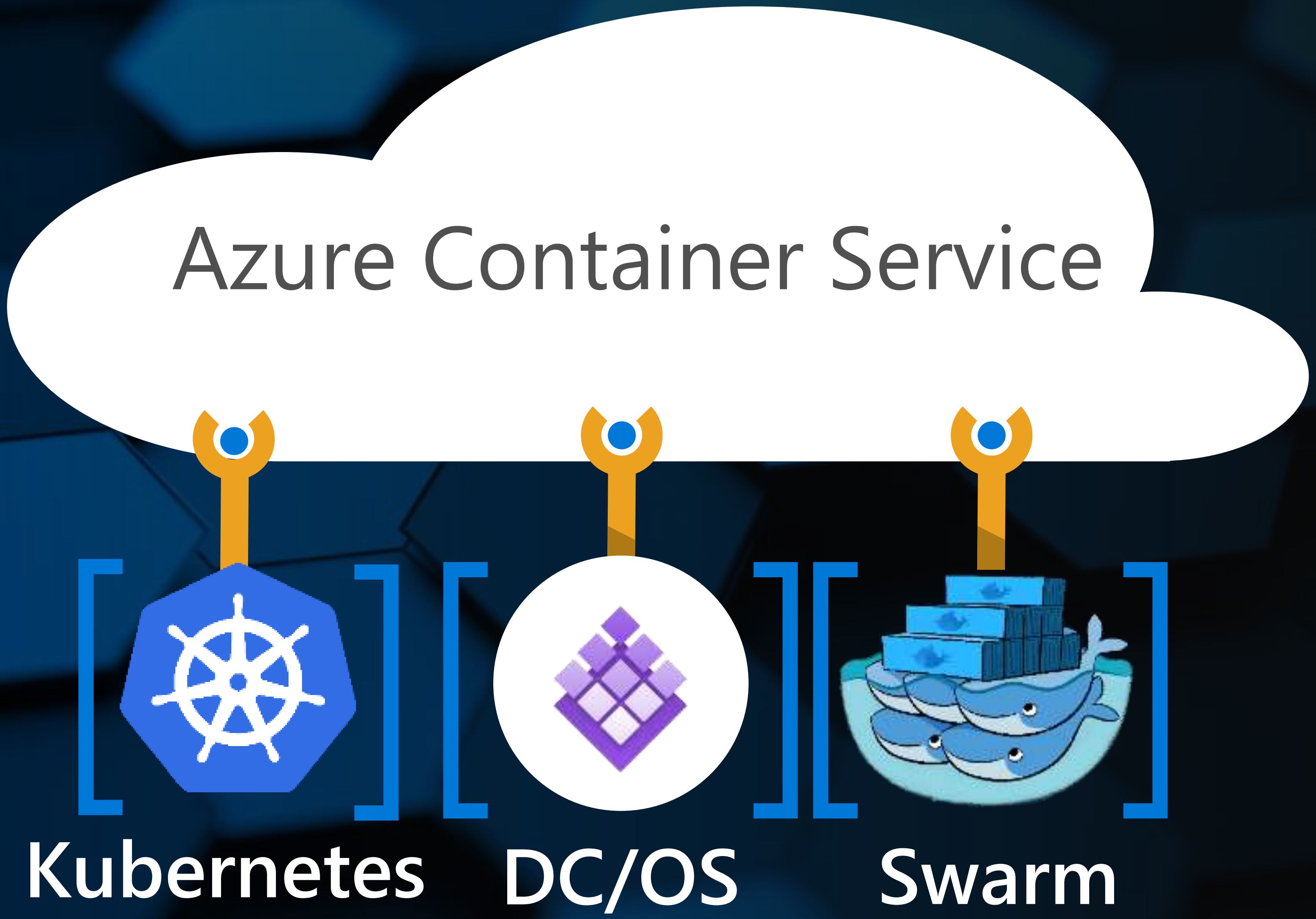
Batch

# ~$ docker run App-A:v1



# DevOps Pipeline





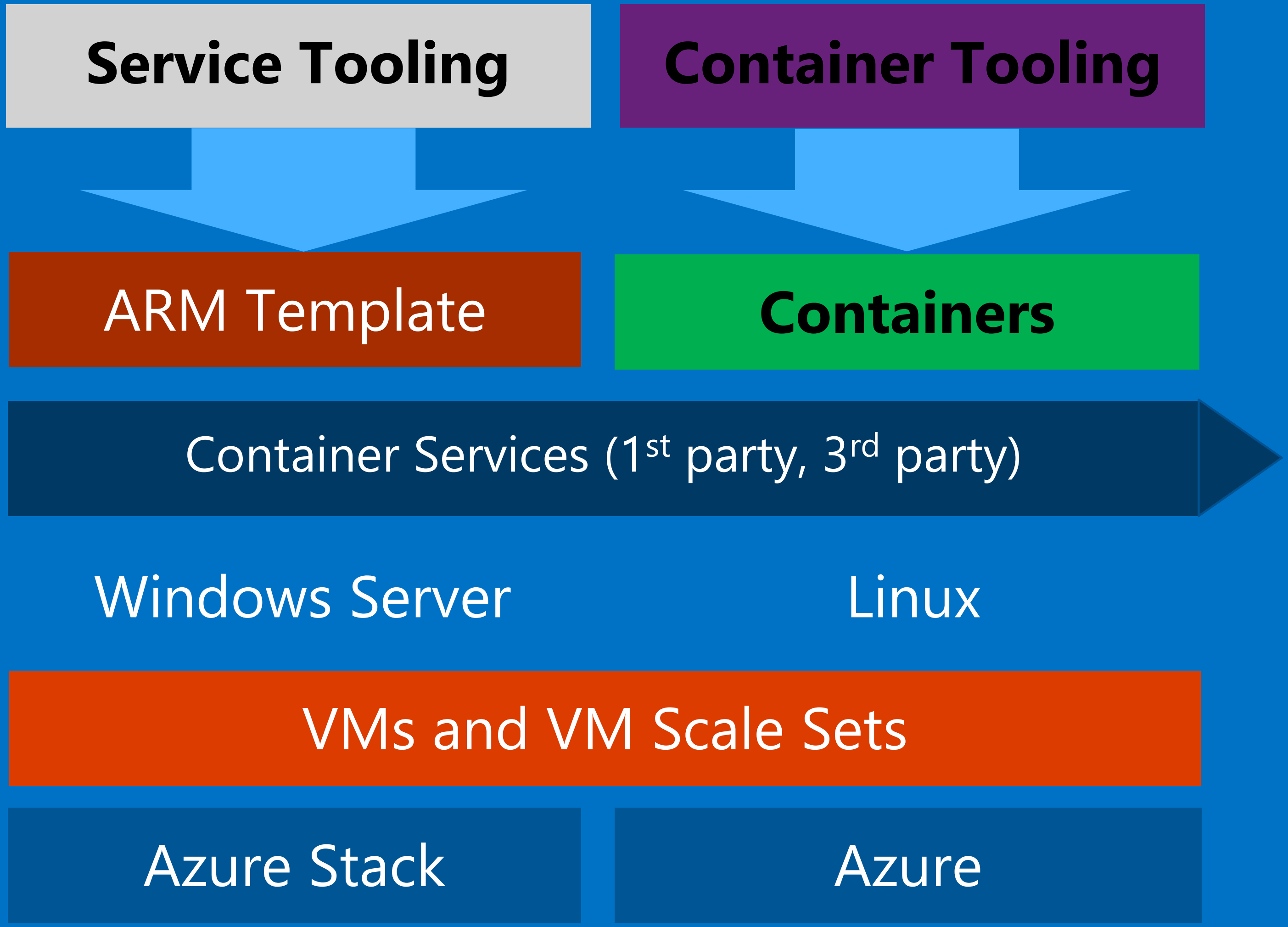
Azure Container Service

Kubernetes

DC/OS

Swarm

# Azure Container Services


Service Tooling

Container Tooling

ARM Template

Containers

Container Services (1st party, 3rd party)

Windows Server

Linux

VMs and VM Scale Sets

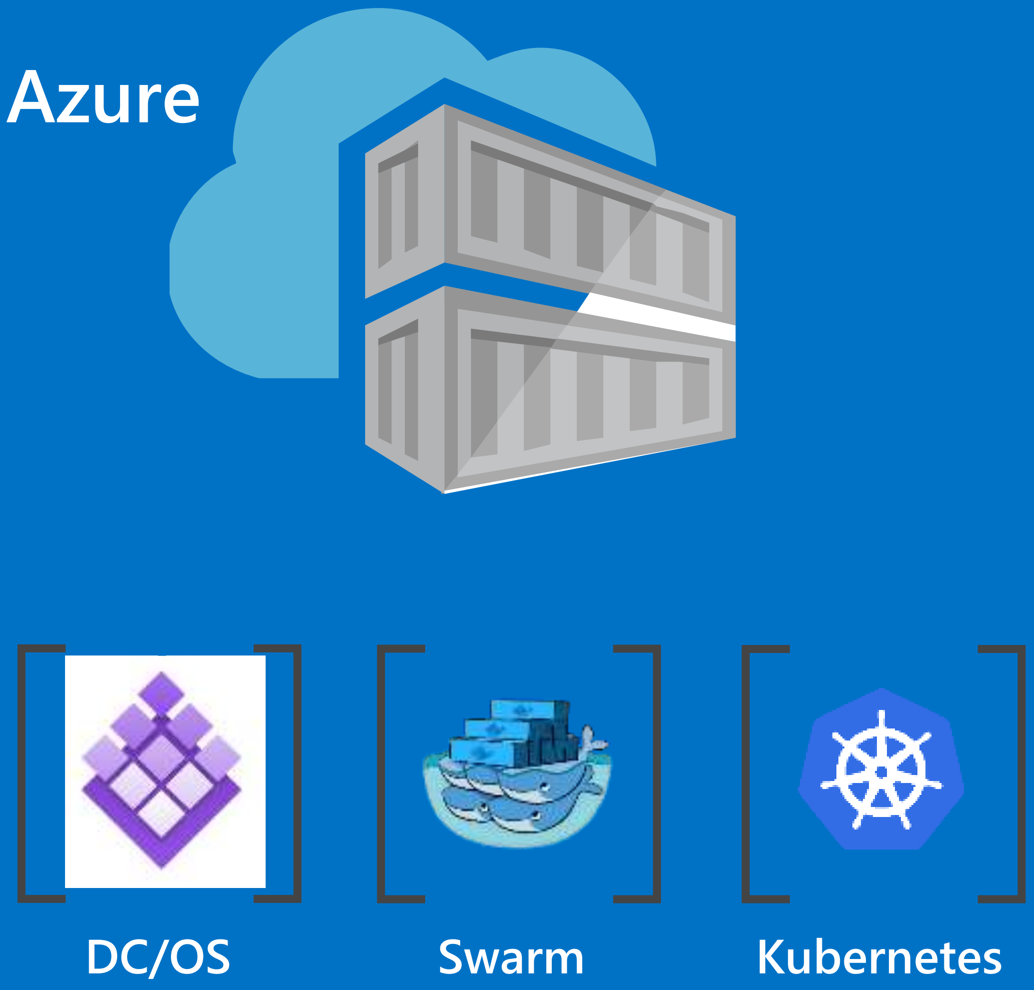
Azure Stack

Azure

| Layer | Supported Technologies |
|---|---|
| **Configuration as Code** | ARM, Dockerfile, Docker Compose, Marathon.json |
| **Host cluster management** | VM Scale Sets |
| **Container orchestration** | Docker Swarm, DC/ OS, Kubernetes |
| **Monitoring** | OMS, App Insights |

# Azure Container Service

- ✓ Container hosting solution optimized for Azure
- ✓ The cloud’s most open option for containers
- ✓ Manage container applications using familiar tools



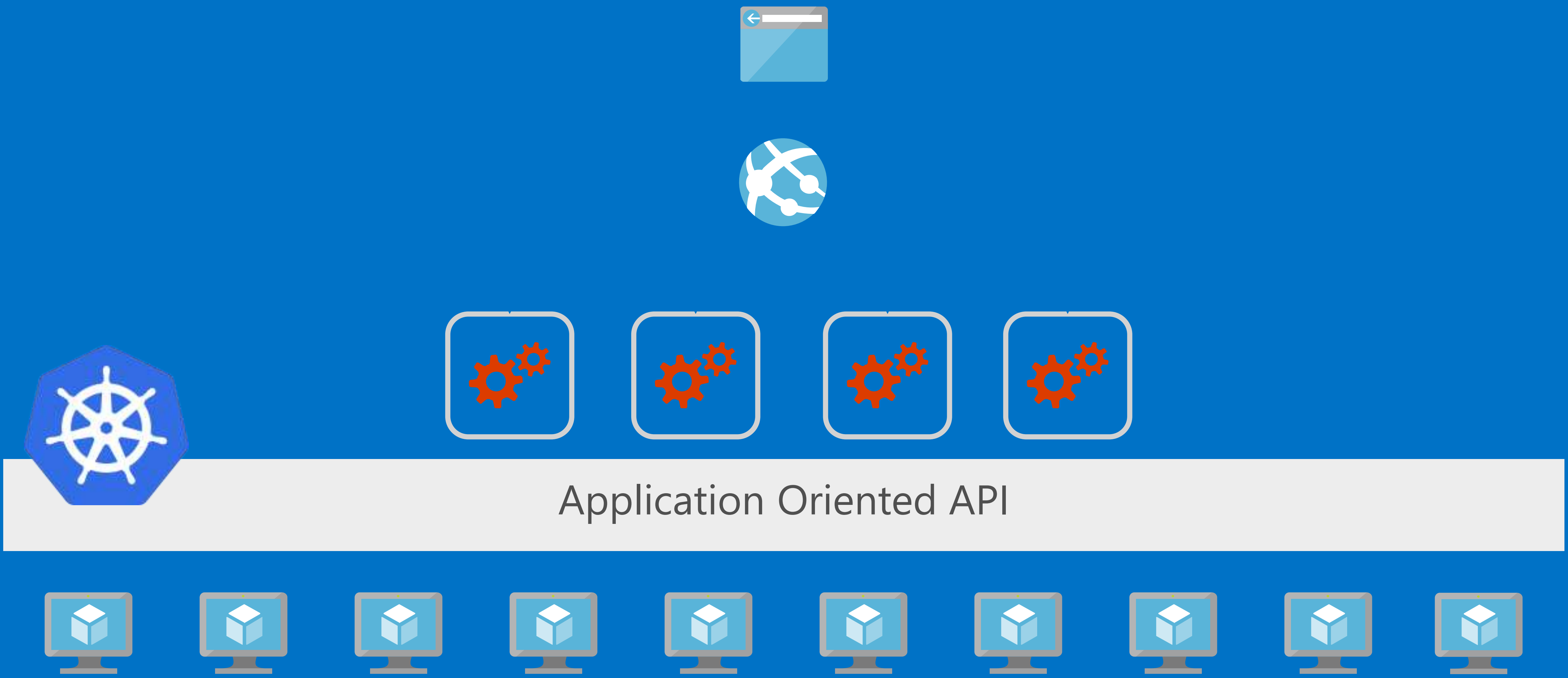
Azure

DC/OS

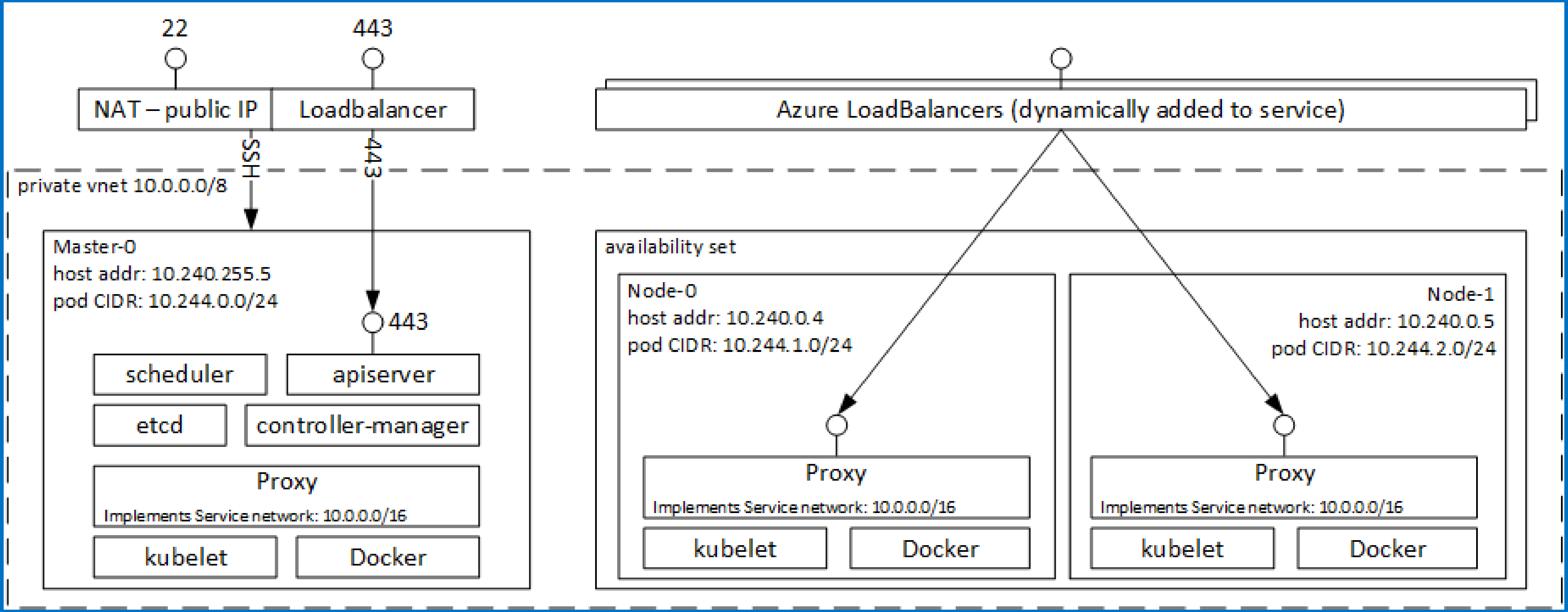
Swarm

Kubernetes

# Application design



# Kubernetes infrastructure setup on Azure



# Kubernetes Objects

- Pods
- Services
- Deployments
- Replica Sets
- …

- Defined via *.yml *.yaml file
- Desired state is specified via „Spec“ section
- Actual state is represented in „Status“ section

# Kubelet

Component which runs on each agent and manages the pod and container lifecycle

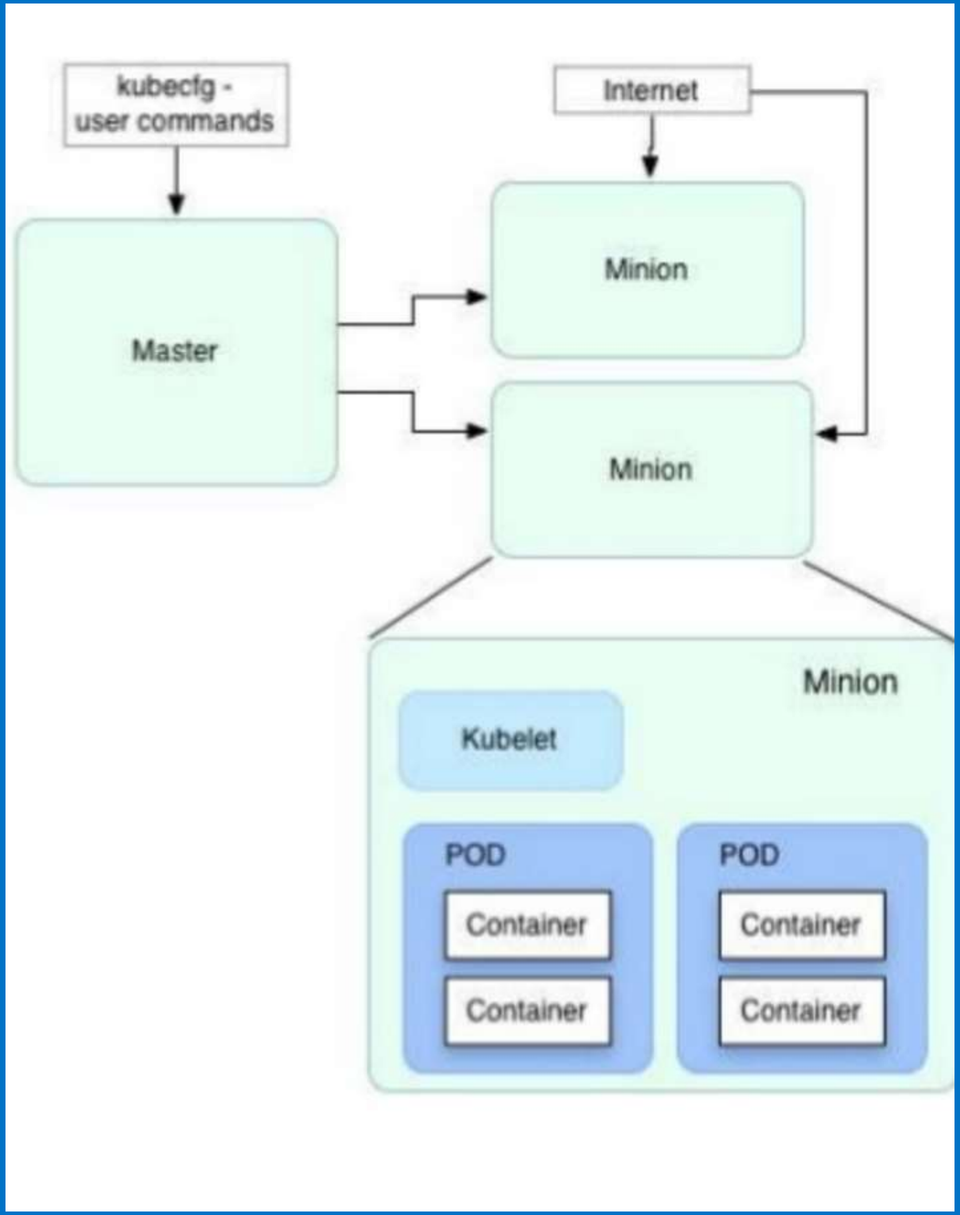
There is 1:1 mapping between a host and a kubelet

Key elements of a kubelet

- Docker client
- Root Directory
- Pod Workers
- Etcd client
- cAdvisor client

Functions performed by kubelet

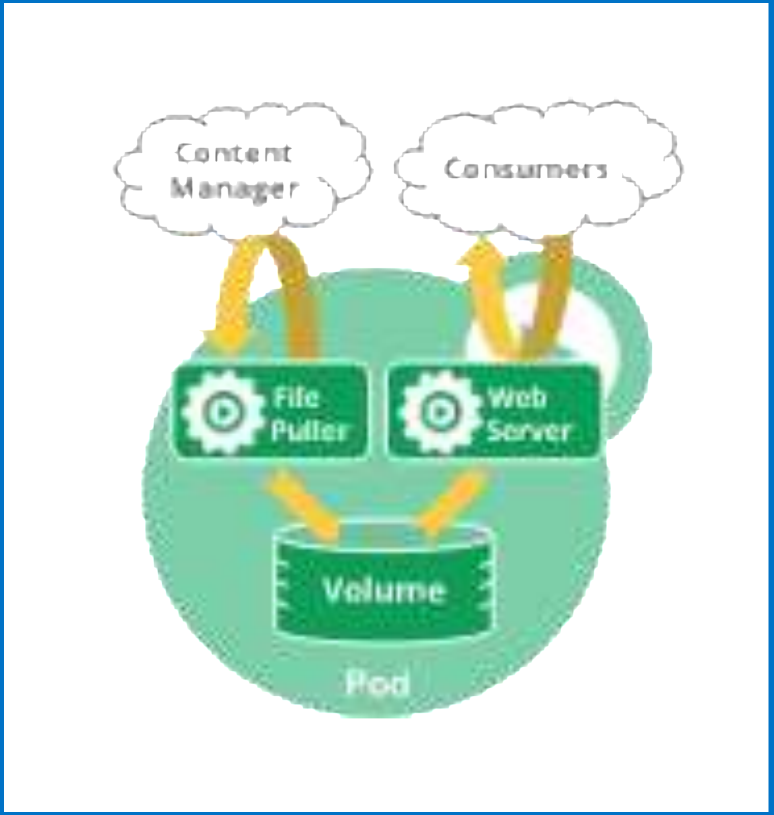
- Run containers in pods
- Bind volumes to pods
- Bind ports to container
- Kill container
- Collect health information for container



# Pod

Pod is a set of containers that can run on a host

- Pod represents a logical construct to bundle one or more applications together
  - Sidecar Model
  - Pod resources deployed to a single agent
- An application-specific "logical host"
- Share:
  - Storage resources
  - Unique network IP
- Options to control how container(s) should run
- Most commonly Docker, but other runtimes are supported

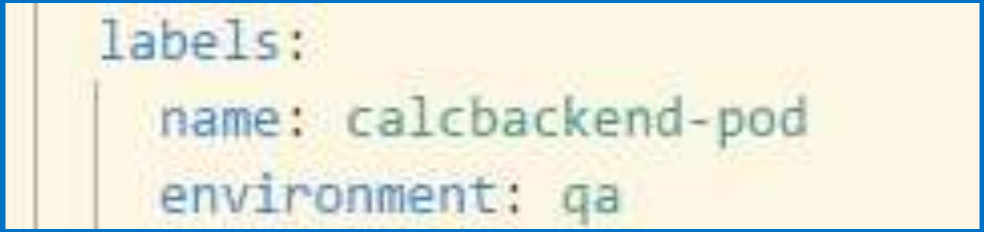


# Application Health and Readiness

- Kubelets on workers use liveness and readiness probes to know when to restart a container or start directing traffic
- Liveness check: when to restart
- Readiness check: when to forward Service traffic to a pod

# Labels & Selectors

- Labels
  - ***Identifying*** attributes on kubernetes objects
  - Key/value based



```
labels:
  name: calcbackend-pod
  environment: qa
```

Can be used for manual query and will be used by e.g. services

*> kubectl get pods -l environment=production,tier=frontend*

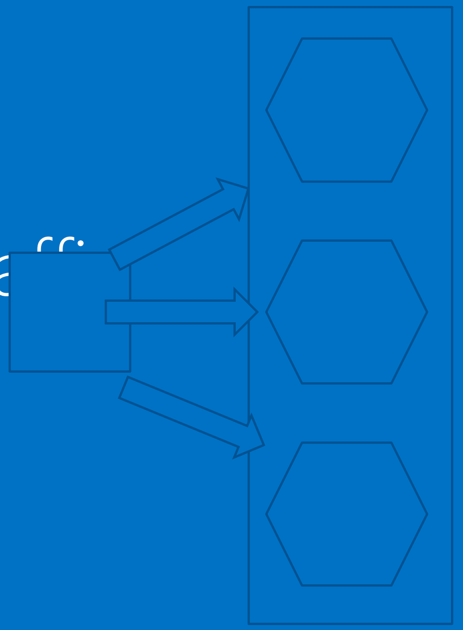
# Service

- Abstraction which defines a logical set of pods

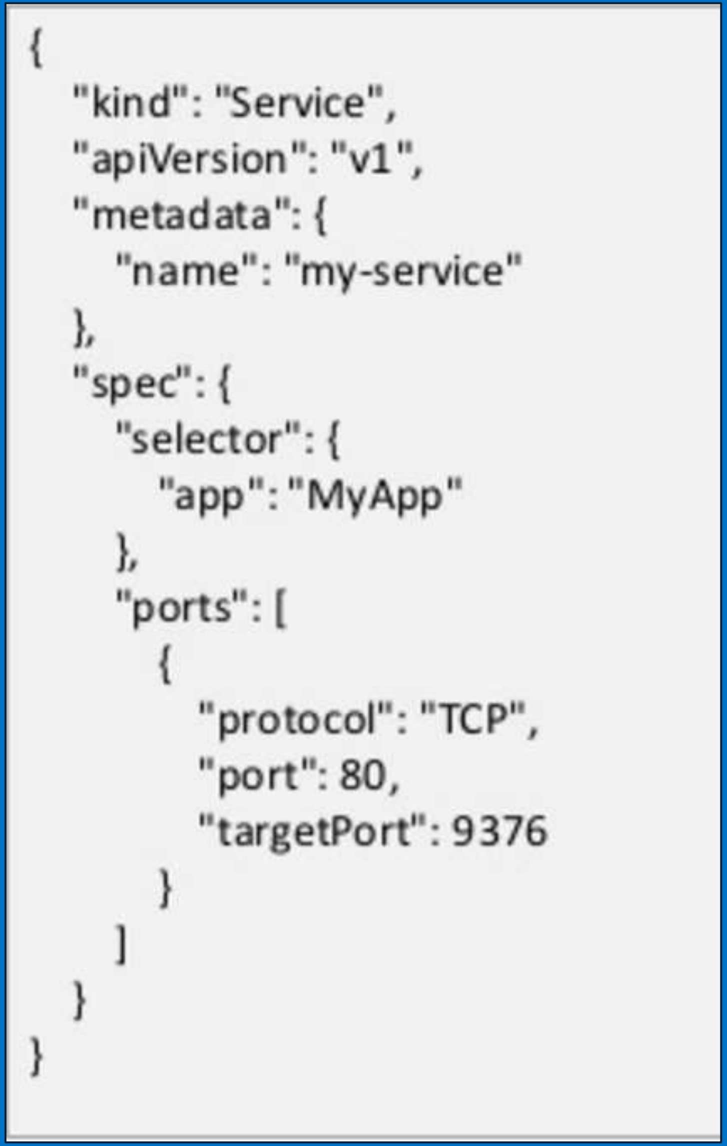
  Mortality of a single pod does **not** affect the availability of a service if the functionality of the service is provided by multiple (and not just one) pods

- Can be used as external interface with public IP

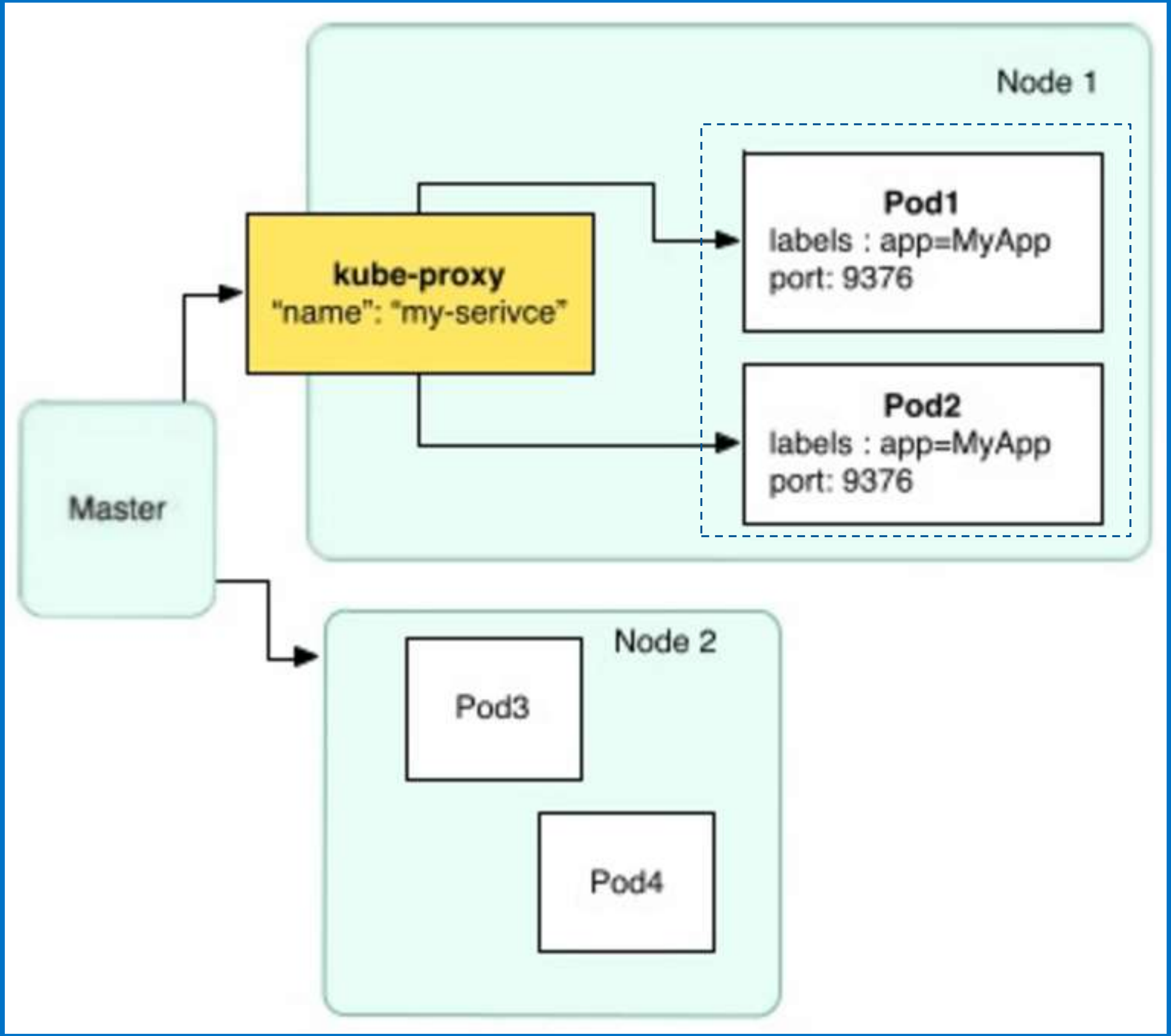
- Services decide via Selectors & Labels where to route traffi



# Services

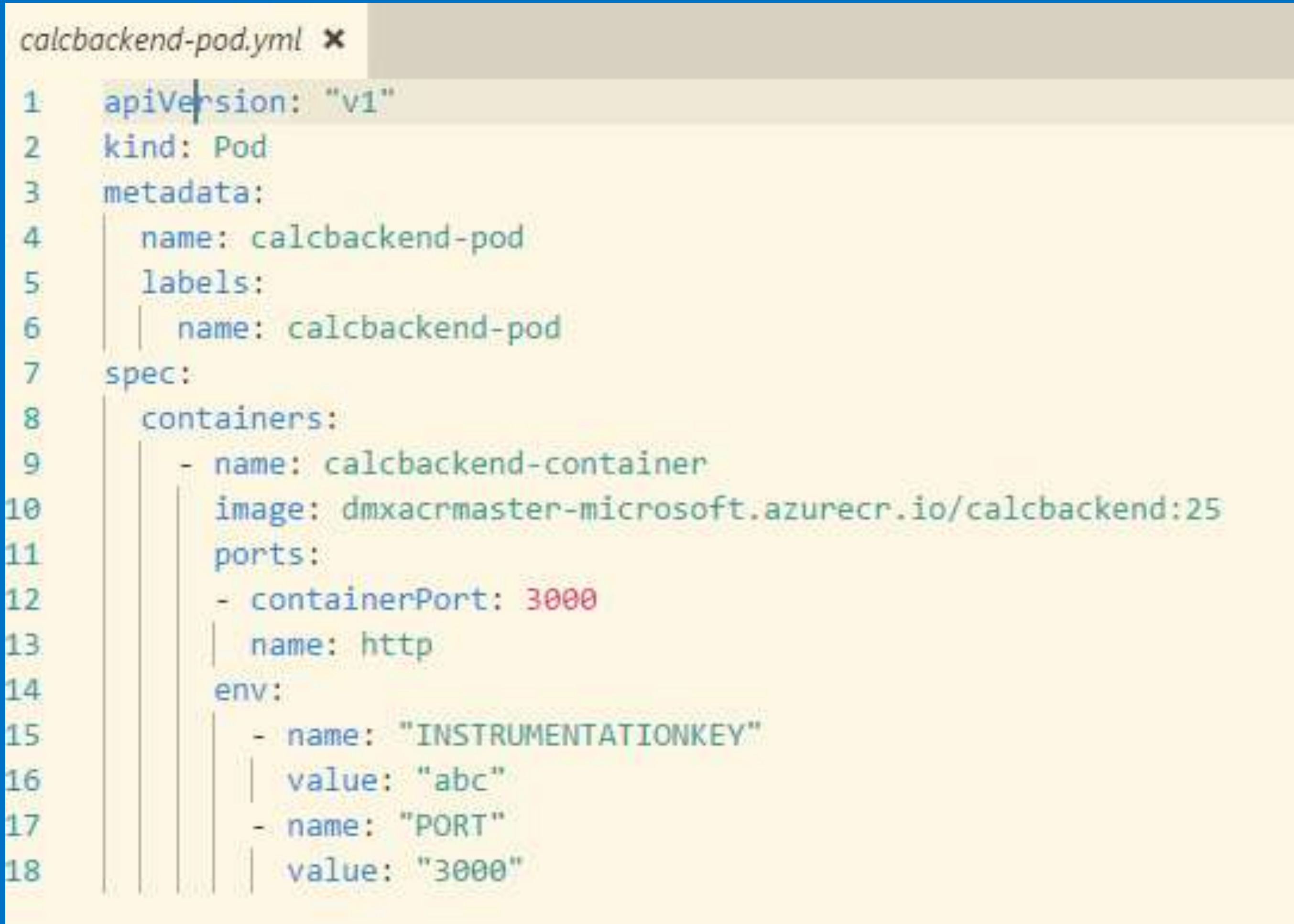


```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "my-service"
  },
  "spec": {
    "selector": {
      "app": "MyApp"
    },
    "ports": [
      {
        "protocol": "TCP",
        "port": 80,
        "targetPort": 9376
      }
    ]
  }
}
```



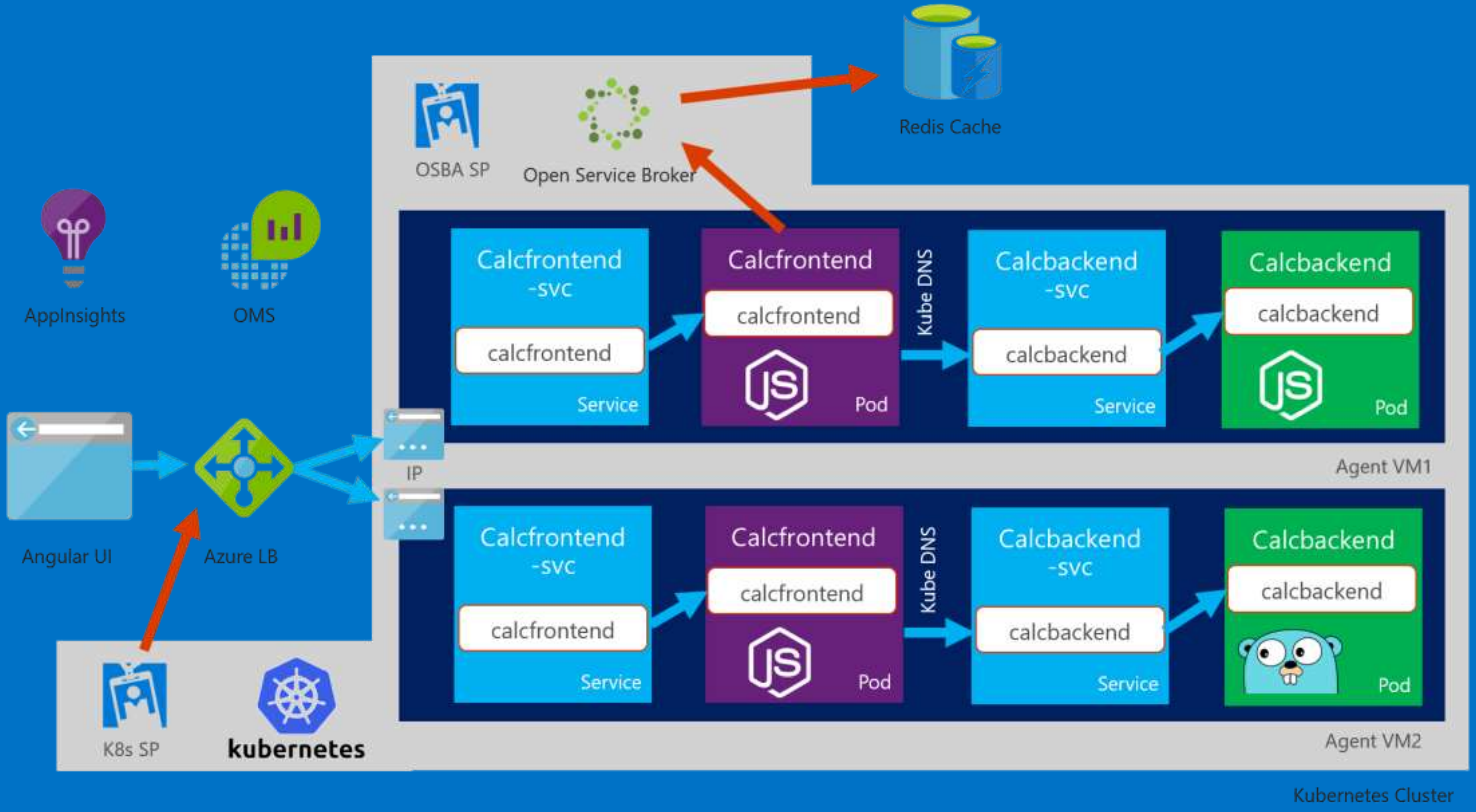
# Kubernetes API

- Create, Query and modify Kubernetes objects
- E.g.
  - kubectl apply –f file.yaml
  - kubeclt delete pods/mypod

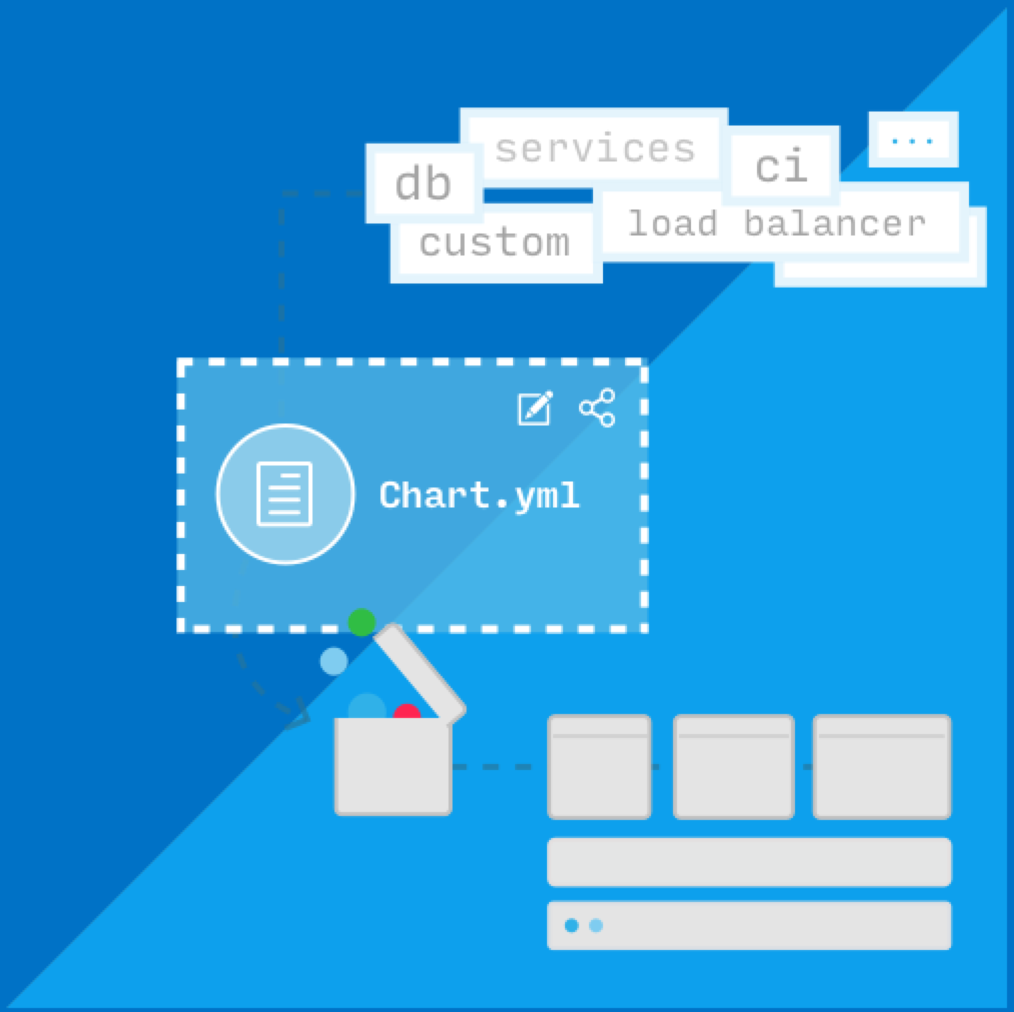


```yaml
apiVersion: "v1"
kind: Pod
metadata:
  name: calcbackend-pod
  labels:
    name: calcbackend-pod
spec:
  containers:
    - name: calcbackend-container
      image: dmxacrmaster-microsoft.azurecr.io/calcbackend:25
      ports:
      - containerPort: 3000
        name: http
      env:
        - name: "INSTRUMENTATIONKEY"
          value: "abc"
        - name: "PORT"
          value: "3000"
```

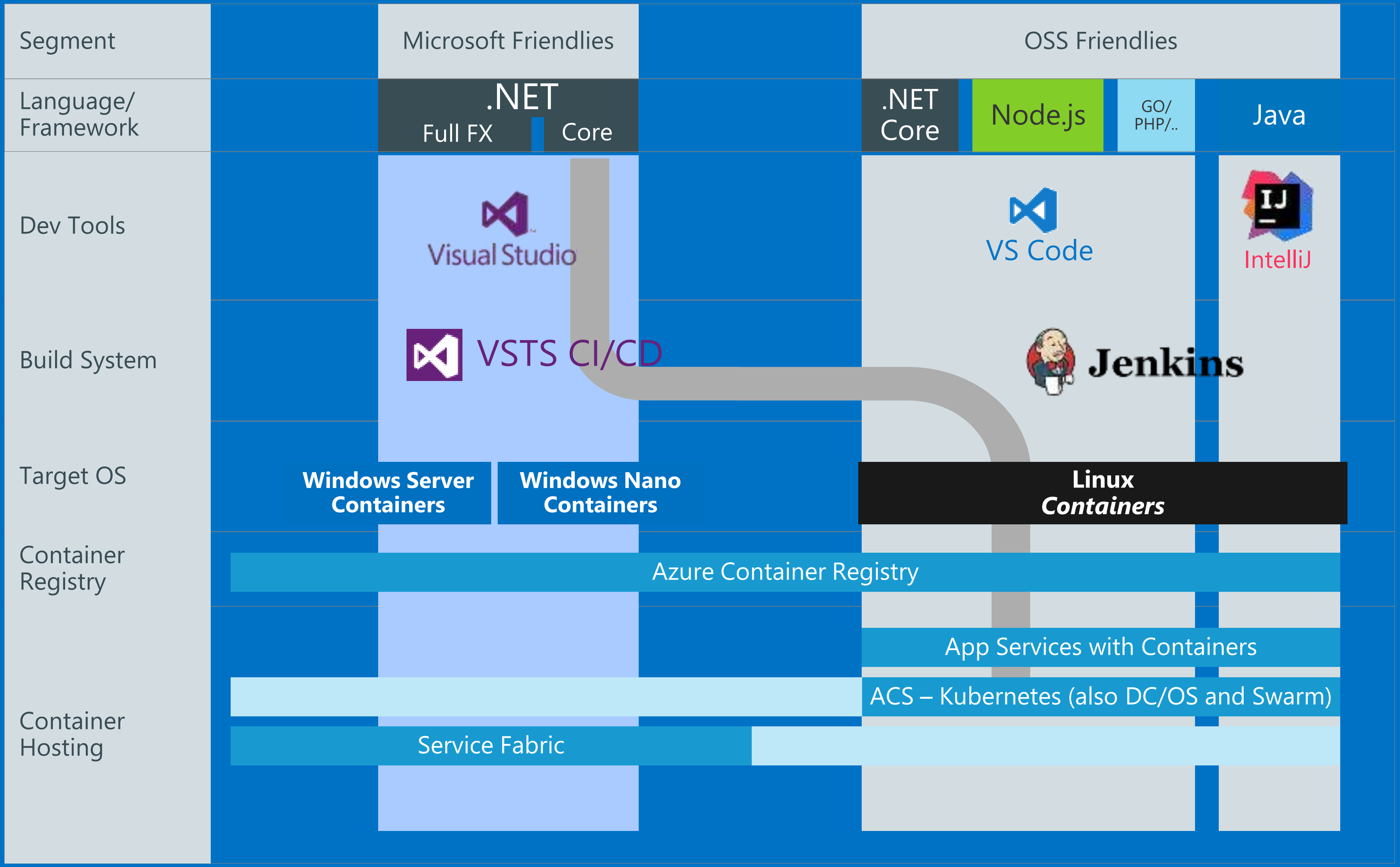
# Mapping from Infrastructure & logical view



# Helm, Charts & Draft



# Microsoft & OSS Toolchains



| Segment | Microsoft Friendlies | OSS Friendlies |
|---|---|---|
| Language/ Framework | .NET<br>Full FX<br>Core | .NET Core<br>Node.js<br>GO/ PHP/..<br>Java |
| Dev Tools | Visual Studio | VS Code<br>IntelliJ |
| Build System | VSTS CI/CD | Jenkins |
| Target OS | Windows Server Containers<br>Windows Nano Containers | Linux *Containers* |
| Container Registry | Azure Container Registry | Azure Container Registry |
| Container Hosting | Service Fabric | App Services with Containers<br>ACS – Kubernetes (also DC/OS and Swarm) |

# Differentiation between Azure & ISV Solutions

Azure Container Instance

Container Service

Service Fabric

Web Apps

Batch

Kubernetes

Mesos DC/OS

Pivotal

OpenShift

Docker Enterprise

# Thank you

Microsoft