Deploying ASP.NET Core Microservices Using Kubernetes and AKS

UNDERSTANDING MICROSERVICES AND CONTAINERS

Marcel de Vries
CTO

@marcelv https://fluentbytes.com

Outline

What is container technology

Why containers for Microservices

Running on a single machine

Production requirements & the need for a cluster orchestrator

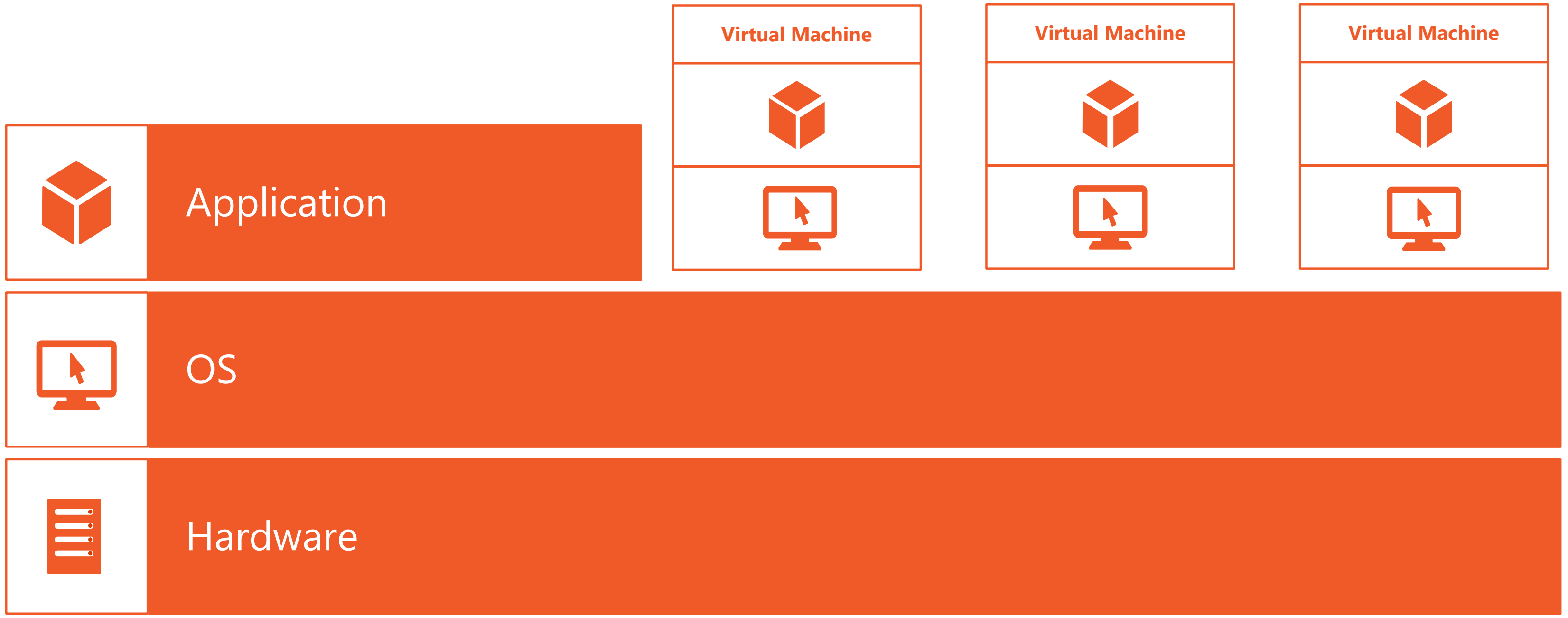
k8s concepts

Deploy a Microservice to k8s from command line

Summary

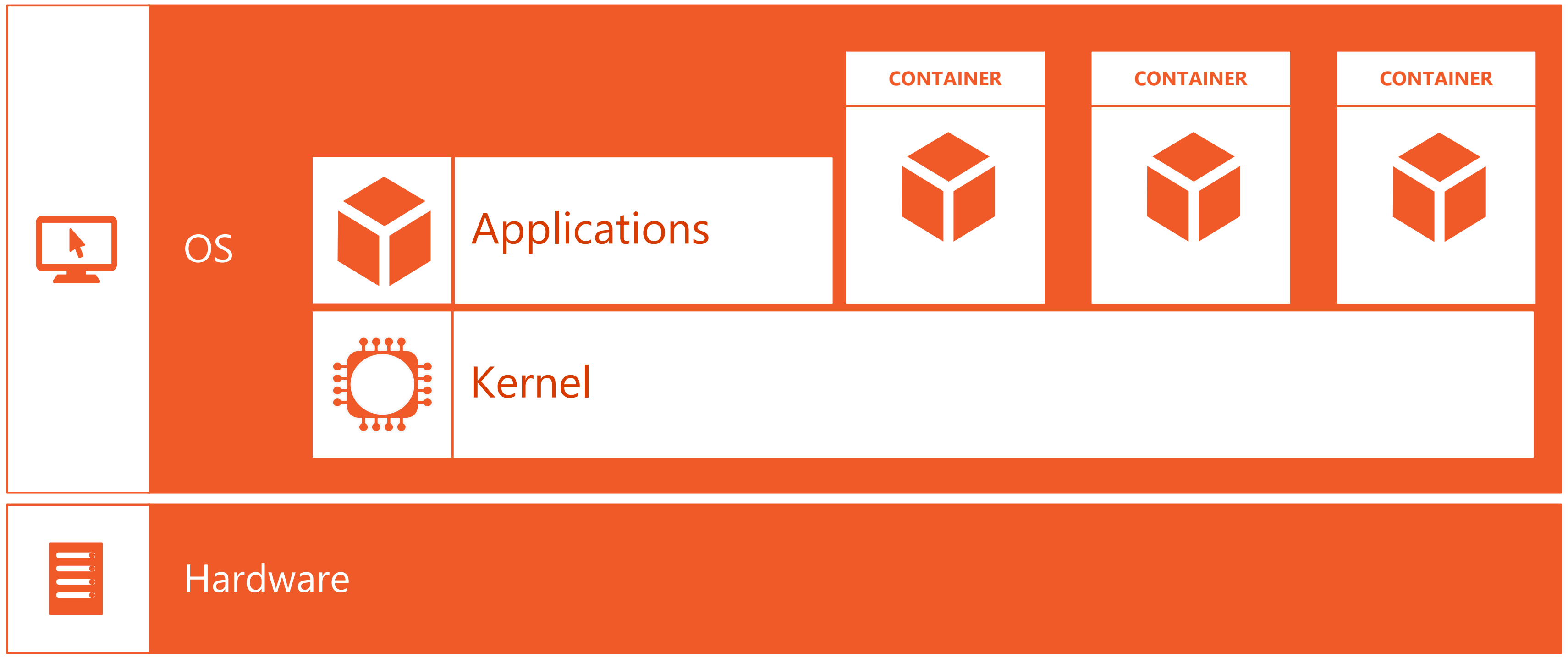
What Is Container Technology?

Traditional virtual machines = hardware virtualization



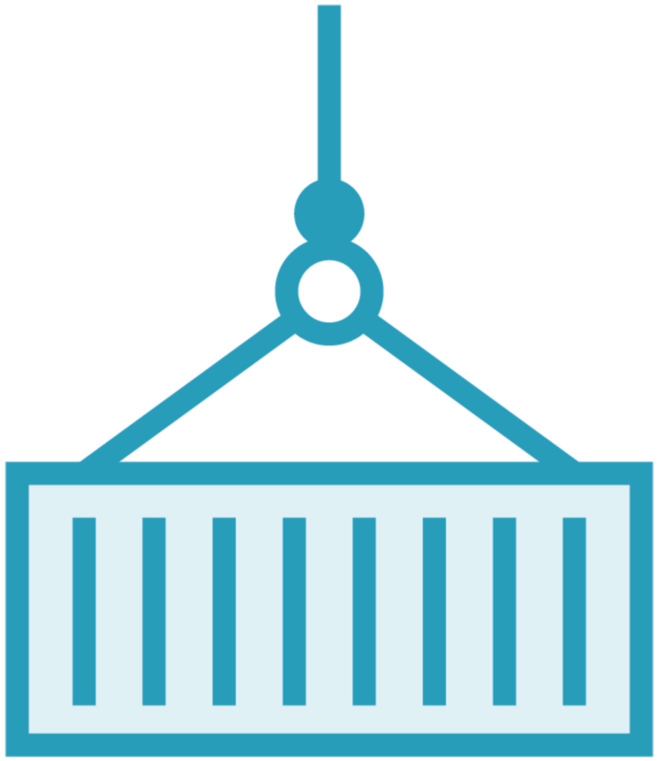
What Is Container Technology?

Containers = Operating system virtualization



Why Containers for Microservices?

Why Containers for Microservices?



Unit of deployment

Simplified testing

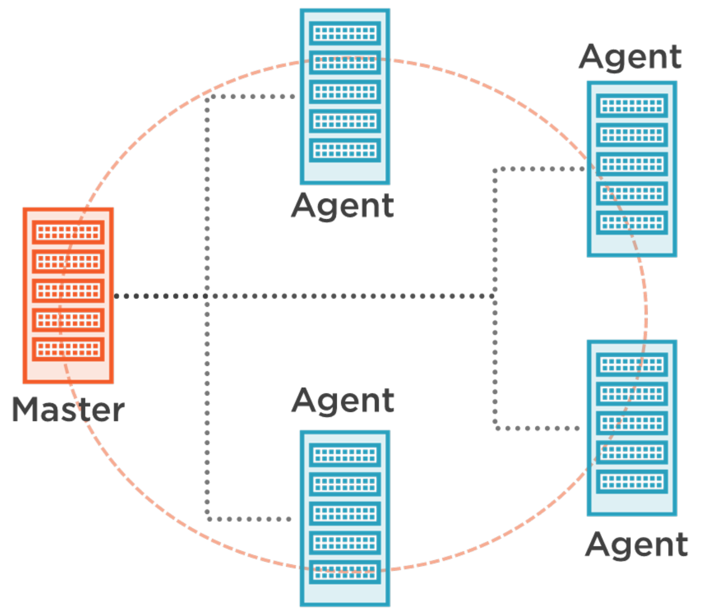
Unit of versioning

Simplified scaling

Demo

Running in Containers on a Single Machine

Production Workloads Run on Clusters



Scalability

Fault Tolerance

Automatic Recovery

Zero Downtime Deployments

Resource Management Cross Machines

Container Composition

Options for Container Clusters

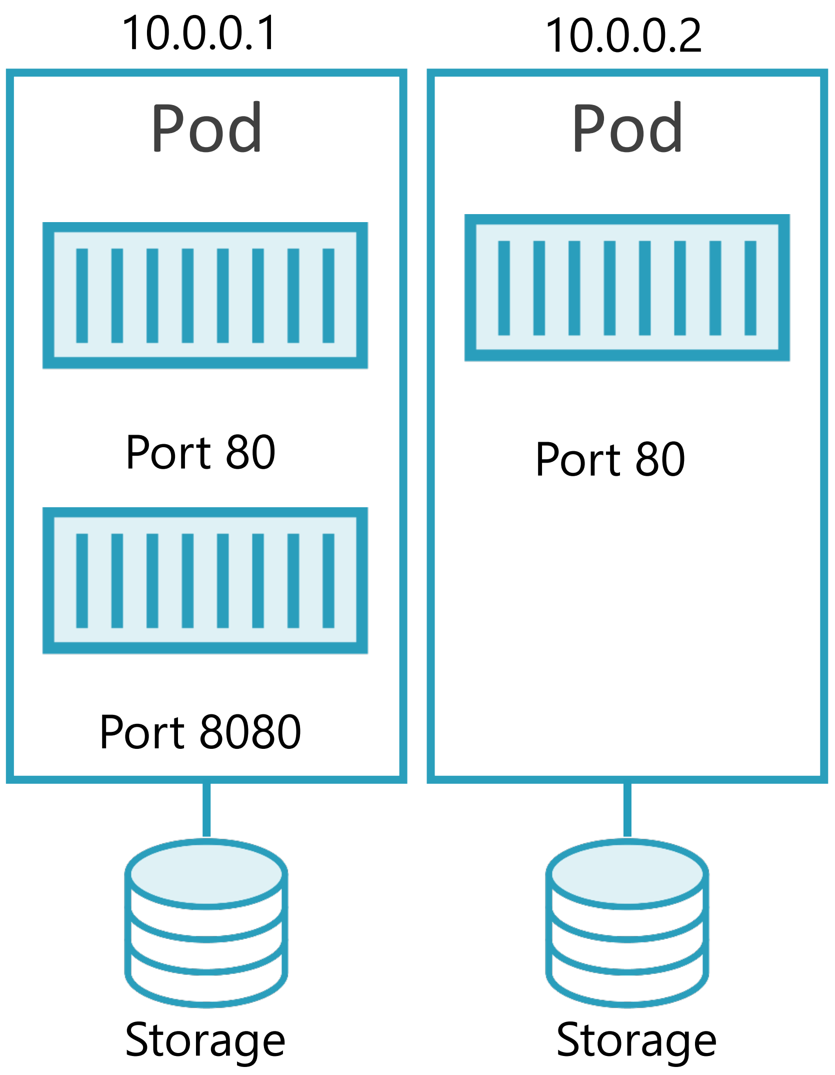
Apache Mesos

Docker Swarm

Azure Service Fabric

Kubernetes

K8s Concepts



A Pod

Group of 1 or more containers

Shared Storage

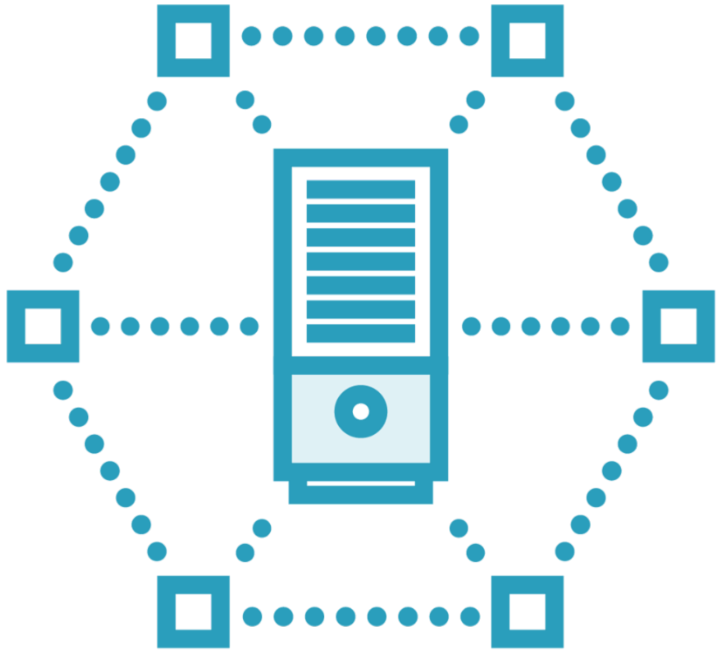
Shared Network

- Same IP-address
- Shared port-range

Well known patterns:

- Sidecar
- Proxy, bridge or adapter

A ReplicaSet and a Deployment



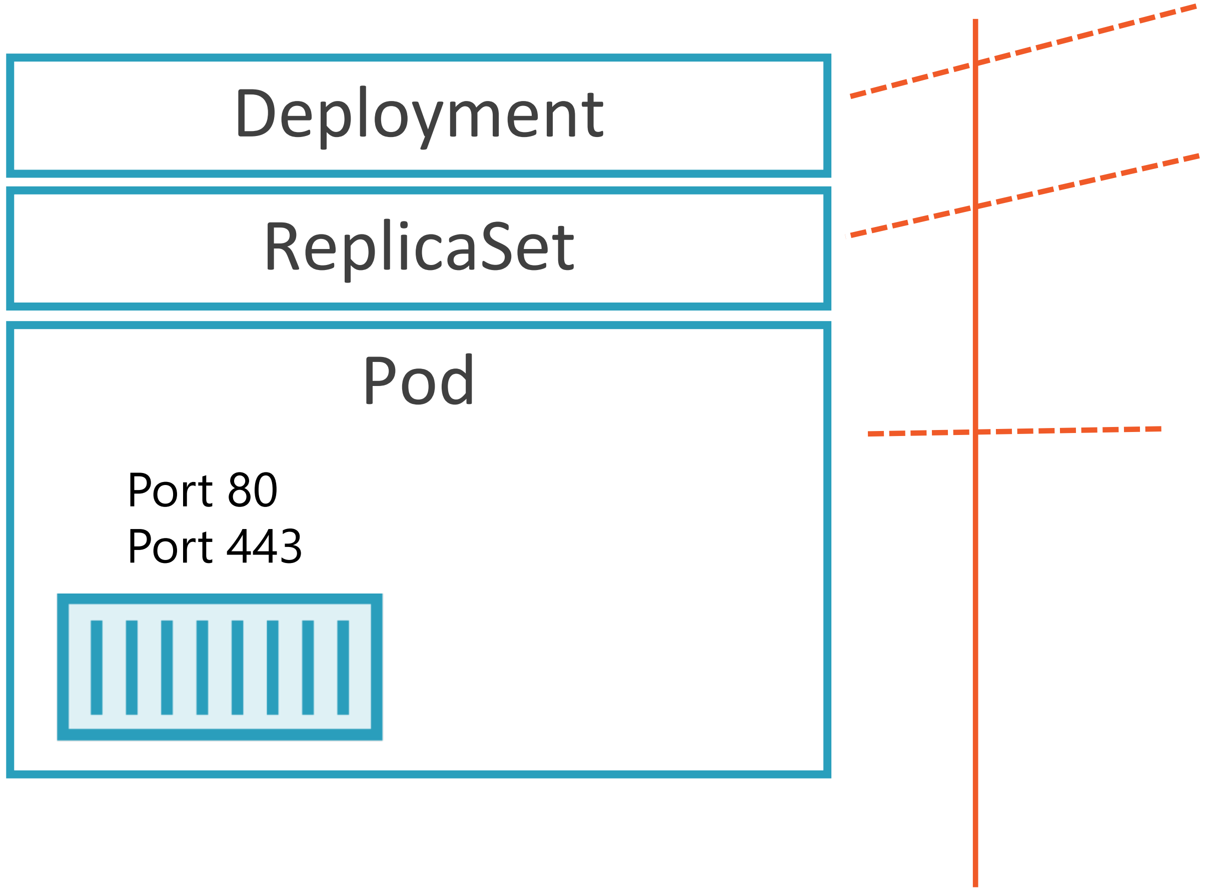
ReplicaSet

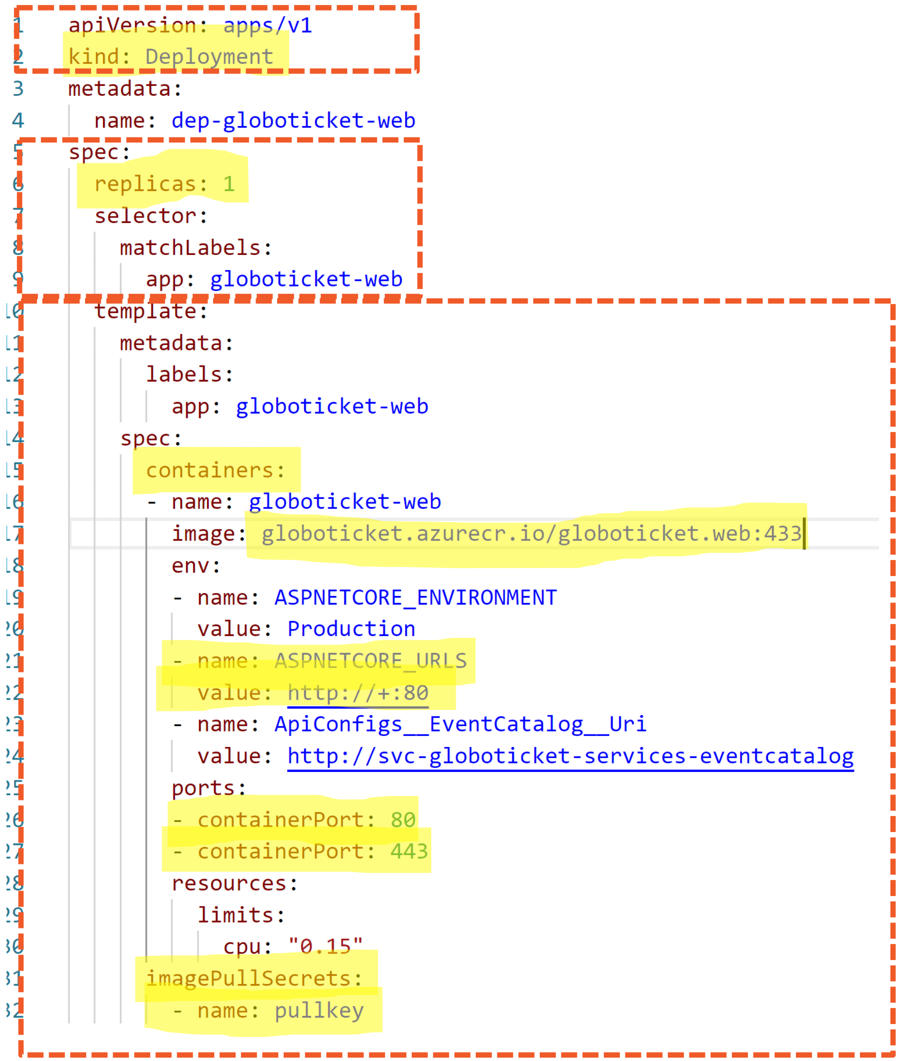
- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time

Deployment

- Provides declarative management for Pods and ReplicaSets
- Deployments own and manage their ReplicaSets

Desired State Defined in Yaml





```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dep-globoticket-web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: globoticket-web
  template:
    metadata:
      labels:
        app: globoticket-web
    spec:
      containers:
      - name: globoticket-web
        image: globoticket.azurecr.io/globoticket.web:433
        env:
        - name: ASPNETCORE_ENVIRONMENT
          value: Production
        - name: ASPNETCORE_URLS
          value: http://+:80
        - name: ApiConfigs__EventCatalog__Uri
          value: http://svc-globoticket-services-eventcatalog
        ports:
        - containerPort: 80
        - containerPort: 443
        resources:
          limits:
            cpu: "0.15"
      imagePullSecrets:
      - name: pullkey
```

Rolling Updates

Zero Downtime Deployments

- New replica set is created
- New pods are created
- Traffic is re-routed
- Old pods are deleted
- Old Replica set is deleted

Can be rolled back

Resource Management

Requests:

- Minimum required resources

Limits:

- Capped resource usage

How to Communicate between Pods?

Pods are mortal

- E.g. new deployment
- Need an abstract way to expose an application running on a set of pods

How to Communicate between Pods?

Important Note:

When building microservices, avoid microservice 2 microservice communication as much as possible.

Breaking up a monolith into multiple services that call each other is an anti pattern! It will result in a distributed monolith and will give you only problems!

A microservice should operate on its own and should not need other services to do its work. If that is the case, you need to revise your design to create better isolated services!

How to Communicate between Pods?

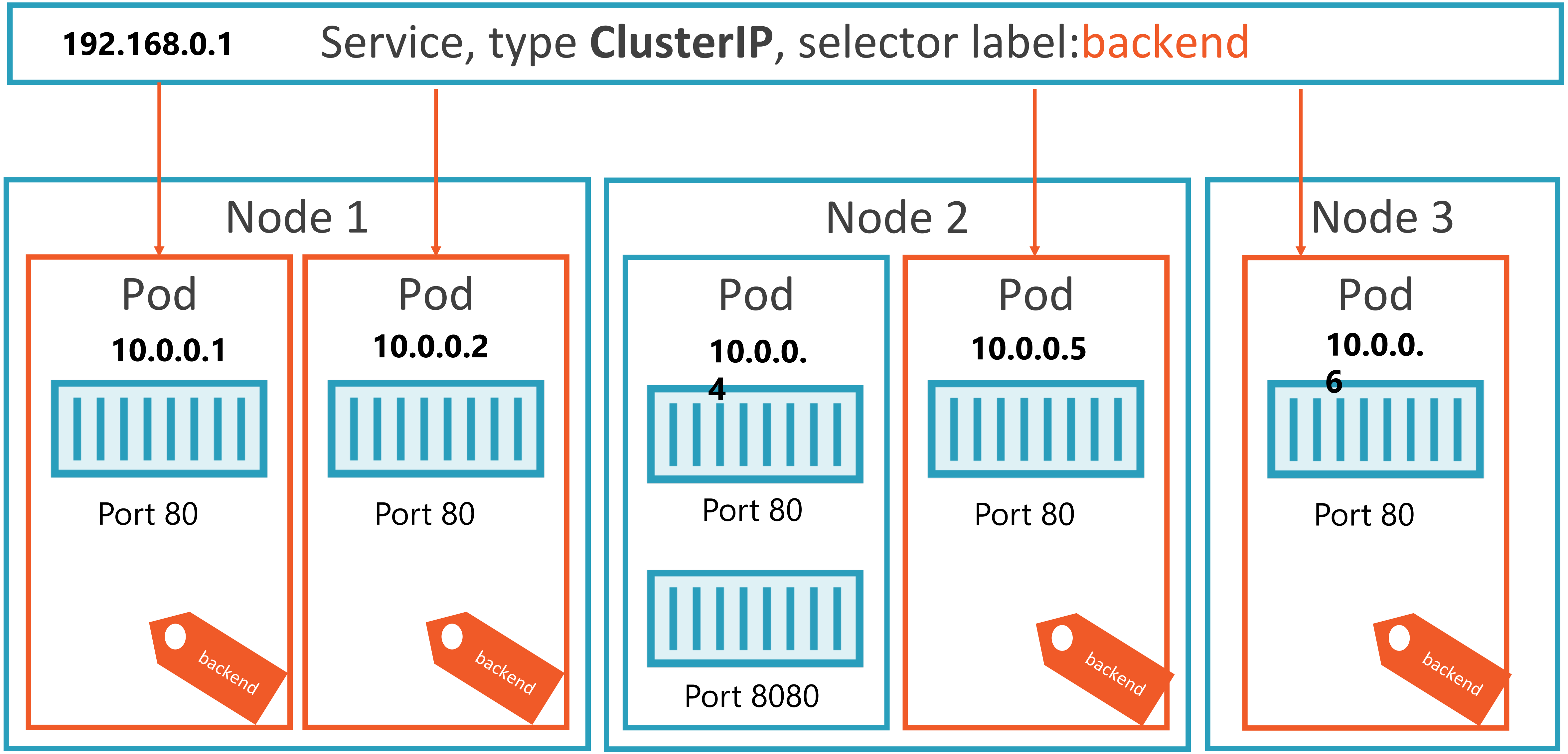
Pods are mortal

- E.g. new deployment
- Need an abstract way to expose an application running on a set of pods

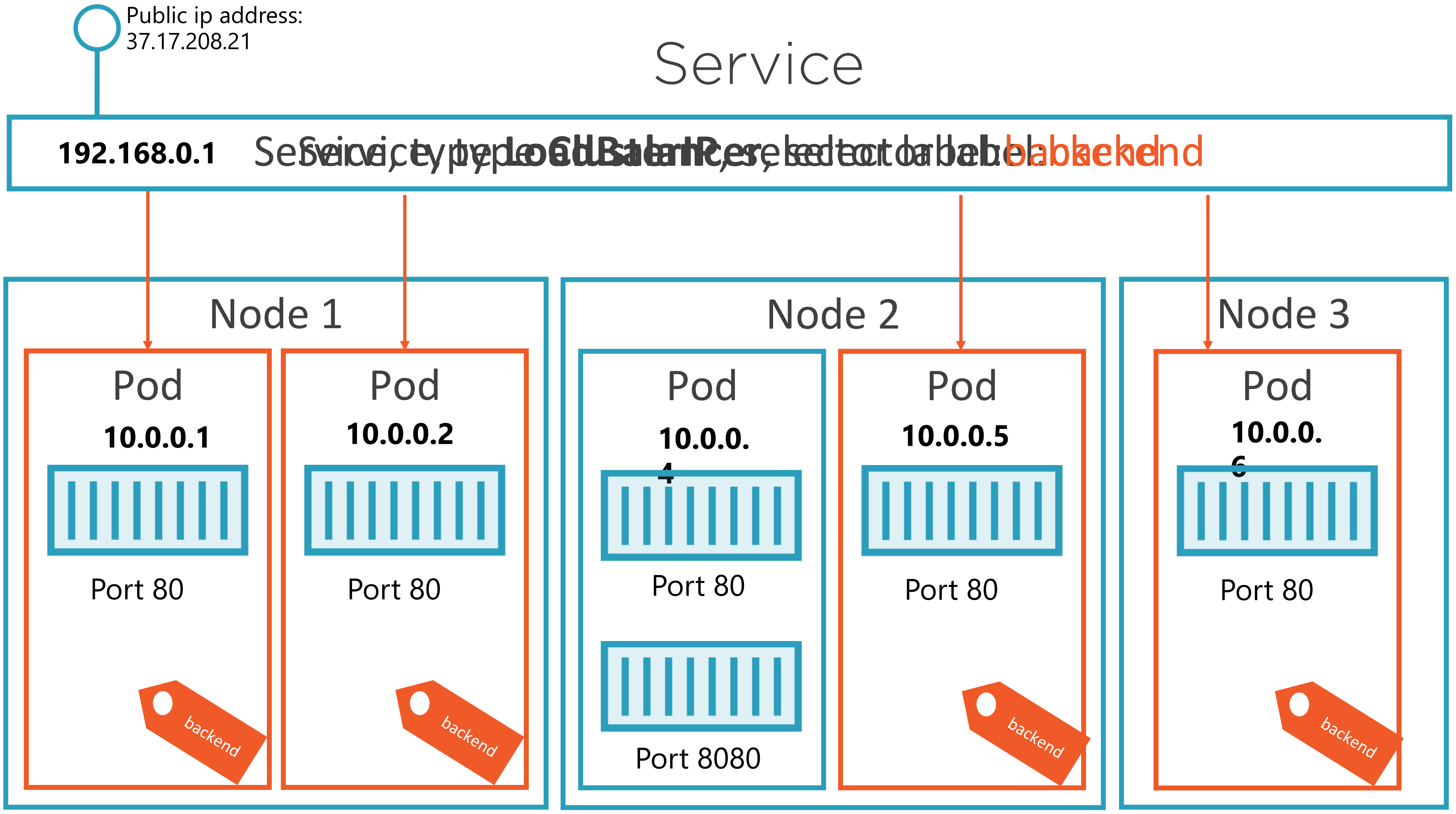
Service

- Single IP address and single DNS name
- Load-balance across pods

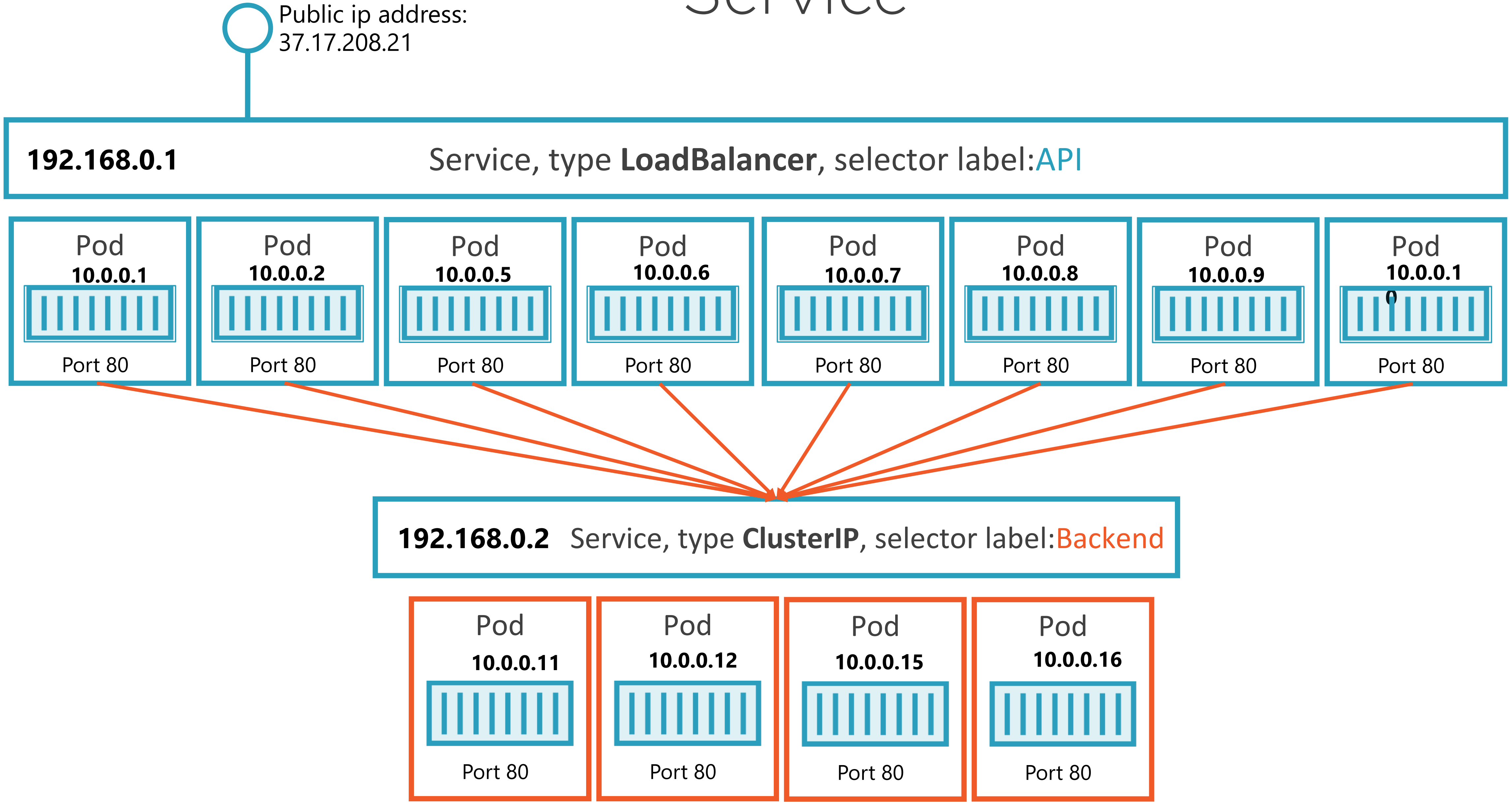
Service


192.168.0.1
Service, type **ClusterIP**, selector label:backend
Node 1
Pod
10.0.0.1
Port 80
backend
Pod
10.0.0.2
Port 80
backend
Node 2
Pod
10.0.0.4
Port 80
Port 8080
Pod
10.0.0.5
Port 80
backend
Node 3
Pod
10.0.0.6
Port 80
backend

Service


Public ip address:
37.17.208.21
192.168.0.1
Service, type LoadBalancer, selector label: backend
Service, type ClusterIP, selector label: backend
Node 1
Pod
10.0.0.1
Port 80
backend
Pod
10.0.0.2
Port 80
backend
Node 2
Pod
10.0.0.4
Port 80
Port 8080
Pod
10.0.0.5
Port 80
backend
Node 3
Pod
10.0.0.6
Port 80
backend

Service


Public ip address:
37.17.208.21
192.168.0.1
Service, type **LoadBalancer**, selector label:API
Pod 10.0.0.1 Port 80
Pod 10.0.0.2 Port 80
Pod 10.0.0.5 Port 80
Pod 10.0.0.6 Port 80
Pod 10.0.0.7 Port 80
Pod 10.0.0.8 Port 80
Pod 10.0.0.9 Port 80
Pod 10.0.0.10 Port 80
192.168.0.2
Service, type **ClusterIP**, selector label:Backend
Pod 10.0.0.11 Port 80
Pod 10.0.0.12 Port 80
Pod 10.0.0.15 Port 80
Pod 10.0.0.16 Port 80

Tip of the Iceberg

Deployment

Replica set

Pod

Label

Rolling update

Health check

Environment variables

Secret

Resource management

Horizontal Pod Autoscaler

Namespace

Service

Ingress

Annotation

Persistent Volume

Cron Job

Daemon Set

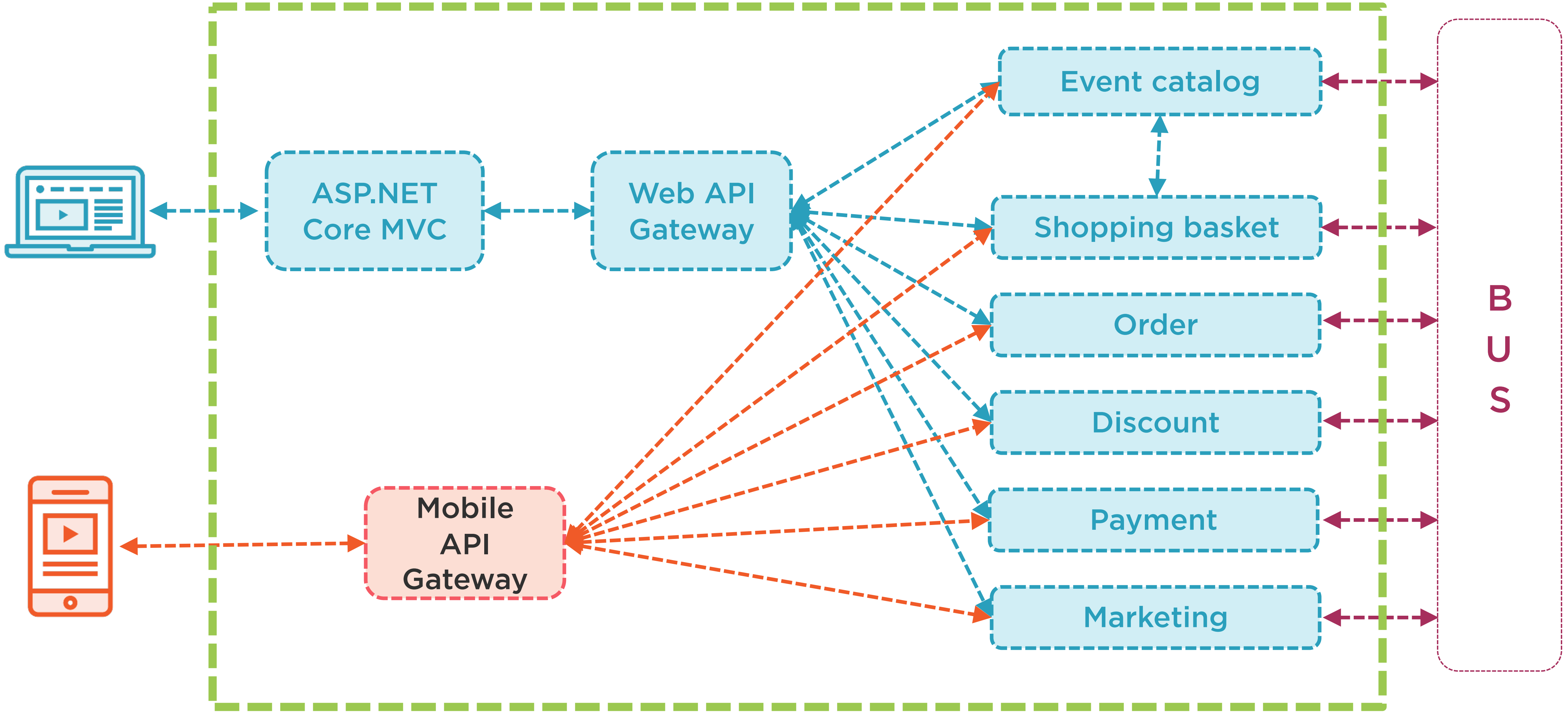
Job

Stateful Set

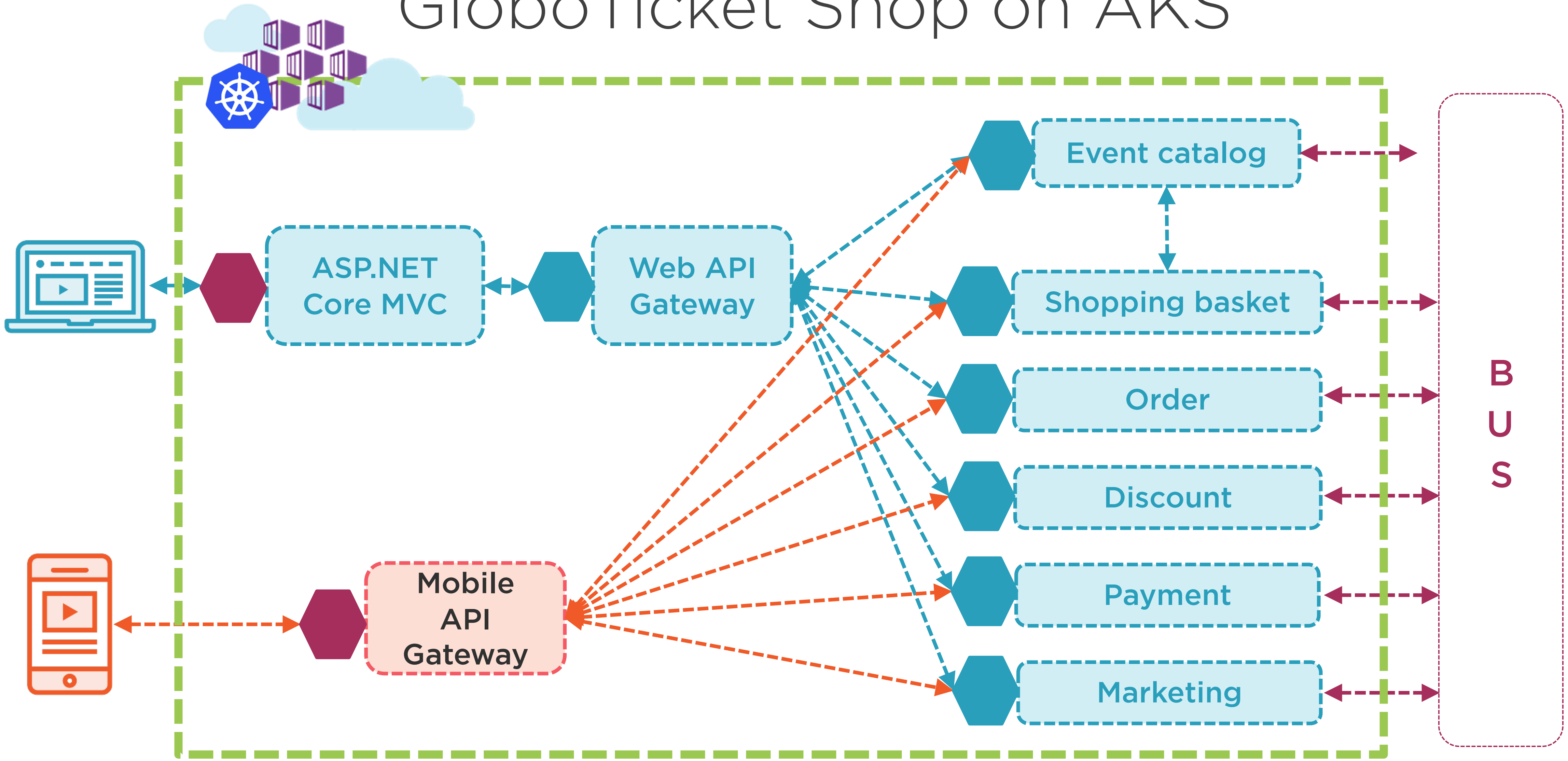
Config Map

...

GloboTicket Shop



GloboTicket Shop on AKS



⬢ Service: type LoadBalancer

⬢ Service: type ClusterIP

A Note on SSL between Containers

SSL in the container, requires distribution of certificates

- Often moved outside the cluster

Use SSL for Container 2 Container when sensitive data is exchanged, and network not trusted

- E.g. passing tokens between containers
- Can be handled by side-car container
  - E.g. envoy
- Or can be handled by service mesh
  - E.g. istio, containerd, etc.

Demo

Deploy the Event Catalog Microservice to K8s from Command Line

Summary

What is container technology

Why containers for Microservices

Running on a single machine

Production requirements & the need for a cluster orchestrator

k8s concepts

Deploy a Microservice to k8s from command line