Kubernetes on Azure

Vishwas Lele @vlele

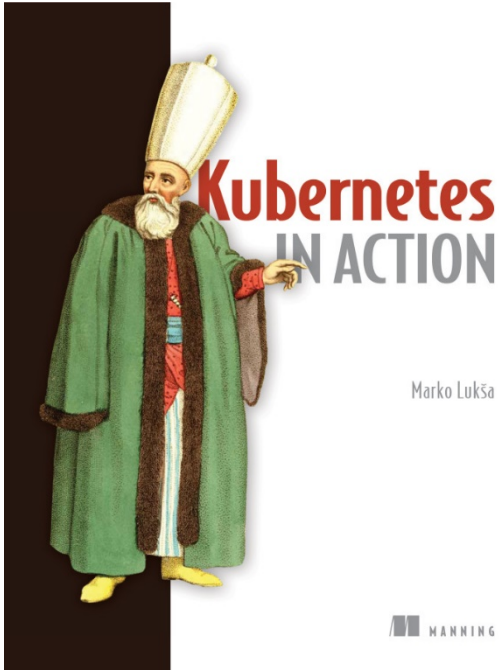
Source Code

- https://github.com/vlele/k8onazure

Attribution

- https://github.com/cncf/presentations/tree/master/kubernetes
- @mrbobbytables
- @jeefy

Book Recommendation

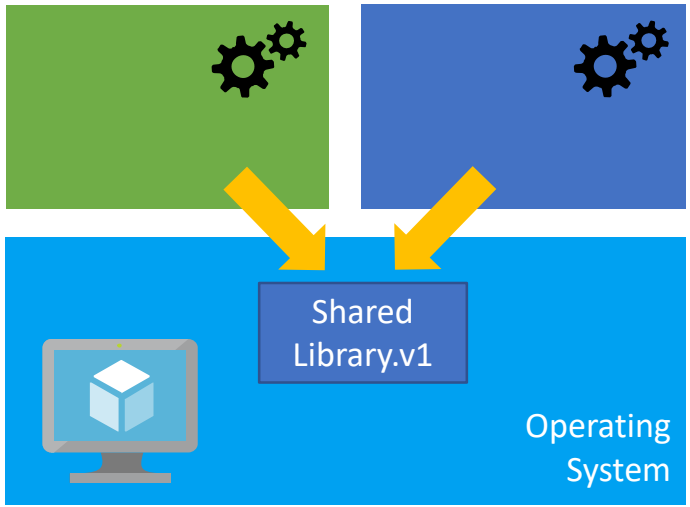


Containers and Orchestrators

Section #1

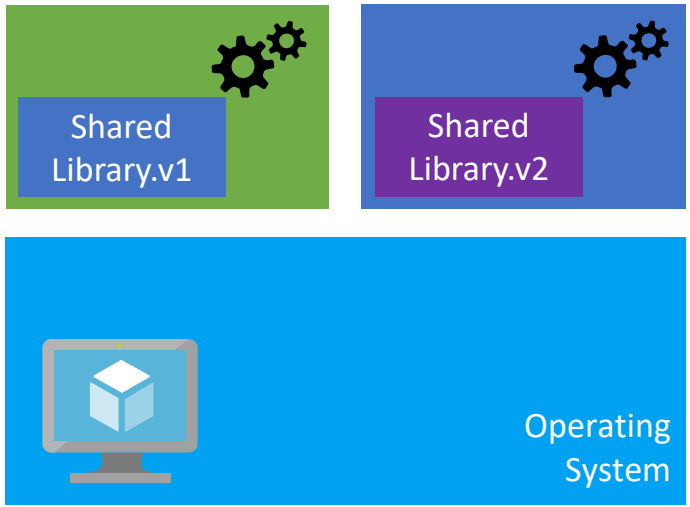
What is a container?

- Application packaging and deployment mechanism



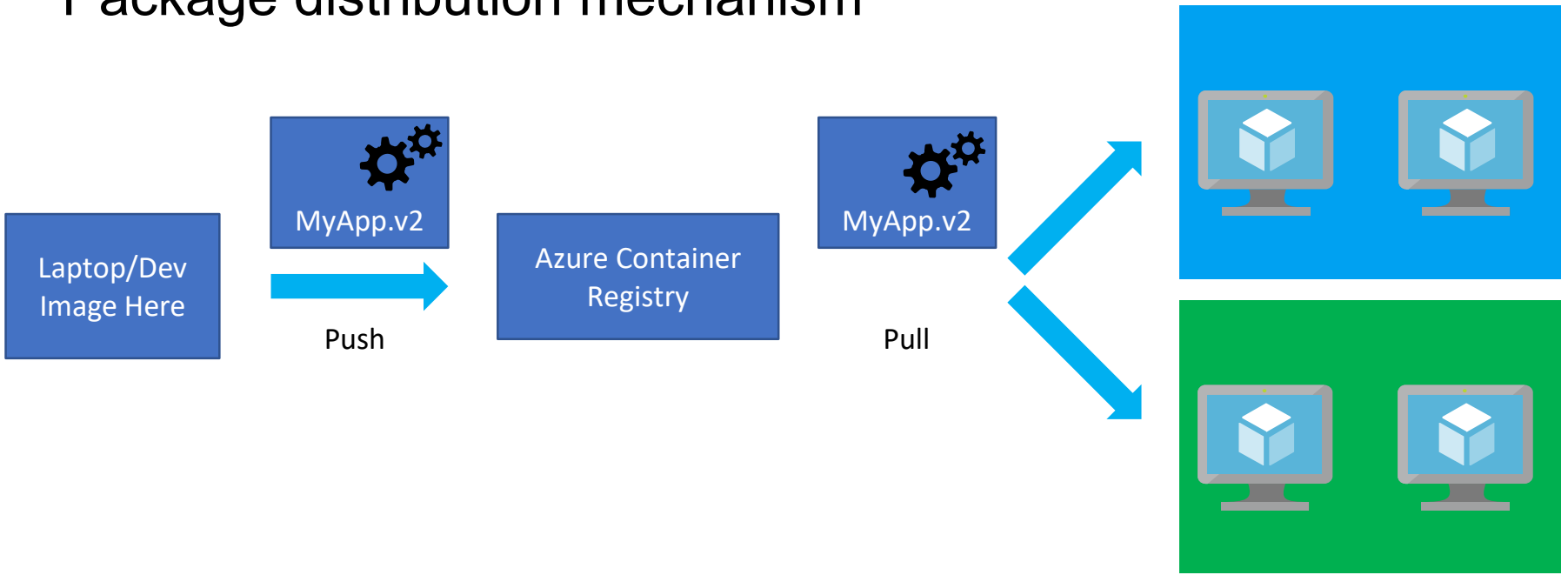
What is a container?

- Application packaging and deployment mechanism



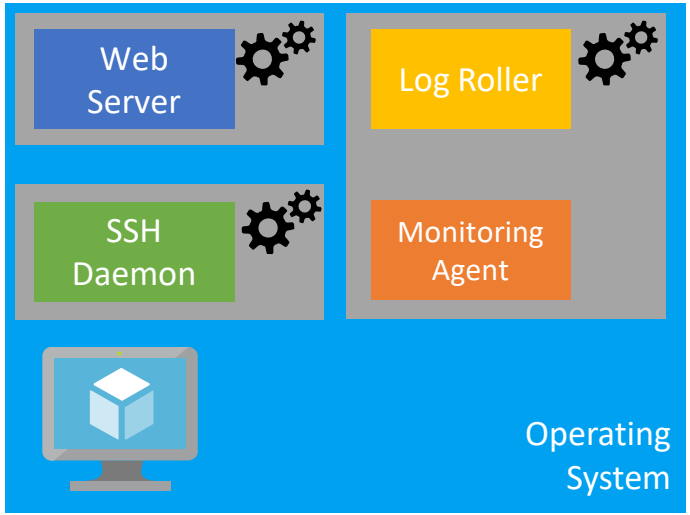
What is a container?

- Package distribution mechanism



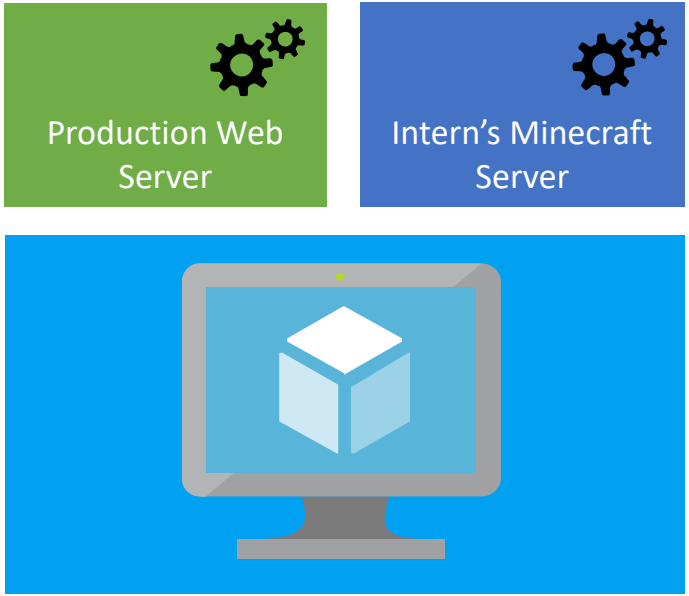
What is a container?

- A clear boundary for your stuff



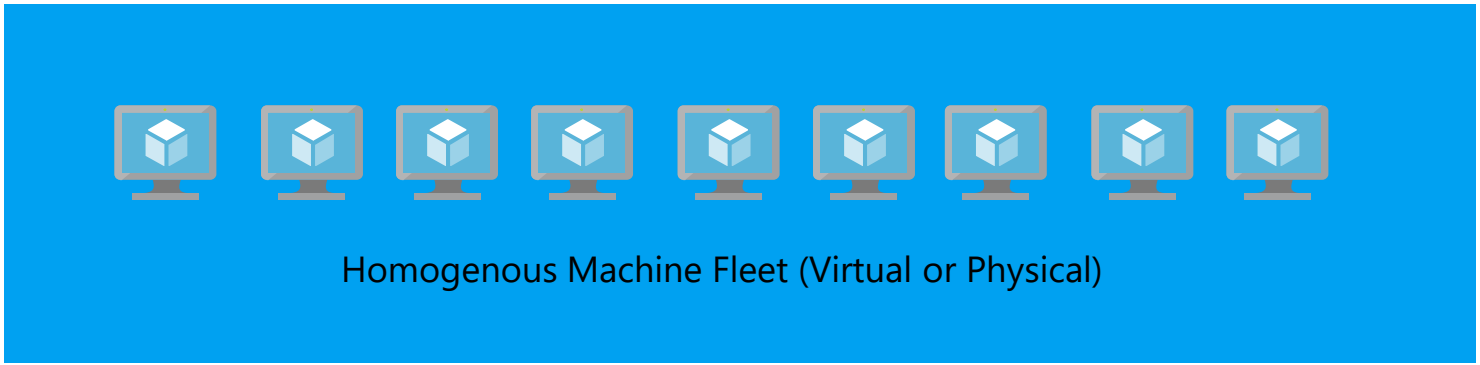
What is a container?

- Runtime isolation



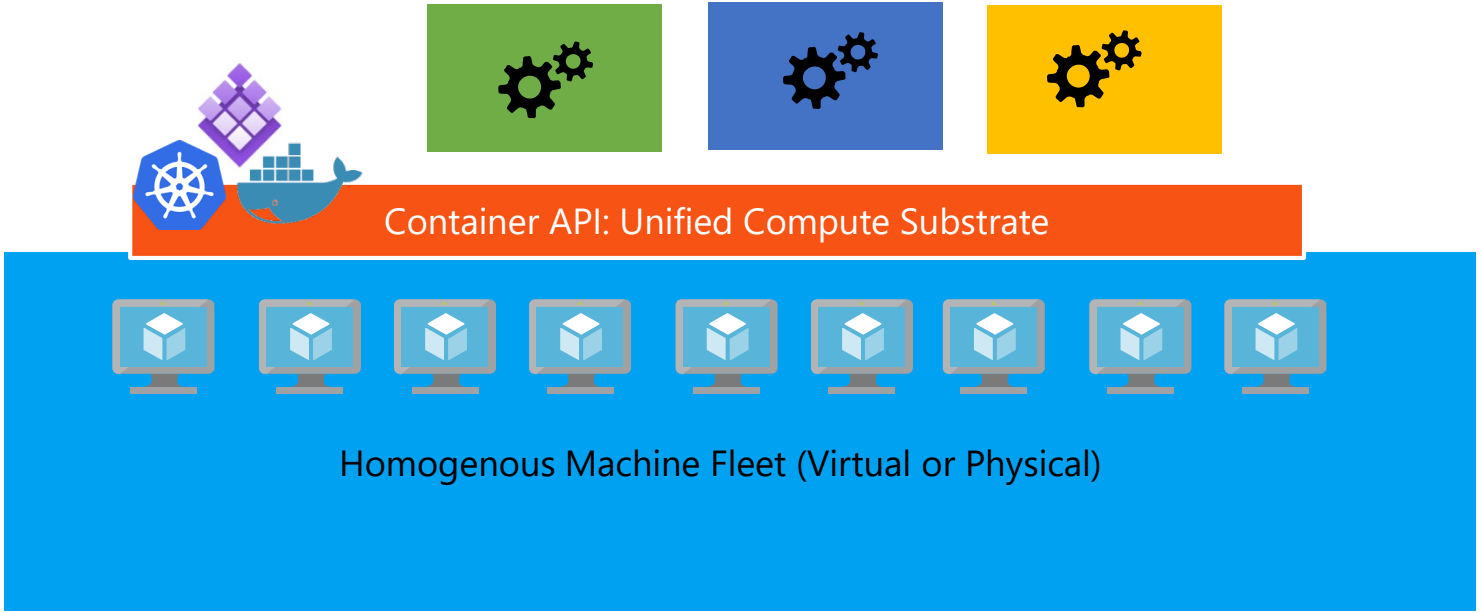
What is a container orchestrator?

- Manage a bunch of machines



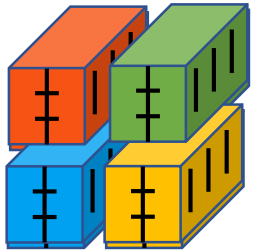
What is a container orchestrator?

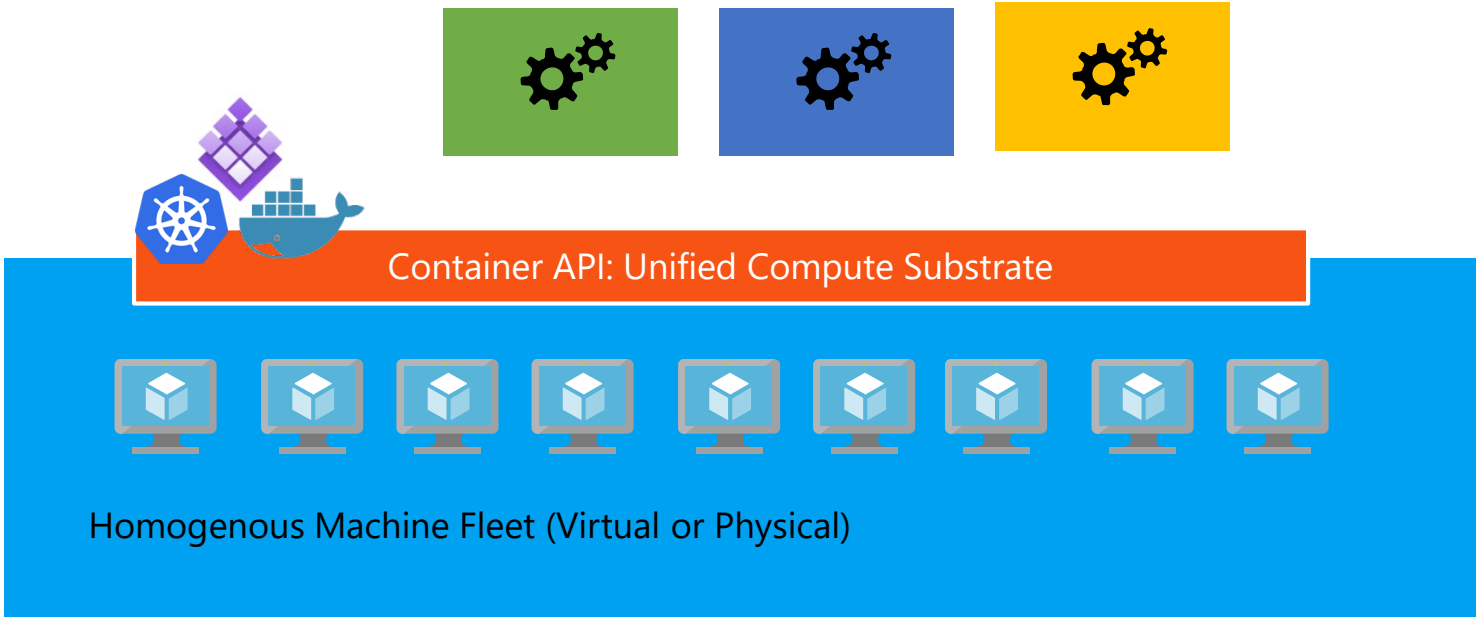
- Decouple machines from applications



What is a container service?

- Make all of this stuff easier





Demo

Container
Registry
Orchestrator

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”

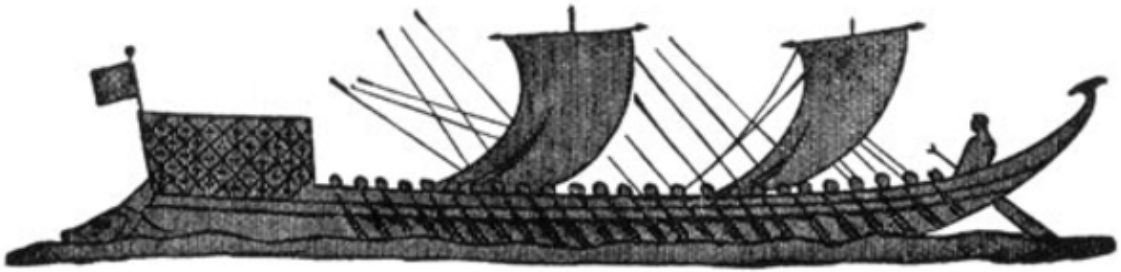
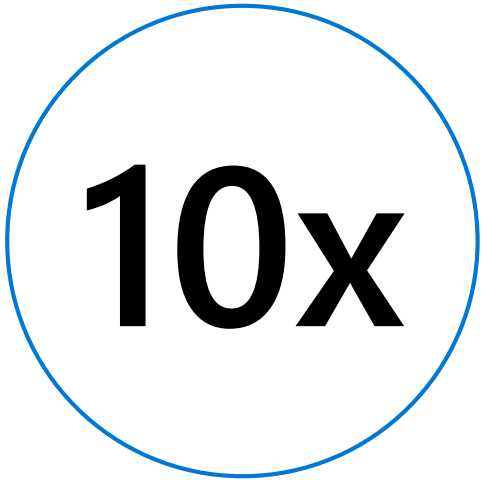


Image Source

Who “Manages” Kubernetes?

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

Azure Kubernetes momentum



Azure Kubernetes Service
usage grew 10x since it was made
generally available in June 2018

Trusted by thousands of customers

J.B. HUNT · Hafslund · MAERSK · AMBITENERGY
webjet.com.au · WhiteSource · falkonry · Aporeto
Altair · SOTETSU · xerox · SIEMENS Healthineers
varian · equinor · Tryg · BOSCH

Dated November 2018

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

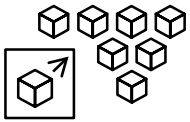
- Significant Operations Overhead
  - No way to handle dozens of services without automation
- Distributed Systems Complexity
  - Performance, latency, security, network topology
- Technology Diversity
  - Microservices can be written different languages, use different libraries
- Testability
  - Difficult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

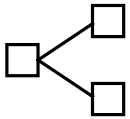
Lift and shift to containers



Cost saving

without refactoring your app

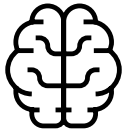
Microservices



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

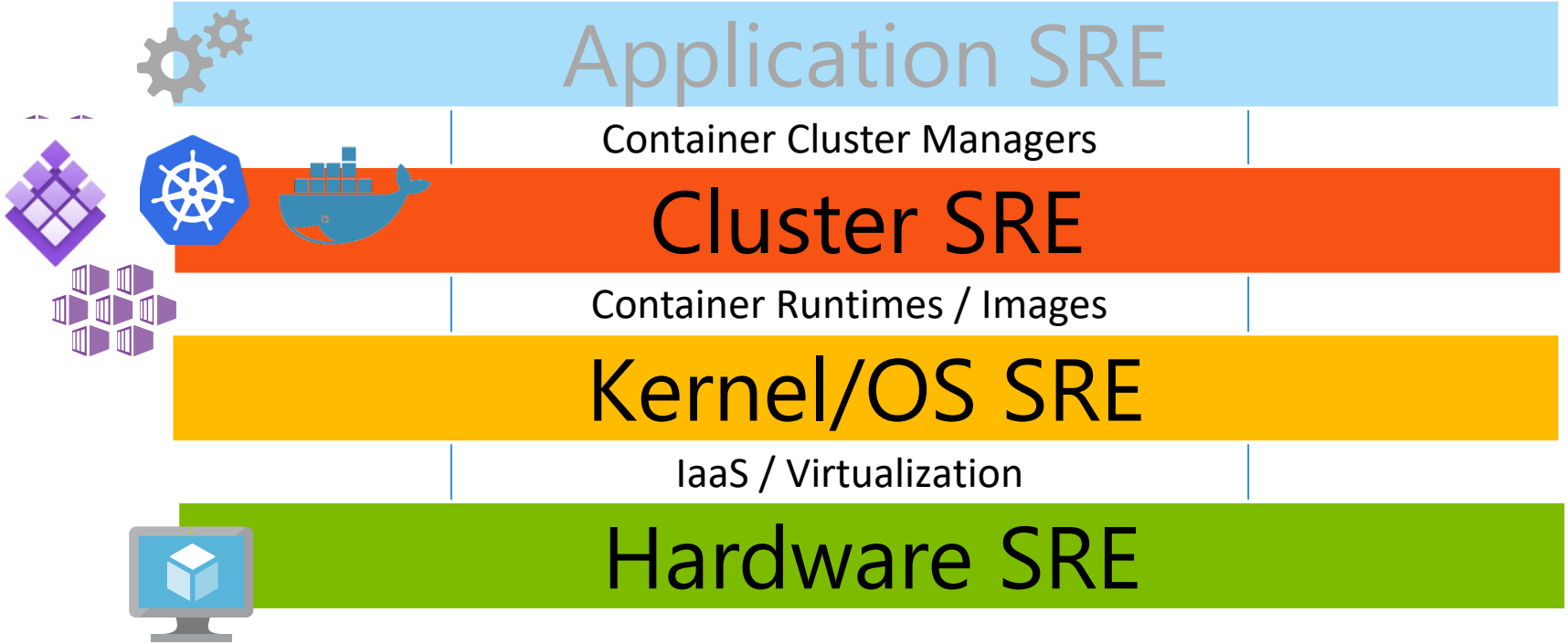
Container Runtimes / Images

Kernel/OS SRE

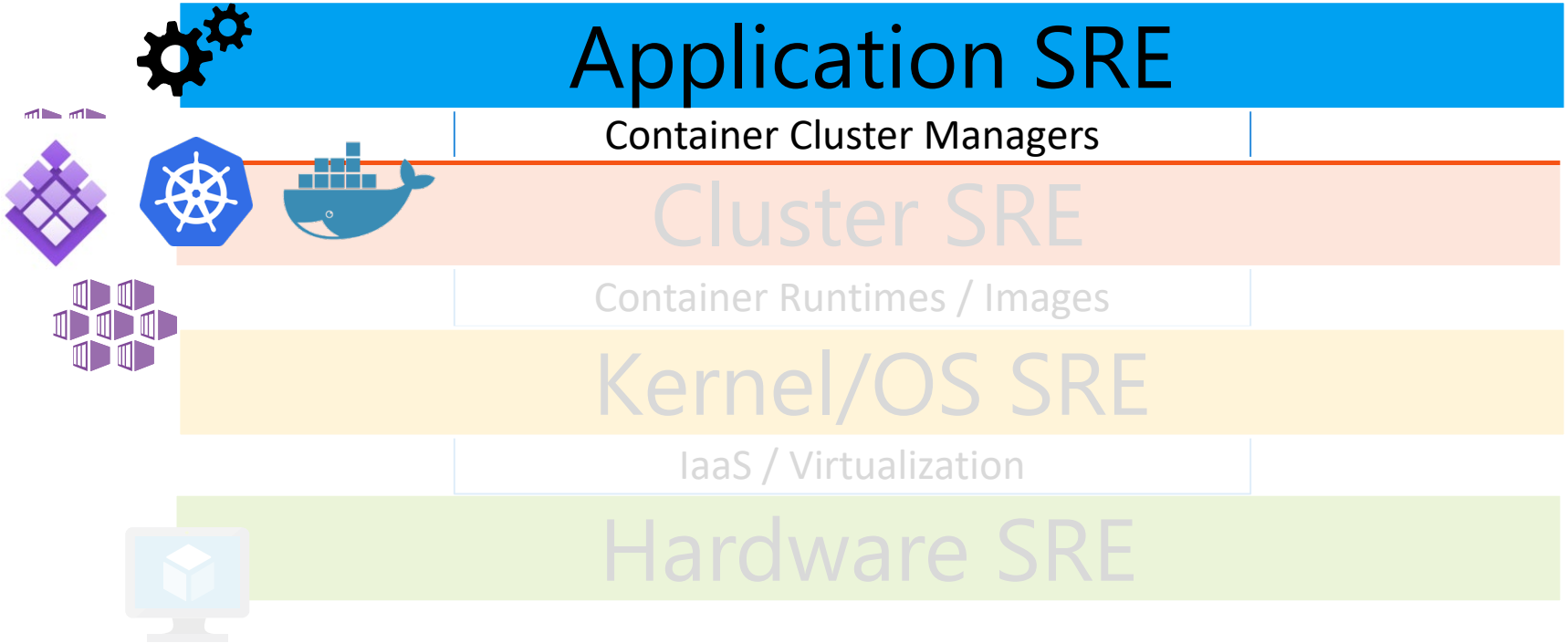
IaaS / Virtualization

Hardware SRE

Decoupling


Application SRE
Container Cluster Managers
Cluster SRE
Container Runtimes / Images
Kernel/OS SRE
IaaS / Virtualization
Hardware SRE

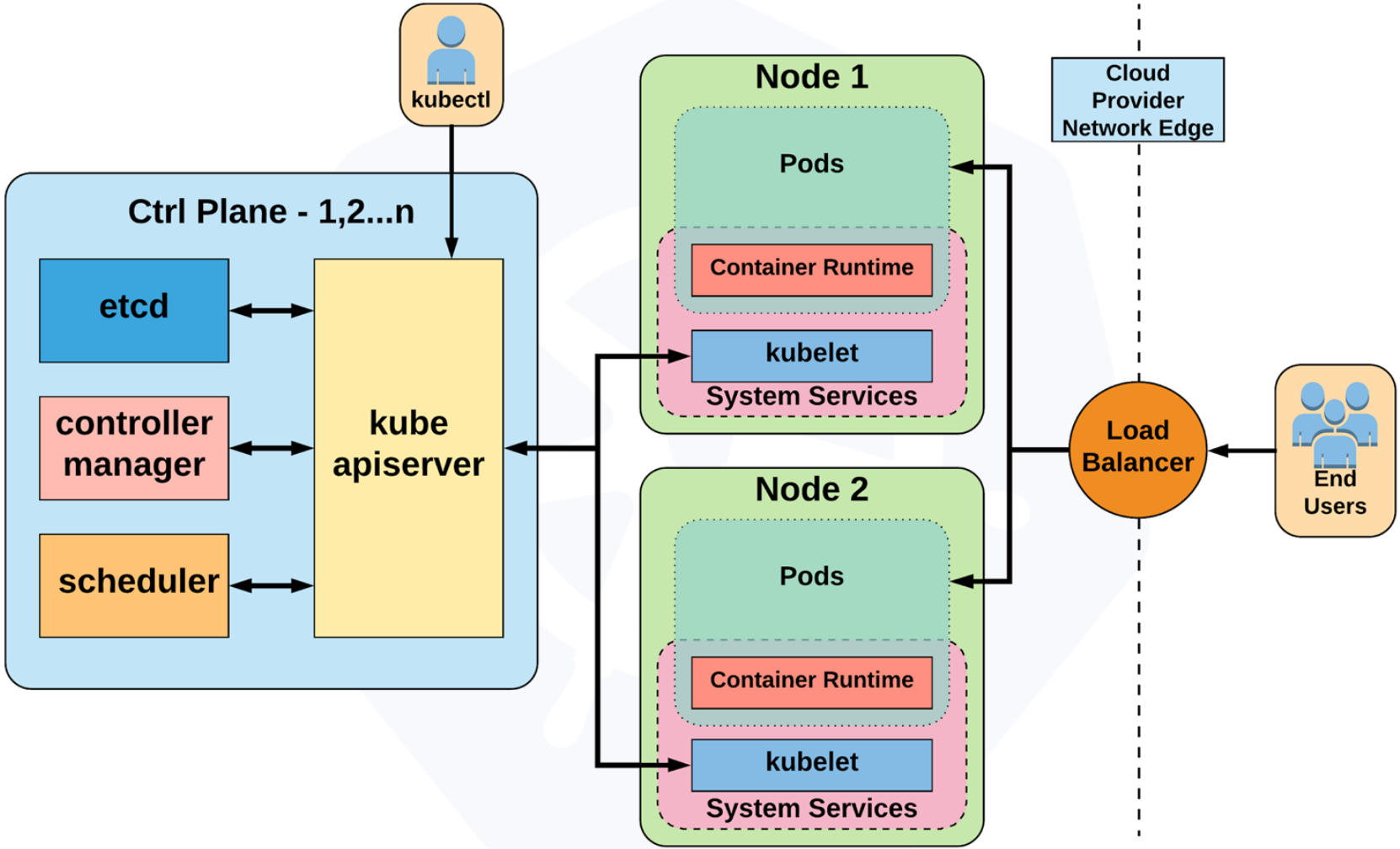
Decoupling


Application SRE
Container Cluster Managers
Cluster SRE
Container Runtimes / Images
Kernel/OS SRE
IaaS / Virtualization
Hardware SRE

Key Concepts

Section #2

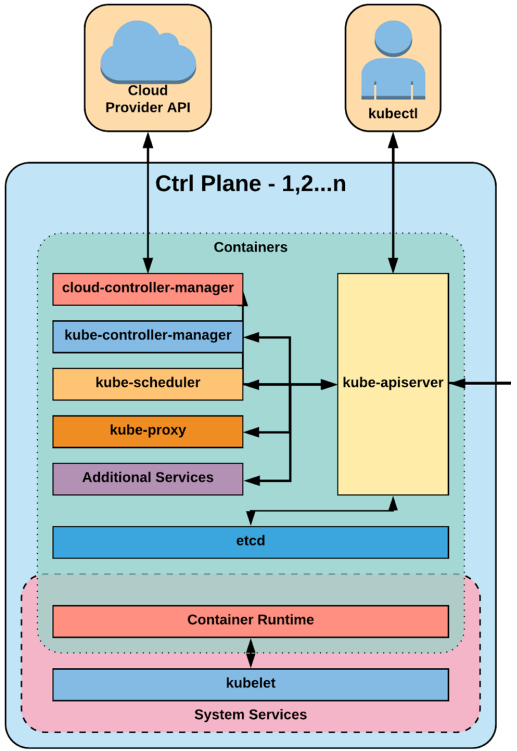
Architecture Overview


kubectl
Ctrl Plane - 1,2...n
etcd
controller manager
scheduler
kube apiserver
Node 1
Pods
Container Runtime
kubelet
System Services
Node 2
Pods
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users

Control Plane Components

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

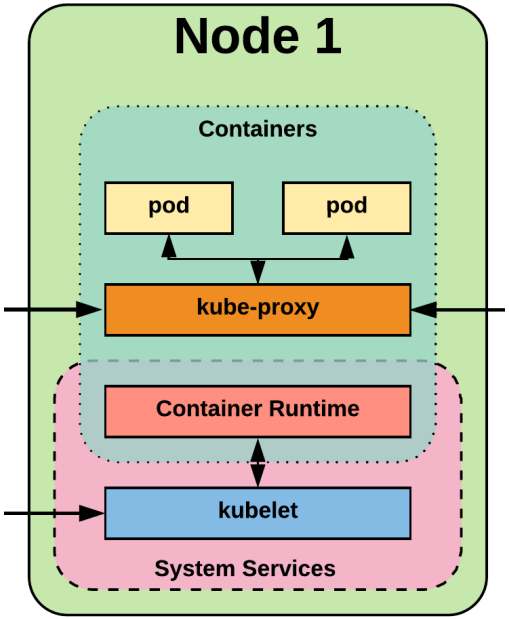
kube-scheduler

- 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

Node Components

- kubelet
- kube-proxy
- Container Runtime Engine



kubelet

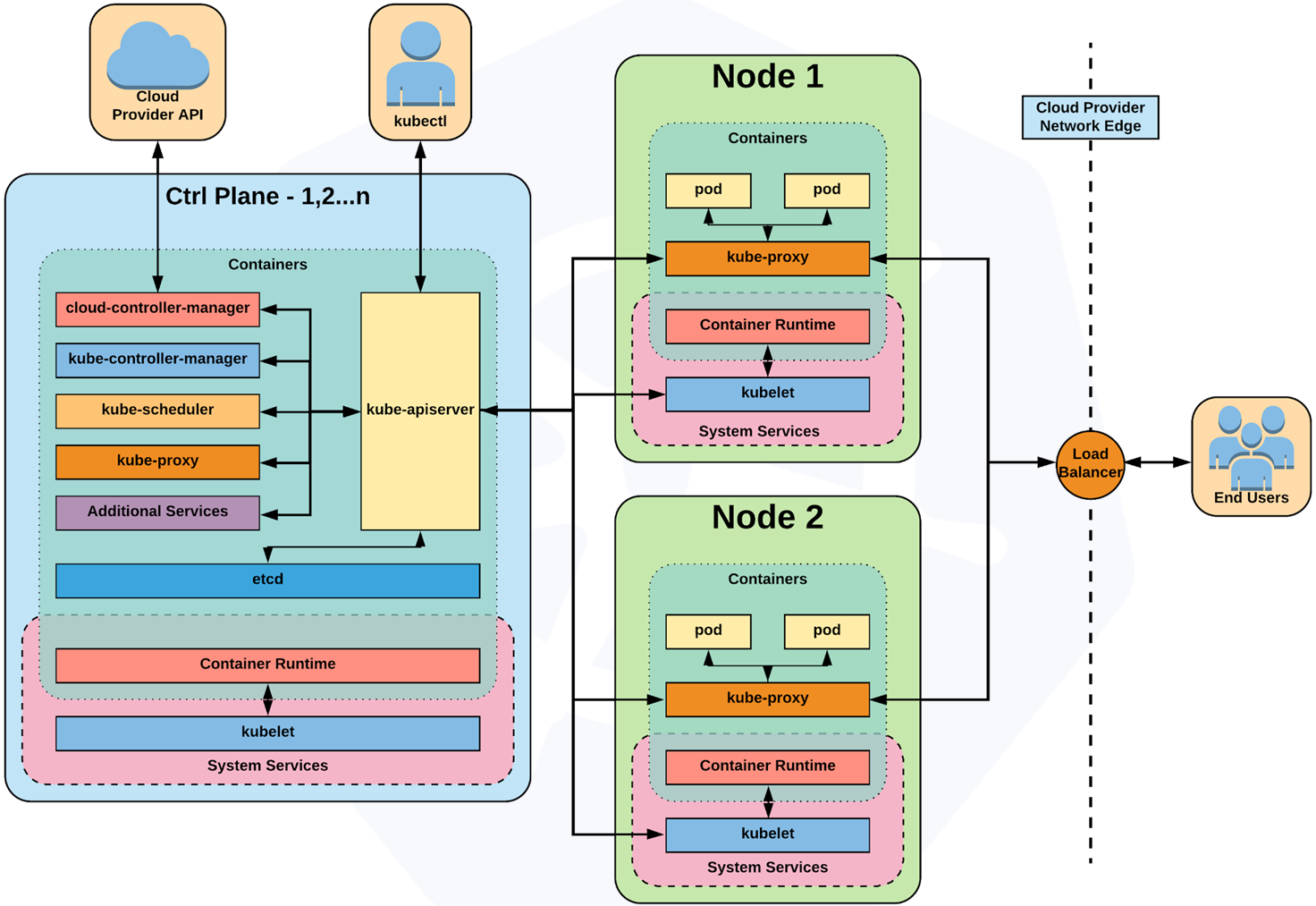
- Acts as the node agent responsible for managing the lifecycle of every pod on its host.
- Kubelet understands YAML container manifests that it can read from several sources:
  - file path
  - HTTP Endpoint
  - etcd watch acting on any changes
  - HTTP Server mode accepting container manifests over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

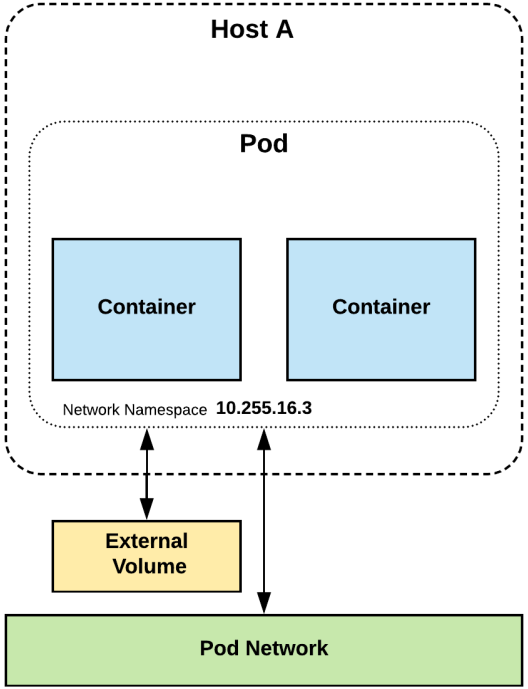

Cloud Provider API
kubectl
Ctrl Plane - 1,2...n
Containers
cloud-controller-manager
kube-controller-manager
kube-scheduler
kube-proxy
Additional Services
kube-apiserver
etcd
Container Runtime
kubelet
System Services
Node 1
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Node 2
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users

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

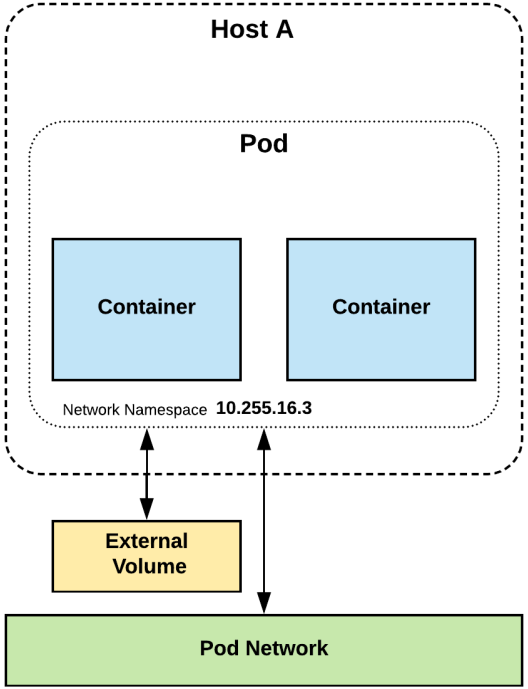
Pods

- **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!

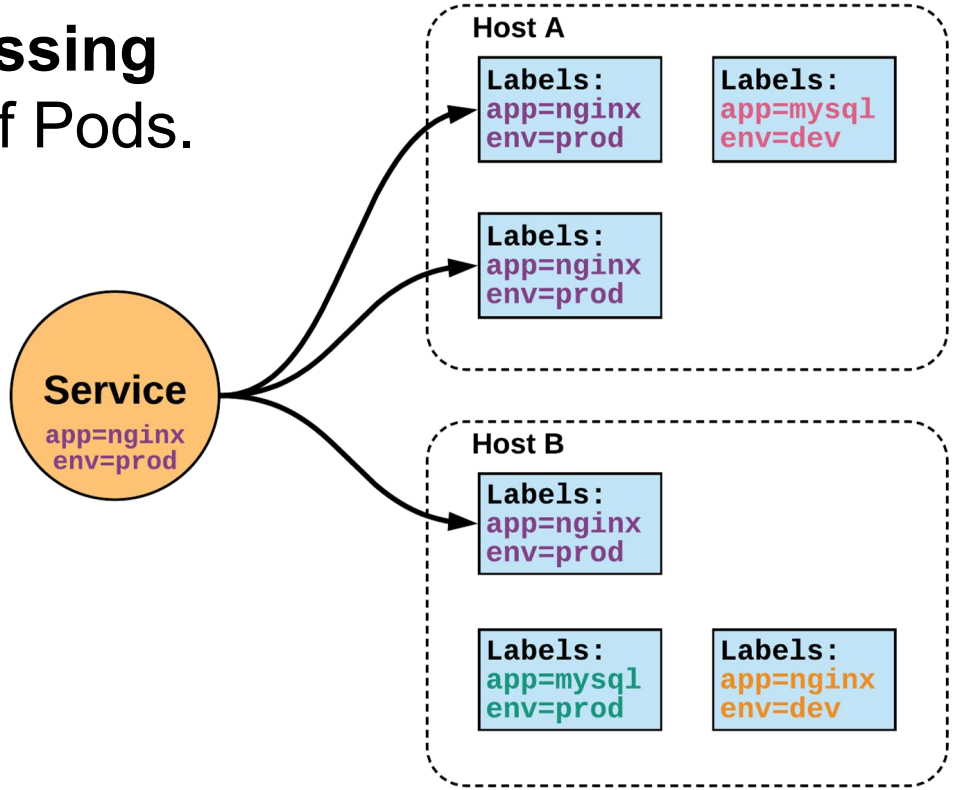


Demo

POD
SERVICE

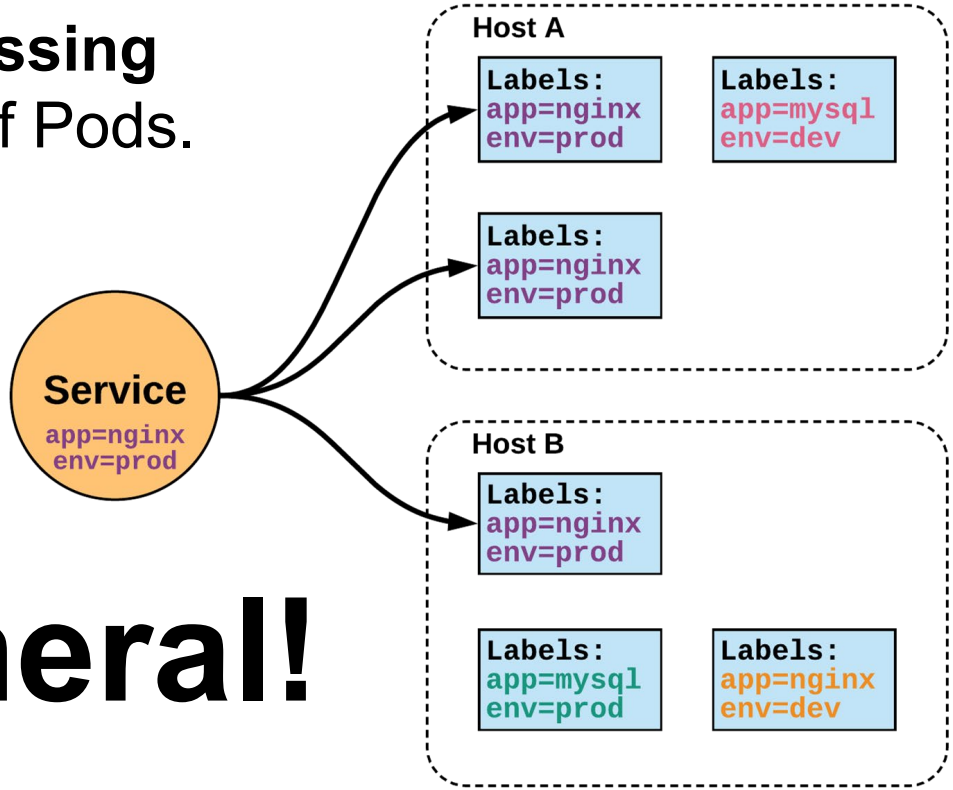
Services

- **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 Ephemeral!

Demo

SERVICE DEPLOYMENT

Concepts and Resources

API Overview

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

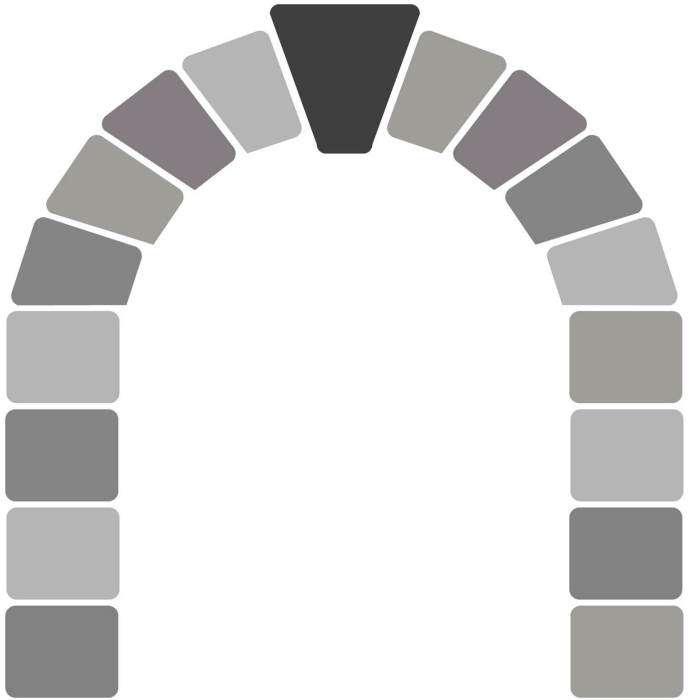


Image Source

API Groups

- Designed to make it extremely simple to both understand and extend.

Format:

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

Examples:

`/apis/apps/v1/deployments`

`/apis/batch/v1beta1/cronjobs`

- 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`.

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 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
```

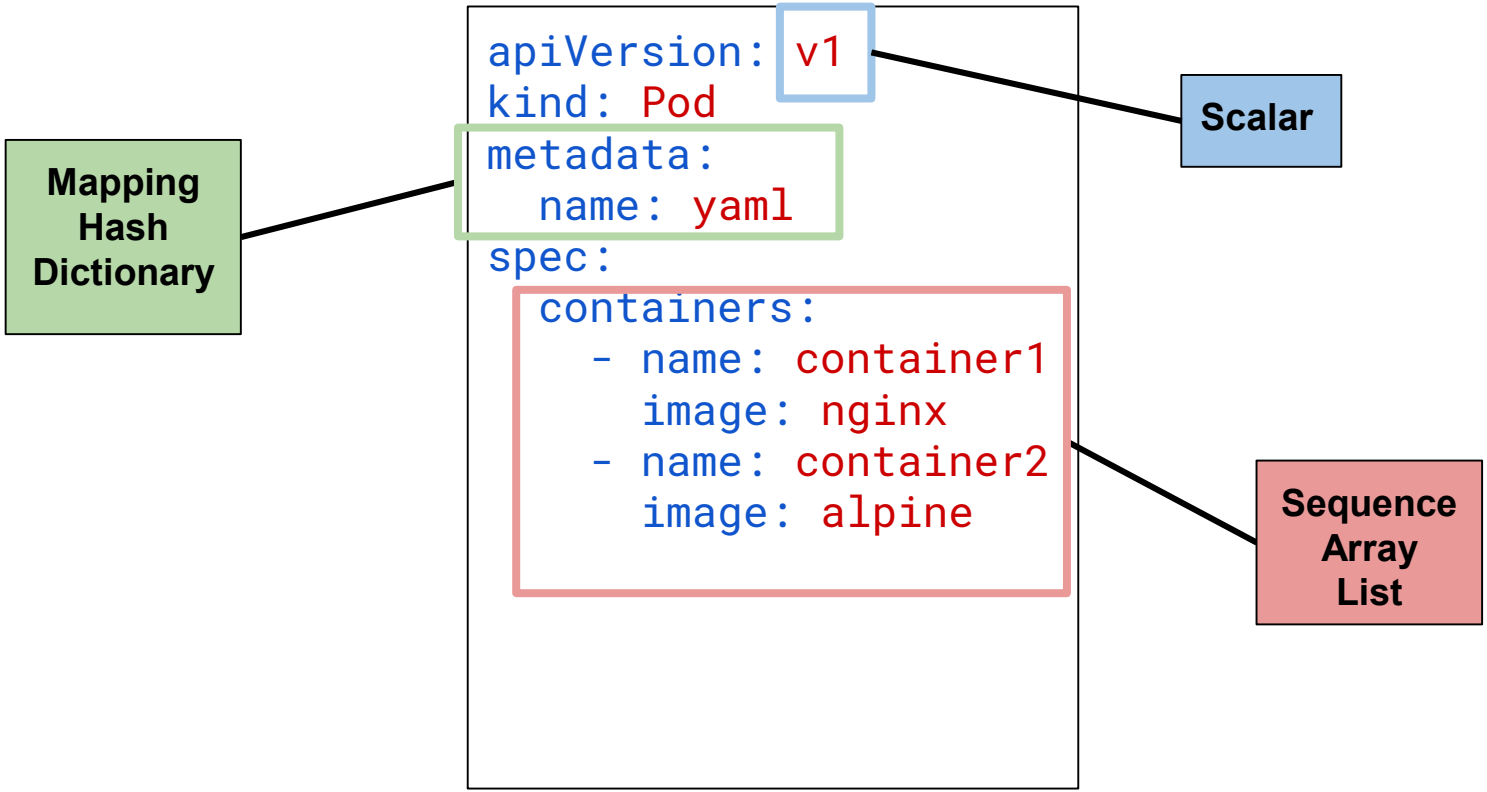
Object Expression - YAML

- 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

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

Object Expression - YAML



YAML vs JSON

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

```json
{
 "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

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

Example Status Snippet

```yaml
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

Core Concepts

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

Namespaces

Pods **Services**

Labels **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
```

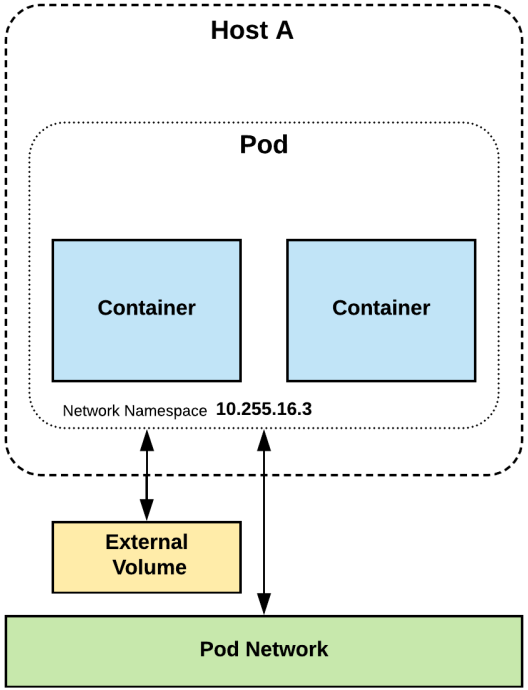
Default Namespaces

- **`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

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

```yaml
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

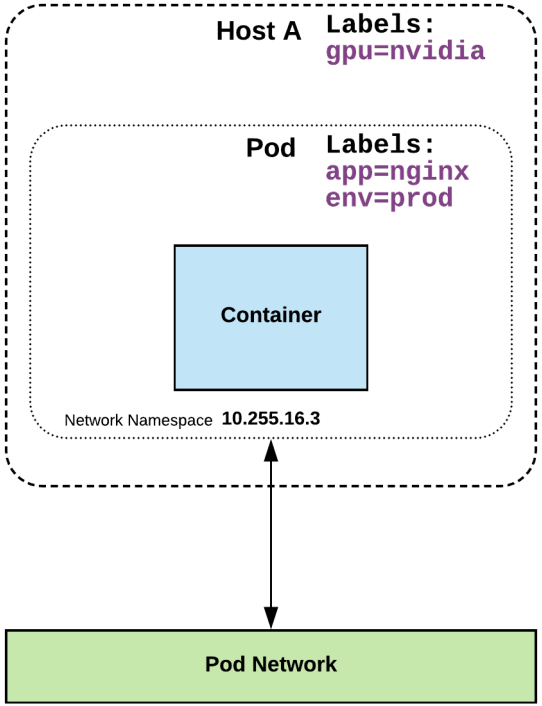
- `name` - The name of the container
- `image` - The container image
- `ports` - array of ports to expose. Can be granted a friendly name and protocol may be specified
- `env` - array of environment variables
- `command` – Entrypoint array (equiv to Docker `ENTRYPOINT`)
- `args` - Arguments to pass to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

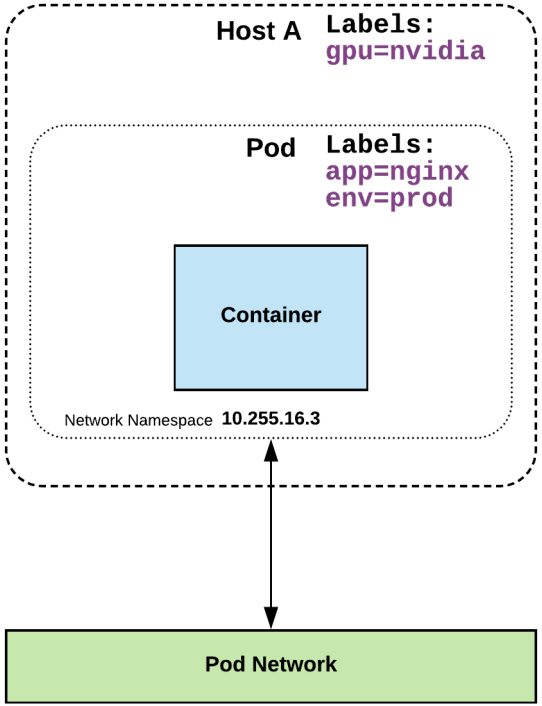
- key-value pairs that are used to identify, describe and group together related sets of objects or resources.
- **NOT** characteristic of uniqueness.
- Have a strict syntax with a slightly limited character set*.



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

Label Example

```yaml
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
```



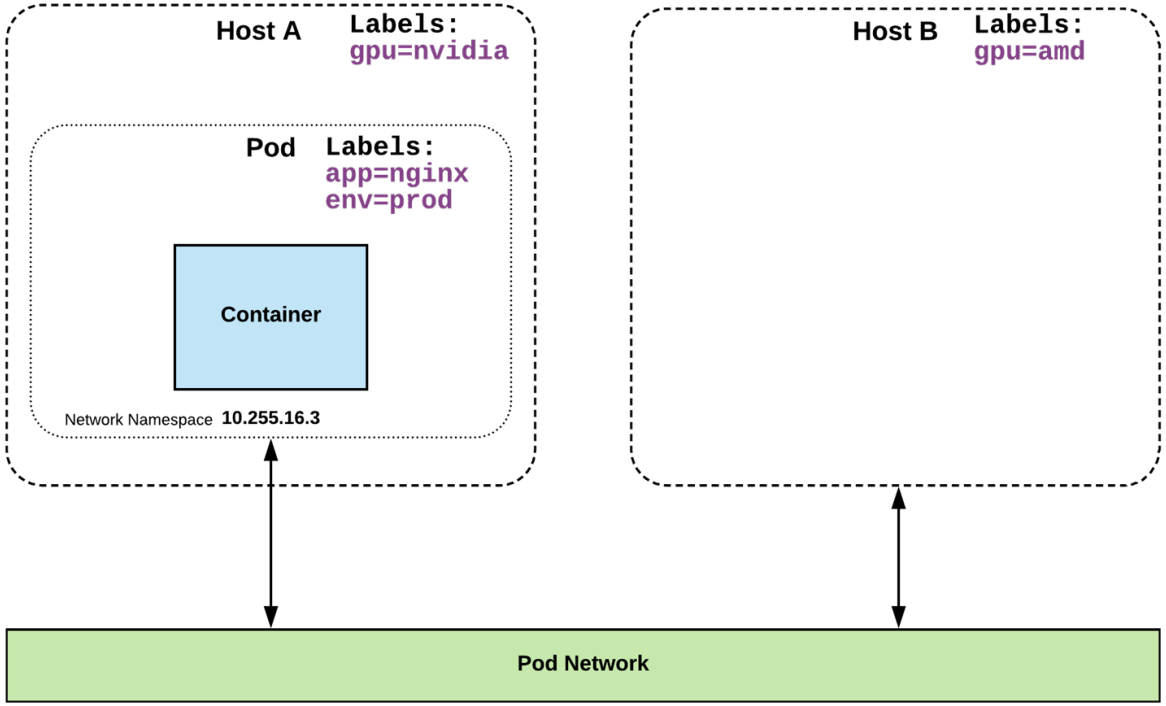
Selectors

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

```yaml
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

```yaml
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

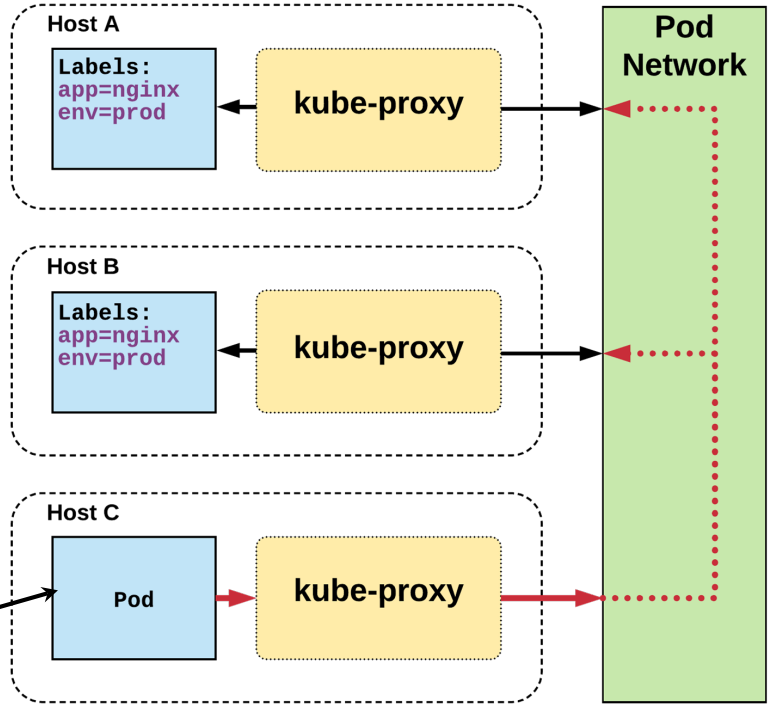
ClusterIP services exposes a service on a strictly cluster internal virtual IP.

```yaml
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
```

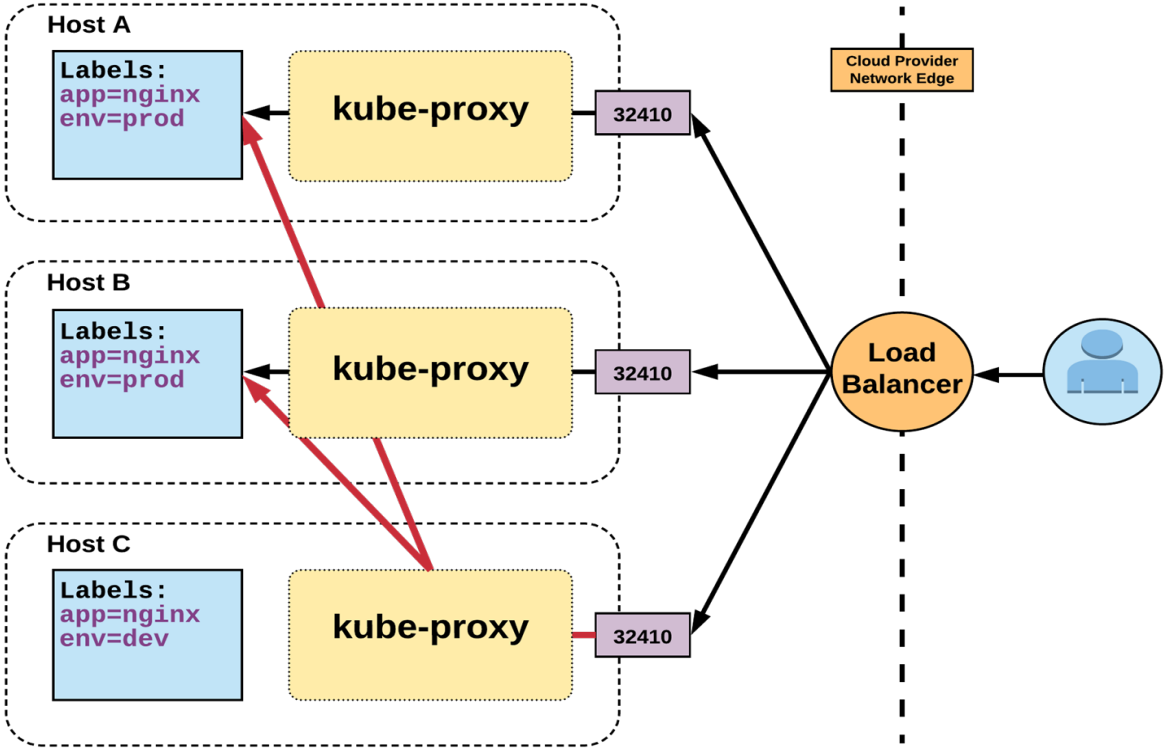


LoadBalancer Service

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

```yaml
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.

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

Demo

PODS
SERVICE
DEPLOYMENT

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.

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

```yaml
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

```yaml
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.

```yaml
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

```yaml
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>

```yaml
  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

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

```
$ 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 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
```

```
/ # 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

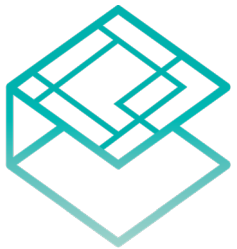
Architecture Overview

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)

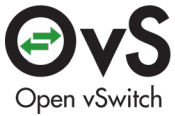
- Pod networking within Kubernetes is plumbed via the Container Network Interface (CNI).
- Functions as an interface between the container runtime and a **network implementation plugin**.
- CNCF Project
- Uses a simple JSON Schema.



CNI

CNI Plugins

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



- 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
- persistentVolumeClaim
- 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 it's own unique `name`.
- `volumeMounts:` A container specific list referencing the Pod volumes by `name`, along with their desired `mountPath`.

```yaml
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 it's own unique `name`.
- `volumeMounts:` A container specific list referencing the Pod volumes by `name`, along with their desired `mountPath`.

```yaml
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 it's own unique `name`.
- `volumeMounts:` A container specific list referencing the Pod volumes by `name`, along with their desired `mountPath`.

```yaml
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: {}
```

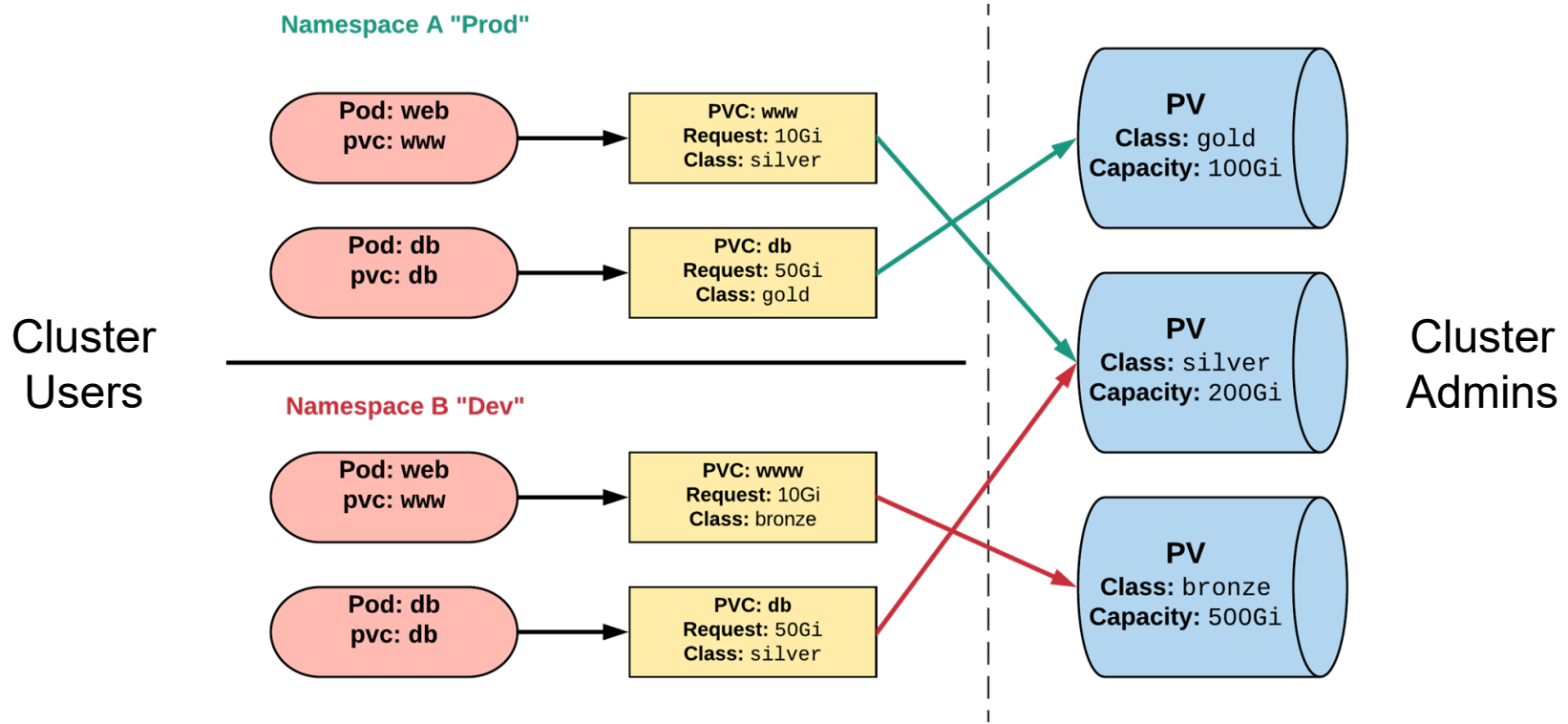
Persistent Volumes

- 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

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

```yaml
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

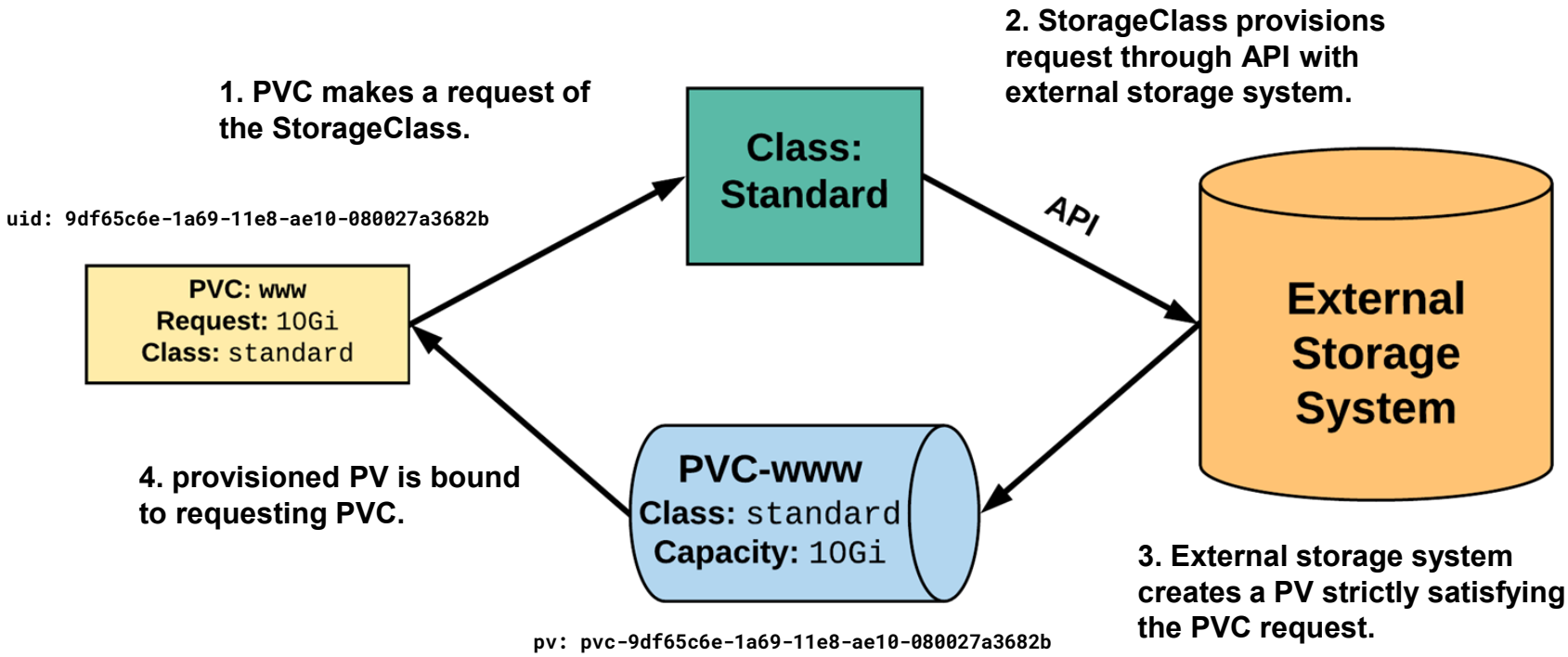
```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-selector-example
  labels:
    type: hostpath
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/mnt/data"
```

```yaml
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

Configuration

- ConfigMap
- Secret

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

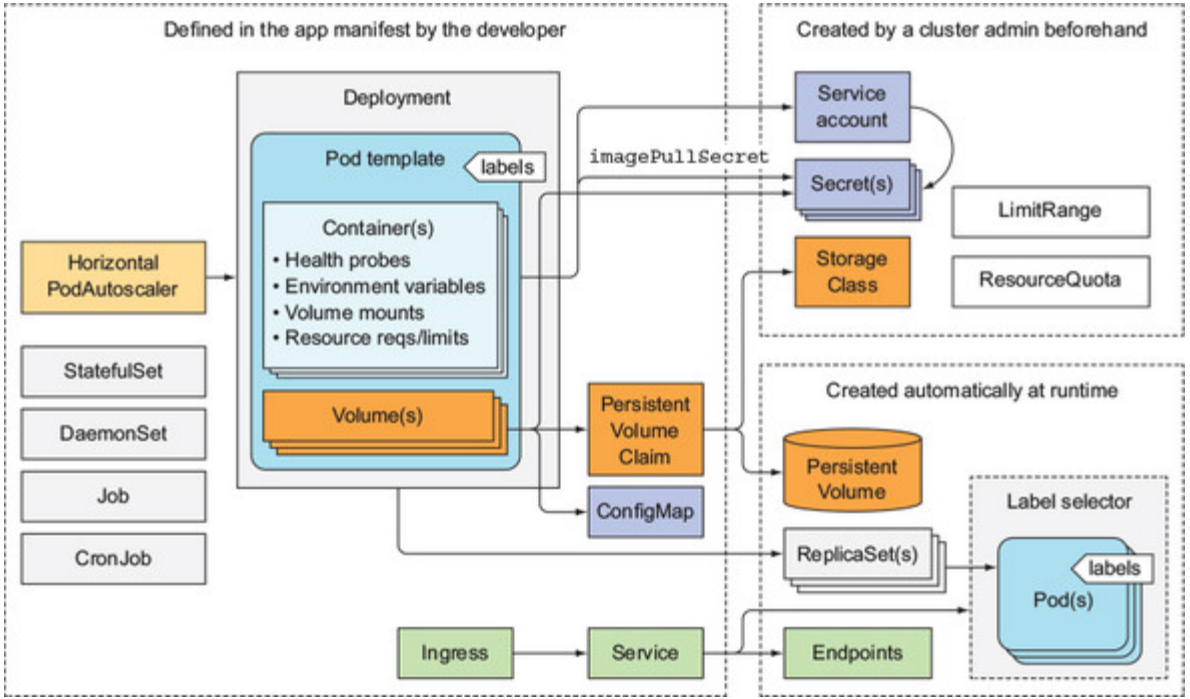
- Functionally identical to a ConfigMap.
- Stored as **base64 encoded content.**
- Encrypted at rest within etcd (**if configured!**).
- Ideal for username/passwords, certificates or other sensitive information that should not be stored in a container.
- Can be created from a manifest, literals, directories, 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: bXlwYXNzd29yZA==
```

Resources in a typical application



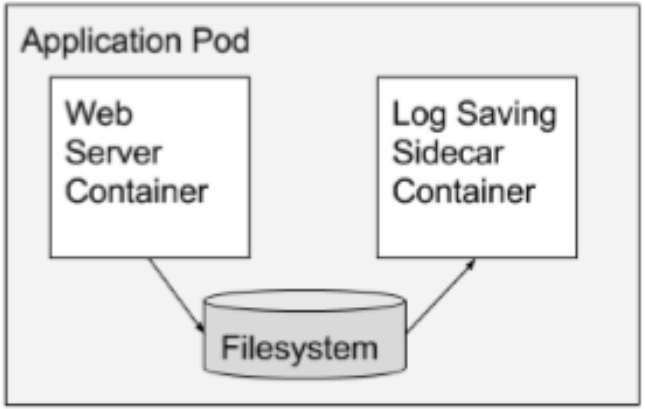
Source:
Kubernetes in Action
Book by
Marko Luksa

Patterns

Section #6

Sidecar Pattern

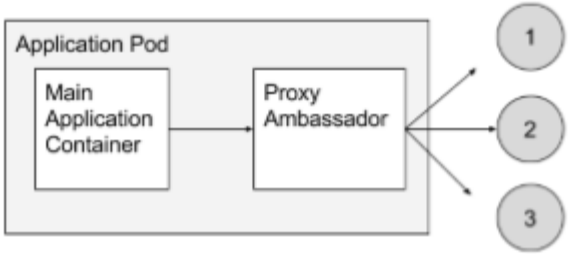
- Main container paired with “sidecar” container
- Containers are on the same machine and share the disk
- “sidecar” can scavenge spare CPU cycles



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

Ambassador Pattern

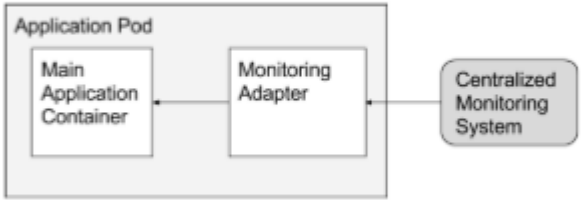
- Sidecar proxies all communication to and from the main container
- Simplified view of “outside”
- Sidecar and main container share the network interface



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

Adapter Pattern

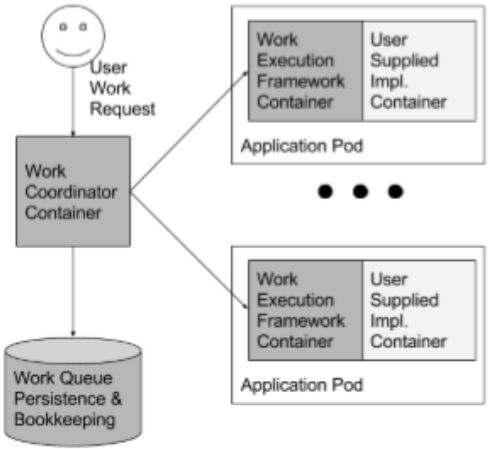
- In contrast to Ambassador pattern, adapter provides standard view of the container to the outside world



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

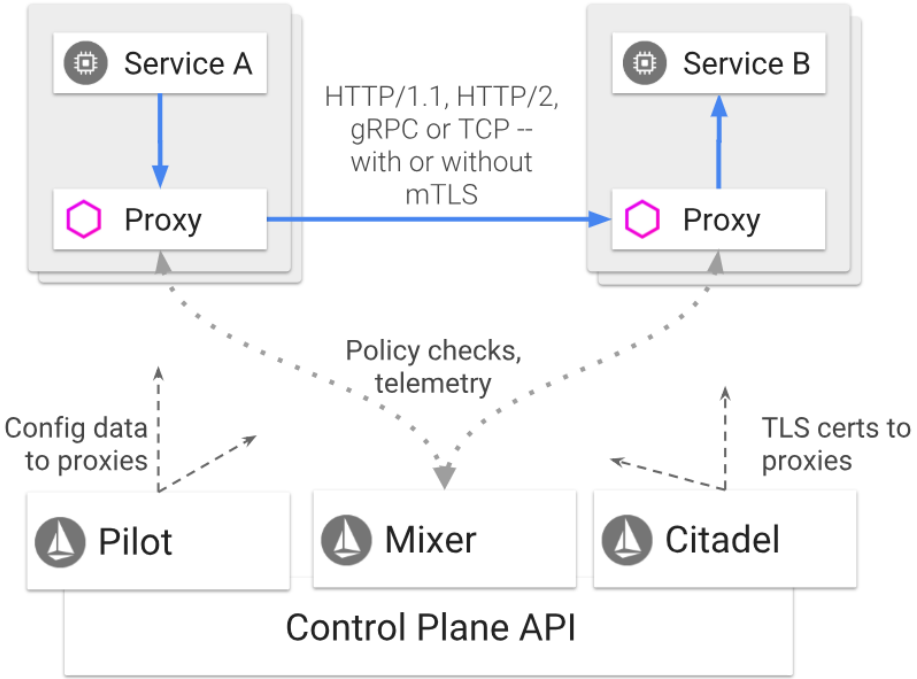
Istio

- 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.

Istio

- 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



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

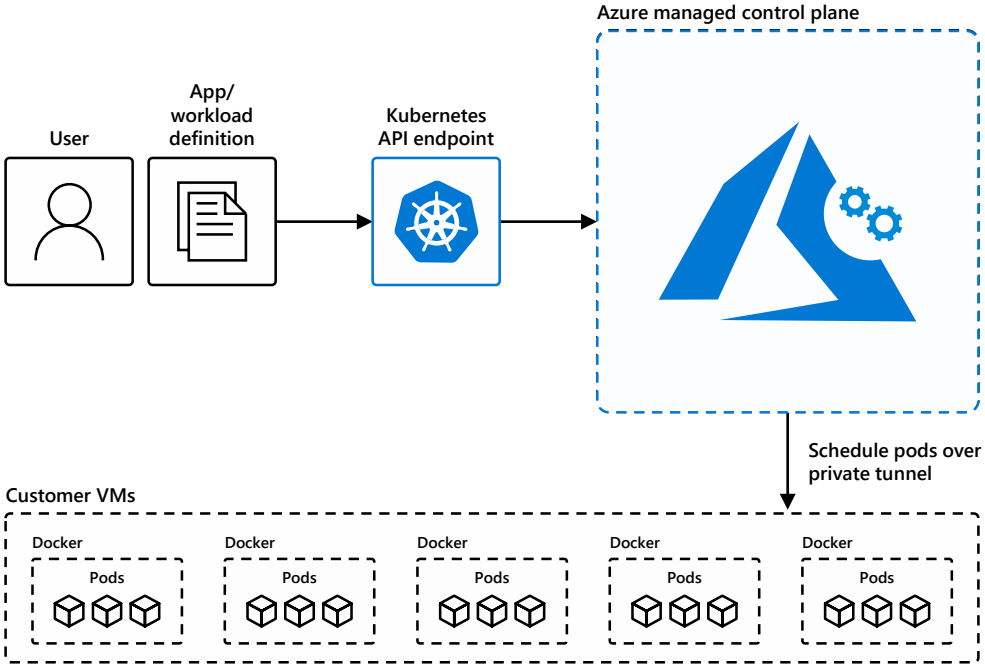
Side car pattern

Azure Kubernetes Service (AKS)

Section #4

AKS at a glance!

- Automated upgrades, patches
- High reliability, availability
- Easy, secure cluster scaling
- Self-healing
- API server monitoring
- At no charge



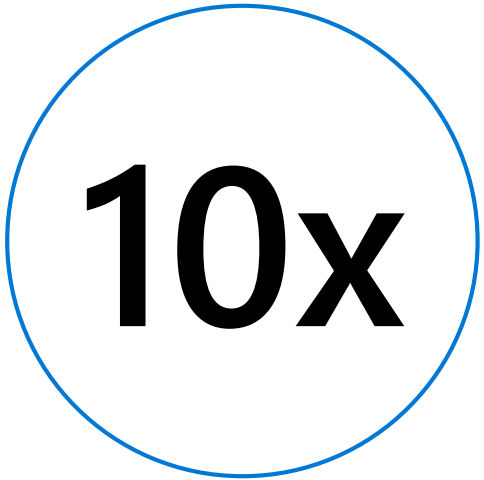
AKS Overview

- 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

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

AKS Momentum



Azure Kubernetes Service
usage grew 10x since it was made
generally available in June 2018

Trusted by thousands of customers

J.B. HUNT · Hafslund · MAERSK · AMBITENERGY

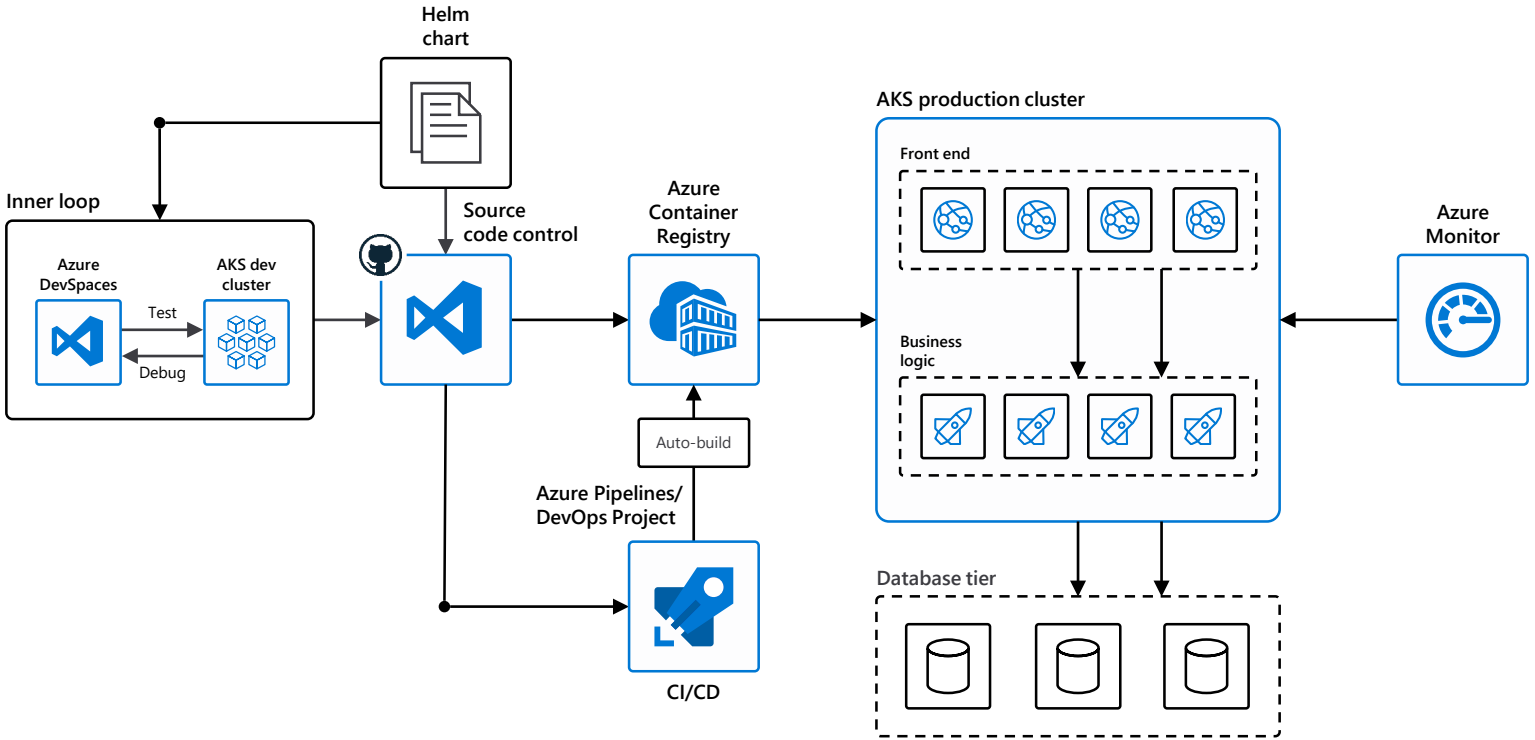
webjet.com.au · WhiteSource · falkonry · Aporeto

Altair · SOTETSU · xerox · SIEMENS Healthineers

varian · equinor · Tryg · BOSCH

Dated November 2018

AKS end to end experience



07/2018

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

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

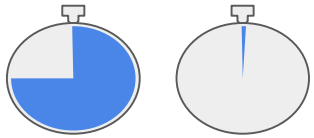
AKS etcd

- SSD-backed
- Fully automated
  - Quorum management
  - Backup, Restore from Azure Blob Storage
- Cosmos DB etcd API

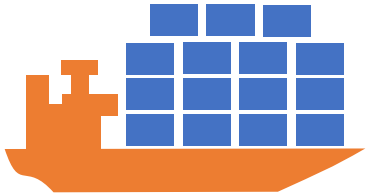
Demo

AKS in the Portal

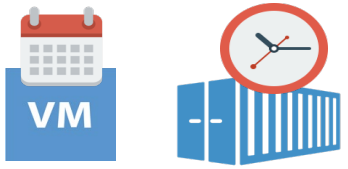
Implications Of Dynamically Orchestrated Containers



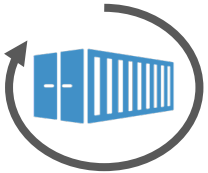
900x
start time



15x
workloads



25x shorter
lifetime

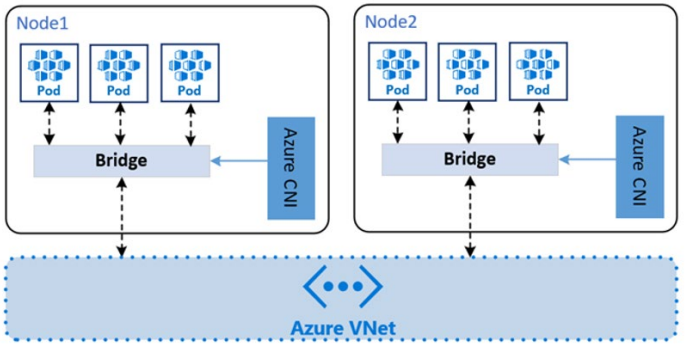


375x
network churn

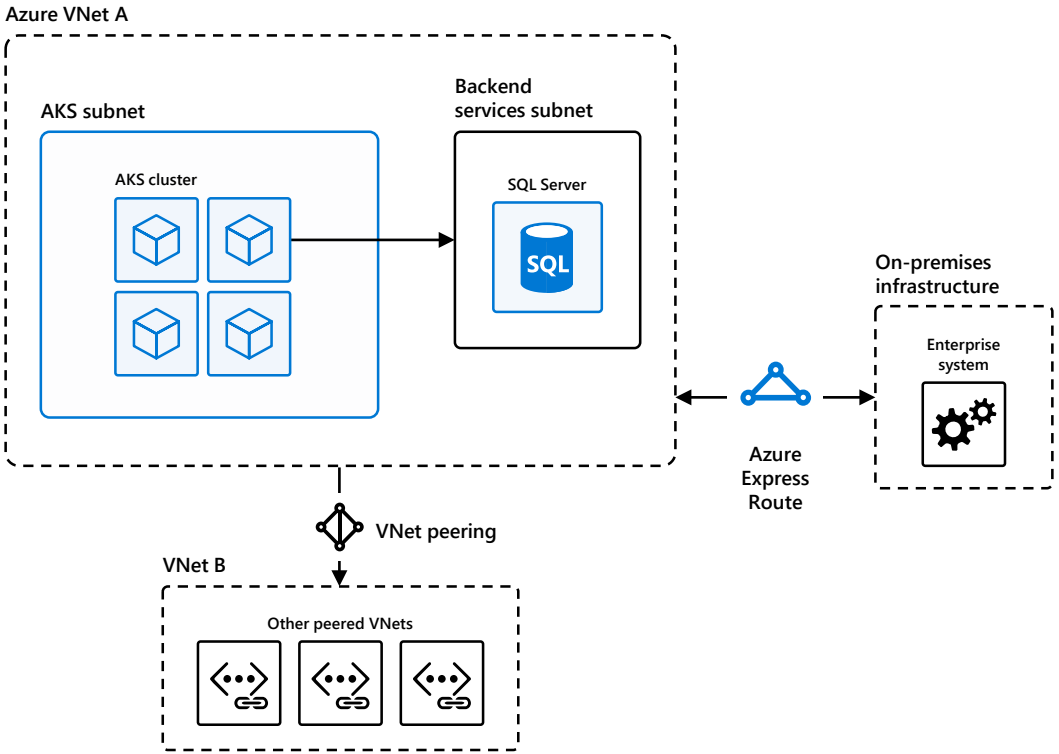
15x
attack surface area

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 in 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

Operational Best Practices: Kubernetes Network Security

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

Baseline traffic patterns

Enforce East-West Policies Within Cluster

Zero Trust Security Model

Multi-factor authentication (mTLS cert + network identity)

Encryption of data-in-transit

Role based access controls

GitOps / policy-as-code

Enterprise Controls

Continuous Compliance

Visibility, monitoring, & alerting

Reporting

Demo

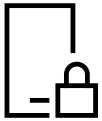
Ingress Controller
Network Policy

AKS Security

AKS Security and Compliance

Control access through AAD and RBAC

Safeguard keys and secrets with Key Vault

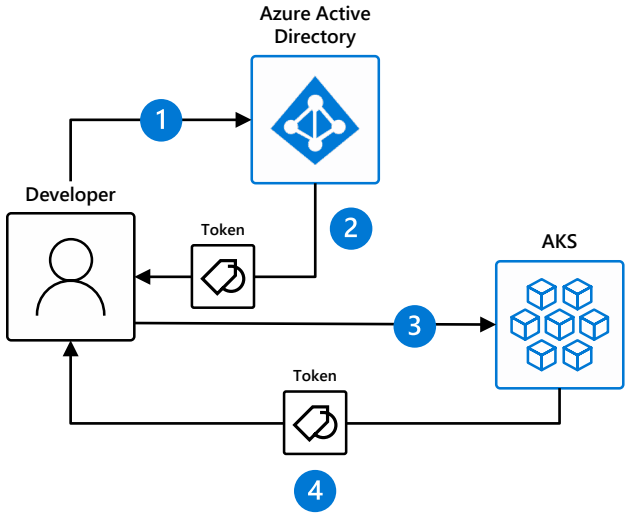


Secure network communications with VNET and CNI

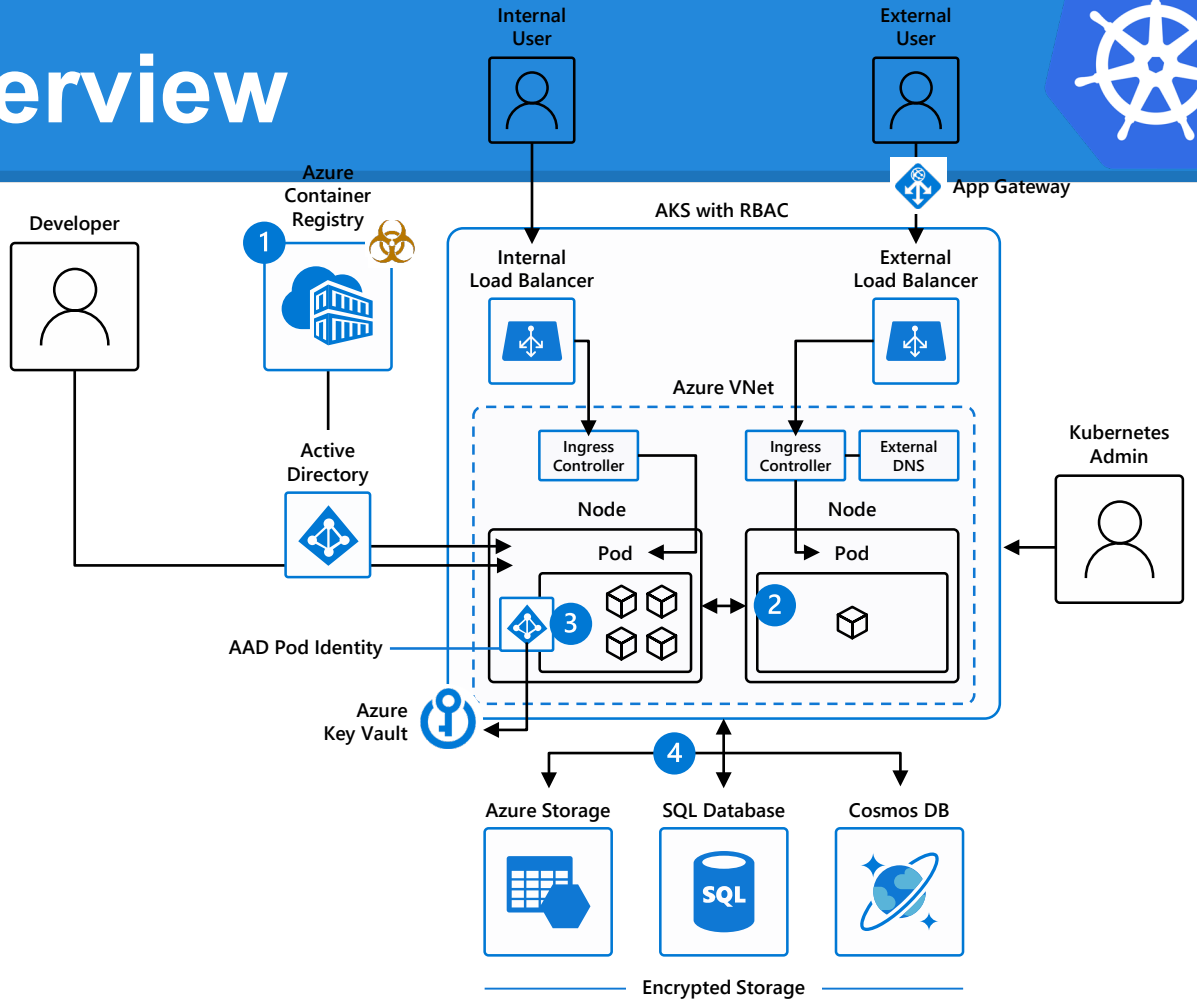
Compliant Kubernetes service with certifications covering SOC, HIPAA, and PCI

Identity and access management through AAD and RBAC

1. A developer authenticates to the AAD token issuance endpoint and requests an access token
2. The AAD token issuance endpoint issues the access token
3. The access token is used to authenticate to the secured resource
4. Data from the secured resource is returned to the web application



Security Overview



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

Demo

Custom Service Accounts

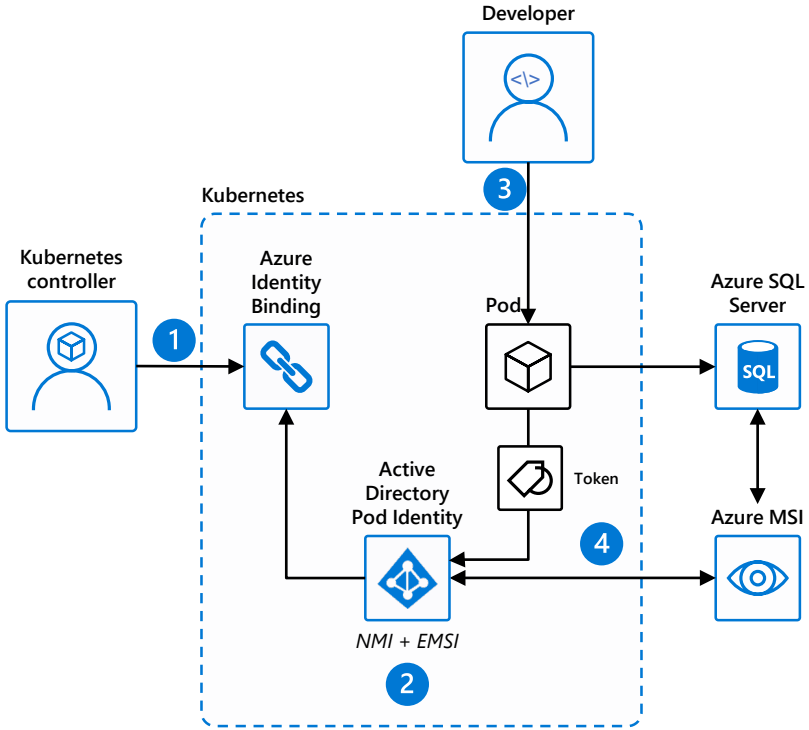
Pod Identity

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 Identify (MSI)

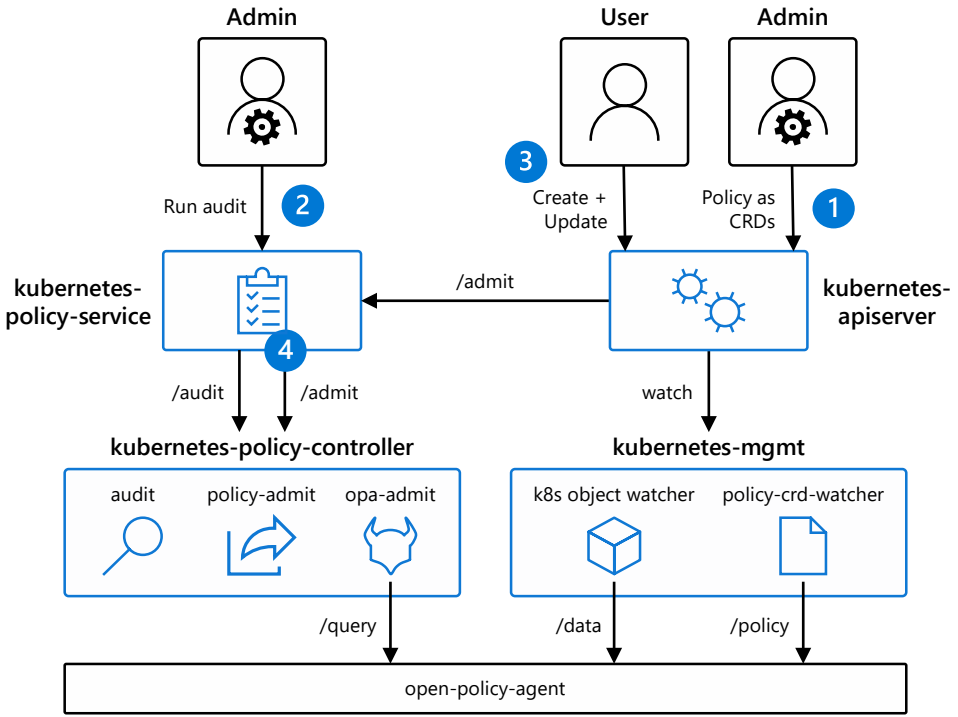
3. Developer creates a pod with a service account. Pod uses standard Azure SDK to fetch a token bound to MSI

4. Pod uses access token to consume other Azure services; services validate token



Policy Controller

1. Admin adds policy for the cluster
2. Admin audits compliance of the cluster using /audit endpoint
3. User uses standard Kubernetes API to operate the cluster and the create actions are guarded by policy
4. Kubernetes-policy-controller provides an admission controller webhook that performs evaluations by calling open-policy-agent (OPA) service

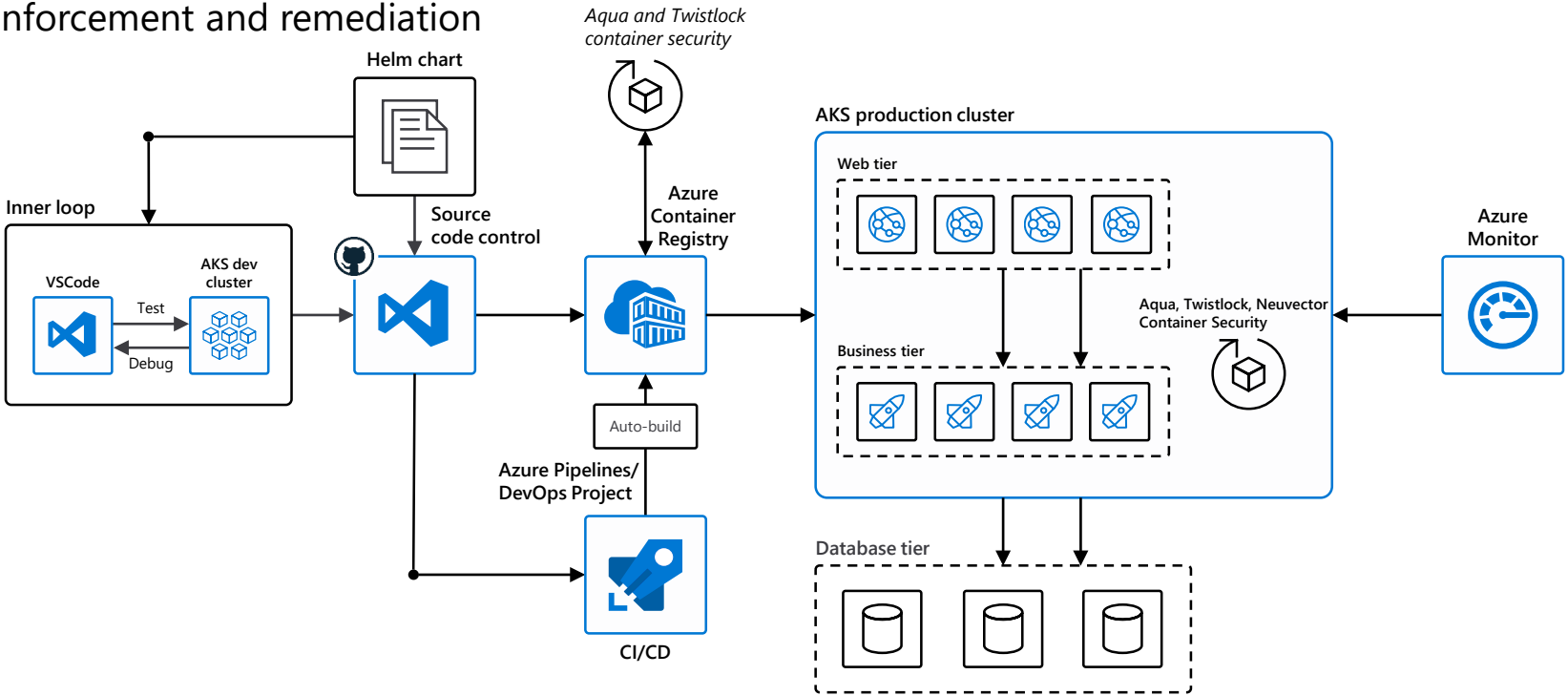


Container Level – The images

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

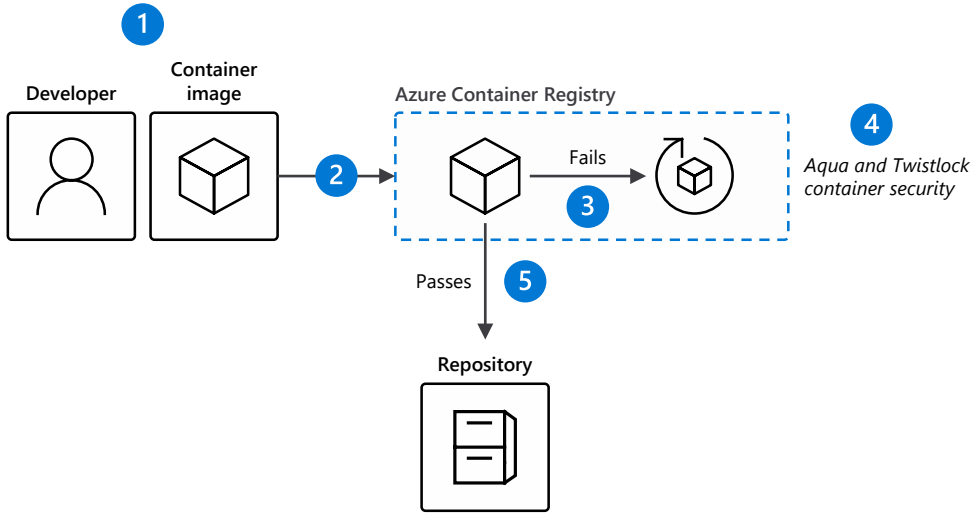
Container Level – Images and Runtime

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



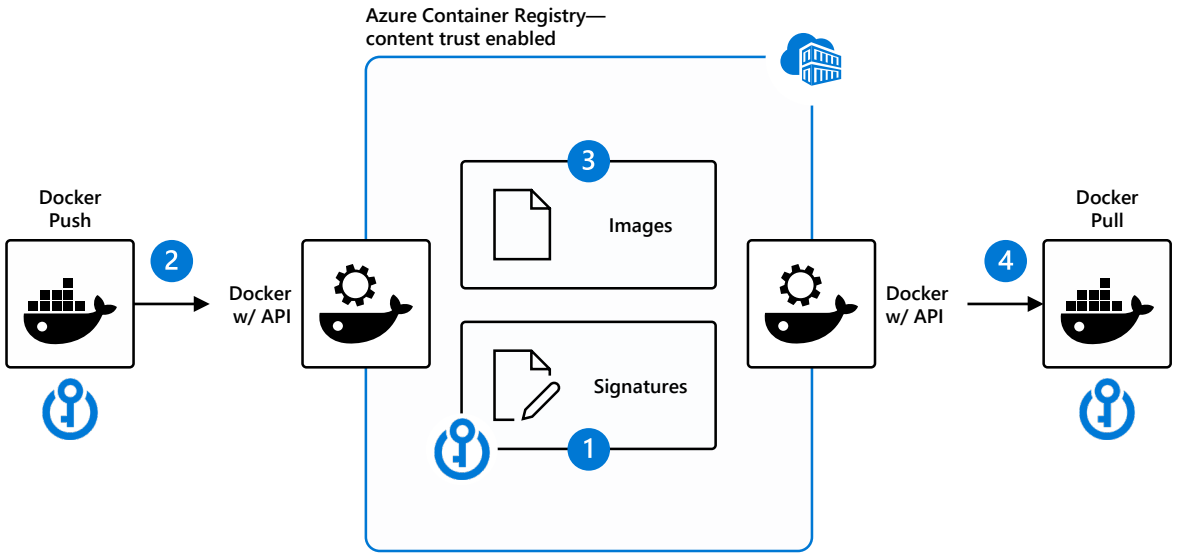
ACR – Vulnerability Scanning

1. Developer/CI system builds container image
2. Image pushed to Azure Container Registry
3. Azure Container Registry quarantines image until scanning passes
4. Azure Container Registry scans content leveraging Aqua, Twistlock
5. Azure Container Registry publishes the image to the repository



ACR – Content Trust

1. A set of cryptographic signing keys are associated with Azure Container Registry and used for image signing
2. Image publisher signs the image and pushes to the Azure Container Registry
3. Signed image is stored in Azure Container Registry
4. When an image consumer pulls a signed image, their Docker client verifies the integrity of the image



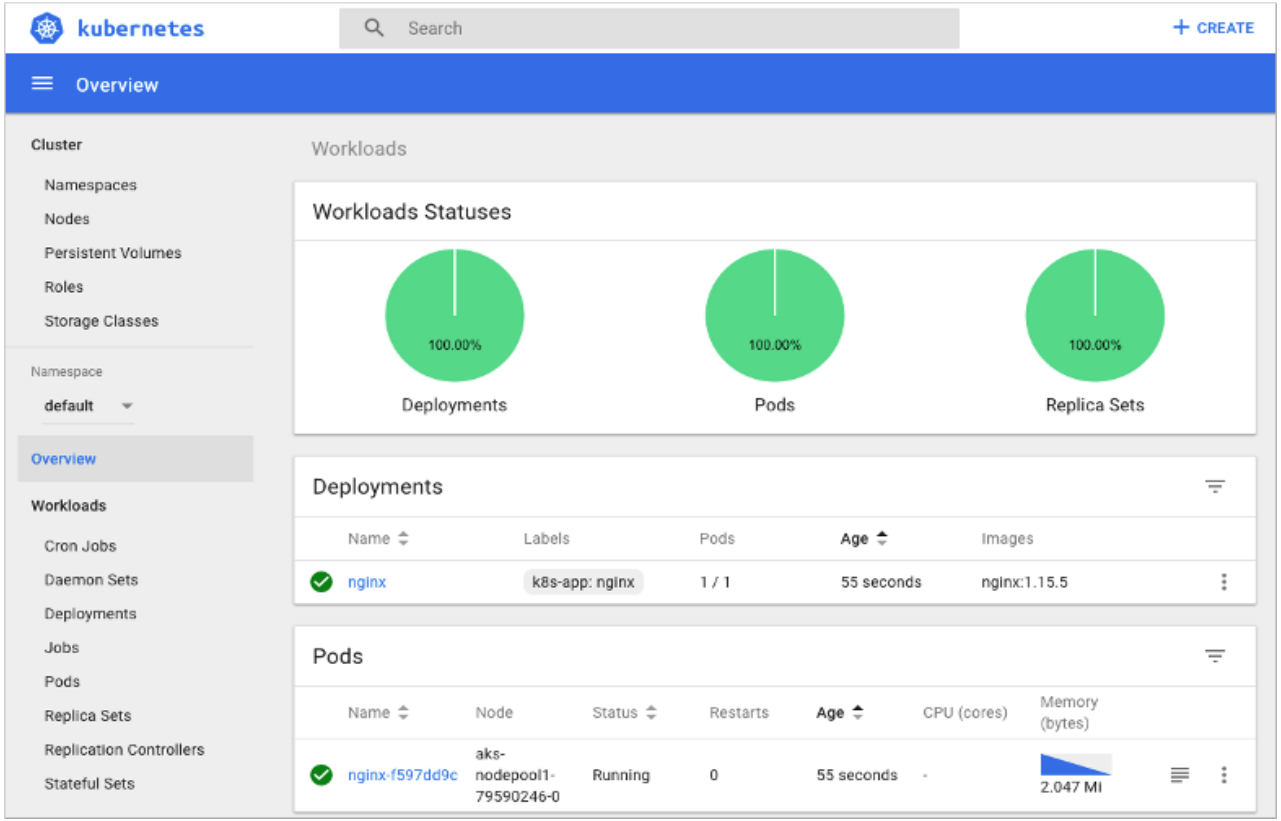
Demo

Custom Security Role
Pod Identity

But we have a policy for tags?!!

- Using Azure Policy
  - Enforce tags on resources groups (Deny Policy)
  - Apply tags on resources under the Infrastructure resource group (Append Policy)
  - Create a remediation policy for all-non compliant resources
- Automate tag assignment for AKS infrastructure resources using Event Grid + Azure Functions or any other means

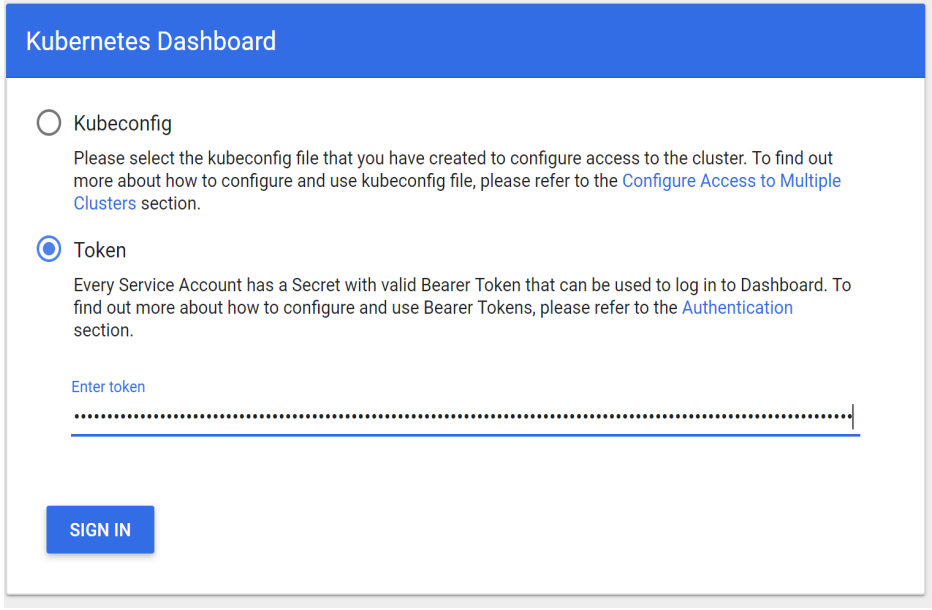
Kubernetes Dashboard



Disable the Dashboard (Recommended)

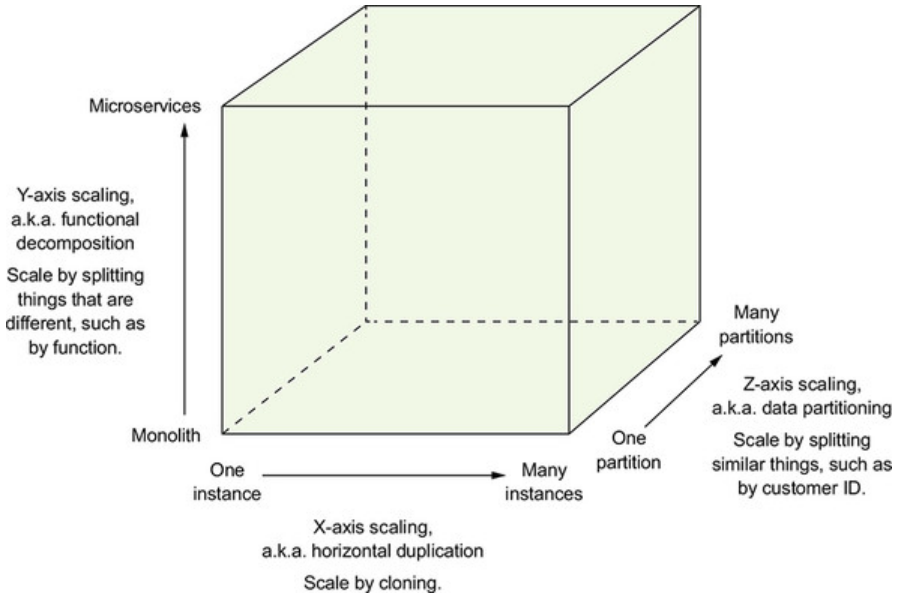
```
$ kubectl get pods -n kube-system | grep "dashboard"
kubernetes-dashboard-cc4cc9f58-whmhv   1/1     Running   0          30d

$ az aks disable-addons -a kube-dashboard -g k8s-demo -n k8s-demo-rbac

$ kubectl get pods -n kube-system | grep "dashboard"
```

Kubernetes Dashboard – less worse




```yaml
spec:
  containers:
    - name: kubernetes-dashboard
      image: kubernetesui/dashboard:v2.0.0-beta1
      imagePullPolicy: Always
      ports:
        - containerPort: 8443
          protocol: TCP
      args:
        - --auto-generate-certificates
        - --authentication-mode=token
      volumeMounts:
        - name: kubernetes-dashboard-certs
          mountPath: /certs
          # Create on-disk volume to store exec logs
        - mountPath: /tmp
          name: tmp-volume
      livenessProbe:
        httpGet:
          scheme: HTTPS
          path: /
          port: 8443
        initialDelaySeconds: 30
        timeoutSeconds: 30
  volumes:
    - name: kubernetes-dashboard-certs
      secret:
        secretName: kubernetes-dashboard-certs
    - name: tmp-volume
      emptyDir: {}
  serviceAccountName: kubernetes-dashboard
```

The Scale Cube



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

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

Demo

Scaling with AKS

Virtual Kubelet with AKS

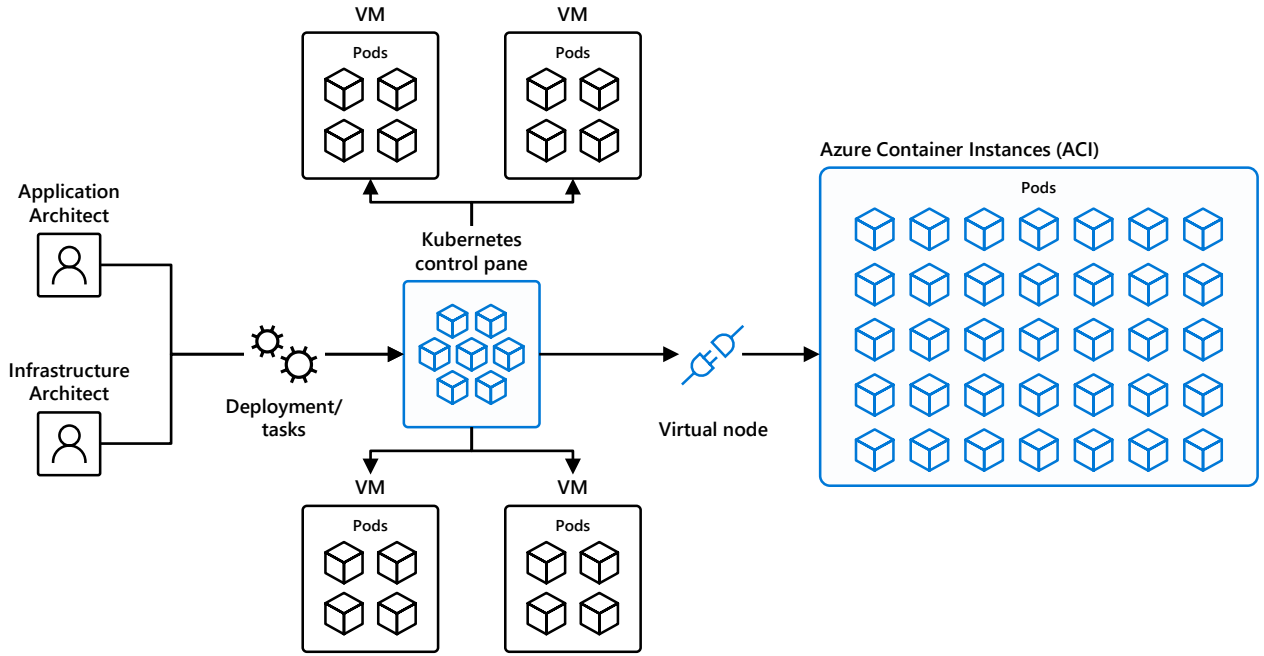
- Containers scheduled on a container instance (ACI), *as if it is a standard Kubernetes node*

```
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
```

AKS virtual node

Elastically provision additional compute capacity in seconds

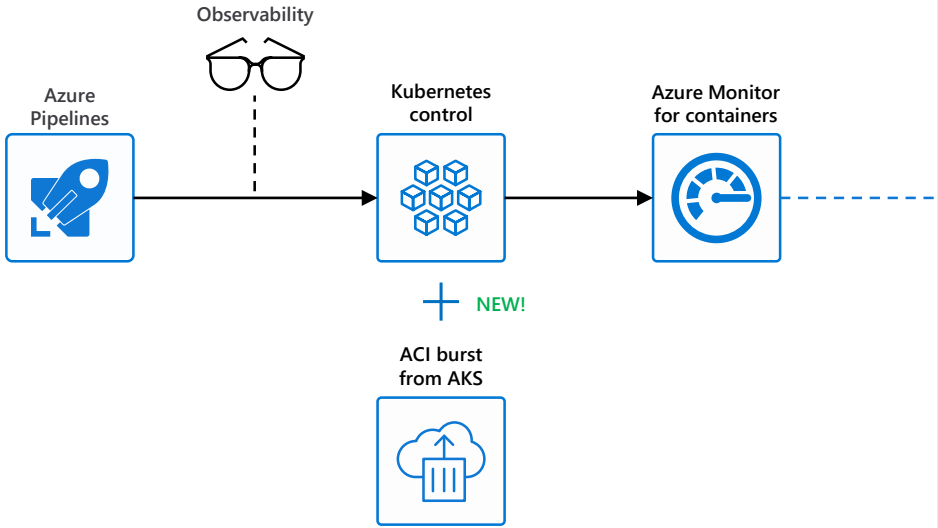


Demo

AKS + ACI

Logging and Monitoring

Azure Monitor for containers



Visualization
Visualize overall health and performance from clusters to containers with drill downs and filters

Insights
Provide insights with multi-cluster health roll up view

Monitor & Analyze
Monitor and analyze Kubernetes and container deployment performance, events, health, and logs

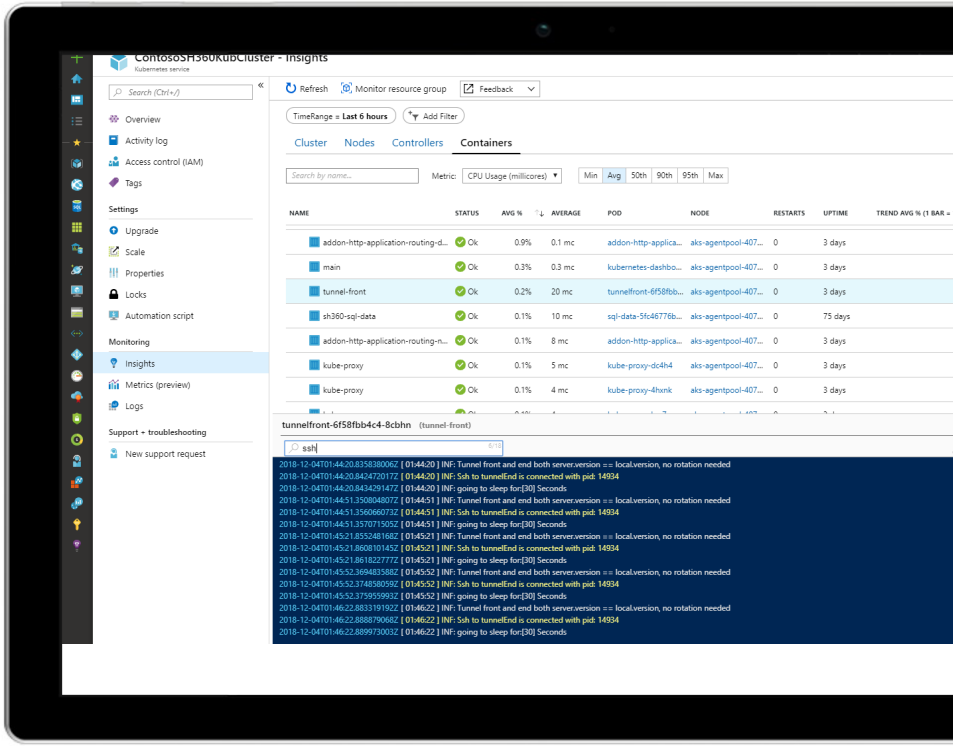
Response
Native alerting with integration to issue managements and ITSM tools

Observability
Observe live container logs on container deployment status

Azure Monitor for containers

4

1. Get detailed insights about your workloads with Azure Monitor
2. See graphical insights about clusters
3. Filter for details about nodes, controllers, and containers
4. Pull events and logs for detailed activity analysis



Monitoring/Logging your cluster

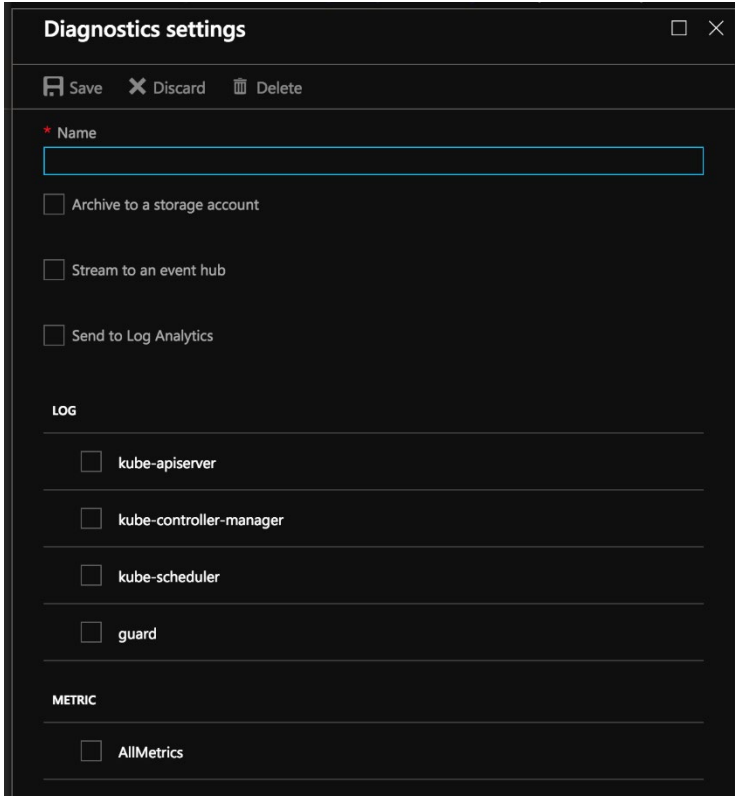
- Log Everything to stdout / stderr
- Key Metrics:
  - Node metrics (CPU Usage, Memory Usage, Disk Usage, Network Usage)
  - Kube_node_status_condition
  - Pod memory usage / limit; memory_failures_total
    - container_memory_working_set_bytes
  - Pod CPU usage average / limit
  - Filesystem Usage / limit
  - Network receive / transmit errors
- Azure Monitor for Containers

In the roadmap

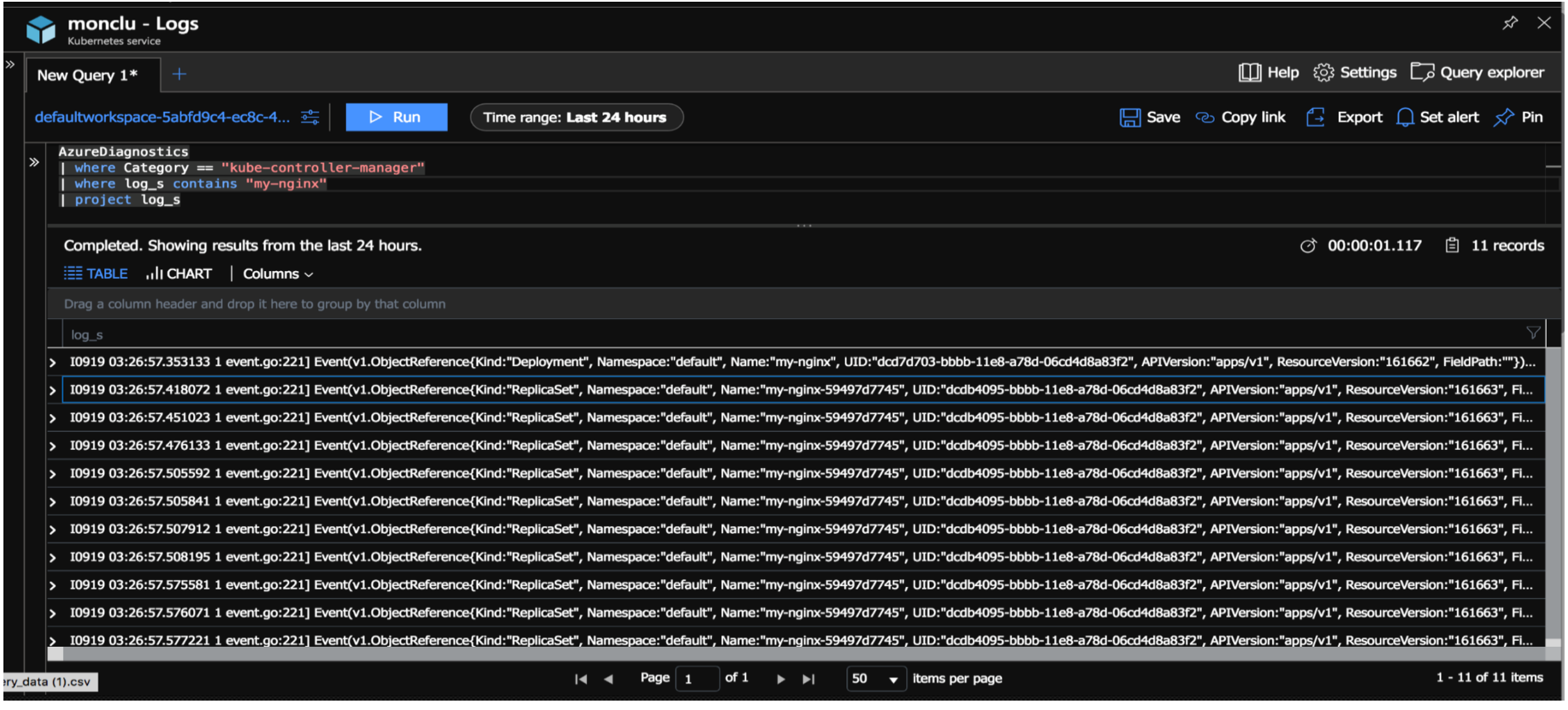
Overview Health of AKS Cluster

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



Example Control Plane Logs



Demo

Prometheus
Azure Monitor

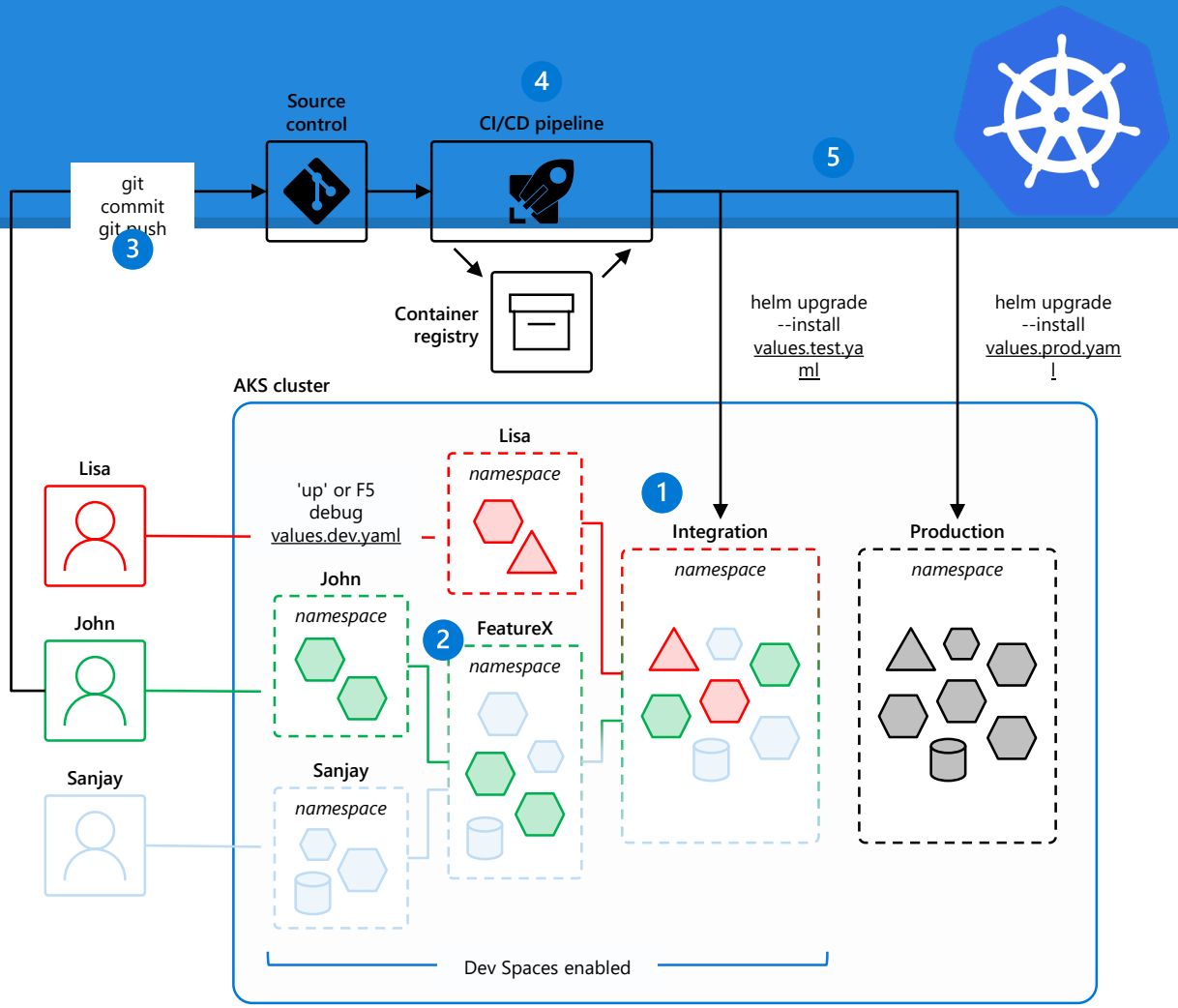
Demo

Azure DevOps

Dev Spaces

1. The "Integration" dev space is running a full baseline version of the entire application
2. John and Sanjay are collaborating on FeatureX; it is setup as a dev space and running all the modified services required to implement a feature
3. Code is committed to the master source control
4. A CI/CD pipeline can be triggered to deploy into "Integration," which updates the teams baseline
5. The same Helm assets used during development are used in later environments by the CD system

*Dev Spaces is enabled per Kubernetes namespaces and can be defined as anything. Any namespace in which Dev Spaces is NOT enabled runs *unaffected*.*



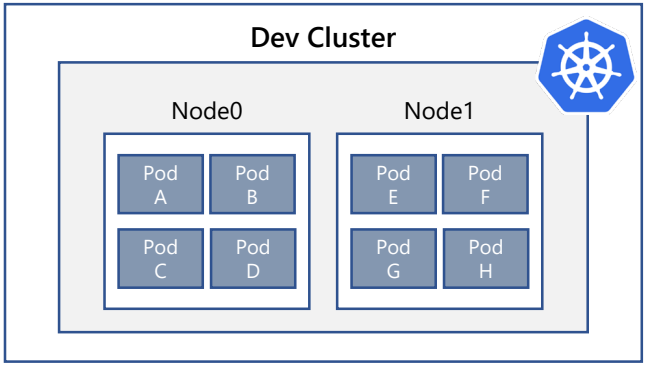
Demo

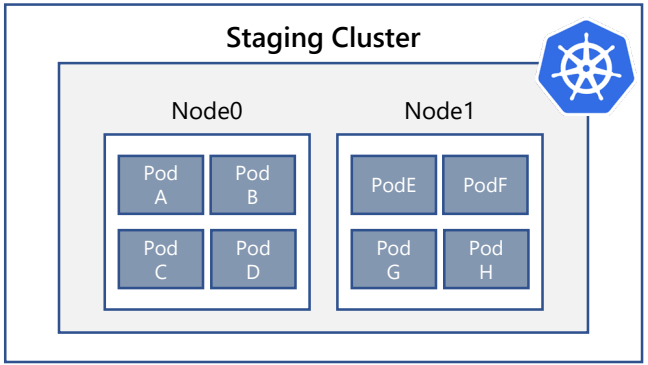
Azure DevOps

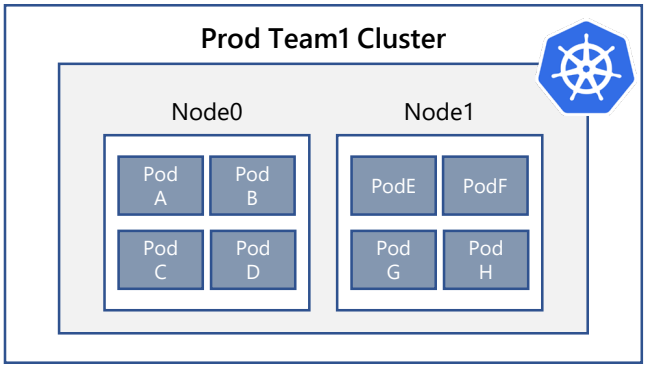
Operational best practices for Azure Kubernetes Service

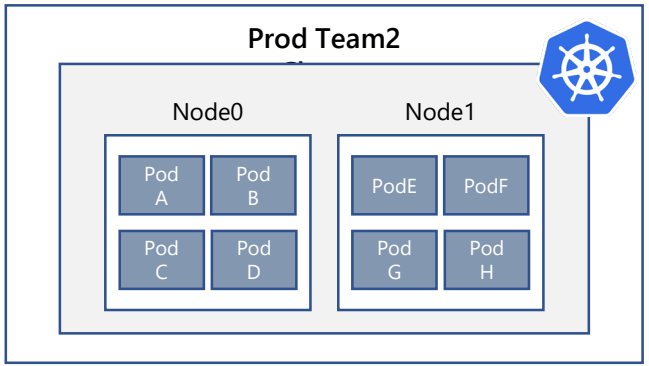
Cluster Isolation and Resource Management

Cluster Isolation Patterns: Physical Isolation

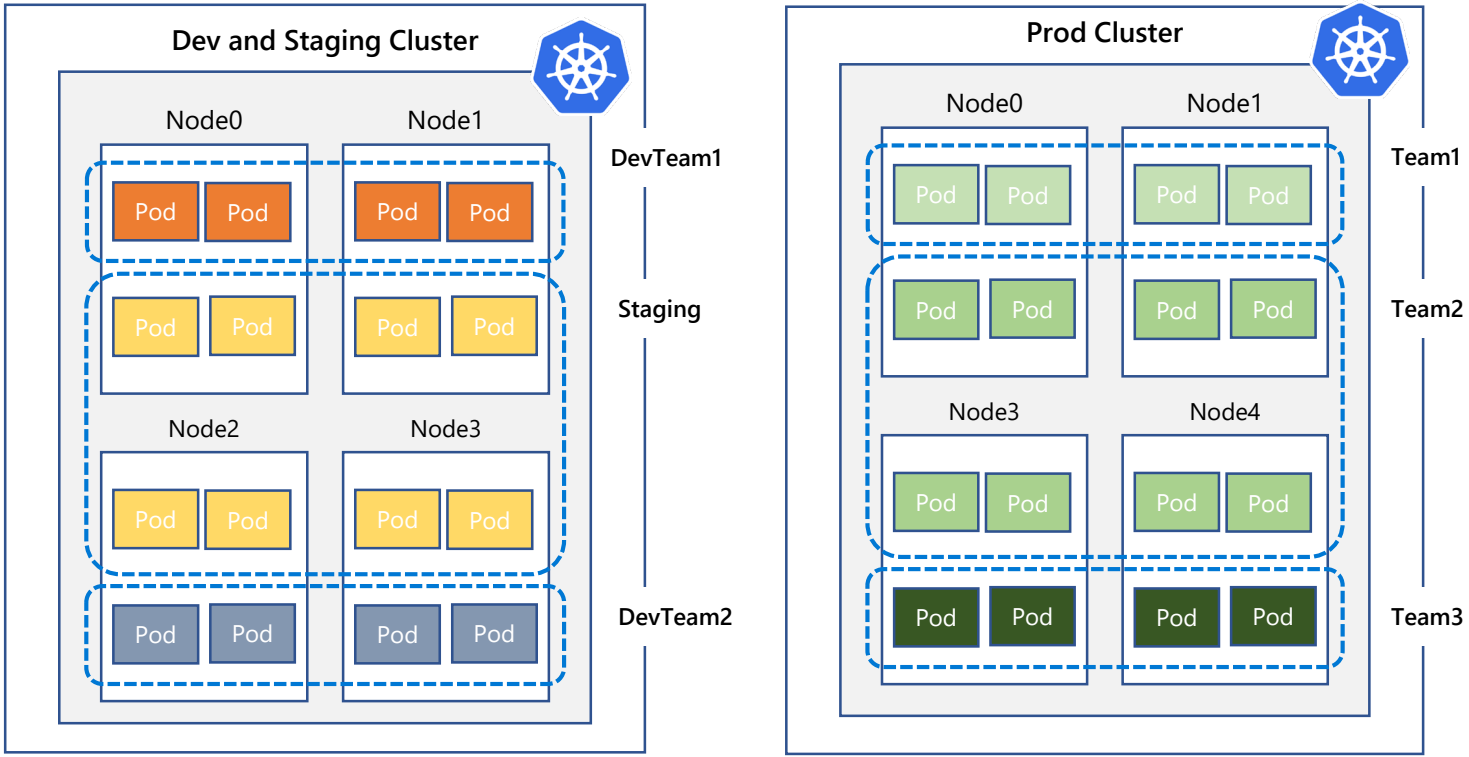








Cluster Isolation Patterns: Logical Isolation



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, Strage,…) 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
```

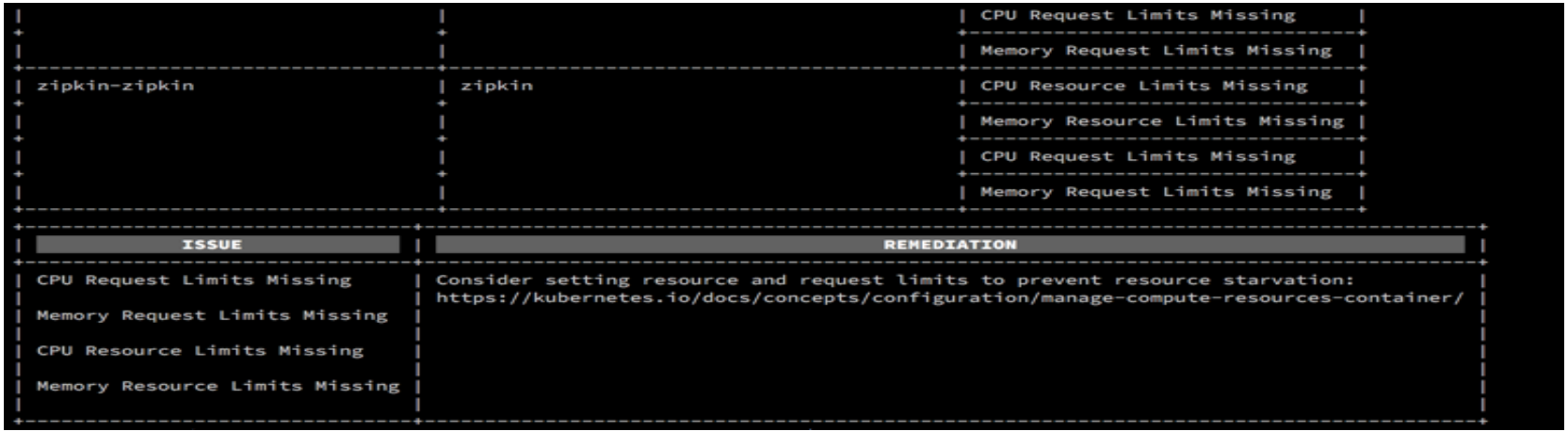
Physical Vs. Logical Isolation

	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



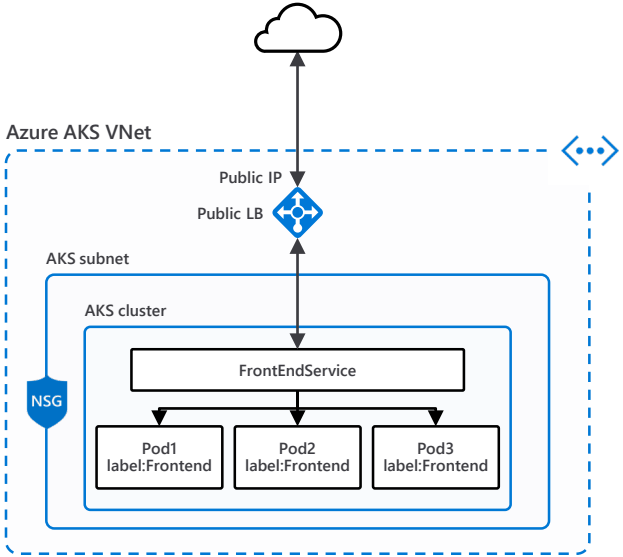
		CPU Request Limits Missing
		Memory Request Limits Missing
zipkin-zipkin	zipkin	CPU Resource Limits Missing
		Memory Resource Limits Missing
		CPU Request Limits Missing
		Memory Request Limits Missing

ISSUE	REMEDIATION
CPU Request Limits Missing Memory Request Limits Missing CPU Resource Limits Missing Memory Resource Limits Missing	Consider setting resource and request limits to prevent resource starvation: https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

Public Service

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

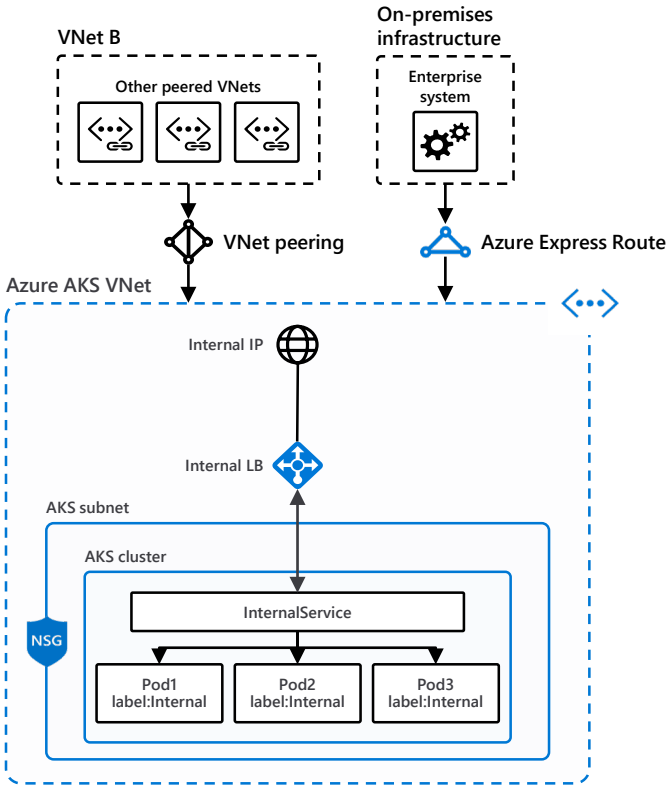
```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontendservice
spec:
  loadBalancerIP: X.X.X.X
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: frontend
```



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
```



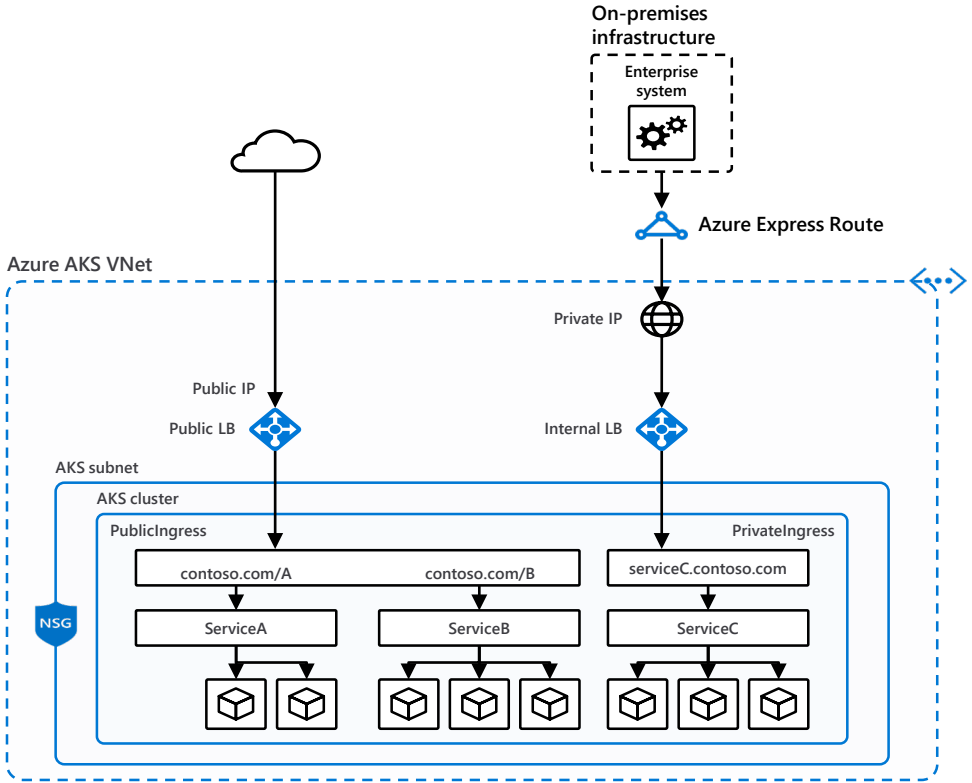
Ingress And Ingress Controllers

- 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
  - Save on public Ips

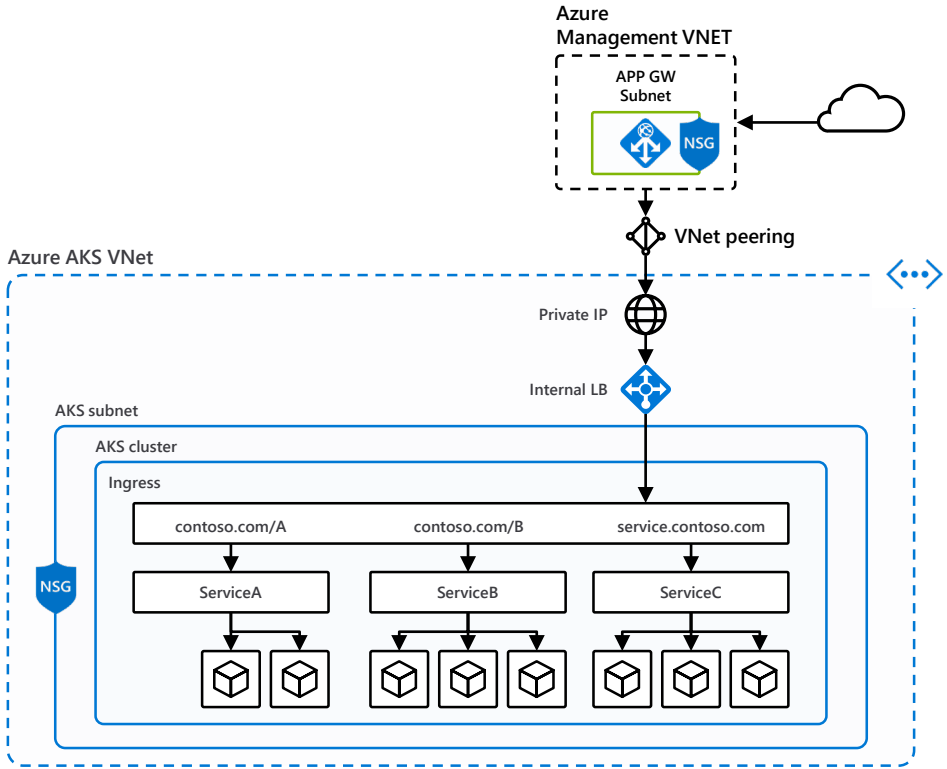
- Ingress controller is a daemon, deployed as a Kubernetes Pod, that watches the Ingress Endpoint for updates. Its job is to satisfy requests for ingresses. Most popular one being Nginx.

Ingress

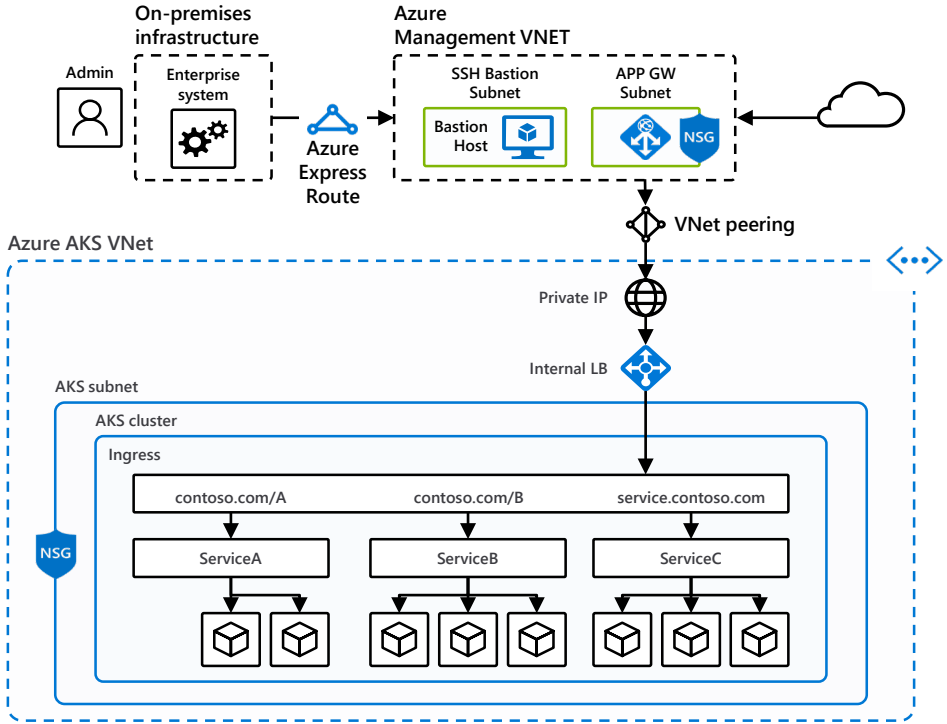
```
kind: Ingress
metadata:
 name: contoso-ingress
   annotations: kubernetes.io/ingress.class:
”PublicIngress"
spec:
 tls:
 - hosts:
  - contoso.com
  secretName: contoso-secret
 rules:
  - host: contoso.com
    http:
     paths:
     - path: /a
       backend:
        serviceName: servicea
       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



Securing Kubernetes Services With WAF



Cluster Management Through Bastion Host



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:
    container.apparmor.security.beta.kubernetes.io
/hello: localhost/k8s-apparmor-example-deny-
write
spec:
  containers:
  - name: hello
    image: busybox
    command: [ "sh", "-c", "echo 'Hello
AppArmor!' && sleep 1h" ]
```

```
$ kubectl exec hello-apparmor touch /tmp/test
touch: /tmp/test: Permission denied
error: error executing remote command: command terminated with non-zero exit
code: Error executing in Docker Container: 1
```

deny-write.profile

```
#include <tunables/global>

profile k8s-apparmor-example-deny-
write flags=(attach_disconnected) {
  #include <abstractions/base>

  file,

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

Pod Level – Pod Security Context

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    fsroup : 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

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 myResourceroup  \
--kubernetes-version 1.10.6
```

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

AKS Availability (Control Plane/Masters)

- AKS offers 99,5 SLO

- What happens if masters are down?
  - API is down, no new management operations (scale, deploy, etc…), but **applications are not impacted**

- How to improve?
  - Planned premium SKU in before the end of 2019
  - Multiple Clusters (less popular)

AKS Availability (Kubernetes Nodes)

- AKS follows the Azure VM availability SLA
  - 2+ Nodes Cluster in VMAS or VMSS(now default) 99,95
  - Cluster in Mutli-AZ 99,99

- How to achieve better availability?

Backup and Restore

- Goal: Restore cluster in case of failure
  - Stateless Cluster: Hooray! just re-deploy
  - Stateful Cluster: Redeploy and restore state
- Both scenarios require you to have the cluster deployment as code to make the re-deployment process faster
- Both scenarios require you to have your application deployed as part of a pipeline
- But what about the STATE?

Restore state

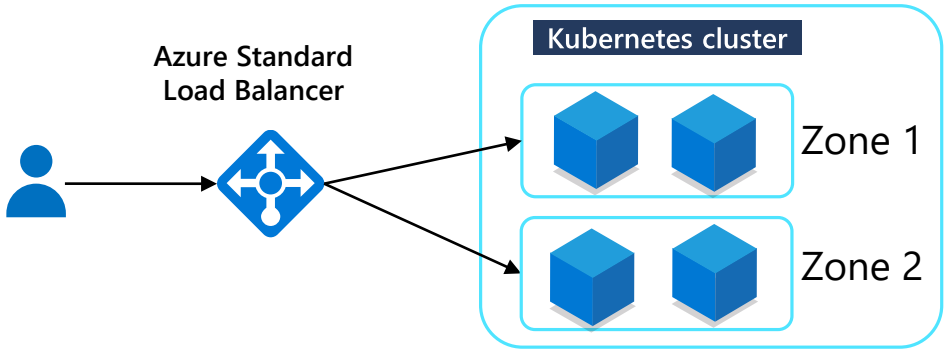
- Application Files
  - Restore from pipeline
  - Restore from a backup solution
- Application Data (Persistent Volumes – Azure Disks)
  - **Azure Snapshot API**
    - Not native to Kubernetes
  - **Heptio/VMware Velero**
    - Kubernetes native and its the recommended way to automate backups
- Note
  - There is no native API which supports copying disks snapshots to another region, only a PowerShell module https://docs.microsoft.com/en-us/azure/virtual-machines/scripts/virtual-machines-windows-powershell-sample-copy-snapshot-to-storage-account

Availability Zones

General Availability

Create an AKS cluster with nodes distributed across Availability Zones

- An AZ is a unique physical location within an Azure region
- Provide a higher level of availability to your applications (99,99%)
- Note that regular Azure Disks are tied to a Zone
- Limited to regions that support Zones (10 regions now)
- Requires Standard Load Balancer SKU (Basic SKU does not support Zones)

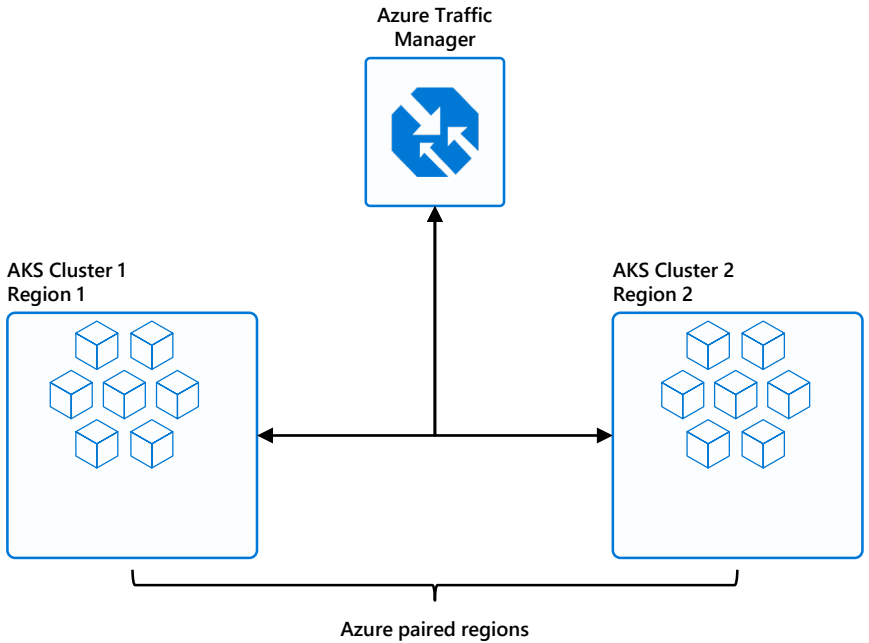


Takeaways

- Availability is a complex topic!
- Everything as code and Backup Strategy is where you start and the most important
- Backups should be shipped to different region(s)
- Multi-AZ and Off-region Backups should be enough for most of your workloads
- Watch out for the Cross-AZ Traffic Charges
- If you dont have Azs and you need >99,95, then Multi-Region. challenge is always the STATE

Multi-Region Clusters

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

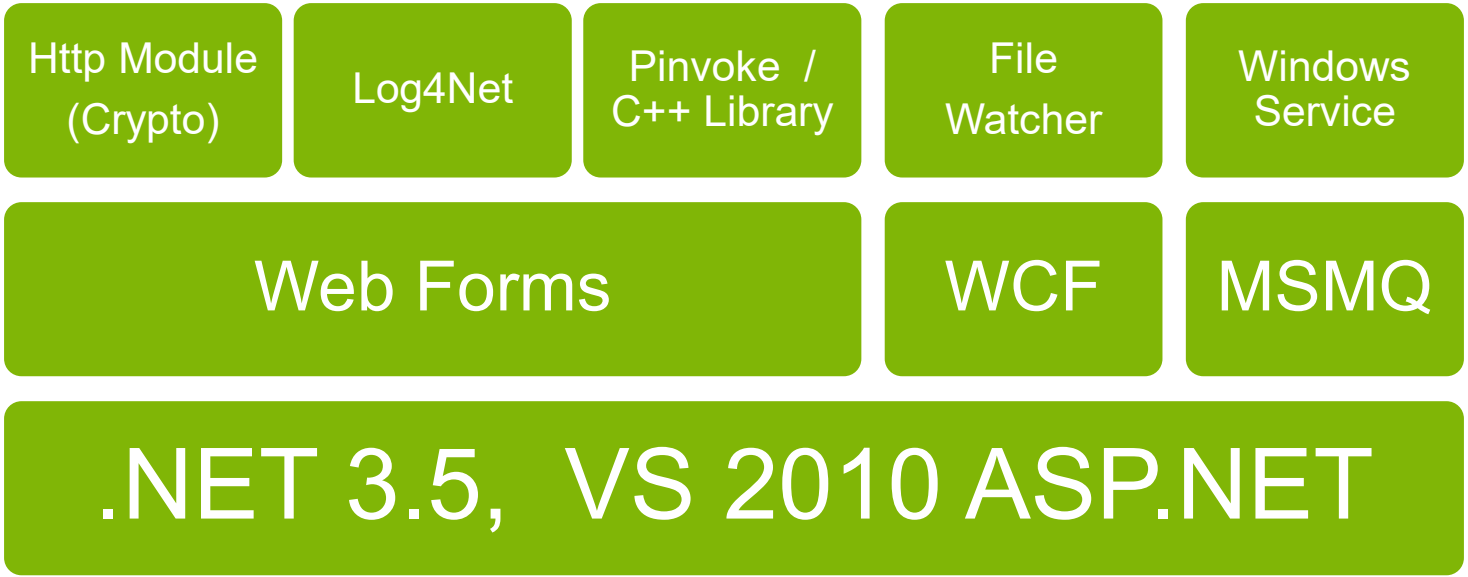


Rehosting legacy apps with Kubernetes

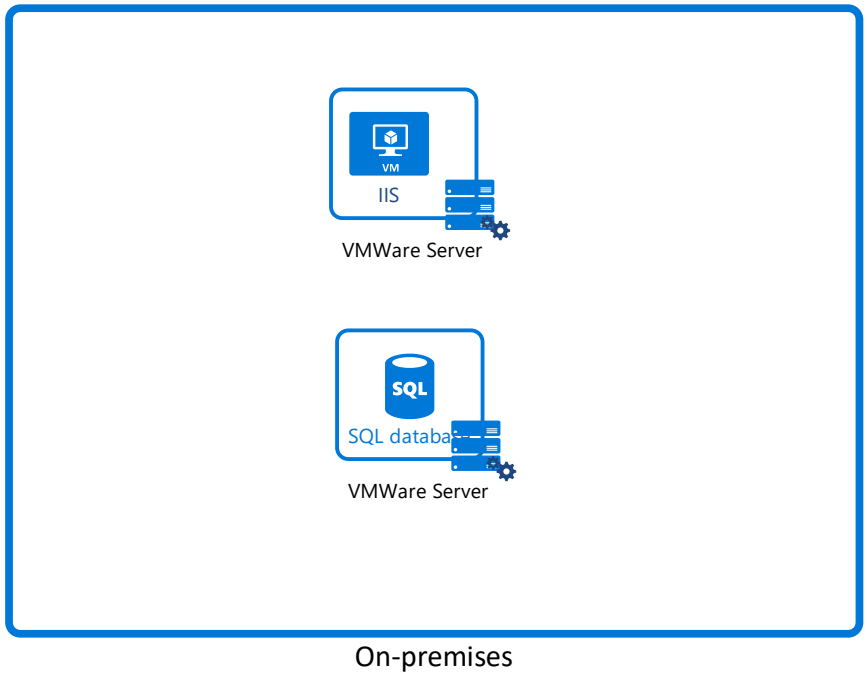
Comparing Migration Approaches

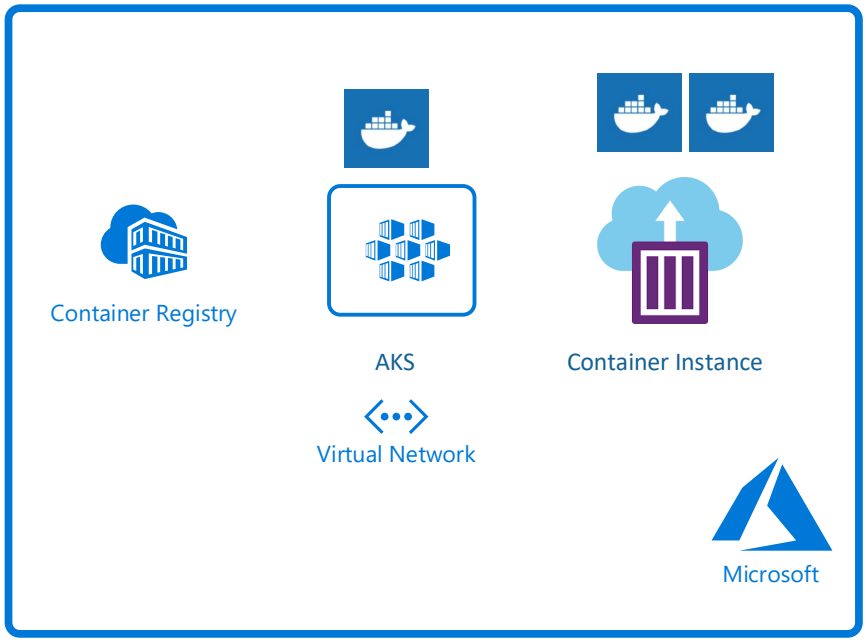
Approach	Code Changes	Operational Costs	Cloud Costs	Leverage Native Cloud Services	Scalability	DevOps Maturity	Code structure	Time to value (end state)	SRE Maturity
Rehost (lift-n-shift)	None	High (Unchanged from on-premises)	High	None	Vertical	Low	Unchanged	High	Low
Replatform (lift-n-reshape)	Low	Medium	Medium	Low Orchestrator / Compute	Horizontal and vertical scaling	Medium-High	Unchanged	Medium	Medium-High
Refactor	Medium	Medium	Low	Medium PaaS Services	Horizontal and vertical	Medium-High	Materially changed	Moderate	Medium-High
Reimagine	High	Low	Minimal Dynamic – usage based	Maximum	Horizontal	High	Rewrite	Low	High

Reference App “Legacy”



Architecture Before and After





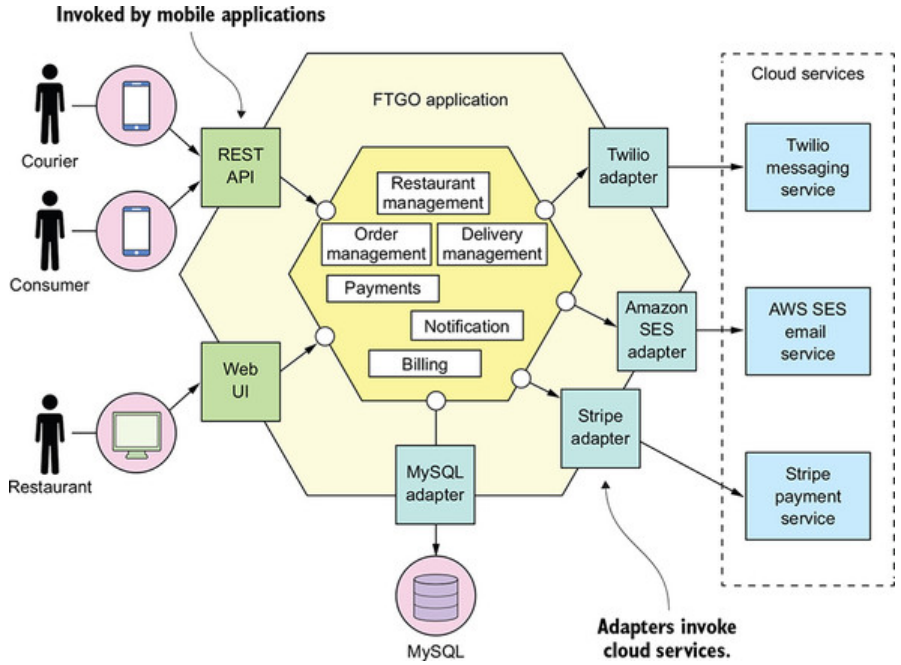
Steps to Modernize with Kubernetes

- App Assessment
- Dependency Analysis
- Containerize
- Hosting the image in an orchestrator
- Fault tolerance and scaling
- Continuous Modernization

Challenges with monolithic apps

- Complexity
- Development is slow
- Path to deployment is slow
- Scaling is difficult
- Lack of fault isolation

Monolithic App

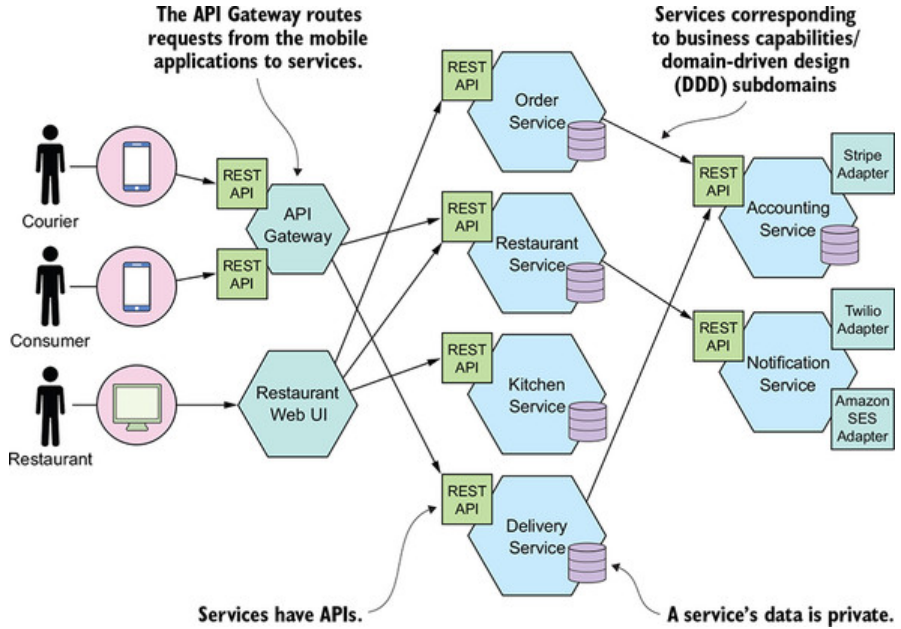


Source: Microservices Patterns With examples in Java

Microservice architecture to the rescue

- Better testability and deploy-ability allows teams to be autonomous
- Small and easy to manage
- Services are independently scalable
- Better fault isolation
- Experiment with new technology

Monolithic app reimagined with microservices



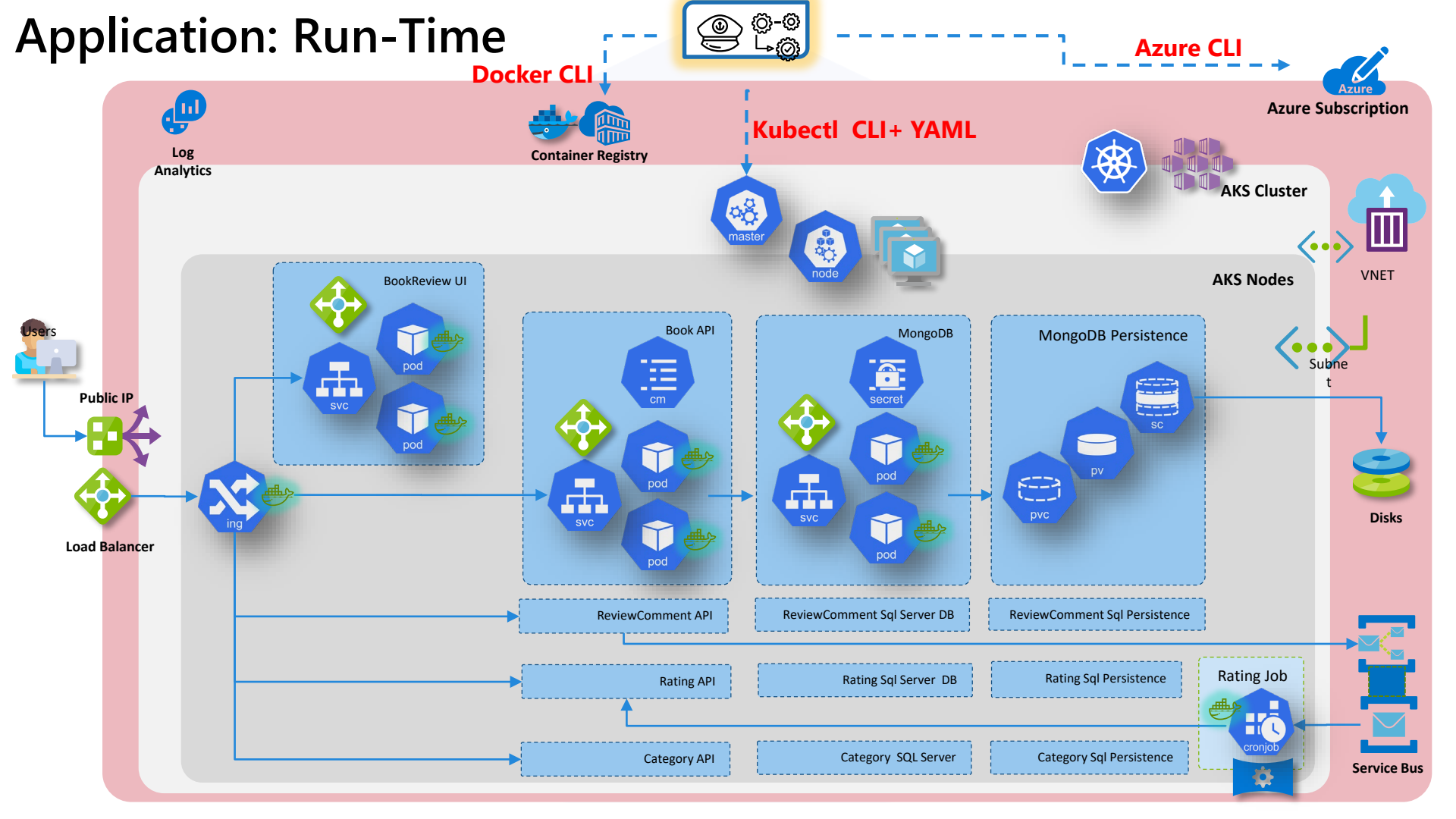
Source: Microservices Patterns With examples in Java

Decomposition Strategies

- Decomposing an application into services
- Defining systems of operations Decompose Business Capability
- Domain Driven Design
- Subdomain - principle
- Single responsibility
- Common Closure principal

Cloud-Native Development with AKS and Cosmos DB

Application: Run-Time



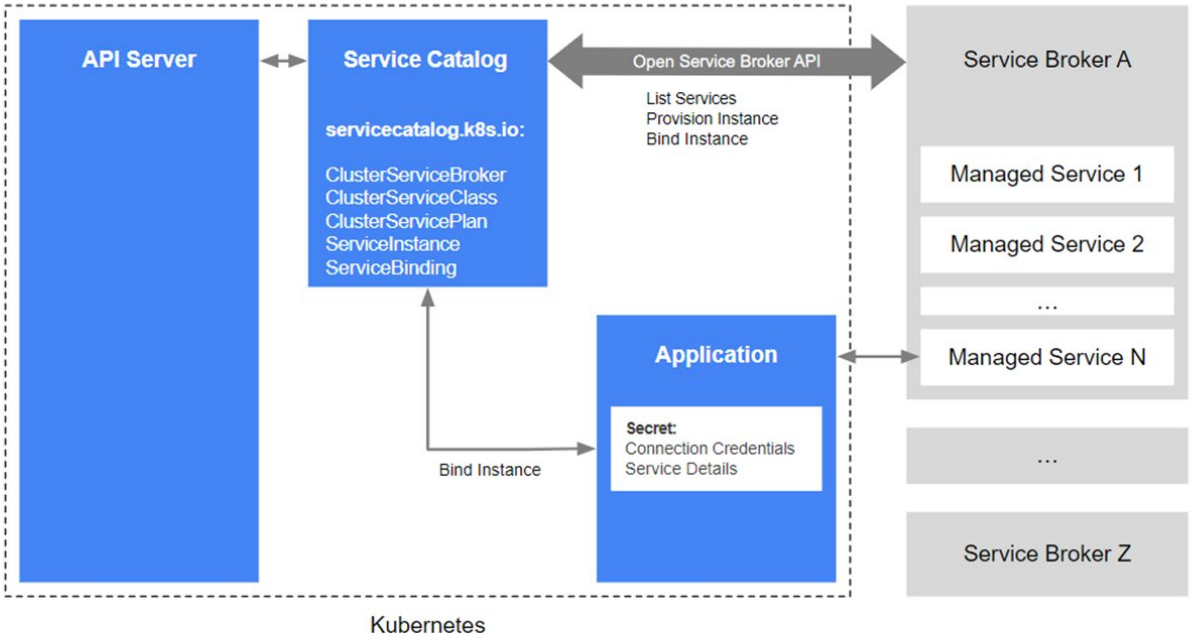
Challenges with Managing the State

- Many recent improvements for stateful apps in Kubernetes
  - Stateful Sets, Container Storage Interface (CSI)
  - Persistent Volumes(PV), Persistent Volume Claims (PVC), Storage Classes
  - Operators
- However, the support is not on par with managed services
  - Hard to manage clusters with (stateless and stateful nodes)
  - Geo distribution
  - Elastic scale out (storage and throughput)
  - Performance guarantees (availability, read, write)

Open Service Broker

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

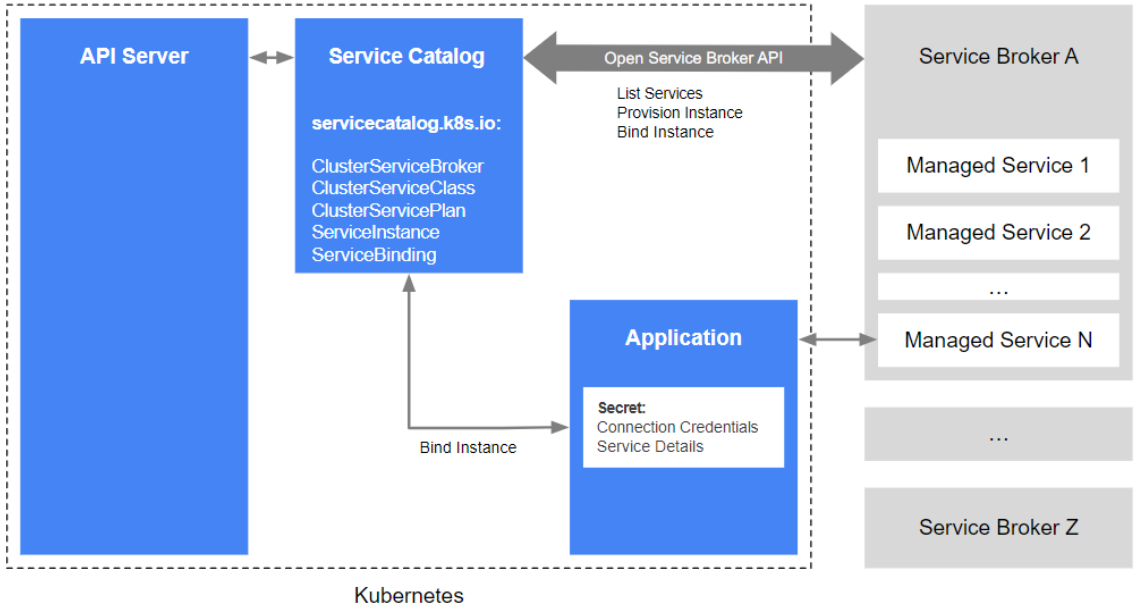
Kubernetes Service Catalog



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

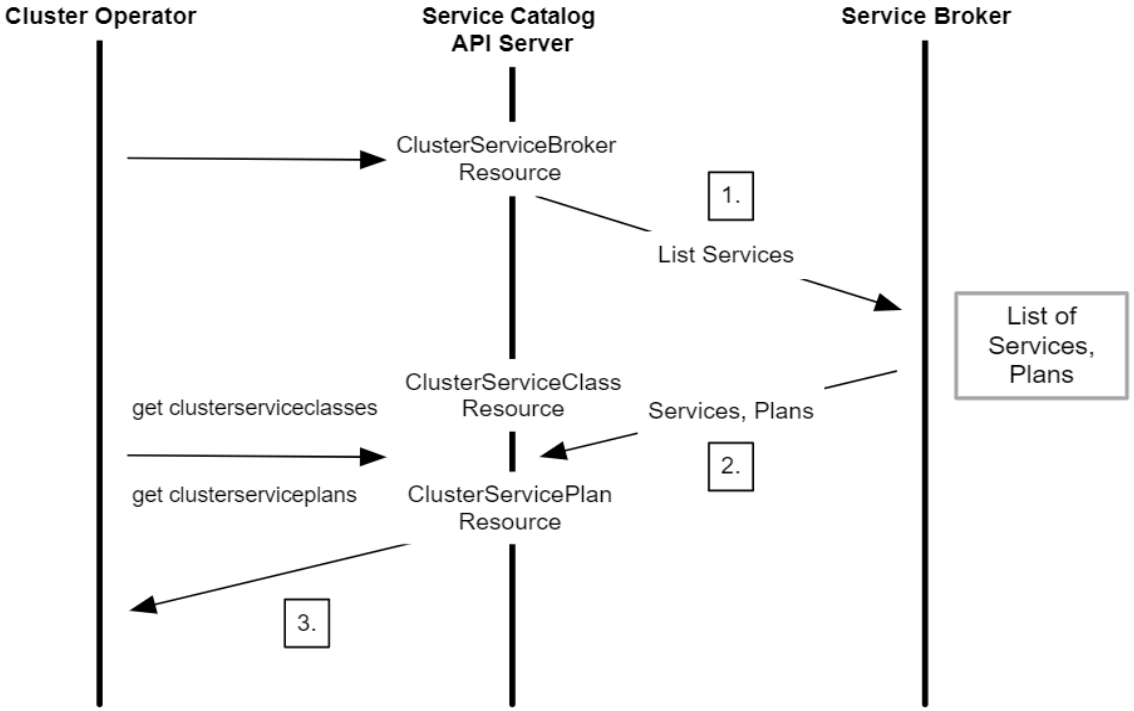
Kubernetes Service Catalog



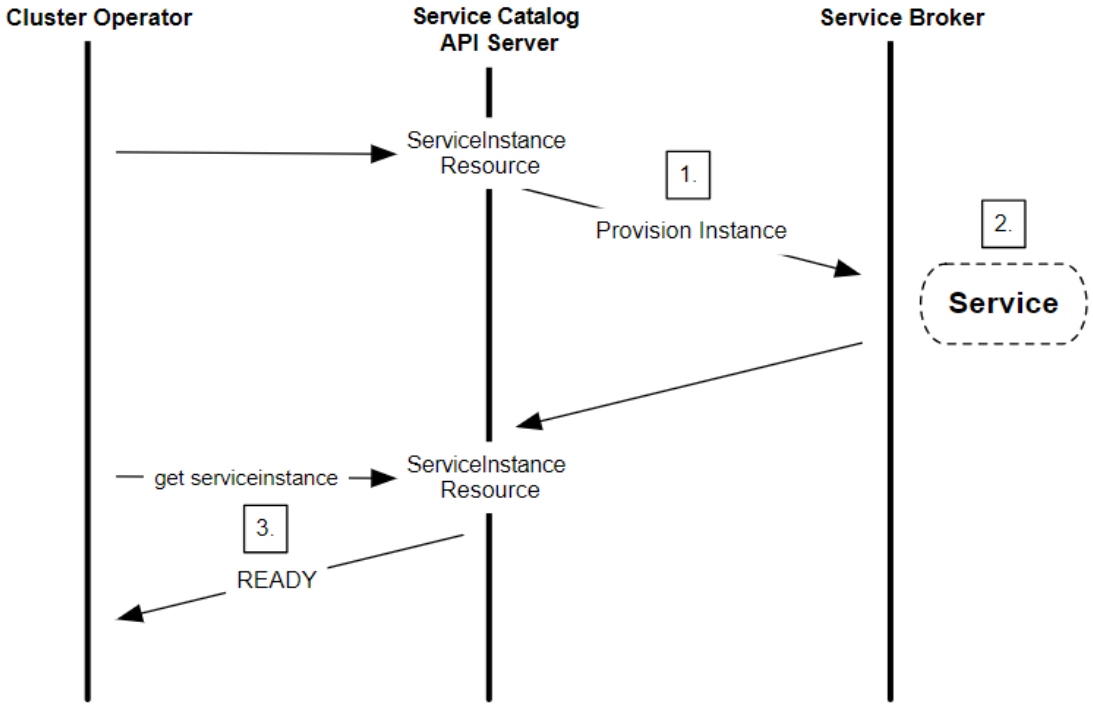
Kubernetes Service Catalog

- 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.

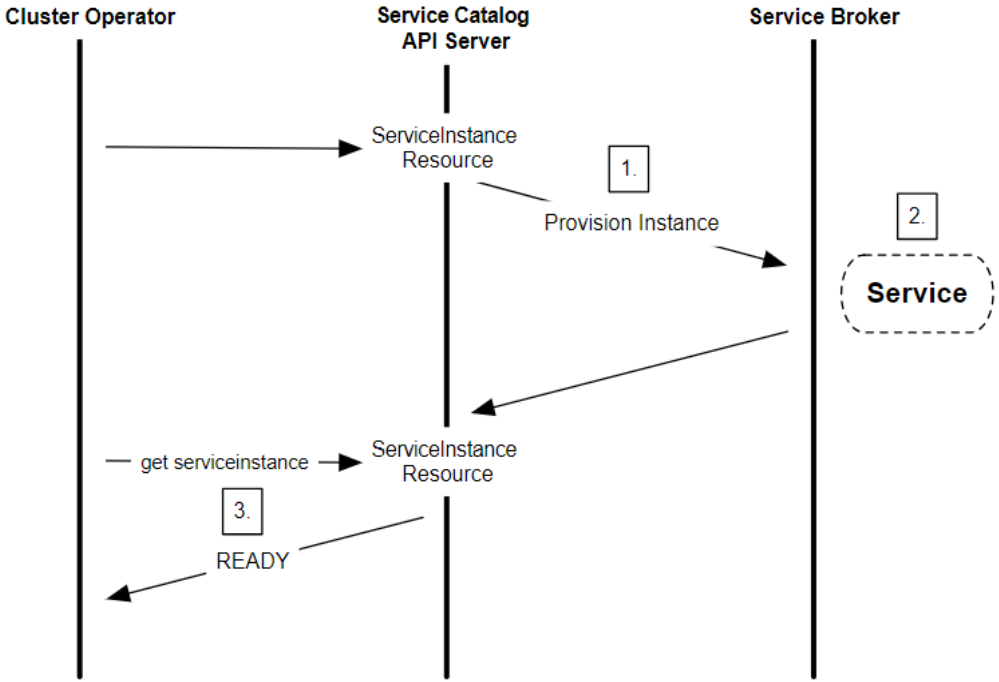
Listing Managed Service Plans



Provisioning a new Instance



Binding to a managed service

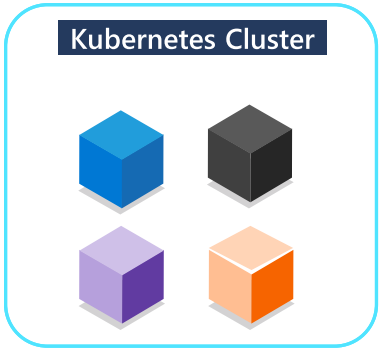


Multiple node pools

General Availability

Mix multiple VM types and configurations in the same cluster

Upgrade and scale each node pool independently



NodePool considerations

- AKS API operations will be decoupled for control plane and node pools.
- Most operations are now at node pool level (scale, upgrade,…)
- You can add taints to the node pool profile that will automatically add them to every new node
- Cluster AutoScaler works on a per node pool basis
- An AKS cluster can have a maximum of 8 node pools
- An AKS cluster can have a maximum of 400 nodes across those node pools
- You can leverage the Public IP per Node feature in selected node pools

Customer responsibilities (Nodes)

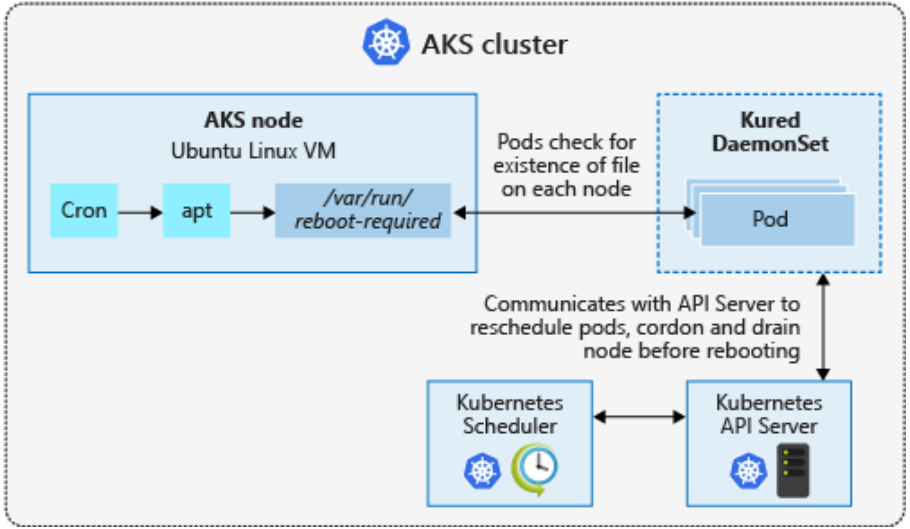
- Reboot nodes for operating system security patches

- Kubernetes version upgrades

- Operating system packages will be applied automatically with the new K8s version

Process Node OS Security Patches

- AKS automatically applies security patches to the nodes on a nightly schedule
- You're responsible to reboot as required
- Regular maintenance recommended
  - Monthly ideal, 3 months maximum
- Use Kured DaemonSet to watch for nodes requiring a reboot and handle the reschedule of the pods and reboot of the nodes. Check AKS documentation for guidance to use Kured



Blue/Green upgrade – Node pools

- There is a new version available
  $ az aks get-upgrades -n k8s-ignite -g k8s
  "kubernetesVersion": "1.15.4"….
- Upgrade the control plane
  $ az aks upgrade --control-plane-only -n k8s-ignite -g k8s –k 1.15.4
- Add a new node pool with the new version
  $ az aks nodepool add -n new-node-pool --cluster-name k8s-ignite -g k8s –k 1.15.4 –c 3
- Taint your new nodes
  $ kubectl taint nodes –l agentpool=new-node-pool nodepool=newNodePool:NoSchedule
- Deploy a test version of your app and do your testing
  $ kubectl apply -f my-test-workload.yaml
- All goes well, deploy a new version of your app with a new end point or Cordon-and-Drain the existing node pool or use Labels

Blue/Green upgrade - New Cluster

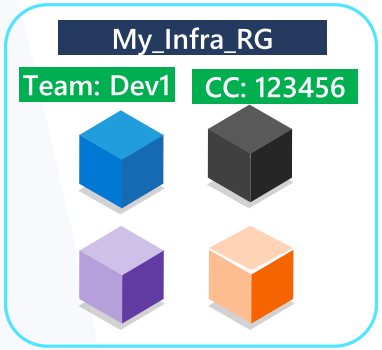
1. There is a new version available
   $ az aks get-upgrades -n k8s-ignite -g k8s
   "kubernetesVersion": "1.15.4”….
2. Create a new cluster
   $ az aks create -g k8s -n k8s-ignite-15.3 --kubernetes-version 1.15.4
3. Deploy a test version of your app and do your testing
   $ kubectl apply -f my-test-workload.yaml
4. All goes well, deploy a new version of your app with a new end point, then switch your DNS records

Upgrade strategies

	AKS Upgrade API	Blue/Green (New Node Pool)	Blue/Green (New Cluster)
How	AKS Upgrade API	New Node Pool Taints and Tolerations	Automation + GLB (Traffic Manager)
Upgrade Duration	Depends on Cluster Size The Larger the cluster the longer the upgrade	Shortest	Short assuming full automation is in place
Risk	Subject to Failure	Safe - with the risk that the control plane upgrade will fail*** No impact on applications	Safest
When to use	Small Production Cluster Non-Mission Critical Workloads Non-Prod Environments	Large Clusters Mission Critical Workloads **assuming the Control Plane failure risk is accepted	Large Clusters Mission Critical Workloads

Tag Passthrough

You can now pass tags to the AKS resource and these will Passthrough to the Infrastructure resource group (MC_)



Custom Infrastructure Resource Group Naming

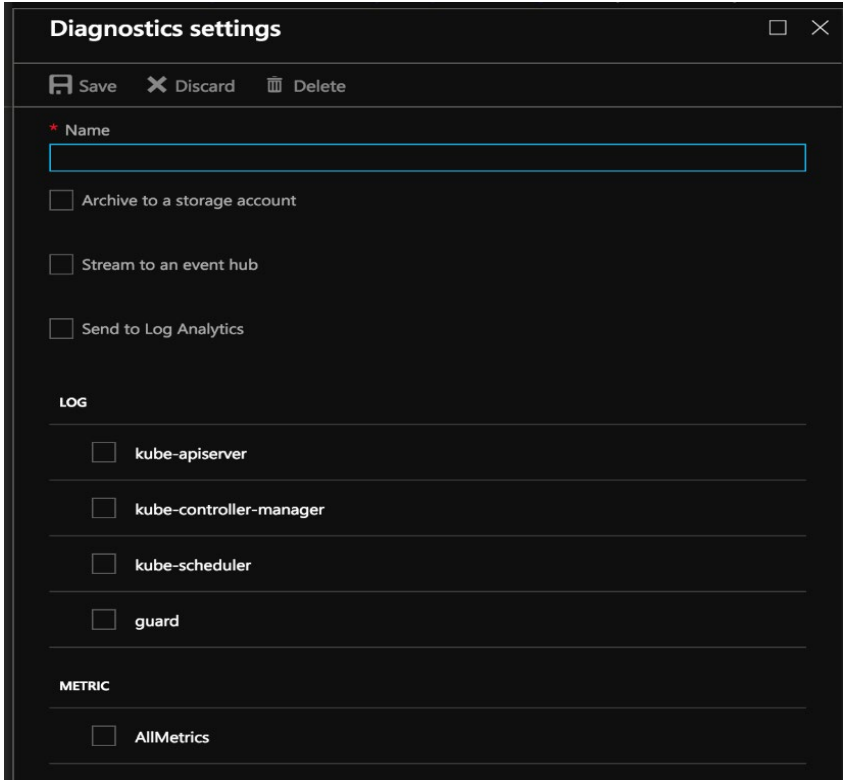
You can now pass a custom name to the Infrastructure RG (MC_)

Source: Microservices Patterns With examples in Java

Enable 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 (new)
  - cluster-autosclaer



Takeaways

- Use Standard Load Balancer
- Use Nodepools!
- Maintain N-1 for minor releases for production workloads
- 3 months upgrade cycle for minor versions is recommended
- Always wait for the 2nd or 3rd patch version
- Automate nodes reboot for security patches using Kured
- Blue/green cluster upgrades for your mission critical workloads

Audience Participation

Networking -
AKS cluster details –
Helm –

Logging and Monitoring --
Network Policy –

Service Catalog –
Security – Hardening

Windows Containers
ACI
Scaling
Patterns

Dev Spaces

BUILD 2019 AKS and COSMOS

Microsoft Build 2019 Session: Architecting Cloud-Native Apps with AKS and Cosmos DB

Migrate and Modernize with Kubernetes on Azure Government

Migrate and Modernize with Kubernetes on Azure Government