

# Kubernetes on Azure



Vishwas Lele @vlele  
Applied Information Sciences

# Agenda



- Why Kubernetes?
  - Kubernetes Concepts – I
  - Kubernetes Concepts – II
  - Patterns
  - Azure Kubernetes Service
  - Best practices
- “More than just microservices”***
- “Essentials”***
- “Advanced Concepts”***
- “Sidecar, Istio, Servicemesh”***
- “Kubernetes on Azure”***
- “Operational best practices”***



# Attribution

- <https://github.com/cncf/presentations/tree/master/kubernetes>
- [@mrbobbytables](#)
- [@jeefy](#)



# Containers and Orchestrators

Section #1



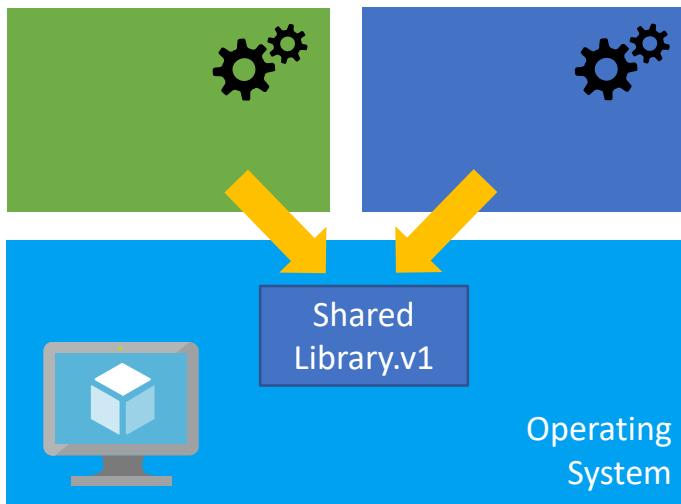
# Go grab a coffee, if you know this already...

- Motivation for containers
  - Immutable images
  - Dependencies
  - Isolation
  - Quotas
- Container Orchestrators
- Orchestration as Service
- Containers as the new foundation



# What is a container?

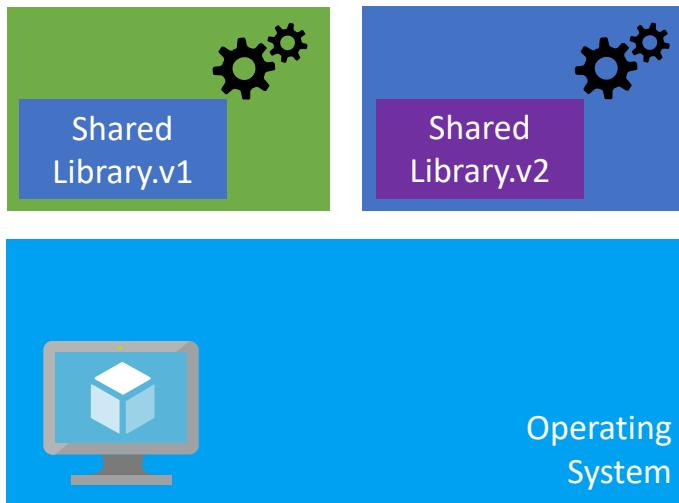
- Application packaging and deployment mechanism





# What is a container?

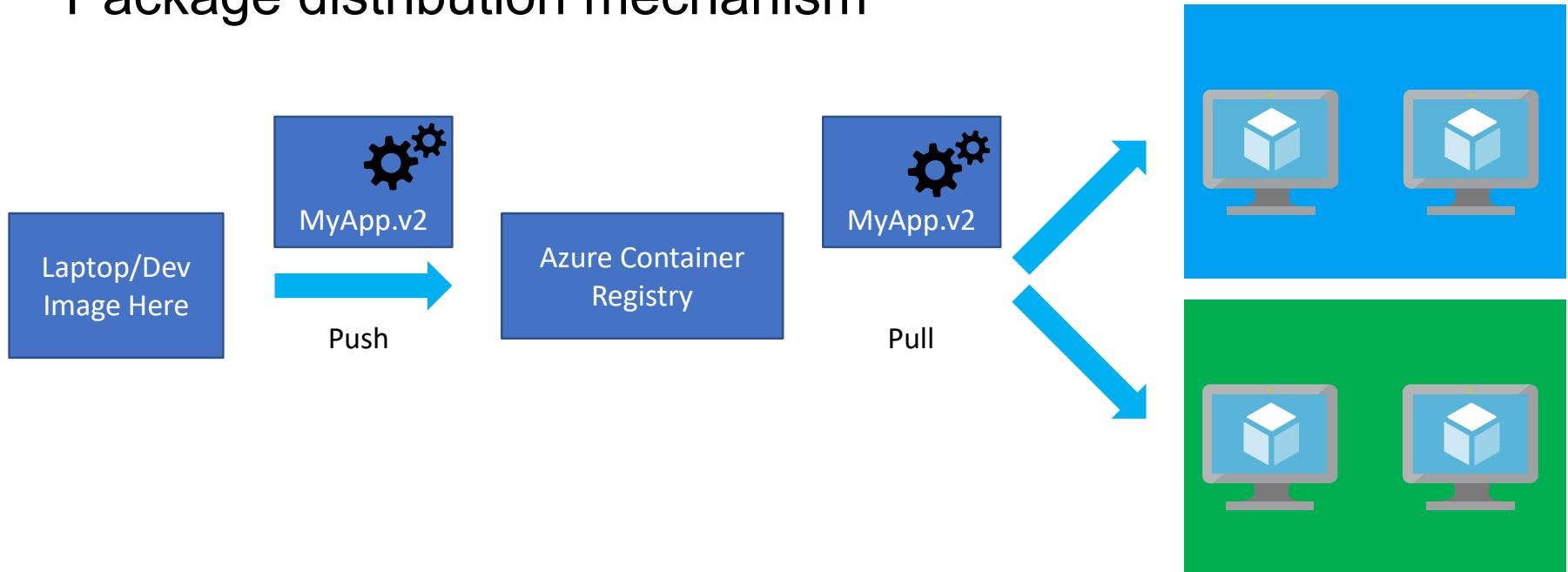
- Application packaging and deployment mechanism





# What is a container?

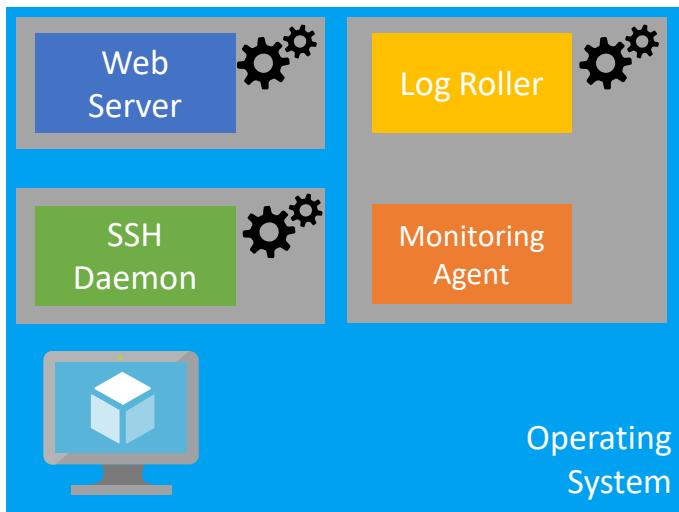
- Package distribution mechanism





# What is a container?

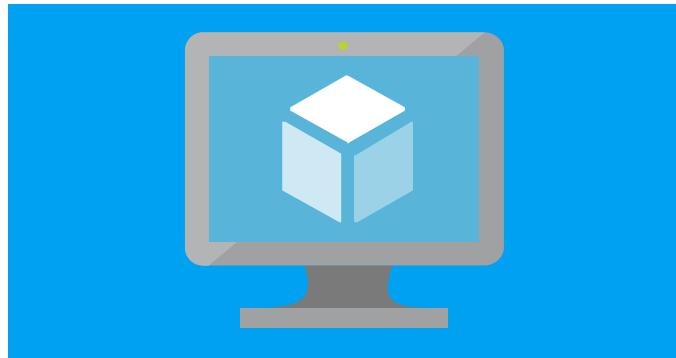
- A clear boundary for your stuff





# What is a container?

- Runtime isolation





# What is a container orchestrator?

- Manage a bunch of machines

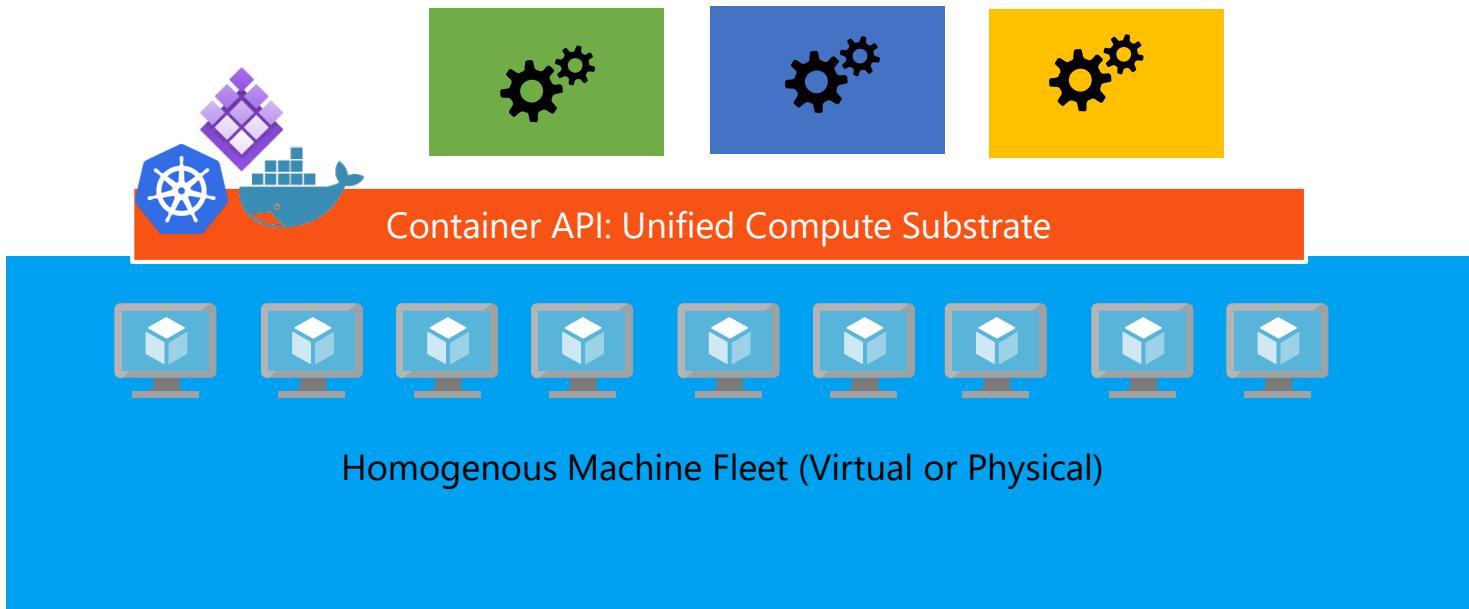


Homogenous Machine Fleet (Virtual or Physical)



# What is a container orchestrator?

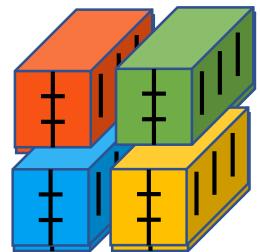
- Decouple machines from applications





# What is a container service?

- Make all of this stuff easier



Container API: Unified Compute Substrate



Homogenous Machine Fleet (Virtual or Physical)

# Demo

Container  
Registry  
Orchestrator

Section #2

# What is Kubernetes?

Section #1



# What is Kubernetes?

- Kubernetes is the open source standard for managing application containers from embedded, edge, on-premises to the cloud
- Built on lessons learnt at Google in developing and running Borg and Omega
  - <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43438.pdf>
- Donated to CNCF in 2014
- Now the most popular GitHub project

# What Does “Kubernetes” Mean?



Greek for “pilot” or  
“Helmsman of a ship”



[Image Source](#)



# Who “Manages” Kubernetes?



**CLOUD NATIVE  
COMPUTING FOUNDATION**

The CNCF is a child entity of the Linux Foundation and operates as a vendor neutral governance group.

# Kubernetes Momentum



- CNCF – All major cloud providers
- 400 CNCF projects
- 4300 folks attended KubeCon in May
- 4 Billion investment in startups
- CERN (210 Clusters)
- JD.Com (20 K servers)
- Over 38,000 stars on Github
- 1700+ Contributors to K8s Core



# Why Kubernetes?

Section #1



# 1/3 Why Kubernetes?

- Extensive App Support- Java, .NET, Spring, Go, Ruby
- Multi-cloud – AKS, AES, GKE
- Community - #1 most active open source project
- Embedded, edge, on-premises, private and public cloud
- Container runtime, networking agnostic Works containerd, Docker,
- Workloads - Stateless and stateful , batch, analytics, off shelf Cassandra
- New App Server!



# 2/3 Why Kubernetes?

- Run production code without setting up servers – Abstracts away the underlying hardware of nodes and provides a uniform interface for workloads
- Visibility and control of code
- Resilient - **loosely coupled** collection of components centered around deploying, maintaining and scaling workloads.
- Works as an engine for resolving state by converging actual and the **desired state** of the system (**self-healing**)
- “Kernel” of distributed systems
- **SAME API** across bare metal and **EVERY** cloud provider!



# 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.”



# 3/3 Why Kubernetes?

- Oh.. Did I say Microservices?
- But first let us define Microservices



# Microservices

- Single responsibility principle == strong cohesion and loosely coupled
- Failure of one microservice does not cascade to other parts
- Autonomous
- Scaled independently
- Deployed and updated independently
- Technology, language or platform agnostic
- Composable

# But Microservices aren't free



- Significant Operations Overhead
  - No way to handle dozens of services without automation
- Distributed Systems Complexity
  - Performance, latency, security, network topology
- Technology Diversity
  - Microservices can be written different languages, use different libraries
- Testability
  - Difficult to recreate environments that are so dynamic



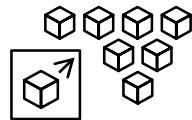
# 3/3 Why Kubernetes? (Continued)

- All **microservices** 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.
- Blue/Green Deployments
- Fire off jobs and scheduled cronjobs
- Manage Stateless and Stateful Applications
- Provide native methods of service discovery
- Easily integrate and support 3rd party apps



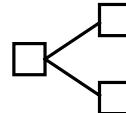
# Top Scenarios For Kubernetes On Azure

Lift and shift  
to containers



**Cost saving**  
without refactoring  
your app

Microservices



**Agility**  
Faster application  
development

Machine  
learning



**Performance**  
Low latency  
processing

IoT



**Portability**  
Build once, run  
anywhere



# Kubernetes and Site Reliability Engineering (SRE)

Section #1

# Improving SRE



Application SRE

Cluster SRE

Kernel/OS SRE



Hardware SRE

# Decoupling



Application SRE

Cluster SRE

Kernel/OS SRE

Azure Virtual Machines



Hardware SRE



# Decoupling



Application SRE

Cluster SRE

Container Runtimes / Images

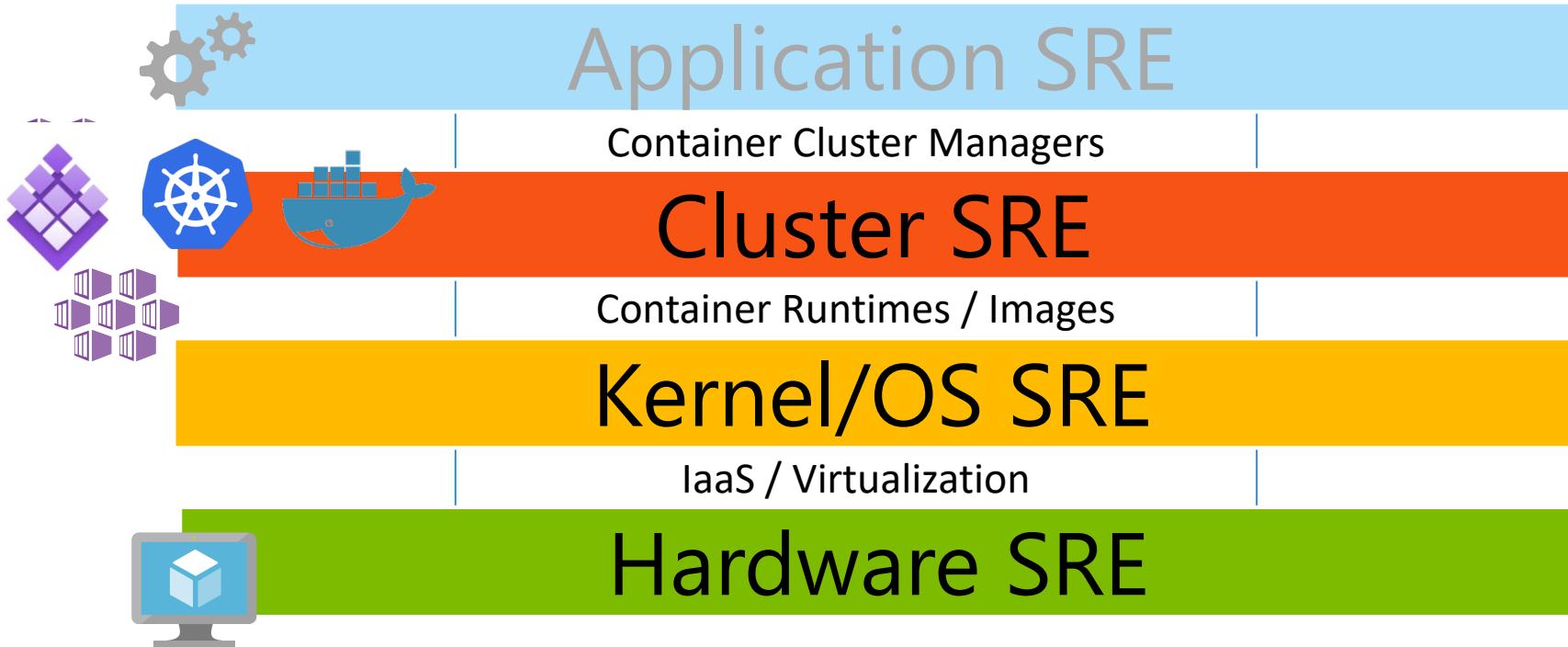
Kernel/OS SRE

IaaS / Virtualization



Hardware SRE

# Decoupling

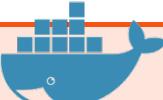


# Decoupling



**Application SRE**

Container Cluster Managers



**Cluster SRE**

Container Runtimes / Images



**Kernel/OS SRE**

IaaS / Virtualization



**Hardware SRE**

# Key Concepts

Section #2

# Kubernetes Key Concepts

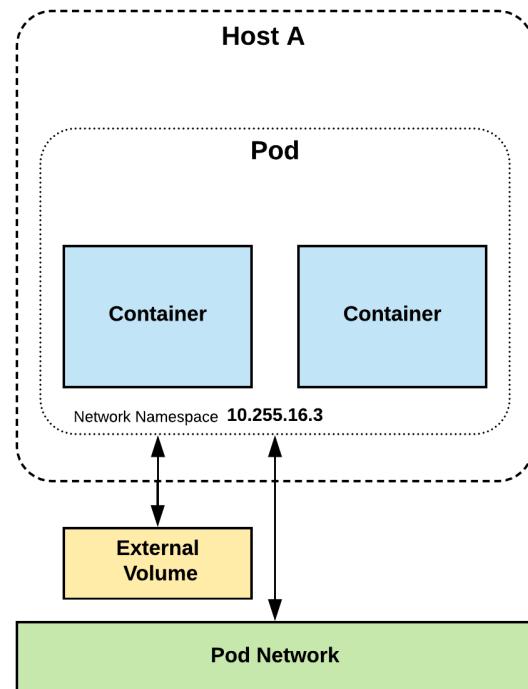


- Pods
- Services
- Control plane components
- Node components
- Optional components
- API and object model
- Core Concepts - Labels and Selectors
- Service Types
- Deployment
- StatefulSets and DaemonSets
- Headless Service
- Jobs
- Cron Jobs

# Pods



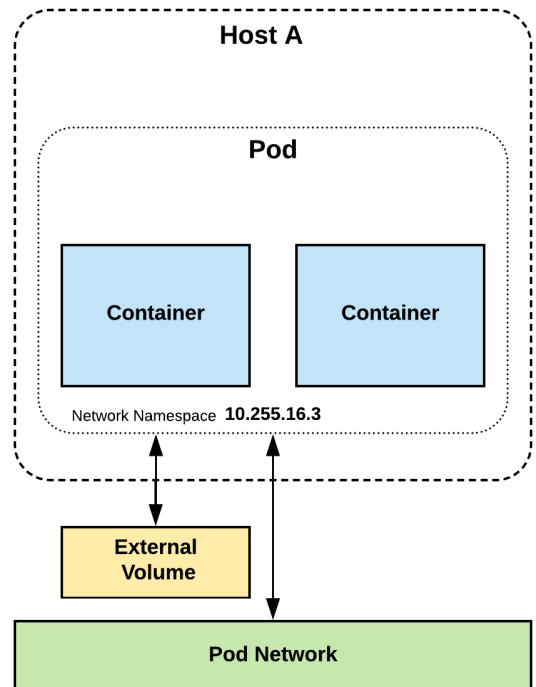
- **Atomic unit** or smallest “*unit of work*” of Kubernetes.
- Pods are **one or MORE containers** that share volumes, a network namespace, and are a part of a **single context**.



# Pods



They are  
also  
Ephemeral!



# Demo

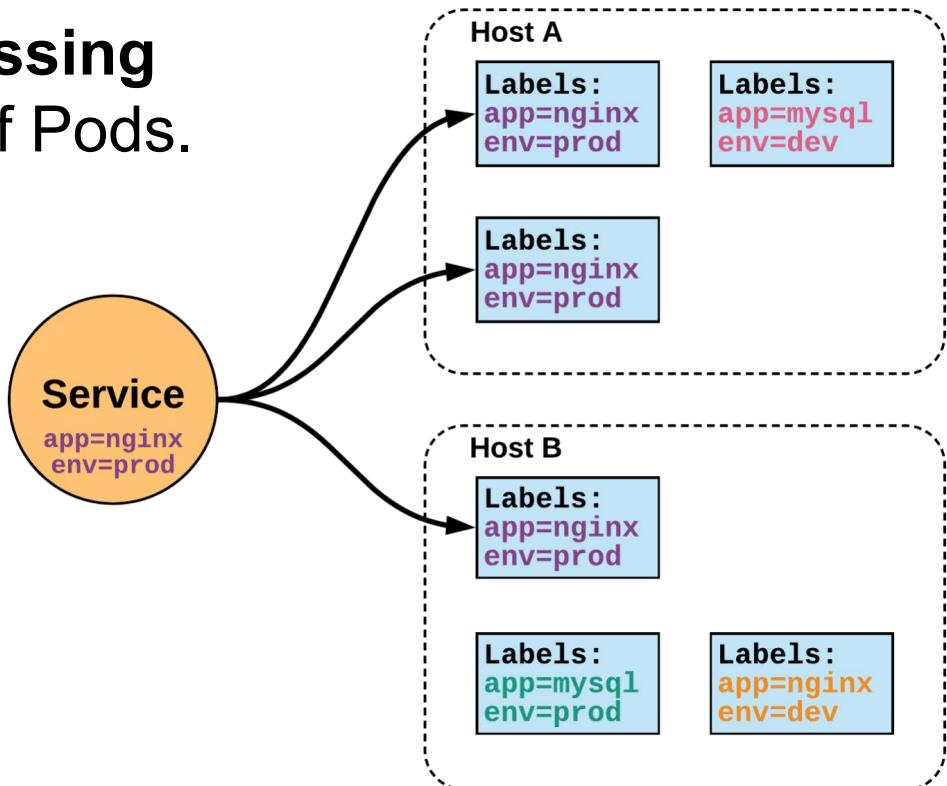
# POD SERVICE

Section #2

# Services



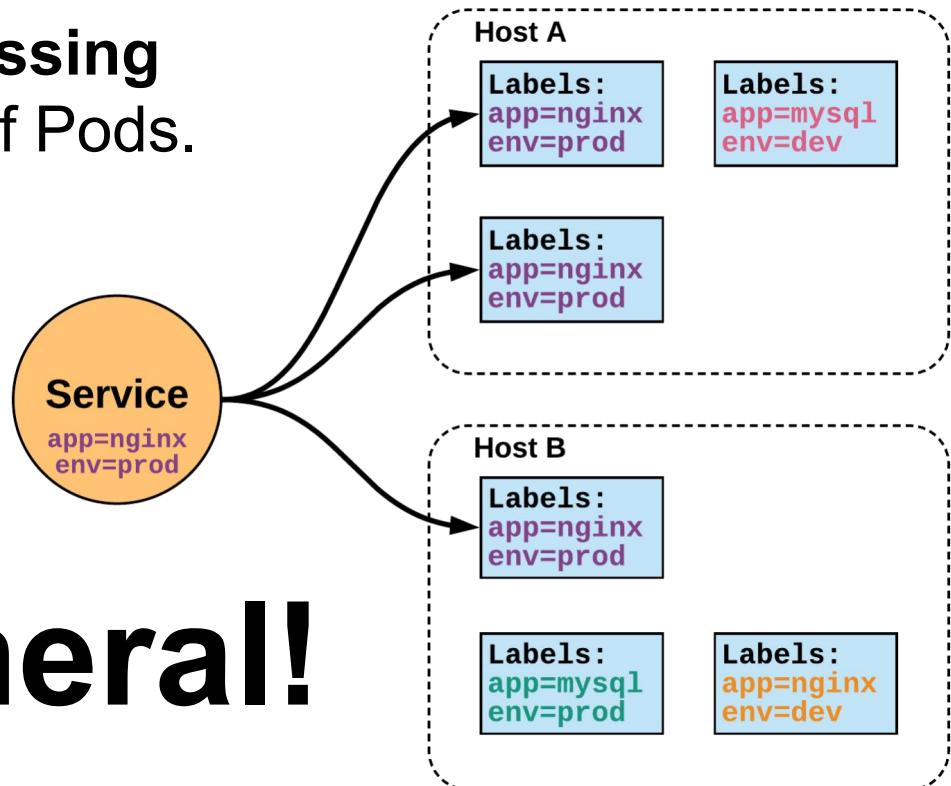
- **Unified method of accessing** the exposed workloads of Pods.
- **Durable resource**
  - static cluster IP
  - static namespaced DNS name



# Services

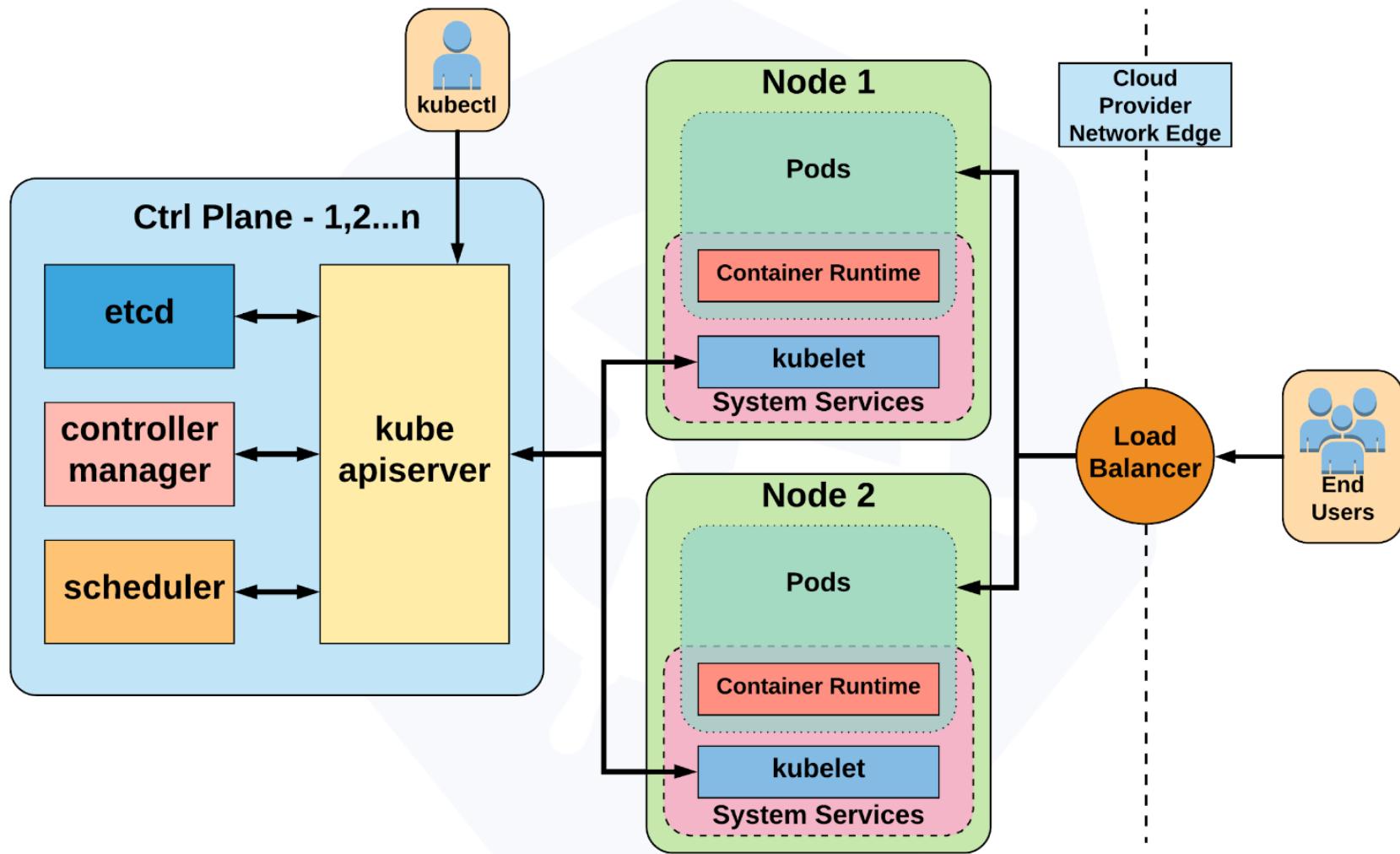


- **Unified method of accessing** the exposed workloads of Pods.
- **Durable resource**
  - static cluster IP
  - static namespaced DNS name



# NOT Ephemerall!

# Architecture Overview



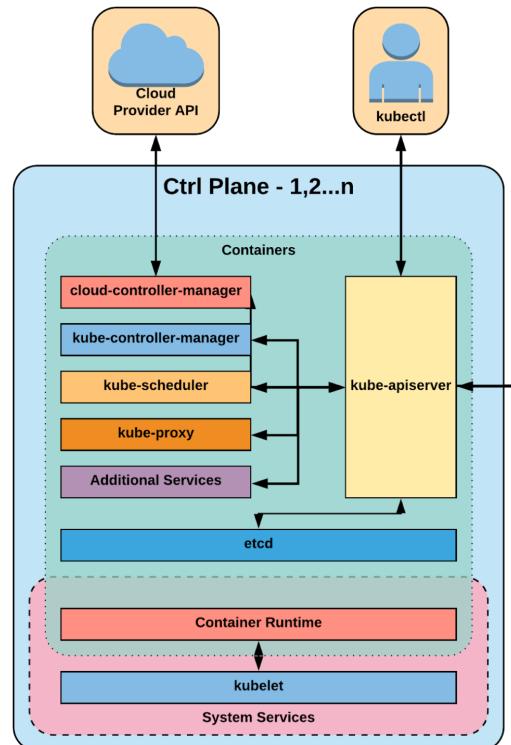
# Control Plane Components

Architecture Overview

# Control Plane Components



- kube-apiserver
- etcd
- kube-controller-manager
- kube-scheduler





# kube-apiserver

- Provides a forward facing REST interface into the kubernetes control plane and datastore.
- All clients and other applications interact with kubernetes **strictly** through the API Server.
- Acts as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore.

# etcd



- etcd acts as the cluster datastore.
- Purpose in relation to Kubernetes is to provide a strong, consistent and highly available key-value store for persisting cluster state.
- Stores objects and config information.





# kube-controller-manager

- Serves as the primary daemon that manages all core component control loops.
- Monitors the cluster state via the apiserver and **steers the cluster towards the desired state.**

List of core controllers: <https://github.com/kubernetes/kubernetes/blob/master/cmd/kube-controller-manager/app/controllermanager.go#L332>



# kube-scheduler

- Verbose policy-rich engine that evaluates workload requirements and attempts to place it on a matching resource.
- Default scheduler uses bin packing.
- Workload Requirements can include: general hardware requirements, affinity/anti-affinity, labels, and other various custom resource requirements.

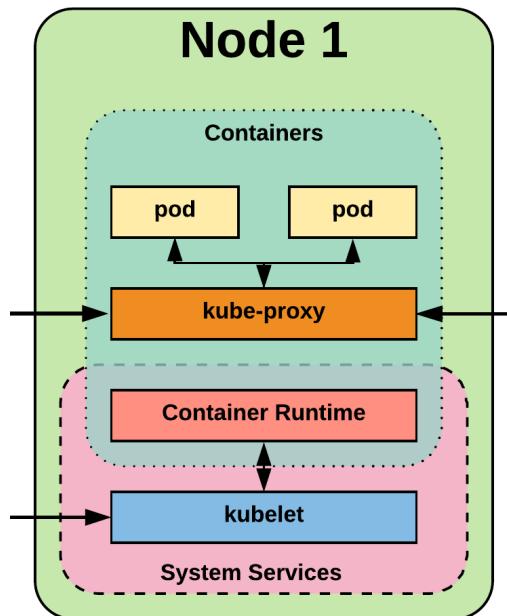
# Node Components

Architecture Overview

# Node Components



- kubelet
- kube-proxy
- Container Runtime Engine



# kubelet



- Acts as the node agent responsible for managing the lifecycle of every pod on its host.
- Kubelet understands YAML container manifests that it can read from several sources:
  - file path
  - HTTP Endpoint
  - etcd watch acting on any changes
  - HTTP Server mode accepting container manifests over a simple API.



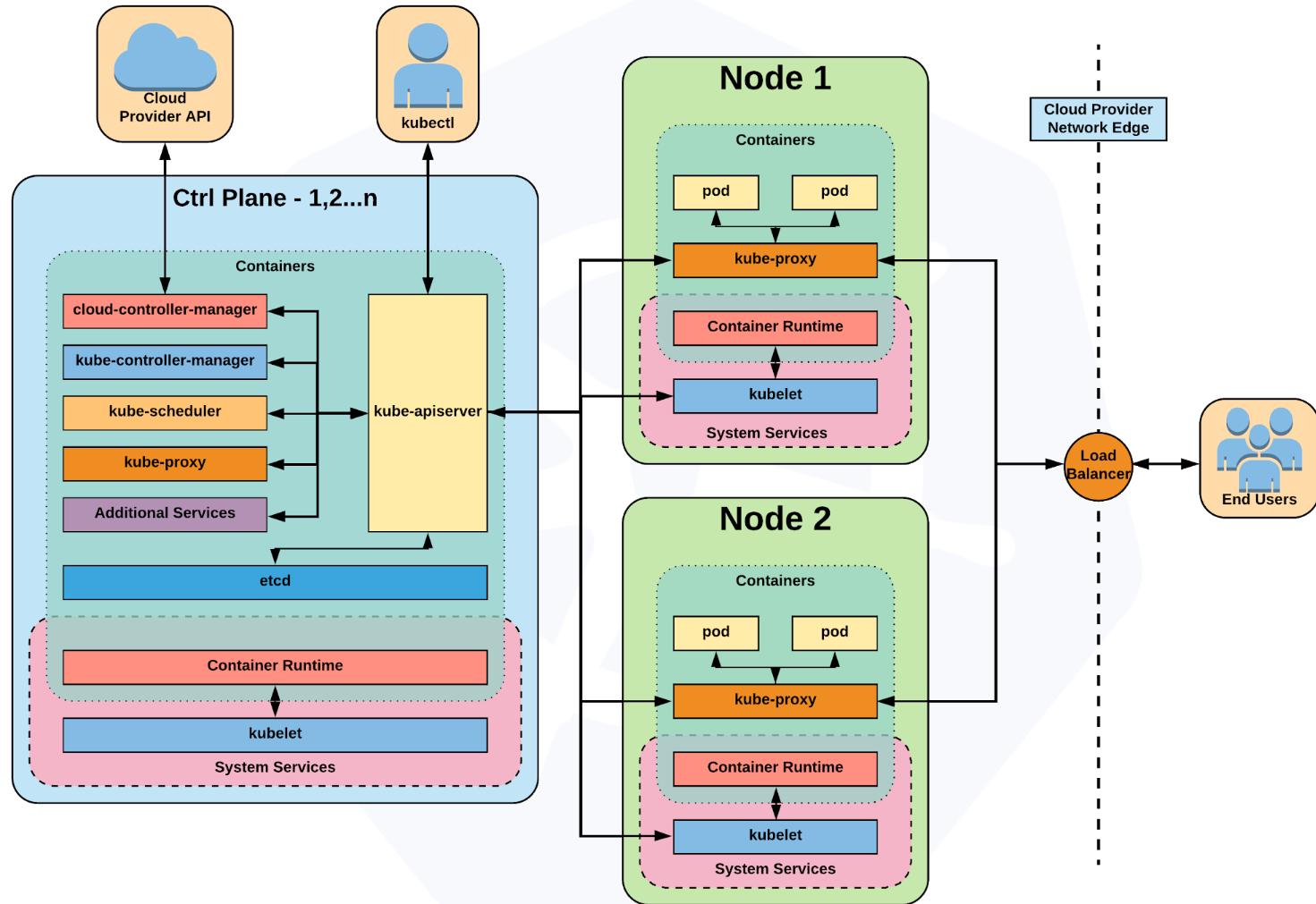
# kube-proxy

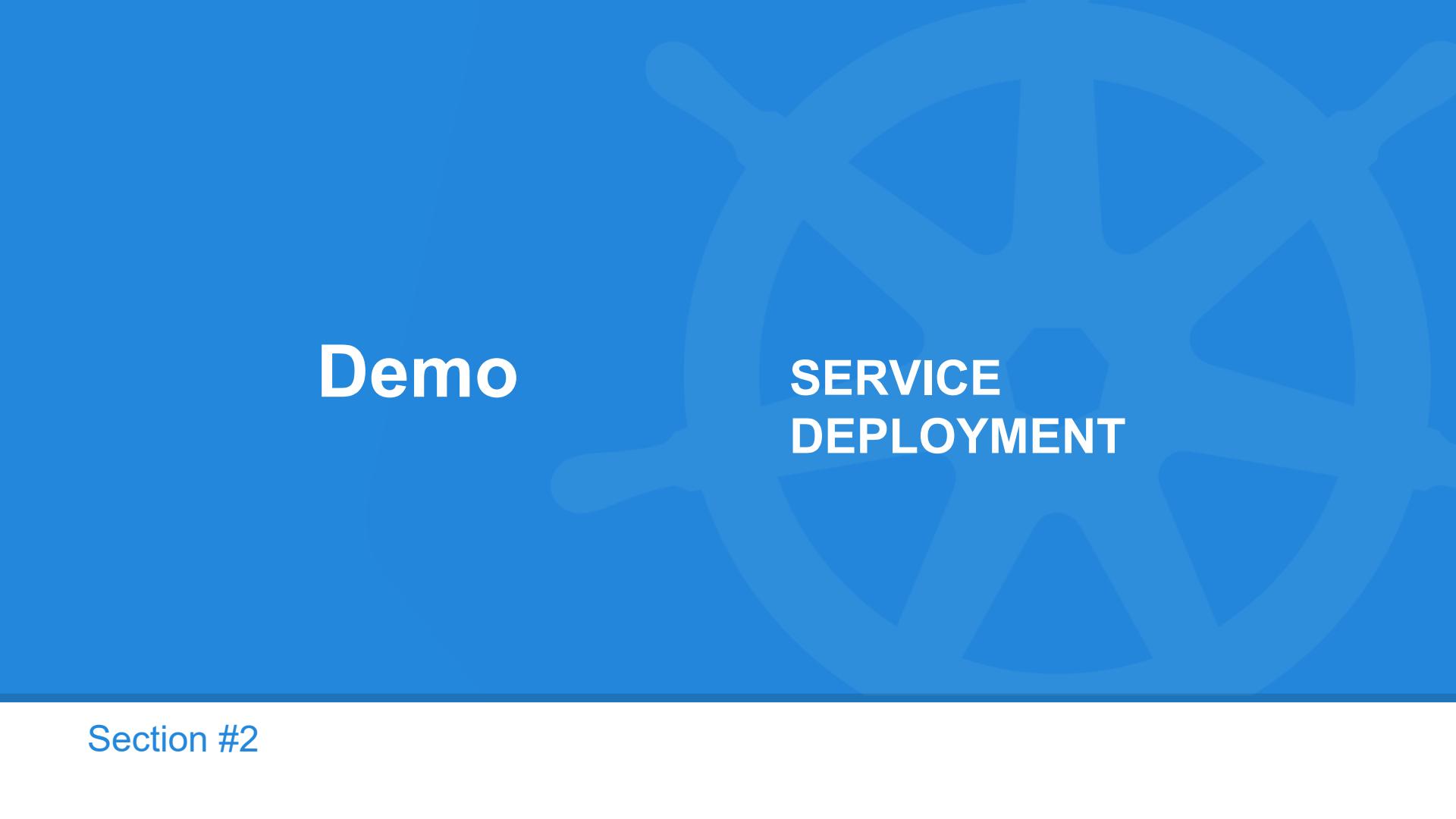
- Manages the network rules on each node.
- Performs connection forwarding or load balancing for Kubernetes cluster services.
- Available Proxy Modes:
  - Userspace
  - iptables
  - ipvs (default if supported)



# Container Runtime Engine

- A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.
  - Containerd (docker)
  - Cri-o
  - Rkt
  - Kata (formerly clear and hyper)
  - Virtlet (VM CRI compatible runtime)



A large, semi-transparent blue circle with a radial gradient pattern, centered behind the text.

**Demo**

**SERVICE  
DEPLOYMENT**

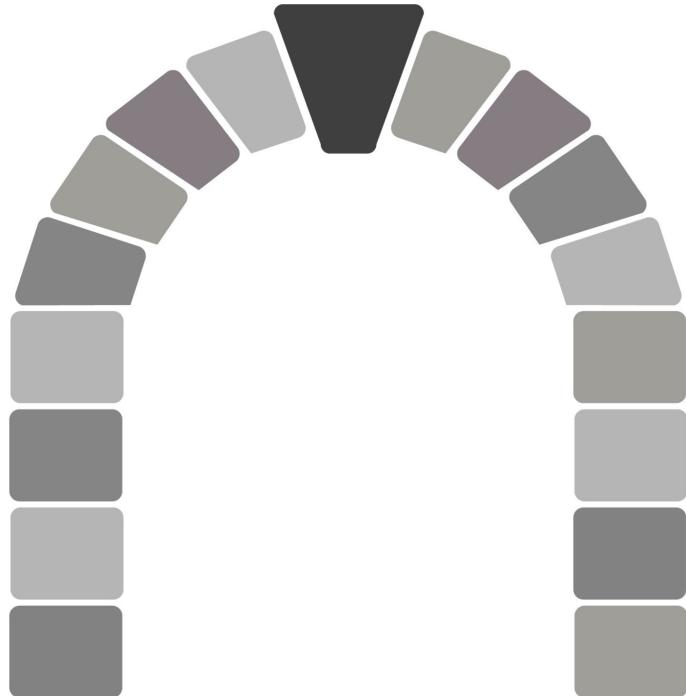
Section #2

# Concepts and Resources

# API Overview



- The **REST API** is the true **keystone** of Kubernetes.
- **Everything** within the Kubernetes is as an **API Object**.



[Image Source](#)

# API Groups



- Designed to make it extremely simple to both understand and extend.
- An API Group is a **REST compatible path** that acts as the type descriptor for a Kubernetes object.
- Referenced within an object as the **apiVersion** and **kind**.

**Format:**

`/apis/<group>/<version>/<resource>`

**Examples:**

`/apis/apps/v1/deployments`

`/apis/batch/v1beta1/cronjobs`

# API Versioning



- Three tiers of API maturity levels.
  - Also referenced within the object **apiVersion**.
- Format:**  
`/apis/<group>/<version>/<resource>`

**Examples:**  
`/apis/apps/v1/deployments`  
`/apis/batch/v1beta1/cronjobs`
- **Alpha:** Possibly buggy, And may change. **Disabled by default.**
  - **Beta:** Tested and considered stable. However API Schema may change. **Enabled by default.**
  - **Stable:** Released, stable and API schema will not change. **Enabled by default.**



# Object Model

- Objects are a “*record of intent*” or a persistent entity that represent the desired state of the object within the cluster.
- All objects **MUST** have `apiVersion`, `kind`, and poses the nested fields `metadata.name`, `metadata.namespace`, and `metadata.uid`.



# Object Model Requirements

- `apiVersion`: Kubernetes API version of the Object
- `kind`: Type of Kubernetes Object
- `metadata.name`: Unique name of the Object
- `metadata.namespace`: Scoped environment name that the object belongs to (will default to current).
- `metadata.uid`: The (generated) uid for an object.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  namespace: default
  uid: f8798d82-1185-11e8-94ce-080027b3c7a6
```



# Object Expression - YAML

- Files or other representations of Kubernetes Objects are generally represented in YAML.
- A “*Human Friendly*” data serialization standard.
- Uses white space (specifically spaces) alignment to denote ownership.
- Three basic data types:
  - **mappings** - hash or dictionary,
  - **sequences** - array or list
  - **scalars** - string, number, boolean etc

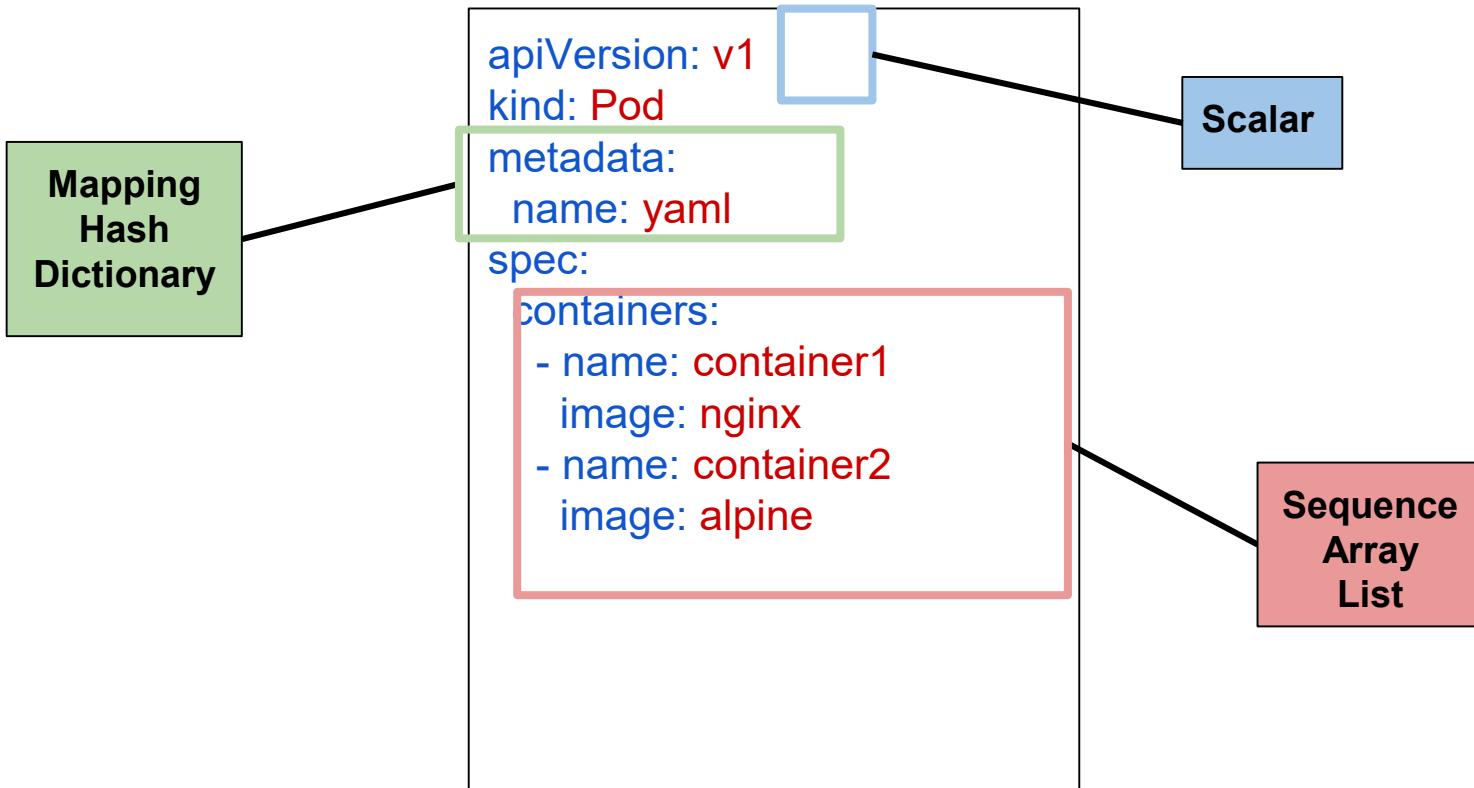


# Object Expression - YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: yaml
spec:
  containers:
    - name: container1
      image: nginx
    - name: container2
      image: alpine
```



# Object Expression - YAML





# YAML vs JSON

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
```

```
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "pod-example"
  },
  "spec": {
    "containers": [
      {
        "name": "nginx",
        "image": "nginx:stable-alpine",
        "ports": [ { "containerPort": 80 } ]
      }
    ]
  }
}
```

# Object Model - Workloads



- Workload related objects within Kubernetes have an additional two nested fields **spec** and **status**.
  - **spec** - Describes the **desired state** or **configuration** of the object to be created.
  - **status** - Is managed by Kubernetes and describes the **actual state** of the object and its history.

# Workload Object Example



## Example Object

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
  ports:
    - containerPort: 80
```

## Example Status Snippet

```
status:
  conditions:
    - lastProbeTime: null
      lastTransitionTime: 2018-02-14T14:15:52Z
      status: "True"
      type: Ready
    - lastProbeTime: null
      lastTransitionTime: 2018-02-14T14:15:49Z
      status: "True"
      type: Initialized
    - lastProbeTime: null
      lastTransitionTime: 2018-02-14T14:15:49Z
      status: "True"
      type: PodScheduled
```

# Core Objects

- Namespaces
- Pods
- Labels
- Selectors
- Services

Concepts and Resources

# Core Concepts



Kubernetes has several core building blocks that make up the foundation of their higher level components.

**Namespaces**

**Pods**

**Labels**

**Services**

**Selectors**

# Namespaces



Namespaces are a logical cluster or environment, and are the primary method of partitioning a cluster or scoping access.

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    app: MyBigWebApp
```

```
$ kubectl get ns --show-labels
NAME      STATUS  AGE   LABELS
default   Active  11h   <none>
kube-public   Active  11h   <none>
kube-system   Active  11h   <none>
prod       Active  6s    app=MyBigWebApp
```



# Default Namespaces

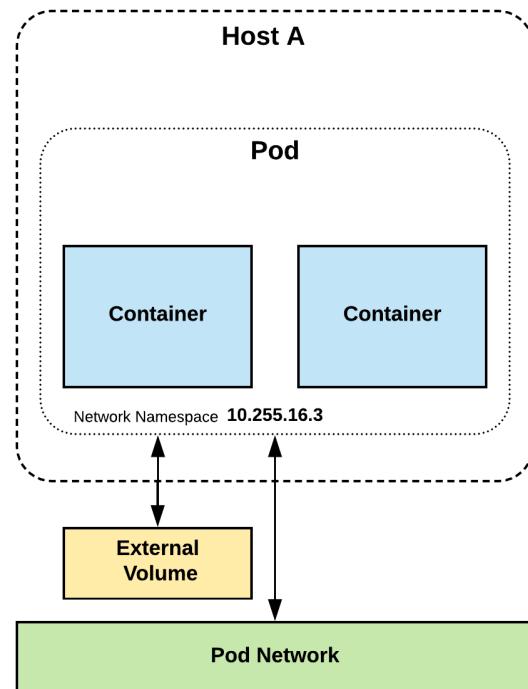
- **default**: The default namespace for any object without a namespace.
- **kube-system**: Acts as the home for objects and resources created by Kubernetes itself.
- **kube-public**: A special namespace; readable by all users that is reserved for cluster bootstrapping and configuration.

```
$ kubectl get ns --show-labels
NAME      STATUS  AGE   LABELS
default   Active  11h   <none>
kube-public   Active  11h   <none>
kube-system  Active  11h   <none>
```

# Pod



- **Atomic unit** or smallest “*unit of work*” of Kubernetes.
- Foundational building block of Kubernetes Workloads.
- Pods are one or more containers that share volumes, a network namespace, and are a part of a **single context**.



# Pod Examples



```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
    - name: content
      image: alpine:latest
      command: ["/bin/sh", "-c"]
      args:
        - while true; do
            date >> /html/index.html;
            sleep 5;
        done
      volumeMounts:
        - name: html
          mountPath: /html
  volumes:
    - name: html
      emptyDir: {}
```

# Key Pod Container Attributes



- **name** - The name of the container
- **image** - The container image
- **ports** - array of ports to expose. Can be granted a friendly name and protocol may be specified
- **env** - array of environment variables
- **command** - Entrypoint array (equiv to Docker **ENTRYPOINT**)
- **args** - Arguments to pass to the command (equiv to Docker **CMD**)

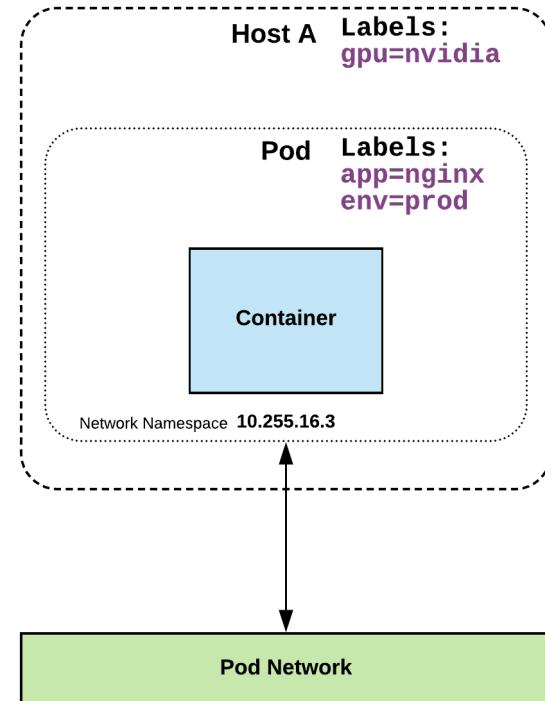
## Container

```
name: nginx
image: nginx:stable-alpine
ports:
- containerPort: 80
  name: http
  protocol: TCP
env:
- name: MYVAR
  value: isAwesome
command: ["/bin/sh", "-c"]
args: ["echo ${MYVAR}"]
```

# Labels



- key-value pairs that are used to identify, describe and group together related sets of objects or resources.
- **NOT** characteristic of uniqueness.
- Have a strict syntax with a slightly limited character set\*.

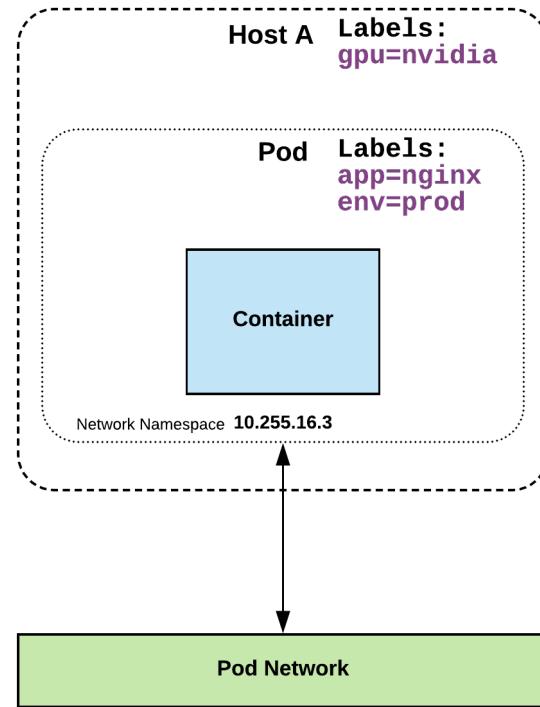


\* <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#syntax-and-character-set>

# Label Example



```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
labels:
  app: nginx
  env: prod
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
```



# Selectors

Selectors use labels to filter or select objects, and are used throughout Kubernetes.

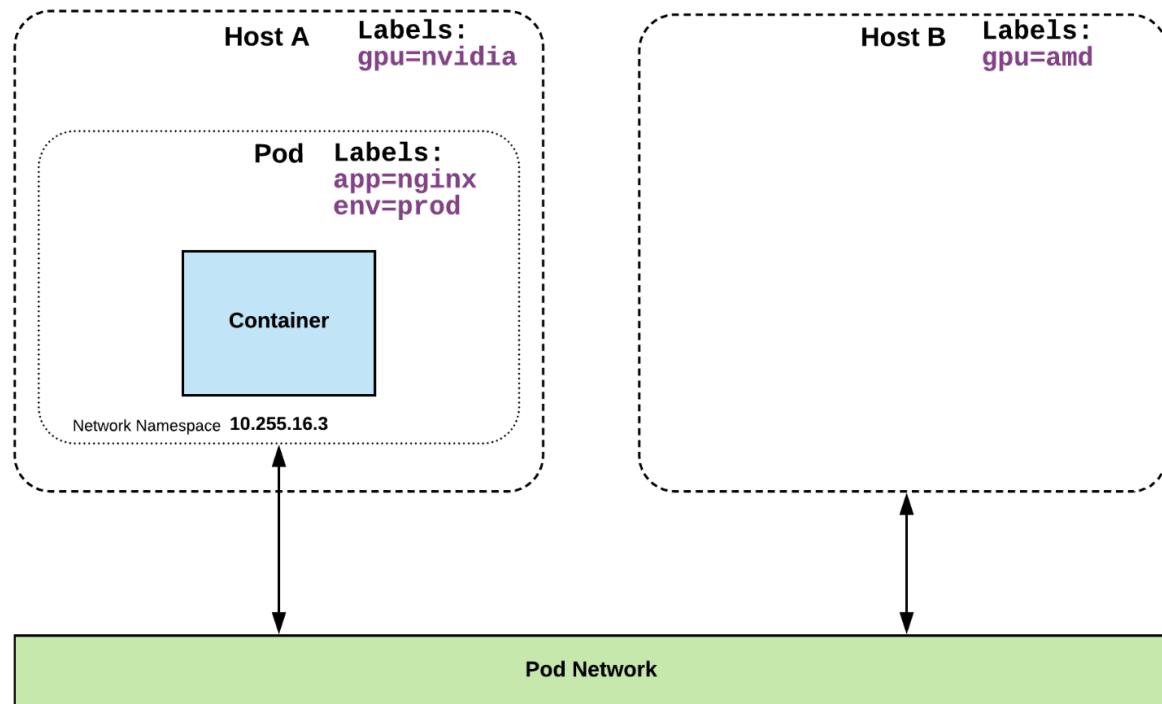
```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
labels:
  app: nginx
  env: prod
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
nodeSelector:
  gpu: nvidia
```





# Selector Example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
nodeSelector:
  gpu: nvidia
```



# Selector Types



**Equality based** selectors allow for simple filtering (`=`, `==`, or `!=`).

```
selector:  
  matchLabels:  
    gpu: nvidia
```

**Set-based** selectors are supported on a limited subset of objects. However, they provide a method of filtering on a set of values, and supports multiple operators including: `in`, `notin`, and `exist`.

```
selector:  
  matchExpressions:  
    - key: gpu  
      operator: in  
      values: ["nvidia"]
```



# Services

- **Unified method of accessing** the exposed workloads of Pods.
- **Durable resource** (unlike Pods)
  - static cluster-unique IP
  - static namespaced DNS name

**<service name>.<namespace>.svc.cluster.local**

# Services



- Target Pods using **equality based selectors**.
- Uses **kube-proxy** to provide simple load-balancing.
- **kube-proxy** acts as a daemon that creates **local entries** in the host's iptables for every service.



# Service Types

There are 4 major service types:

- **ClusterIP** (default)
- **NodePort**
- **LoadBalancer**
- **ExternalName**

# ClusterIP Service



**ClusterIP** services exposes a service on a strictly cluster internal virtual IP.

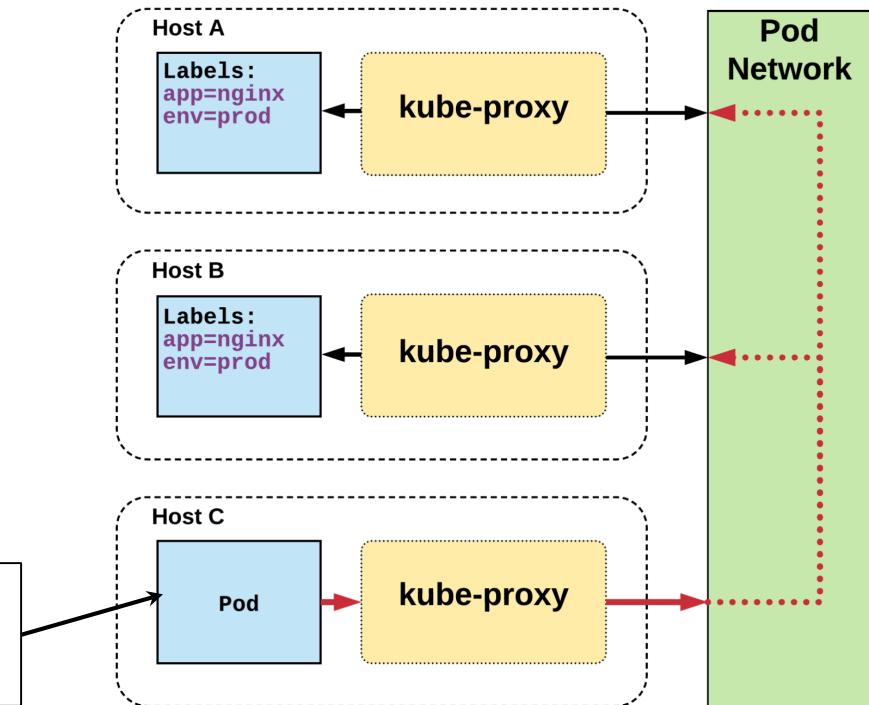
```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  selector:
    app: nginx
    env: prod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

# Cluster IP Service



```
Name: example-prod  
Selector: app=nginx,env=prod  
Type: ClusterIP  
IP: 10.96.28.176  
Port: <unset> 80/TCP  
TargetPort: 80/TCP  
Endpoints: 10.255.16.3:80,  
           10.255.16.4:80
```

```
/ # nslookup example-prod.default.svc.cluster.local  
Name: example-prod.default.svc.cluster.local  
Address 1: 10.96.28.176 example-prod.default.svc.cluster.local
```



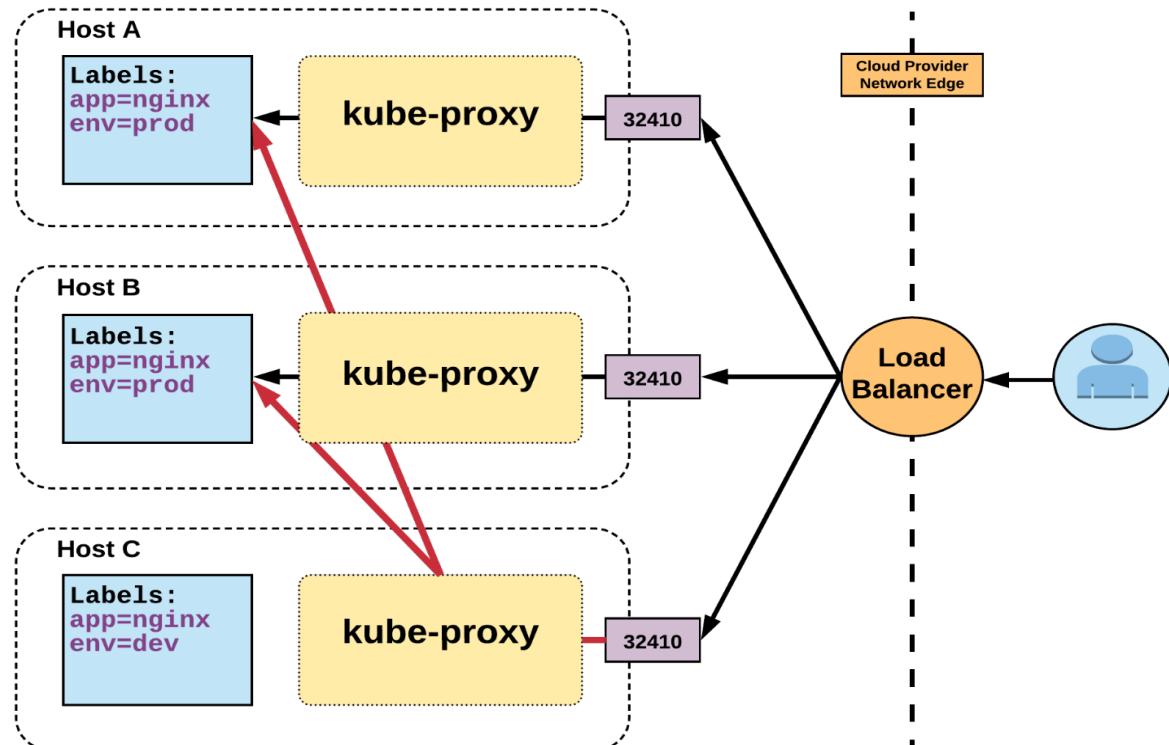
# LoadBalancer Service



- **LoadBalancer** services extend **NodePort**.
- Works in conjunction with an external system to map a cluster external IP to the exposed service.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: LoadBalancer
  selector:
    app: nginx
    env: prod
  ports:
    protocol: TCP
    port: 80
    targetPort: 80
```

# LoadBalancer Service



```
Name: example-prod
Selector: app=nginx,env=prod
Type: LoadBalancer
IP: 10.96.28.176
LoadBalancer
Ingress: 172.17.18.43
Port: <unset> 80/TCP
TargetPort: 80/TCP
NodePort: <unset> 32410/TCP
Endpoints: 10.255.16.3:80,
           10.255.16.4:80
```



# ExternalName Service

- **ExternalName** is used to reference endpoints **OUTSIDE** the cluster.
- Creates an internal **CNAME** DNS entry that aliases another.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: ExternalName
  spec:
    externalName: example.com
```

# Demo

PODS  
SERVICE  
DEPLOYMENT

Section #2

# Workloads

- **ReplicaSet**
- **Deployment**
- **DaemonSet**
- **StatefulSet**
- **Job**
- **CronJob**

# Workloads



Workloads within Kubernetes are higher level objects that manage Pods or other higher level objects.

In **ALL CASES** a Pod Template is included, and acts the base tier of management.

# Pod Template



- Workload Controllers manage instances of Pods based off a provided template.
- Pod Templates are Pod specs with limited metadata.
- Controllers use Pod Templates to make actual pods.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
labels:
  app: nginx
spec:
  containers:
    - name: nginx
      image: nginx
```

```
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx
```

# ReplicaSet



- Primary method of managing pod replicas and their lifecycle.
- Includes their scheduling, scaling, and deletion.
- Their job is simple: **Always ensure the desired number of pods are running.**





# ReplicaSet

- **replicas:** The desired number of instances of the Pod.
- **selector:** The label selector for the **ReplicaSet** will manage **ALL** Pod instances that it targets; whether it's desired or not.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-example
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  template:
    <pod template>
```

# ReplicaSet



```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-example
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  template:
    metadata:
      labels:
        app: nginx
        env: prod
    spec:
      containers:
        - name: nginx
          image: nginx:stable-alpine
          ports:
            - containerPort: 80
```

```
$ kubectl get pods
NAME      READY   STATUS  RESTARTS AGE
rs-example-9l4dt  1/1   Running 0       1h
rs-example-b7bcg  1/1   Running 0       1h
rs-example-mkll2  1/1   Running 0       1h
```

```
$ kubectl describe rs rs-example
Name:           rs-example
Namespace:      default
Selector:       app=nginx,env=prod
Labels:         app=nginx
               env=prod
Annotations:   <none>
Replicas:      3 current / 3 desired
Pods Status:   3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx
           env=prod
  Containers:
    nginx:
      Image:  nginx:stable-alpine
      Port:   80/TCP
      Environment: <none>
      Mounts:  <none>
      Volumes: <none>
  Events:
    Type  Reason     Age From           Message
    ----  ----     --  --   -----
    Normal SuccessfulCreate 16s replicaset-controller Created pod: rs-example-mkll2
    Normal SuccessfulCreate 16s replicaset-controller Created pod: rs-example-b7bcg
    Normal SuccessfulCreate 16s replicaset-controller Created pod: rs-example-9l4dt
```



# Deployment

- Declarative method of managing Pods via **ReplicaSets**.
- Provide rollback functionality and update control.
- Updates are managed through the **pod-template-hash** label.
- Each iteration creates a unique label that is assigned to both the **ReplicaSet** and subsequent Pods.



# Deployment



- `revisionHistoryLimit`: The number of previous iterations of the Deployment to retain.
- `strategy`: Describes the method of updating the Pods based on the `type`. Valid options are **RollingUpdate** or **Recreate**.
  - **RollingUpdate**: Cycles through updating the Pods according to the parameters: `maxSurge` and `maxUnavailable`.
  - **Recreate**: All existing Pods are killed before the new ones are created.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-example
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    <pod template>
```



# DaemonSet

- Ensure that all nodes matching certain criteria will run an instance of the supplied Pod.
- They **bypass** default scheduling mechanisms.
- Are ideal for cluster wide services such as log forwarding, or health monitoring.
- Revisions are managed via a **controller-revision-hash** label.



# DaemonSet



- `revisionHistoryLimit`: The number of previous iterations of the DaemonSet to retain.
- `updateStrategy`: Describes the method of updating the Pods based on the `type`. Valid options are **RollingUpdate** or **OnDelete**.
  - **RollingUpdate**: Cycles through updating the Pods according to the value of `maxUnavailable`.
  - **OnDelete**: The new instance of the Pod is deployed **ONLY** after the current instance is deleted.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ds-example
spec:
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  template:
    spec:
      nodeSelector:
       .nodeType: edge
<pod template>
```



# StatefulSet

- Tailored to managing Pods that must persist or maintain state.
- Pod identity including **hostname**, **network**, and **storage** **WILL** be persisted.
- Assigned a unique ordinal name following the convention of '*<statefulset name>-<ordinal index>*'.





# StatefulSet

- Pod lifecycle will be ordered and follow consistent patterns.
- Ordinal pattern is also passed to the Pod's network Identity and volumes.
- Revisions are managed via a **controller-revision-hash** label



# StatefulSet



```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sts-example
spec:
  replicas: 2
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: stateful
  serviceName: app
  updateStrategy:
    type: RollingUpdate
  rollingUpdate:
    partition: 0
  template:
    metadata:
      labels:
        app: stateful
```

<continued>

<continued>

```
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
      volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
    - metadata:
        name: www
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: standard
        resources:
          requests:
            storage: 1Gi
```



# StatefulSet

- **revisionHistoryLimit:** The number of previous iterations of the StatefulSet to retain.
- **serviceName:** The name of the associated headless service; or a service without a **ClusterIP**.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sts-example
spec:
  replicas: 2
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: stateful
  serviceName: app
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      partition: 0
  template:
    <pod template>
```

# Headless Service



<StatefulSet Name>-<ordinal>.<service name>.<namespace>.svc.cluster.local

```
apiVersion: v1
Kind: Service
metadata:
  name: app
spec:
  clusterIP: None
  selector:
    app: stateful
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

```
/ # dig app.default.svc.cluster.local +noall +answer
; <>> DiG 9.11.2-P1 <>> app.default.svc.cluster.local +noall +answer
;; global options: +cmd
app.default.svc.cluster.local. 2 IN A      10.255.0.5
app.default.svc.cluster.local. 2 IN A      10.255.0.2
```

```
/ # dig sts-example-0.app.default.svc.cluster.local +noall +answer
; <>> DiG 9.11.2-P1 <>> sts-example-0.app.default.svc.cluster.local +noall +answer
;; global options: +cmd
sts-example-0.app.default.svc.cluster.local. 20 IN A 10.255.0.2
```

```
$ kubectl get pods
NAME      READY   STATUS  RESTARTS AGE
sts-example-0  1/1    Running 0       11m
sts-example-1  1/1    Running 0       11m
```

```
/ # dig sts-example-1.app.default.svc.cluster.local +noall +answer
; <>> DiG 9.11.2-P1 <>> sts-example-1.app.default.svc.cluster.local +noall +answer
;; global options: +cmd
sts-example-1.app.default.svc.cluster.local. 30 IN A 10.255.0.5
```



# StatefulSet

- **updateStrategy:** Describes the method of updating the Pods based on the **type**. Valid options are **OnDelete** or **RollingUpdate**.
  - **OnDelete:** The new instance of the Pod is deployed **ONLY** after the current instance is deleted.
  - **RollingUpdate:** Pods with an ordinal greater than the **partition** value will be updated in one-by-one in reverse order.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sts-example
spec:
  replicas: 2
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: stateful
  serviceName: app
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      partition: 0
  template:
    <pod template>
```



# StatefulSet

- **volumeClaimTemplates:** Template of the persistent volume(s) request to use for each instance of the StatefulSet.

```
spec:  
  containers:  
    - name: nginx  
      image: nginx:stable-alpine  
      ports:  
        - containerPort: 80  
      volumeMounts:  
        - name: www  
          mountPath: /usr/share/nginx/html  
  volumeClaimTemplates:  
    - metadata:  
        name: www  
      spec:  
        accessModes: [ "ReadWriteOnce" ]  
        storageClassName: standard  
        resources:  
          requests:  
            storage: 1Gi
```



# Job

- Job controller ensures one or more pods are executed and successfully terminate.
- Will continue to try and execute the job until it satisfies the completion and/or parallelism condition.
- Pods are **NOT** cleaned up until the job itself is deleted.\*



# Job



- **backoffLimit**: The number of failures before the job itself is considered **failed**.
- **completions**: The total number of successful completions desired.
- **parallelism**: How many instances of the pod can be run concurrently.
- **spec.template.spec.restartPolicy**: Jobs only support a **restartPolicy** of type **Never** or **OnFailure**.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-example
spec:
  backoffLimit: 4
  completions: 4
  parallelism: 2
  template:
    spec:
      restartPolicy: Never
    <pod-template>
```

# CronJob



An extension of the Job Controller, it provides a method of executing jobs on a cron-like schedule.

CronJobs within Kubernetes  
use **UTC ONLY**.



# CronJob



- **schedule:** The cron schedule for the job.
- **successfulJobHistoryLimit:** The number of successful jobs to retain.
- **failedJobHistoryLimit:** The number of failed jobs to retain.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-example
spec:
  schedule: "*/1 * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      completions: 4
      parallelism: 2
      template:
        <pod template>
```

# Demo

- **ReplicaSet**
- **Deployment**
- **DaemonSet**
- **StatefulSet**
- **Job**
- **CronJob**

# Advanced Concepts

Networking  
Storage  
Configuration

Section #3

# Networking

Architecture Overview



# Kubernetes Networking

- **Pod Network**

- Cluster-wide network used for pod-to-pod communication managed by a CNI (Container Network Interface) plugin.

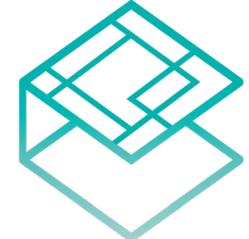
- **Service Network**

- Cluster-wide range of **Virtual IPs** managed by **kube-proxy** for service discovery.

# Container Network Interface (CNI)



- Pod networking within Kubernetes is plumbed via the Container Network Interface (CNI).
- Functions as an interface between the container runtime and a **network implementation plugin**.
- CNCF Project
- Uses a simple JSON Schema.



C N I

# CNI Plugins



- Azure CNI
- Amazon ECS
- Calico
- Cillium
- Contiv
- Contrail
- Flannel



cilium



- GCE
- kube-router
- Multus
- OpenVSwitch
- Romana
- Weave



# Fundamental Networking Rules



- All containers within a pod can communicate with each other unimpeded.
- All Pods can communicate with all other Pods without NAT.
- All nodes can communicate with all Pods (and vice-versa) without NAT.
- The IP that a Pod sees itself as is the same IP that others see it as.



# Fundamentals Applied

- **Container-to-Container**

- Containers within a pod exist within the **same network namespace** and share an IP.
- Enables intrapod communication over *localhost*.

- **Pod-to-Pod**

- Allocated **cluster unique IP** for the duration of its life cycle.
- Pods themselves are fundamentally ephemeral.

# Storage

- **Volumes**
- **Persistent Volumes**
- **Persistent Volume Claims**
- **StorageClass**

# Storage



Pods by themselves are useful, but many workloads require exchanging data between containers, or persisting some form of data.

For this we have **Volumes**, **PersistentVolumes**, **PersistentVolumeClaims**, and **StorageClasses**.



# Volumes

- Storage that is tied to the **Pod's Lifecycle**.
- A pod can have one or more types of volumes attached to it.
- Can be consumed by any of the containers within the pod.
- Survive Pod restarts; however their durability beyond that is dependent on the Volume Type.



# Volume Types

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- configMap
- csi
- downwardAPI
- emptyDir
- fc (fibre channel)
- flocker
- gcePersistentDisk
- gitRepo
- glusterfs
- hostPath
- iscsi
- local
- nfs
- persistentVolume  
Claim
- projected
- portworxVolume
- quobyte
- rbd
- scaleIO
- secret
- storageos
- vsphereVolume



Persistent Volume Supported

# Volumes



- **volumes:** A list of volume objects to be attached to the Pod. Every object within the list must have its own unique **name**.
- **volumeMounts:** A container specific list referencing the Pod volumes by **name**, along with their desired **mountPath**.

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
          readOnly: true
        - name: content
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args:
            - while true; do
                date >> /html/index.html;
                sleep 5;
            done
      volumeMounts:
        - name: html
          mountPath: /html
    volumes:
      - name: html
        emptyDir: {}
```

# Volumes



- **volumes:** A list of volume objects to be attached to the Pod. Every object within the list must have its own unique **name**.
- **volumeMounts:** A container specific list referencing the Pod volumes by **name**, along with their desired **mountPath**.

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
          readOnly: true
        - name: content
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args:
            - while true; do
                date >> /html/index.html;
                sleep 5;
            done
      volumeMounts:
        - name: html
          mountPath: /html
  volumes:
    - name: html
      emptyDir: {}
```



# Volumes

- **volumes:** A list of volume objects to be attached to the Pod. Every object within the list must have its own unique **name**.
- **volumeMounts:** A container specific list referencing the Pod volumes by **name**, along with their desired **mountPath**.

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
          readOnly: true
        - name: content
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args:
            - while true; do
                date >> /html/index.html;
                sleep 5;
            done
      volumeMounts:
        - name: html
          mountPath: /html
  volumes:
    - name: html
      emptyDir: {}
```



# Persistent Volumes

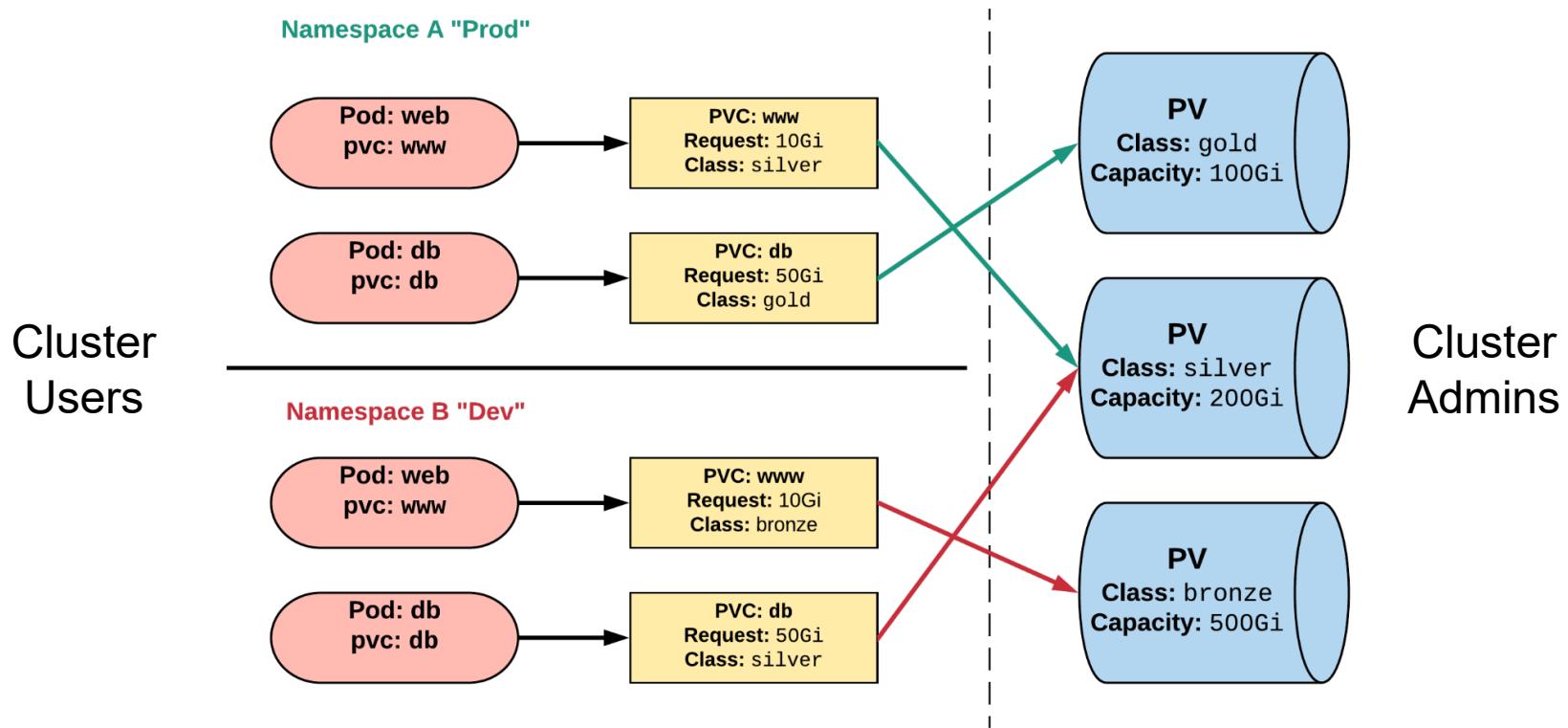
- A **PersistentVolume** (PV) represents a storage resource.
- PVs are a **cluster wide resource** linked to a backing storage provider: NFS, Azure Disk, RBD etc.
- Generally provisioned by an administrator.
- Their lifecycle is handled independently from a pod
- **CANNOT** be attached to a Pod directly. Relies on a **PersistentVolumeClaim**

# PersistentVolumeClaims



- A **PersistentVolumeClaim** (PVC) is a **namespaced** request for storage.
- Satisfies a set of requirements instead of mapping to a storage resource directly.
- Ensures that an application's '*claim*' for storage is portable across numerous backends or providers.

# Persistent Volumes and Claims



# PVs and PVCs with Selectors



```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-selector-example
  labels:
    type: hostpath
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/mnt/data"
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-selector-example
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      type: hostpath
```

# PVs and PVCs with Selectors



```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-selector-example
  labels:
    type: hostpath
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/mnt/data"
```

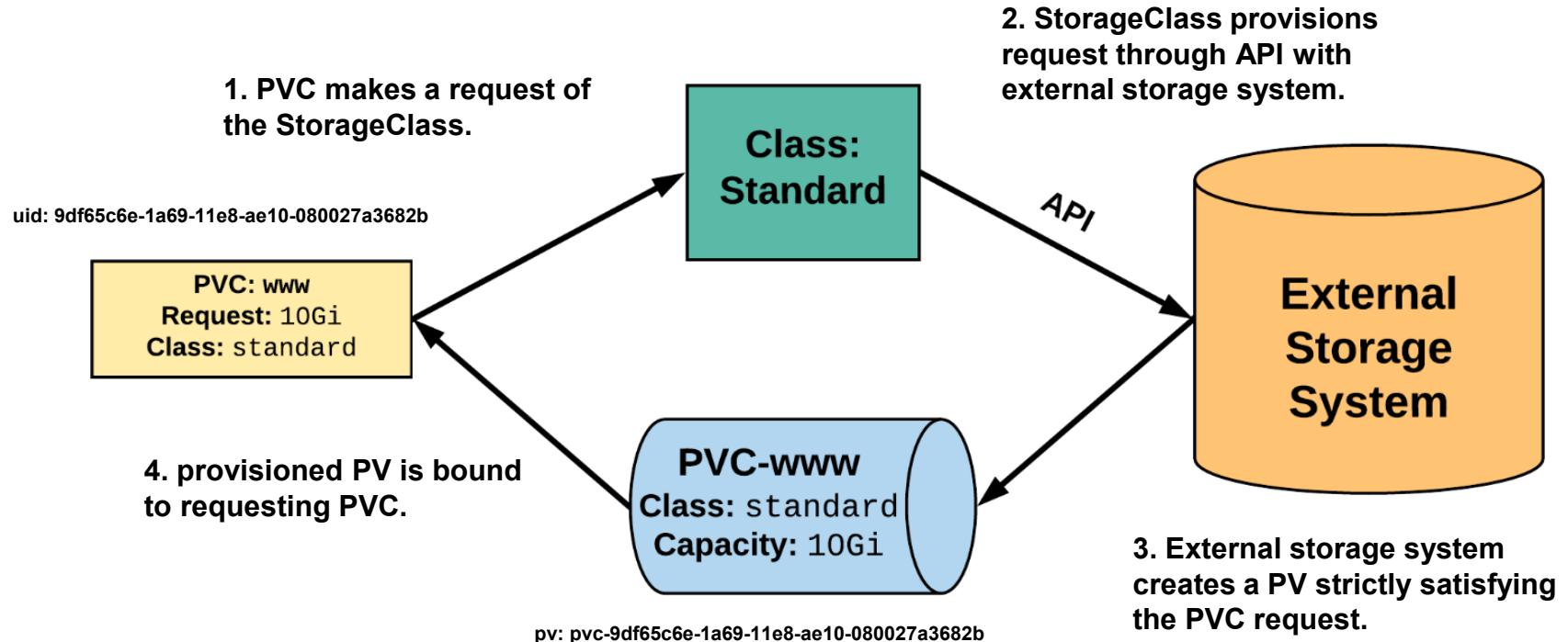
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-selector-example
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      type: hostpath
```



# StorageClass

- Storage classes are an abstraction on top of an external storage resource (PV)
- Work hand-in-hand with the external storage system to enable **dynamic provisioning** of storage
- Eliminates the need for the cluster admin to pre-provision a PV

# StorageClass





# Available StorageClasses

- AWSElasticBlockStore
- AzureFile
- AzureDisk
- CephFS
- Cinder
- FC
- Flocker
- GCEPersistentDisk
- Glusterfs
- iSCSI
- Quobyte
- NFS
- RBD
- VsphereVolume
- PortworxVolume
- ScaleIO
- StorageOS
- Local

# Demo

PVC  
PV

Section #2

# Configuration

- **ConfigMap**
- **Secret**

Concepts and Resources

# Configuration



Kubernetes has an integrated pattern for decoupling configuration from application or container.

This pattern makes use of two Kubernetes components: **ConfigMaps** and **Secrets**.



# ConfigMap

- Externalized data stored within kubernetes.
- Can be referenced through several different means:
  - environment variable
  - a command line argument (via env var)
  - injected as a file into a volume mount
- Can be created from a manifest, literals, directories, or files directly.

# ConfigMap



**data:** Contains key-value pairs of ConfigMap contents.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-example
data:
  state: Michigan
  city: Ann Arbor
  content: |
    Look at this,
    its multiline!
```

# ConfigMap Example



All produce a **ConfigMap** with the same content!

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-example
data:
  city: Ann Arbor
  state: Michigan
```

```
$ kubectl create configmap literal-example \
> --from-literal="city=Ann Arbor" --from-literal=state=Michigan
configmap "literal-example" created
```

```
$ cat info/city
Ann Arbor
$ cat info/state
Michigan
$ kubectl create configmap dir-example --from-file=cm/
configmap "dir-example" created
```

```
$ cat info/city
Ann Arbor
$ cat info/state
Michigan
$ kubectl create configmap file-example --from-file=cm/city --from-file=cm/state
configmap "file-example" created
```

# Secret



- Functionally identical to a ConfigMap.
- Stored as **base64 encoded content**.
- Encrypted at rest within etcd (**if configured!**).
- Ideal for username/passwords, certificates or other sensitive information that should not be stored in a container.
- Can be created from a manifest, literals, directories, or from files directly.

# Secret



- **type:** There are three different types of secrets within Kubernetes:
  - **docker-registry** - credentials used to authenticate to a container registry
  - **generic/Opaque** - literal values from different sources
  - **tls** - a certificate based secret
- **data:** Contains key-value pairs of base64 encoded content.

```
apiVersion: v1
kind: Secret
metadata:
  name: manifest-secret
type: Opaque
data:
  username: ZXhhbXBsZQ==
  password: bXIwYXNzd29yZA==
```

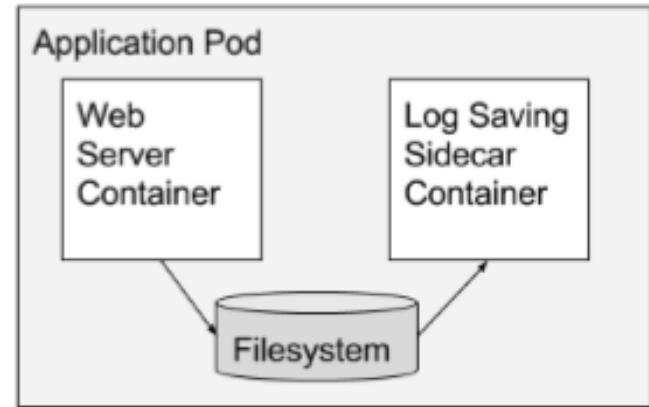
# Patterns

Section #6

# Sidecar Pattern



- Main container paired with “sidecar” container
- Containers are on the same machine and share the disk
- “sidecar” can scavenge spare CPU cycles

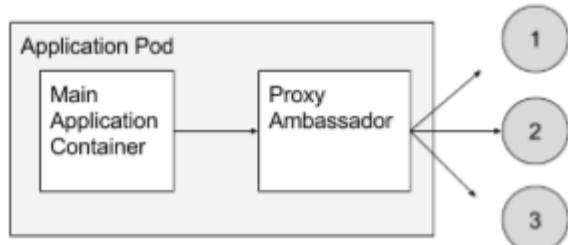


Design patterns for container-based distributed systems Brendan Burns David Oppenheimer

# Ambassador Pattern



- Sidecar proxies all communication to and from the main container
- Simplified view of “outside”
- Sidecar and main container share the network interface

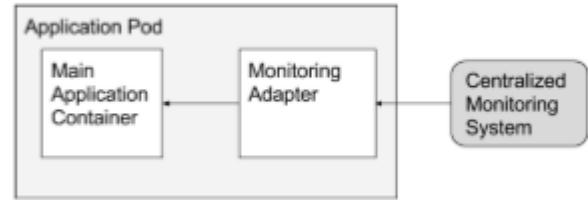


Design patterns for container-based distributed systems Brendan Burns David Oppenheimer



# Adapter Pattern

- In contrast to Ambassador pattern, adapter provides standard view of the container to the outside world

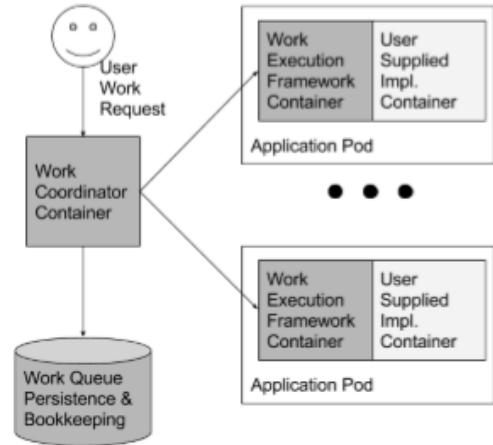


Design patterns for container-based distributed systems Brendan Burns David Oppenheimer



# Work queue Pattern

- All the logic to convert a file system into a work queue is implemented by the sidecar



Design patterns for container-based distributed systems Brendan Burns David Oppenheimer

# Microservices Challenge



- As the # of microservices grow in size and complexity, it becomes harder to understand and manage. This includes
  - discovery, load balancing, failure recovery, metrics, and monitoring.
  - A/B testing, canary releases, rate limiting, access control, and end-to-end authentication.



# Service Mesh

- A network of microservices that make up such applications and the interactions between them.

# Istio



- Makes it easy to create a network of deployed services with load balancing, service-to-service authentication, monitoring, and more, without any changes in service code.
- Deployed using a special sidecar proxy

# Istio

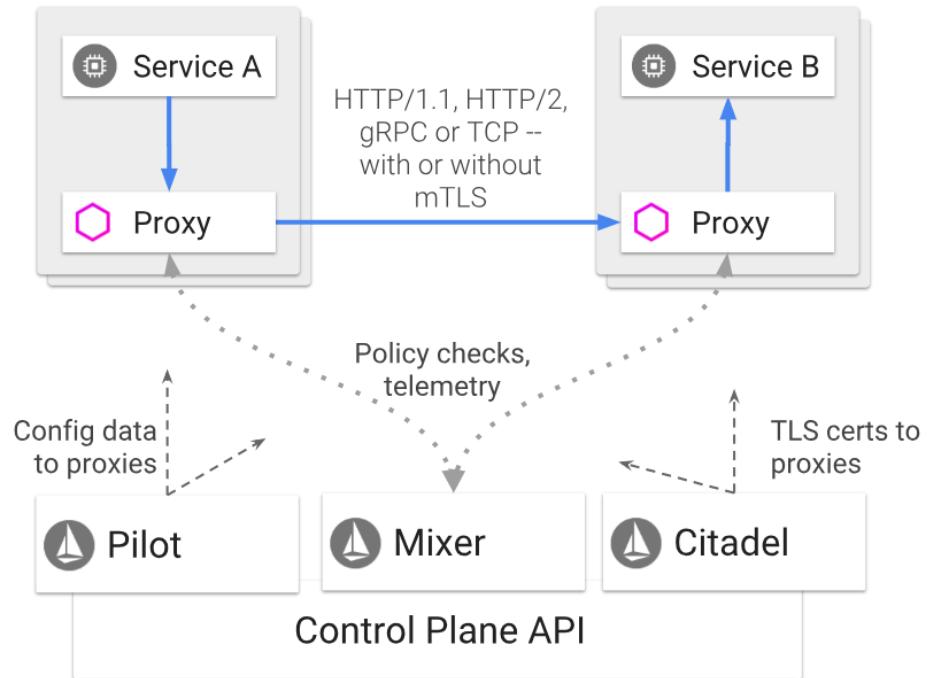


- Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic.
- Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection.
- A pluggable policy layer and configuration API supporting access controls, rate limits and quotas.



- Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress.
- Secure service-to-service communication in a cluster with strong identity-based authentication and authorization.

# Istio Architecture



# Istio Architecture



- The **data plane** is composed of a set of intelligent proxies ([Envoy](#)) deployed as sidecars. These proxies mediate and control all network communication between microservices along with [Mixer](#), a general-purpose policy and telemetry hub.

# Istio Architecture



- The **control plane** manages and configures the proxies to route traffic. Additionally, the control plane configures Mixers to enforce policies and collect telemetry.



# Envoy

- Istio uses an extended version of the [Envoy](#) proxy.
- Envoy is deployed as a **sidecar** to the relevant service in the same Kubernetes pod.

# Envoy



- Dynamic service discovery
- Load balancing
- TLS termination
- HTTP/2 and gRPC proxies
- Circuit breakers
- Health checks
- Staged rollouts with %-based traffic split
- Fault injection
- Rich metrics

# Demo

## Side car pattern Istio

Section #2

# Azure Kubernetes Service (AKS)

Section #4

# AKS at a glance!



Built-in  
auto scaling



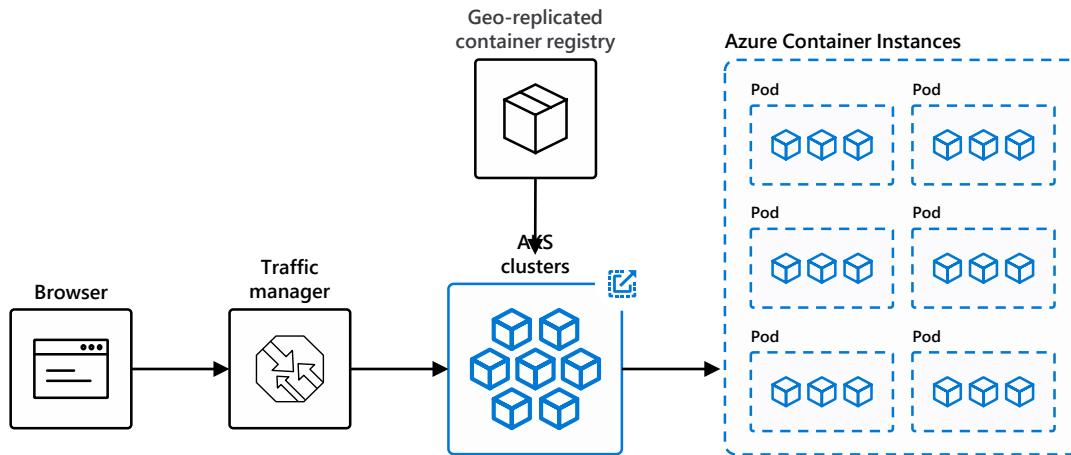
Global  
data center



Elastically burst  
using ACI



Geo-replicated  
container registry





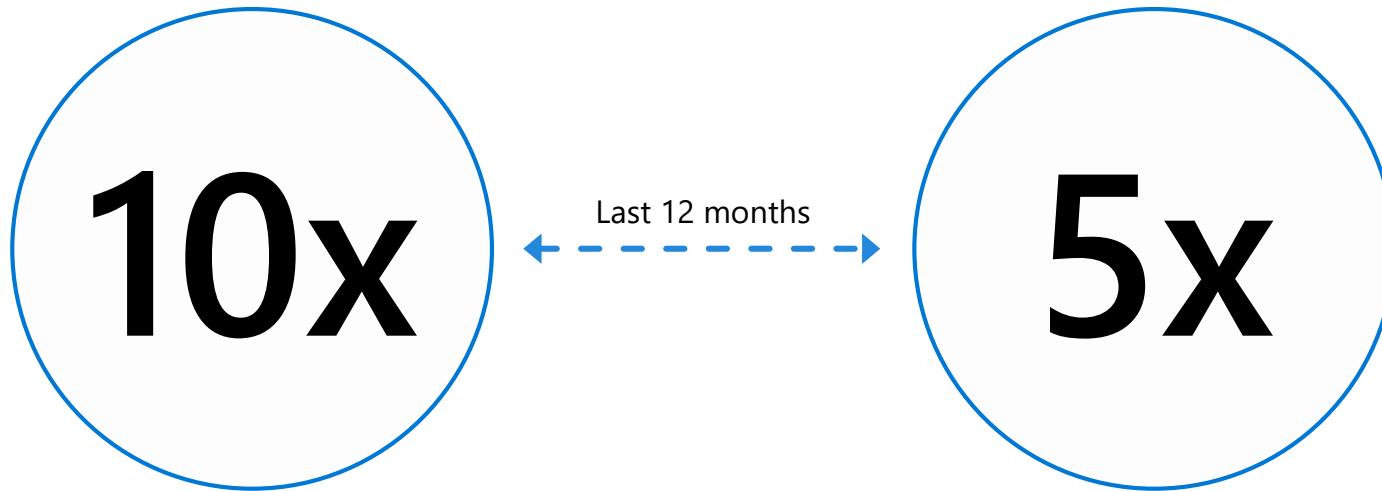
- CNCF Certified Managed Kubernetes
- Identity and Security (RBAC)
- Integrated Logging (apps, nodes, controllers)
- Scaling (Cluster Nodes, Pods)
- Cluster Node Upgrades
- GPU enabled

# AKS



- Storage Support
- VNET Integration / Ingress
- Dev Tooling
- Private Container Registry
- Kubernetes Certification
- Regulatory Compliance

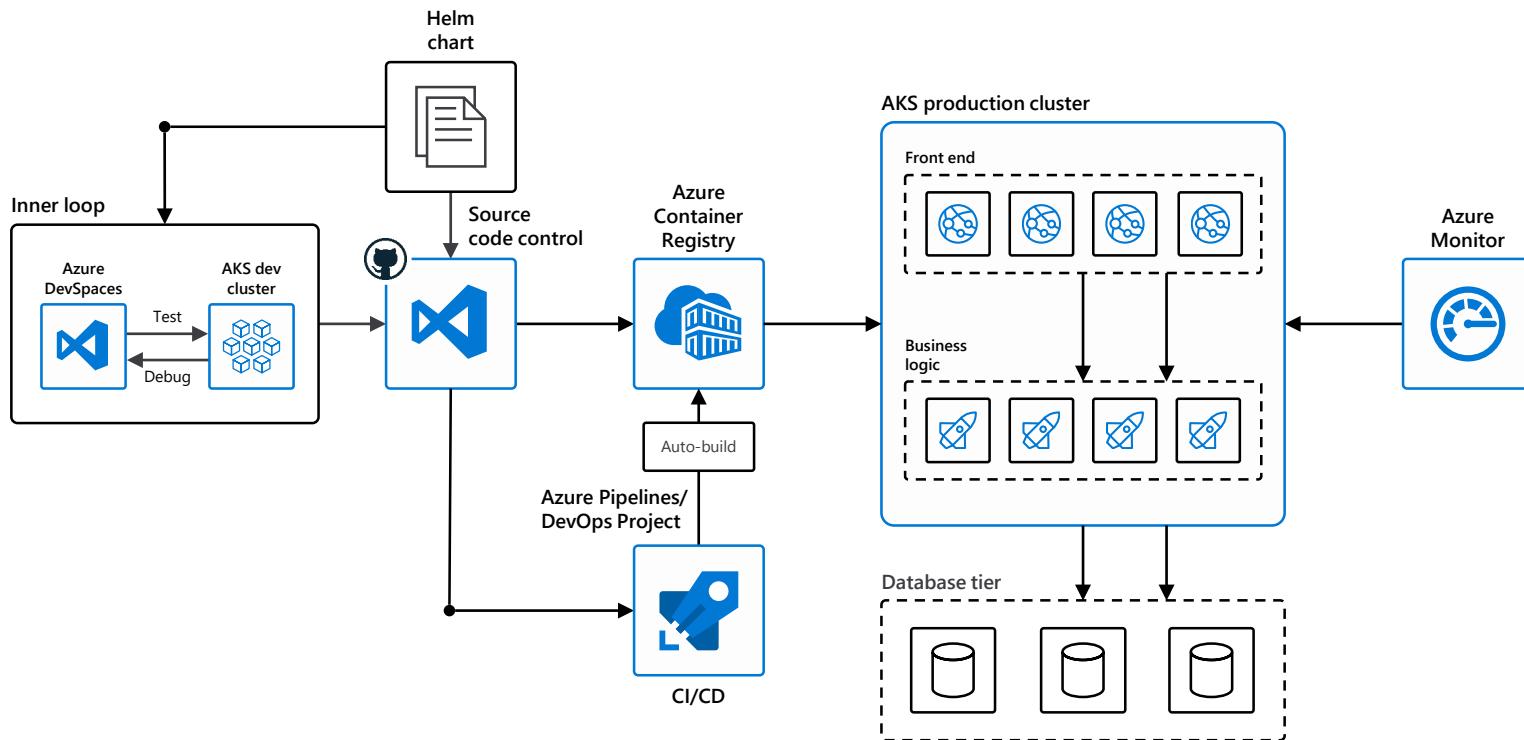
# AKS Momentum



Kubernetes on Azure  
**usage** grew 10x

Kubernetes on Azure  
**customers** grew 5x

# AKS end to end experience

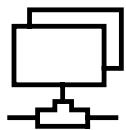


# Demo

## Azure DevOps Dev Space

Section #2

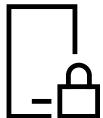
# AKS Security and Compliance



Control access through  
AAD and RBAC



Safeguard keys and  
secrets with Key Vault



Secure network  
communications with  
VNET and CNI



Compliant Kubernetes  
service with  
certifications covering  
SOC, HIPAA, and PCI



# AKS Overview



- Fully managed Kubernetes control plane
- Quick provisioning (<2min)
- SLA-available service (99.95%)
- Transparently scales with cluster size:
  - 1, 50, 100, 250, 500, etc.
- Upstream Kubernetes
  - Extensions from ecosystem



# AKS Operations

- Automated minor version upgrades
- Control Plane Self-healing
  - Component monitoring and restart
  - Storage scaling, upgrades, and recovery
- Control Plane Log Shipping
  - OMS
  - Custom endpoint (syslog, partner APIs)



# AKS Operations (Continued)

- Control plane lifecycle events
- Long-term event storage
- Stream control plane metrics to OMS
  - Bring your own aggregation

# AKS etcd



- SSD-backed
- Fully automated
  - Quorum management
  - Backup, Restore from Azure Blob Storage



# AKS Networking

- Agent Pools in New and Custom VNETs
- Configurable addressing:
  - podCIDR, serviceCIDR
- CNI Support
  - Azure, Calico, etc.
- Azure Application Gateway Ingress Controller



# AKS Security

- TLS across all components
- Secrets backed by Azure Key Vault
- Automated minor version upgrades
  - Control Plane
  - Agent Pools
- Off-cluster audit log



# AKS Identity

- Azure Active Directory Integration
  - kubectl OIDC Authentication
  - AAD Device Auth for 2FA
- Kubernetes RBAC

# Demo

## AKS and Azure AD RBAC

Section #2



# Deploy AKS Cluster

```
az ad sp create-for-rbac --skip-assignment
```

```
az acr show --resource-group myResourceGroup --name <acrName> --query "id" --output tsv
```

```
az role assignment create --assignee <appId> --scope <acrId> --role Reader
```

```
az aks create \  
  --resource-group myResourceGroup \  
  --name myAKSCluster \  
  --node-count 1 \  
  --service-principal <appId> \  
  --client-secret <password> \  
  --generate-ssh-keys
```

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```



# Supported Kubernetes Versions

- AKS supports four minor versions of Kubernetes:
  - The current minor version that is released upstream (n)
  - Three previous minor versions
- If you are on the n-4 version, out of the SLO. A successful upgrade to n-3 gets you back in the SLO



# Virtual Kubelet with AKS

- Containers scheduled on a container instance (ACI), \*as if it is a standard Kubernetes node\*

```
az aks install-connector --resource-group myAKSCluster --name myAKSCluster --connector-name virtual-kubelet --os-type Both
```

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-23443254-0	Ready	agent	16d	v1.9.6
aks-nodepool1-23443254-1	Ready	agent	16d	v1.9.6
aks-nodepool1-23443254-2	Ready	agent	16d	v1.9.6
virtual-kubelet-virtual-kubelet-linux	Ready	agent	4m	v1.8.3
virtual-kubelet-virtual-kubelet-win	Ready	agent	4m	v1.8.3



# Cluster Auto Scaler

- As resource demands increase, the cluster autoscaler allows your cluster to grow, based on constraints you set.
- Scans periodically for pending pods
- When used with the horizontal pod autoscaler (HPA), the HPA will update pod replicas and resources as per demand



# Persistent volumes with Azure Files

- Create an Azure file share
- Create a Kubernetes secret (credentials to access the fil share)
- Mount the file share as a volume



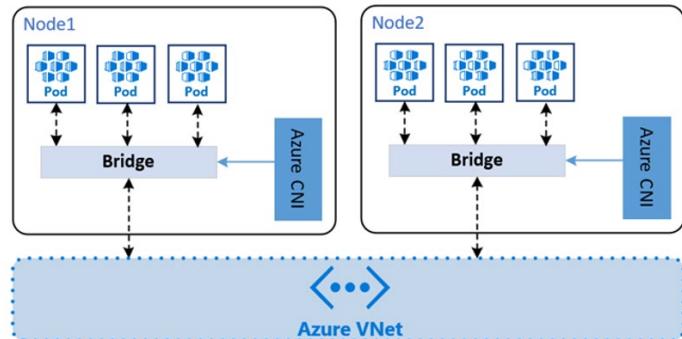
# Network configuration and AKS

- **Basic**

- Completely managed
- No control over VNET
- Based Kubenet plug-in

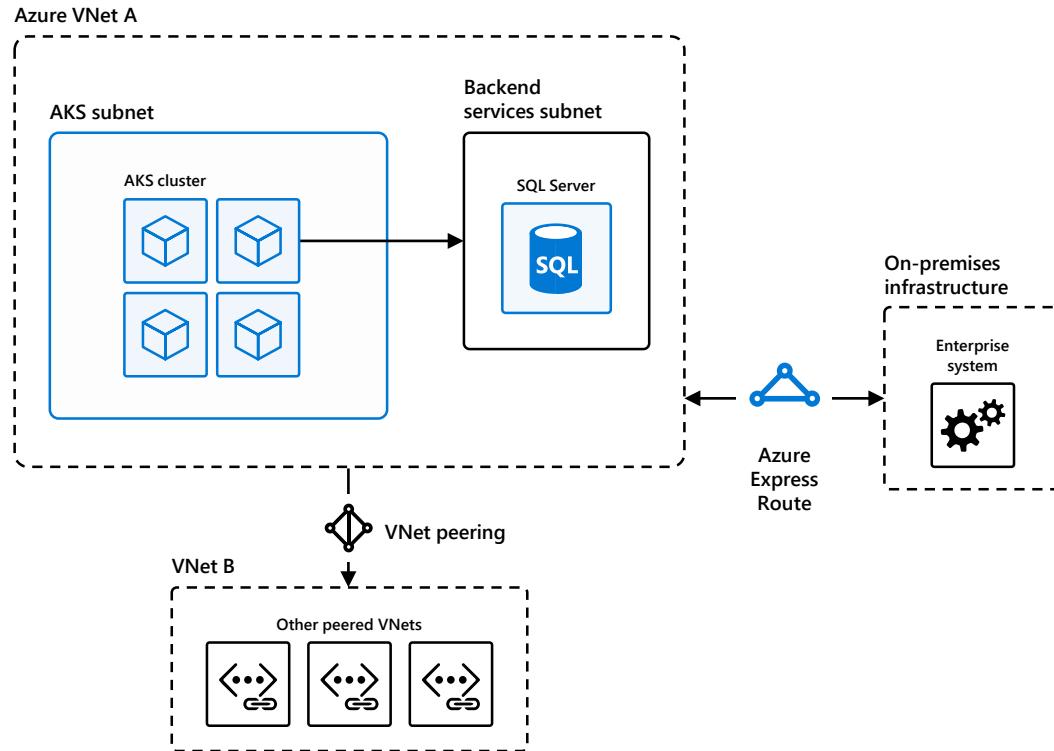
- **Advanced**

- Control over VNET
- Based on Azure CNI plug-in





# AKS With Advanced Networking



# AKS Advanced Networking 1/2



- Deploy AKS cluster existing VNET, create a new VNET
- Every Pod in the cluster is assigned an IP address in the VNET
- Connect to other services on-peered VNET or on-premises
- Pods can connect to



# AKS Advanced Networking 2/2

- Pods can connect to Service Endpoints (like Cosmos, Storage )
- Use User Defined Routes to route traffics from pods to NVAs
- Expose a Service internally or externally via Azure Load Balancer
- One cluster per subnet

# Prereqs: AKS Advanced Networking



- Size of VNET
- Subnet planning – must accommodate all nodes, pods, AKS and Azure resources
- Max pods per node (limits)
- Cannot deploy VMs to the cluster subnet
- Different subnet for Kubernetes services?



# AKS Advanced Networking

- Static IP Address
  - To avoid losing IP assigned to a service
- Internal Load Balancer
  - Expose the services to internal endpoints

`annotations: service.beta.kubernetes.io/azure-load-balancer-internal: "true"`
- Ingress Controller
  - Load balancing, SSL termination, virtual hosting



# AKS Advanced Networking

- Application Routing
- Ingress Controller
  - Implements Ingress resources
- External DNS Controller
  - Watches for Ingress resources and creates a DNS record

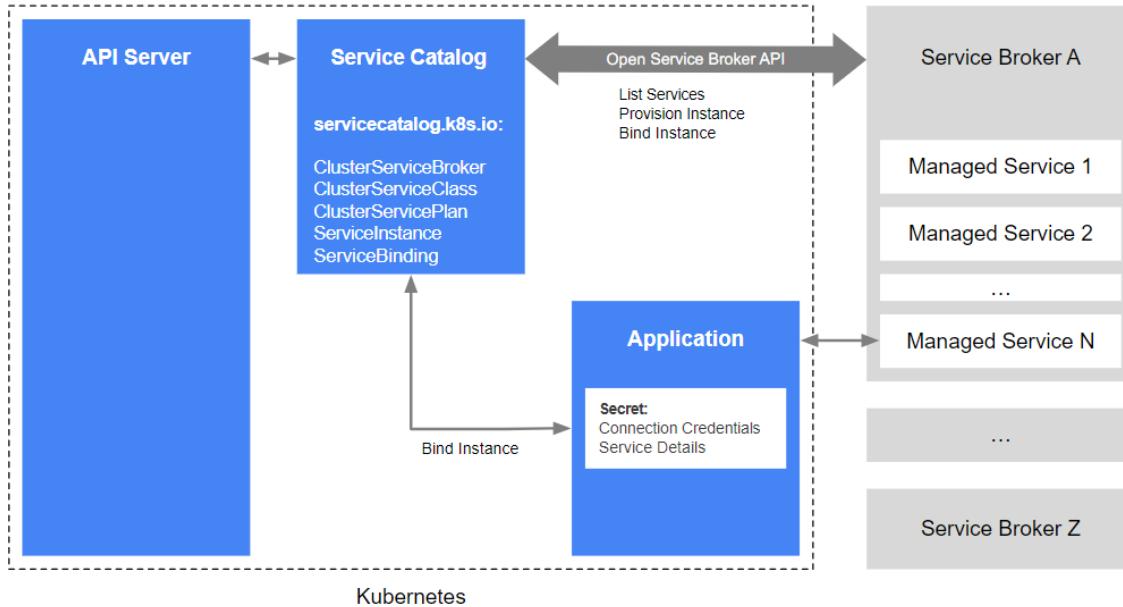


# Kubernetes Service Catalog

- Service Catalog – API that enables services running in Kubernetes to use external managed services
- It provides a way to list, provision and bind with External Service Brokers ( without needing detailed knowledge of the service)
- Open Service Broker API – endpoint for a services offered by third-party



# Kubernetes Service Catalog



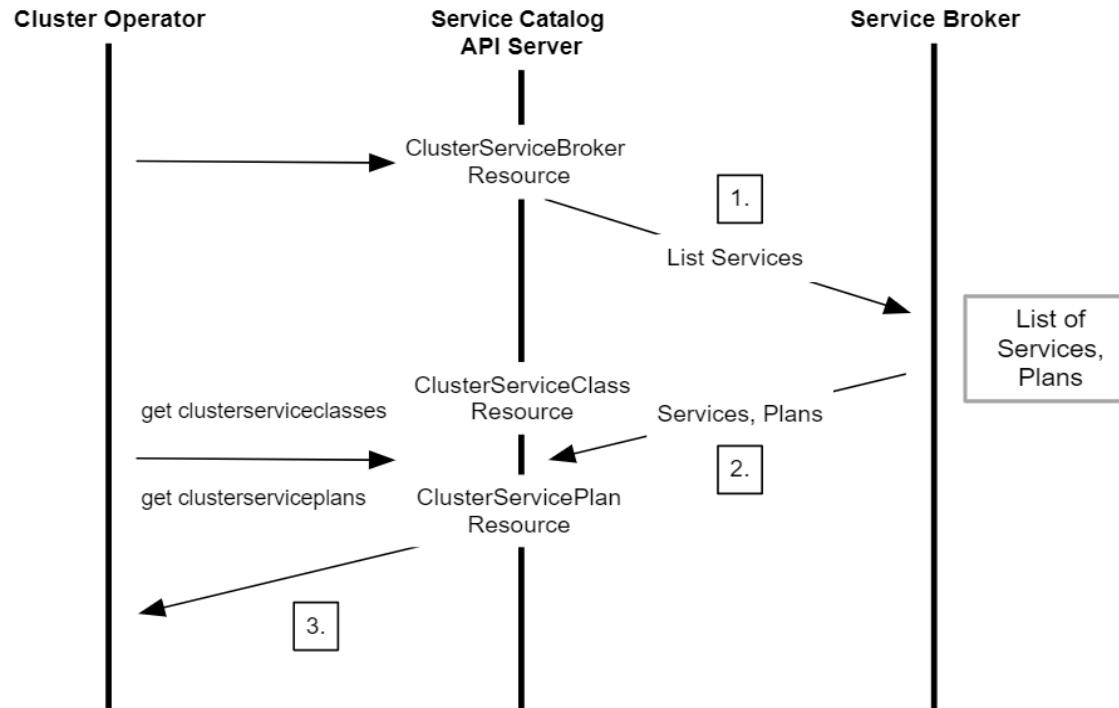


# Kubernetes Service Catalog

- Listing the managed services and Service Plans available from a service broker.
- Provisioning a new instance of the managed service.
- Binding to the managed service, which returns the connection credentials.
- Mapping the connection credentials into the application.

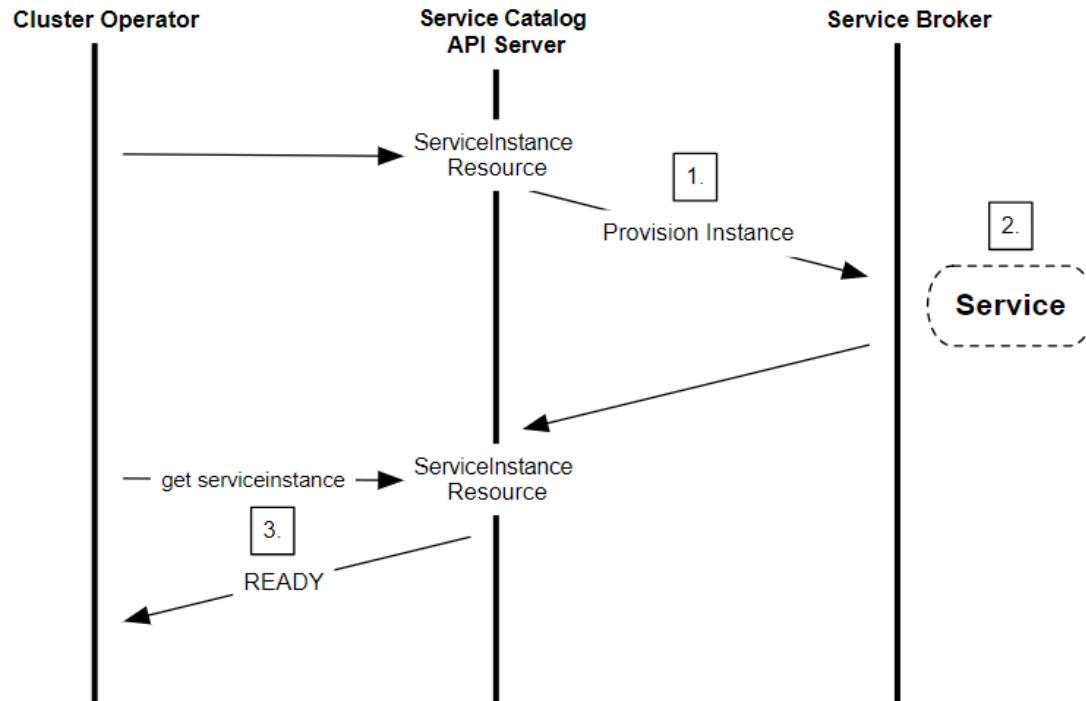


# Listing Managed Service Plans



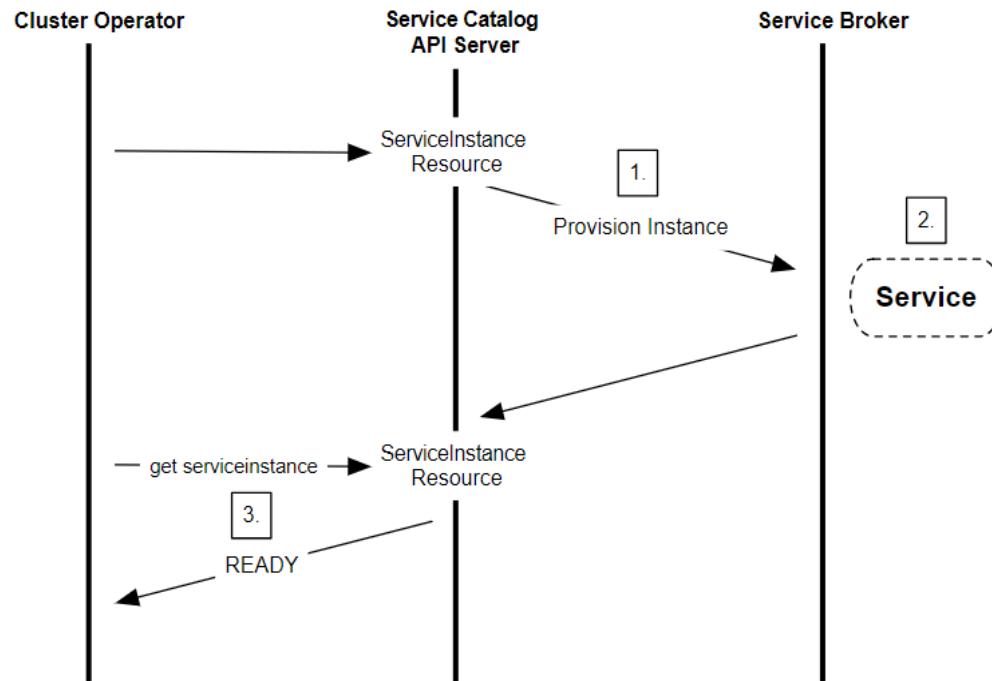


# Provisioning a new Instance





# Binding to a managed service



# Demo

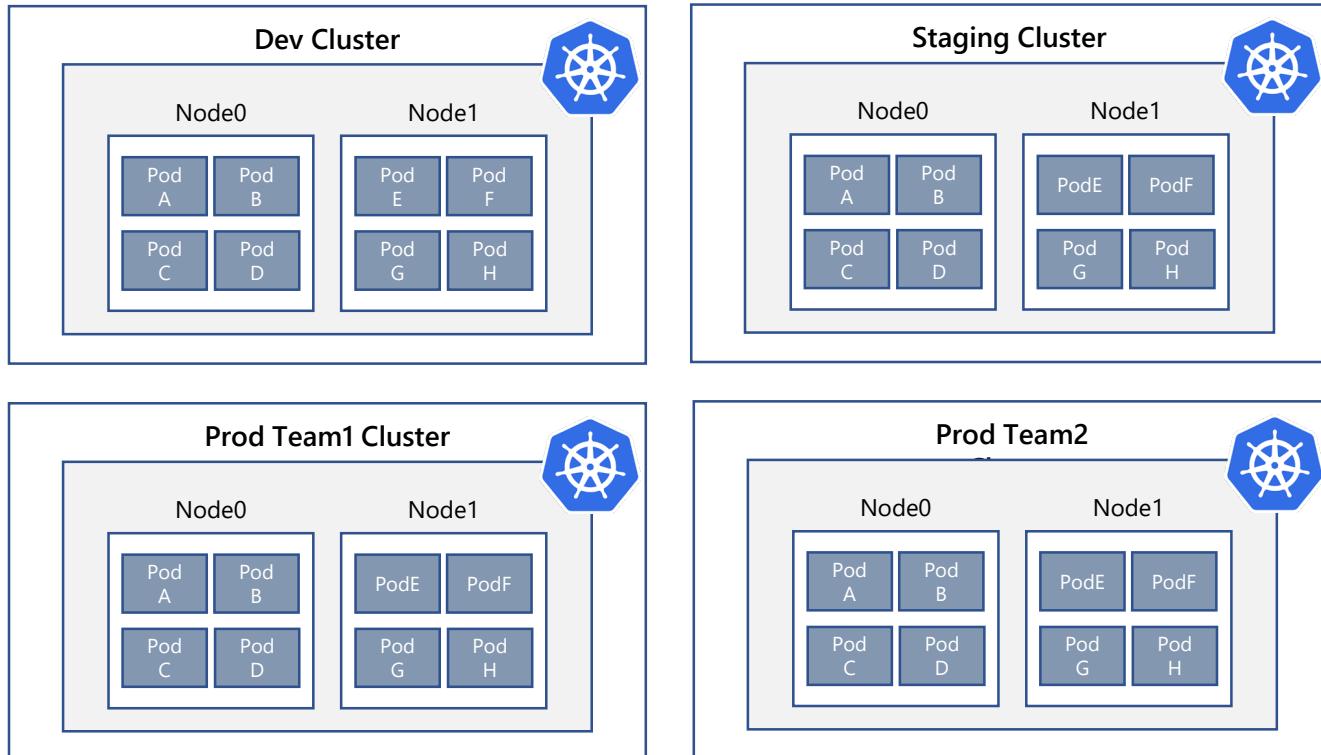
## Virtual Kubelet Service Catalog

Section #2

# Operational best practices for Azure Kubernetes Service

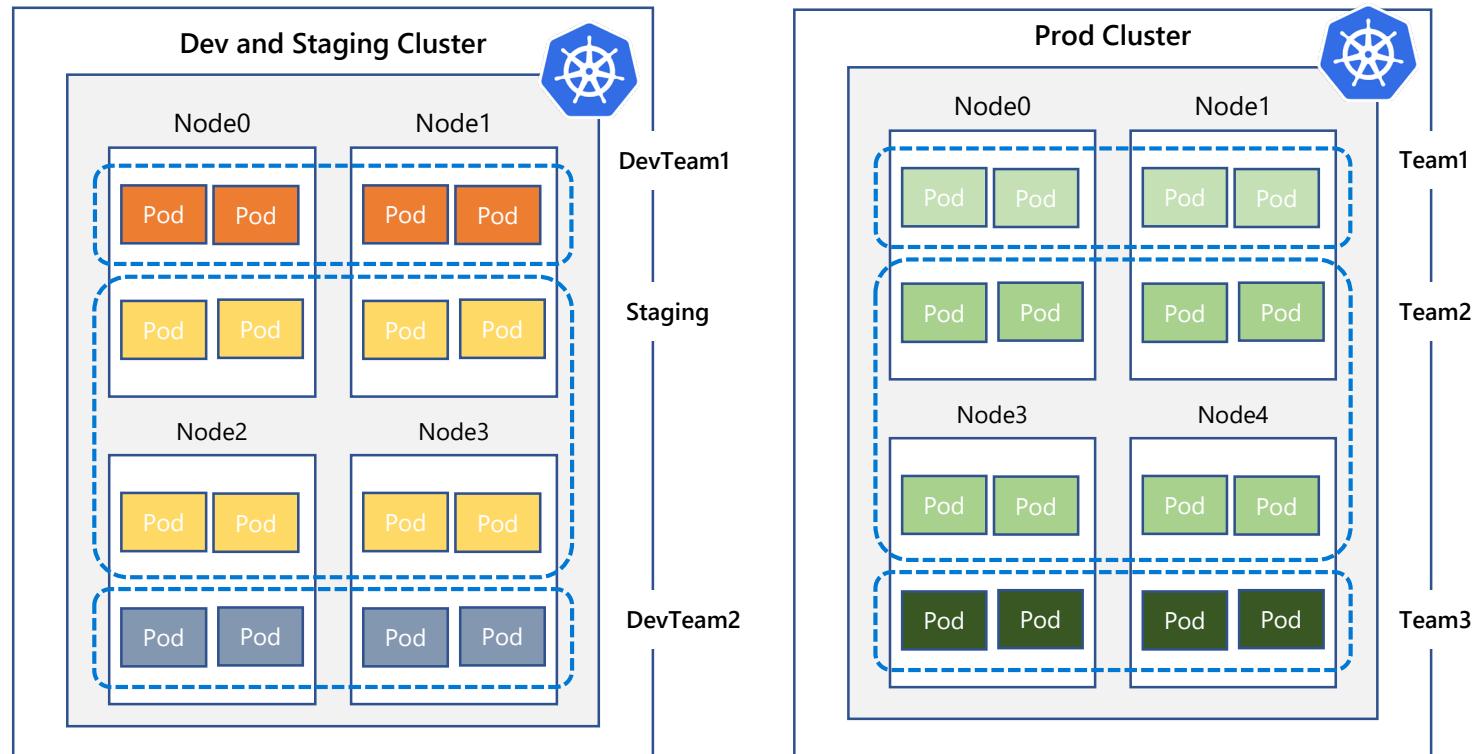
# Cluster Isolation and Resource Management

# Cluster Isolation Patterns: Physical Isolation





# Cluster Isolation Patterns: Logical Isolation





# Kubernetes Namespaces

- **Namespaces** Object is the logical Isolation boundary
- Kubernetes has features to help us safely isolate tenants
  - **Scheduling**: Resource Quota
  - **Network Isolation** using Network Policies
  - **Authentication and Authorization**: RBAC and Pod Security Policy
- Note: **Container Level isolation** still need to be done to achieve hard Isolation



# Kubernetes Resource Quotas

- Constraints that limit aggregate resource consumption per namespace
- You can limit Compute Resources (CPU, Memory, Storage, ...) and/or limit the number of Objects (Pods, Services, etc..) and
- When enabled, users must specify requests or limits, otherwise the quota system will fail the request.
- Kubernetes will not overcommit

**Create a namespace:**

```
$ kubectl create namespace ignite
```

**Apply a resource quota to the namespace:**

admin/resource/ignite.yaml

apiVersion: v1

kind: ResourceQuota

metadata:

name: mem-cpu-demo

spec:

hard:

requests.cpu: "1"

requests.memory: 1Gi

limits.cpu: "2"

limits.memory: 2Gi



# Physical Vs. Logical Isolation

	Physical	Logical
Pod Density	Low to Medium	Medium to High
Cost	\$\$	\$
Kubernetes Experience	Low to Medium	Medium to High
Security	High (Surface is small)	High*
Blast Radius of Changes	Small	Big
Management and Operations	Owner Team	Single or Cross Functional Team

\*Logical Isolation via Namespaces can achieve hard isolation assuming the cluster admin has applied all the required security controls



# Kube-Advisor

- Diagnostic tool for Kubernetes clusters. At the moment, it returns pods that are missing resource and request limits.
- More info can be found at <https://github.com/Azure/kube-advisor>

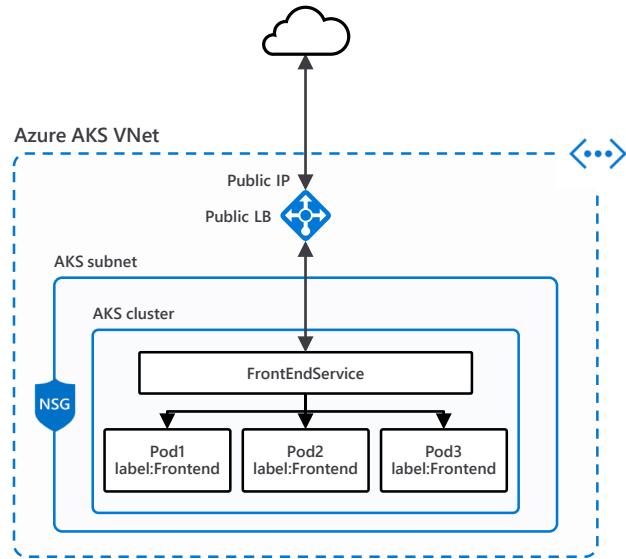
		CPU Request Limits Missing	Memory Request Limits Missing
		CPU Resource Limits Missing	Memory Resource Limits Missing
		CPU Request Limits Missing	Memory Request Limits Missing
ISSUE		REMEDIALION	
CPU Request Limits Missing		Consider setting resource and request limits to prevent resource starvation: <a href="https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/">https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/</a>	
Memory Request Limits Missing			
CPU Resource Limits Missing			
Memory Resource Limits Missing			



# Public Service

- Service Type LoadBalancer
- Basic Layer4 Load Balancing (TCP/UDP)
- Each service as assigned an IP on the ALB

```
apiVersion: v1
kind: Service
metadata:
  name: frontendservice
spec:
  loadBalancerIP: X.X.X.X
  type: LoadBalancer
  ports:
    - port: 80
  selector:
    app: frontend
```

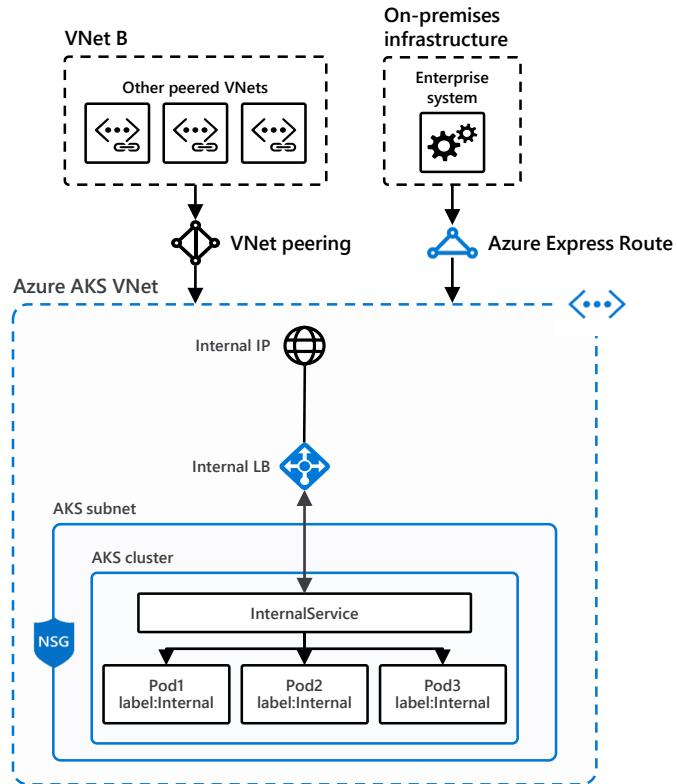




# Internal Service

- Used for internal services that should be accessed by other VNets or On-Premise only

```
apiVersion: v1
kind: Service
metadata:
  name: internalservice
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal:
    "true"
spec:
  type: LoadBalancer
  loadBalancerIP: 10.240.0.25
  ports:
  - port: 80
  selector:
    app: internal
```





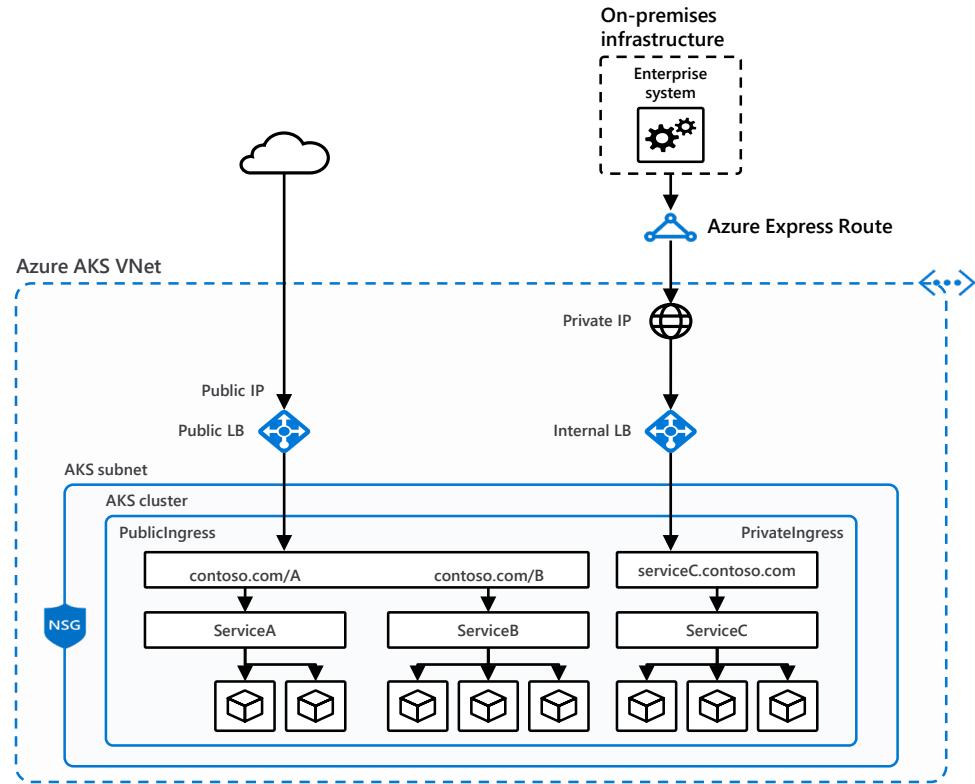
# Ingress And Ingress Controllers

- **Ingress** is a Kubernetes API that manages external access to the services in the cluster
  - Supports HTTP and HTTPS
  - Path and Subdomain based routing
  - SSL Termination
  - Serve on public IPs
- **Ingress controller** is a daemon, deployed as a Kubernetes Pod, that watches the Ingress Endpoint for updates. Its job is to satisfy requests for ingresses. Most popular one being **Nginx**.

# Ingress

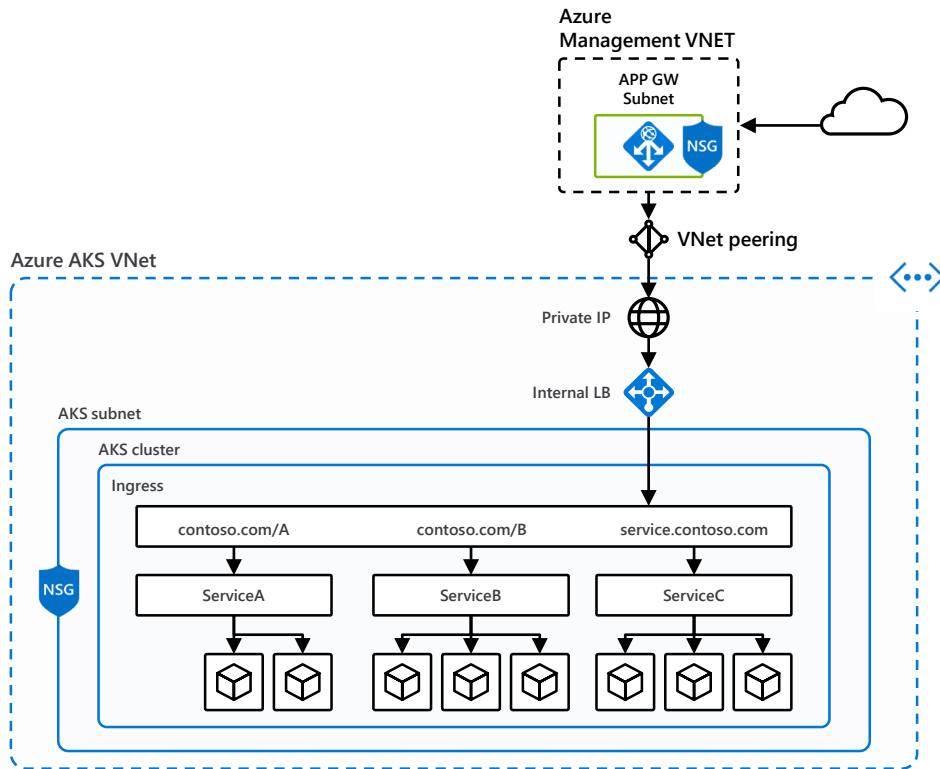


```
kind: Ingress
metadata:
  name: contoso-ingress
  annotations: kubernetes.io/ingress.class: "PublicIngress"
spec:
  tls:
    - hosts:
        - contoso.com
      secretName: contoso-secret
  rules:
    - host: contoso.com
      http:
        paths:
          - path: /a
            backend:
              serviceName: servicea
              servicePort: 80
          - path: /b
            backend:
              serviceName: serviceb
              servicePort: 80
```



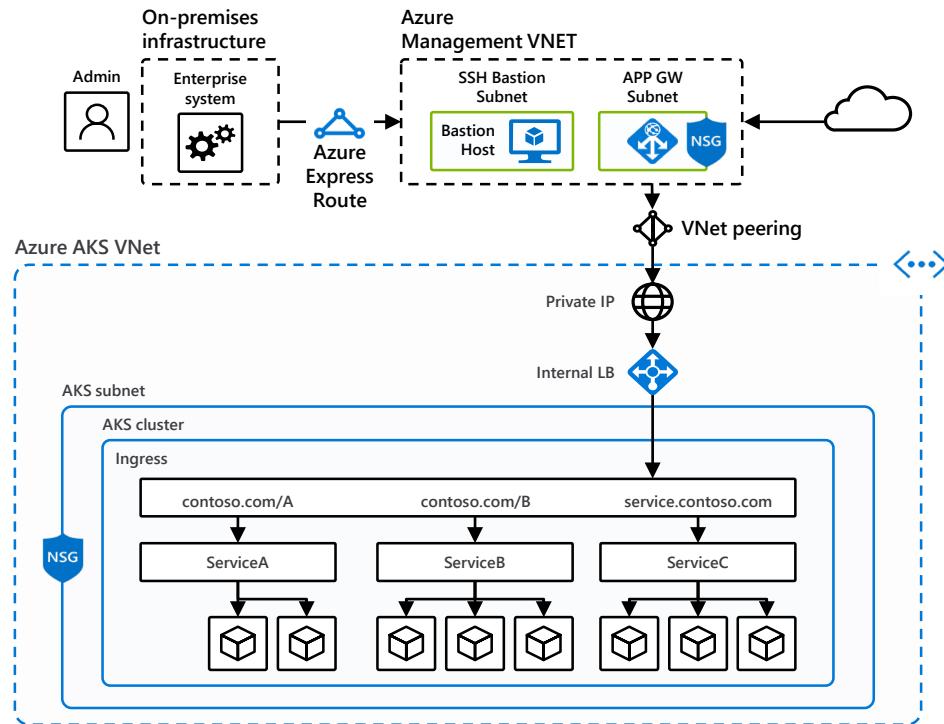


# Securing Kubernetes Services With WAF

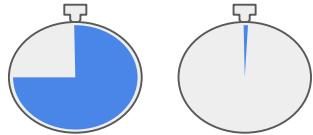




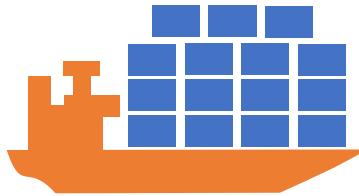
# Cluster Management Through Bastion Host



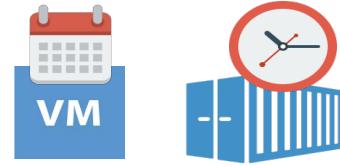
# Implications Of Dynamically Orchestrated Containers



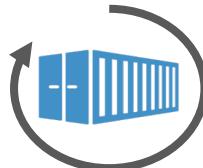
**900x**  
start time



**15x**  
workloads



**25x shorter**  
lifetime



**375x**  
network churn



**15x**  
attack surface area

# Operational Best Practices: Kubernetes Network Security



Define micro-service connectivity (e.g. as part of CI/CD workflow)  
Baseline traffic patterns

Role based access controls  
GitOps / policy-as-code

Enforce East-West Policies Within Cluster

Enterprise Controls

Zero Trust Security Model

Continuous Compliance

Multi-factor authentication (mTLS cert + network identity)

Encryption of data-in-transit

Visibility, monitoring, & alerting

Reporting

# Cluster Level Security



# Cluster Level Security

- Securing endpoints for API server and cluster nodes
  - Ensuring authentication and authorization (AAD + RBAC)
  - Setting up & keeping least privileged access for common tasks



# Cluster Level Security

- Securing endpoints for API server and cluster nodes
  - Ensuring authentication and authorization (AAD + RBAC)
  - Setting up & keeping least privileged access for common tasks
  - Admission Controllers
    - DenyEscalatingExec
    - ValidatingAdmissionWebhooks
    - MutatingAdmissionWebhooks
    - ServiceAccount
    - Coming soon:
      - NodeRestriction
      - PodSecurityPolicy

# Cluster Level – Nodes, Upgrade and Patches



- Regular maintenance, security and cleanup tasks
  - Maintain, update and upgrade hosts and kubernetes
  - Monthly ideal, 3 months minimum
  - Security patches
    - AKS automatically applies security patches to the nodes on a nightly schedule
    - You're responsible to reboot as required
    - Kured DaemonSet  
<https://github.com/weaveworks/kured>

## Upgrade to version 1.10.6

```
$ az aks upgrade --name myAKSCluster \  
--resource-group myResourceGroup \  
--kubernetes-version 1.10.6
```

- **SSH Access**
  - DenyEscalatingExec
- **Running benchmarks and tests to validate cluster setup**
  - Kube-bench
  - Aqua Hunter
  - Others

# Container Level Security and Isolation



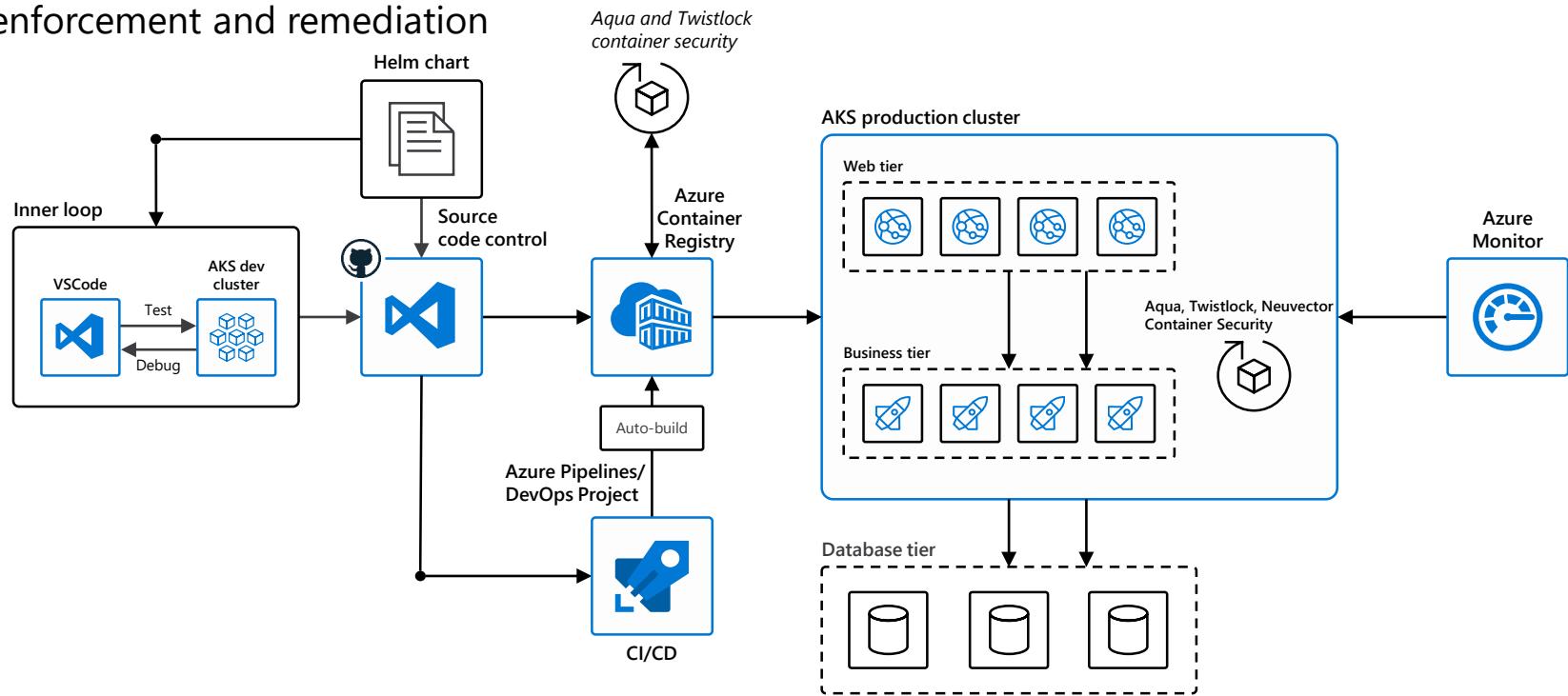
# Container Level – The images

- Trusted Registry
- Regularly apply security updates to the container images



# Container Level – Images and Runtime

- Scan your images, scan your containers
- Runtime enforcement and remediation





# Container Level – The access

- Avoid access to HOST IPC namespace - only if absolutely necessary
- Avoid access to Host PID namespace - only if absolutely necessary
- Avoid root / privileged access
  - Consider Linux Capabilities

# Container Level – App Armor



## Securing a Pod with a deny-write.profile

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-apparmor
  annotations:
    kubernetes.io/appArmor.securityLevel: beta.kubernetes.io
    /hello: localhost/k8s-apparmor-example-deny-
    write
  command: [ "sh", "-c", "echo 'Hello
AppArmor!' && sleep 1h" ]
spec:
  containers:
  - name: hello
    image: busybox
    command: [ "sh", "-c", "echo 'Hello
AppArmor!' && sleep 1h" ]
```

## deny-write.profile

```
#include <tunables/global>

profile k8s-apparmor-example-deny-
write flags=(attach_disconnected) {

  #include <abstractions/base>
  terminated with non-zero exit
  file,

  # Deny all file writes.
  deny /** w,
}
```

# Pod Level Security



# Pod Level – Pod Security Context

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: ignite.azurecr.io/nginx-demo
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
    seLinuxOptions:
      level: "s0:c123,c456"
```



# Pod level

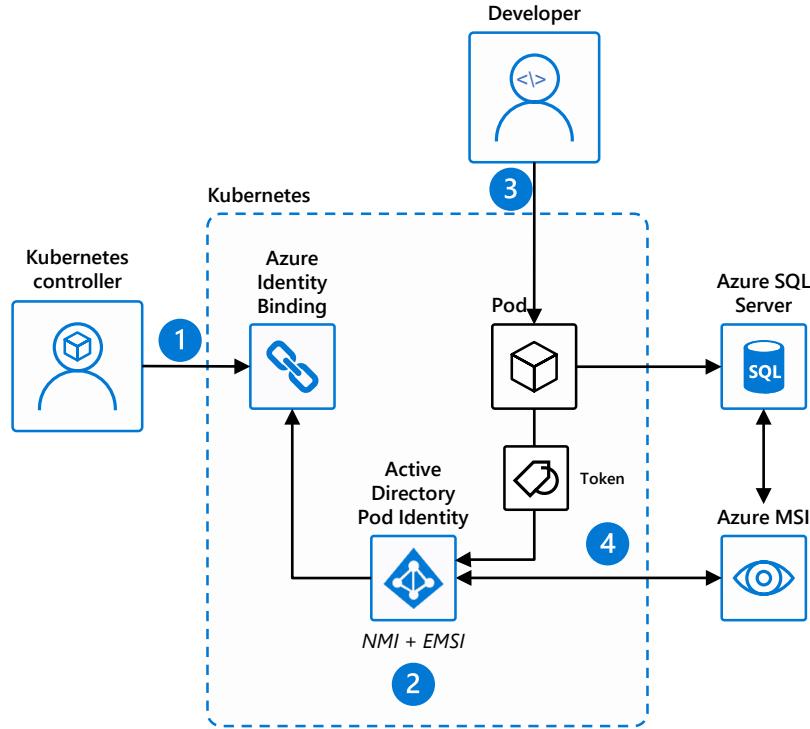
- Pod Security Context
- Pod Security Policies
- AlwaysPull Images

# Securing Workloads



# Pod Identity

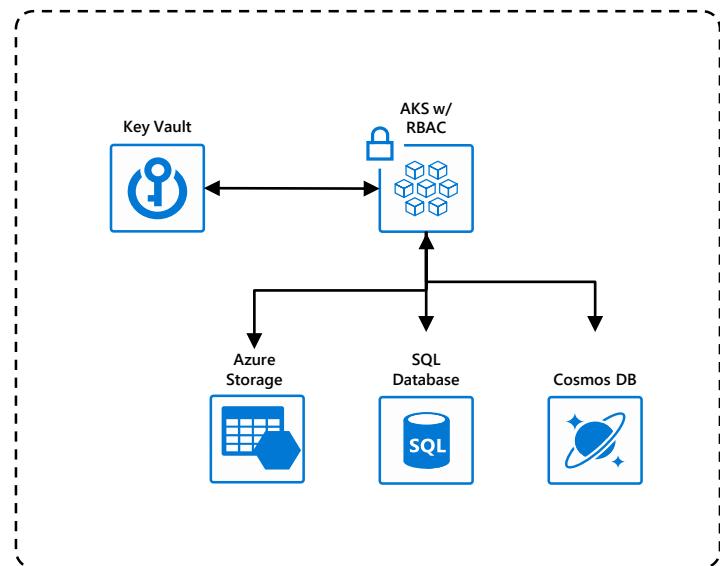
1. Kubernetes operator defines an identity map for K8s service accounts
2. Node Managed Identity (NMI) watches for mapping reaction and syncs to Managed Service Identity (MSI)
3. Developer creates a pod with a service account. Pod uses standard Azure SDK to fetch a token bound to MSI
4. Pod uses access token to consume other Azure services; services validate token





# Securing workloads

- Managing secrets and privileged information
  - Azure Key Vault





# Securing workloads

- Service Endpoints
- Filter secrets from the logs
- Encrypted Service to Service Communication
  - mTLS between services
  - Service Meshes



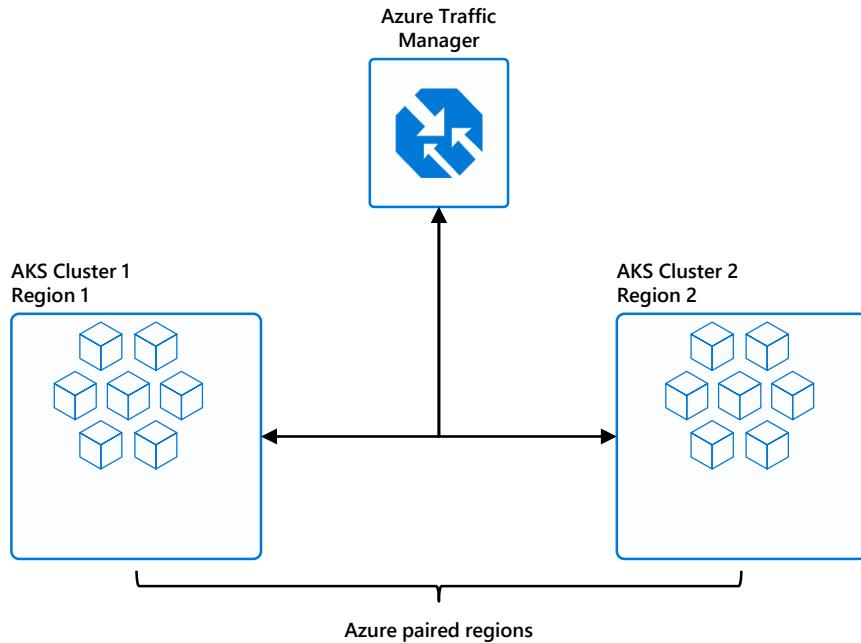
# Manual scaling is Tedious and Ineffective

- Horizontal pod autoscaling(HPA) -> Scaling pods/containers
- Cluster Autoscaling -> Scaling infrastructure/VM's
- AKS + ACI + VK for burst scenarios -> Scaling pods/containers



# Multi-Region Clusters

- Minimize downtime risk
- One live region
  - Another backup
  - Or weighted traffic
- A/B testing



# Logging and Monitoring



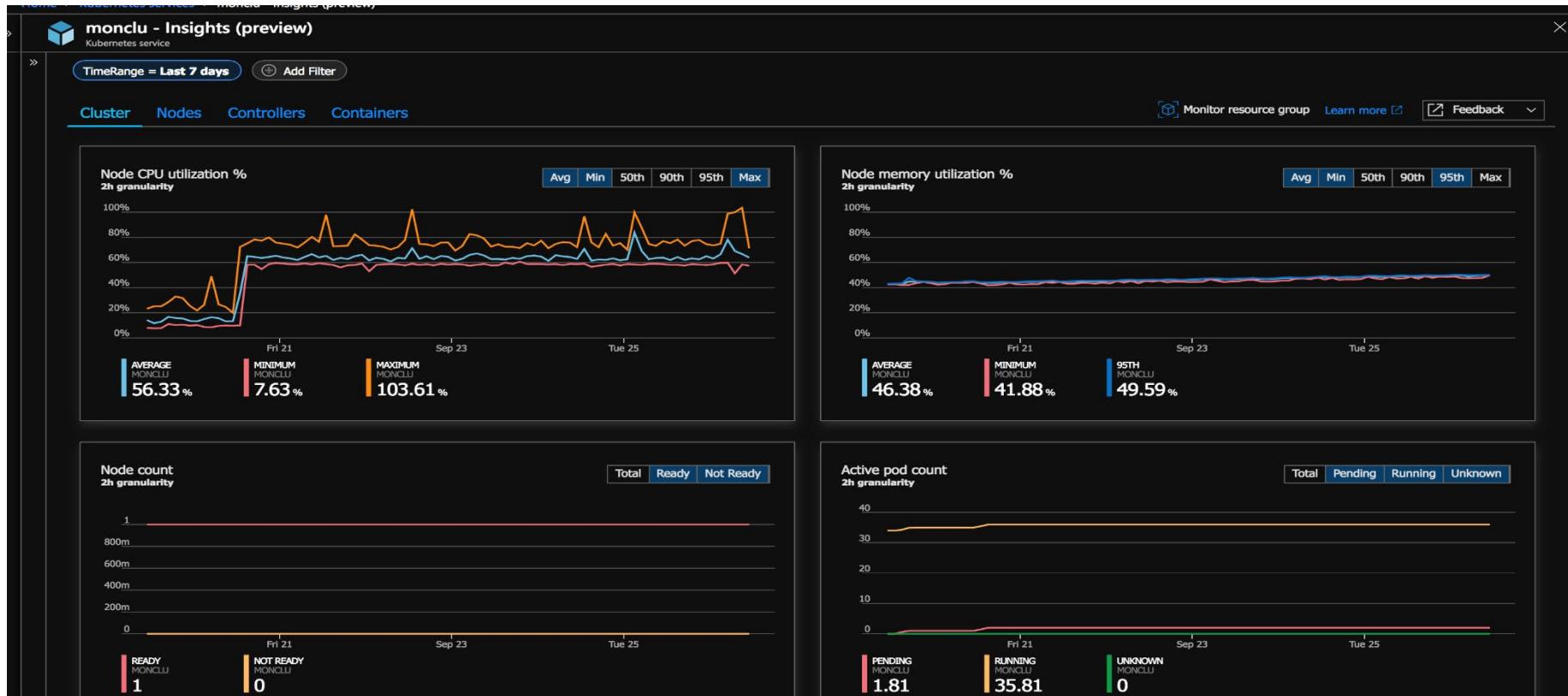
# Monitoring/Logging your cluster

- Log Everything to stdout / stderr
- Key Metrics:
  - Node metrics (CPU Usage, Memory Usage, Disk Usage, Network Usage)
  - Kube\_node\_status\_condition
  - Pod memory usage / limit; memory\_failures\_total
    - container\_memory\_working\_set\_bytes
  - Pod CPU usage average / limit
  - Filesystem Usage / limit
  - Network receive / transmit errors
- Azure Monitor for Containers

In the roadmap



# Overview Health of AKS Cluster





# Customer Control Plane Logs

- Use the Azure portal to enable diagnostics logs
- Pipe logs to log analytics, event hub or a storage account
- Metrics available today
  - Kube-controller-manager
  - Kube-api-server
  - Kube-scheduler
  - Audit logs on the roadmap

Diagnostics settings

Save Discard Delete

\* Name

Archive to a storage account

Stream to an event hub

Send to Log Analytics

**LOG**

kube-apiserver

kube-controller-manager

kube-scheduler

guard

**METRIC**

AllMetrics



# Example Control Plane Logs

monclu - Logs  
Kubernetes service

New Query 1 \* +

defaultworkspace-5abfd9c4-ec8c-4... | Run Time range: Last 24 hours

AzureDiagnostics  
| where Category == "kube-controller-manager"  
| where log\_s contains "my-nginx"  
| project log\_s

Completed. Showing results from the last 24 hours. 00:00:01.117 11 records

TABLE CHART Columns

Drag a column header and drop it here to group by that column

log_s
I0919 03:26:57.353133 1 event.go:221] Event{v1.ObjectReference{Kind:"Deployment", Namespace:"default", Name:"my-nginx", UID:"dcd7d703-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161662", FieldPath:""})...
I0919 03:26:57.418072 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.451023 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.476133 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.505592 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.505841 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.507912 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.508195 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.575581 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.576071 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...
I0919 03:26:57.577221 1 event.go:221] Event{v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"my-nginx-59497d7745", UID:"dcdb4095-bbbb-11e8-a78d-06cd4d8a83f2", APIVersion:"apps/v1", ResourceVersion:"161663", Fi...

Query\_data (1).csv | Page 1 of 1 50 Items per page 1 - 11 of 11 items