Kubernetes

An Introduction

Kubernetes Overview

What Does “Kubernetes” Mean?

Greek for “pilot” or

“Helmsman of a ship”

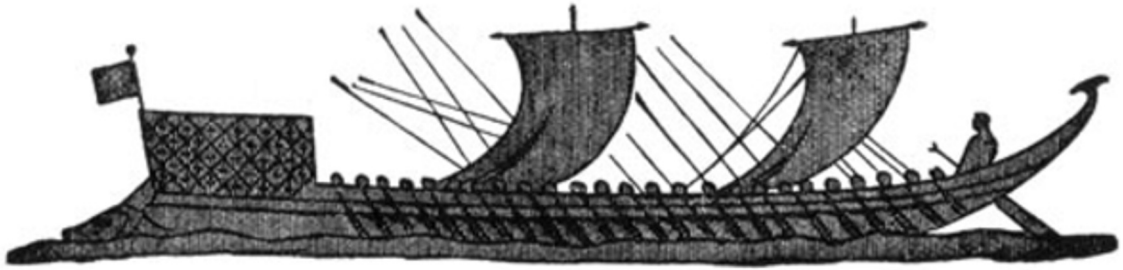


Image Source

What is Kubernetes?

- Project that was spun out of Google as an open source container orchestration platform.
- Built from the lessons learned in the experiences of developing and running Google's Borg and Omega.
- Designed from the ground-up as a **loosely coupled** collection of components centered around deploying, maintaining and scaling workloads.

What Does Kubernetes do?

- Known as the **linux kernel of distributed systems**.
- **Abstracts away the underlying hardware** of the nodes and provides a uniform interface for workloads to be both deployed and consume the shared pool of resources.

Decouples Infrastructure and Scaling

- **All services** 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.

Self Healing

Kubernetes will **ALWAYS** try and steer the cluster to its desired state.

- **Me:** “I want 3 healthy instances of selenium chrome nodes 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.”

What can Kubernetes REALLY do?

- Autoscale Workloads
- 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

Most Importantly...

Use the **SAME** API
across bare metal and
EVERY cloud provider!!!

Who “Manages” Kubernetes?

CLOUD NATIVE
COMPUTING FOUNDATION

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

Project Stats

- Over 99,000 stars on GitHub
- 3450+ Contributors to K8s Core
- Most discussed Repository by a large margin

Where Can we use K8s in Automation?

Selenium Grid

- The Infrastructure to run our automated tests against different browsers, versions etc.
- Need to scale up on high demand and scale down upon test completion

Can’t Docker Compose do this?

- Ephemeral
- Multiple : Org wide Scalable Grid
- High Availability ?
- Service Discovery & Load Balancing?

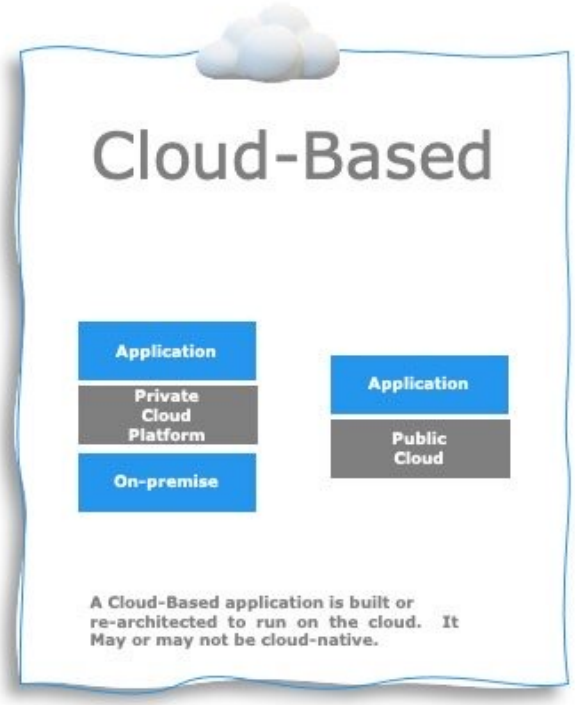
Continuous Testing

Costlier to Scale in Cloud

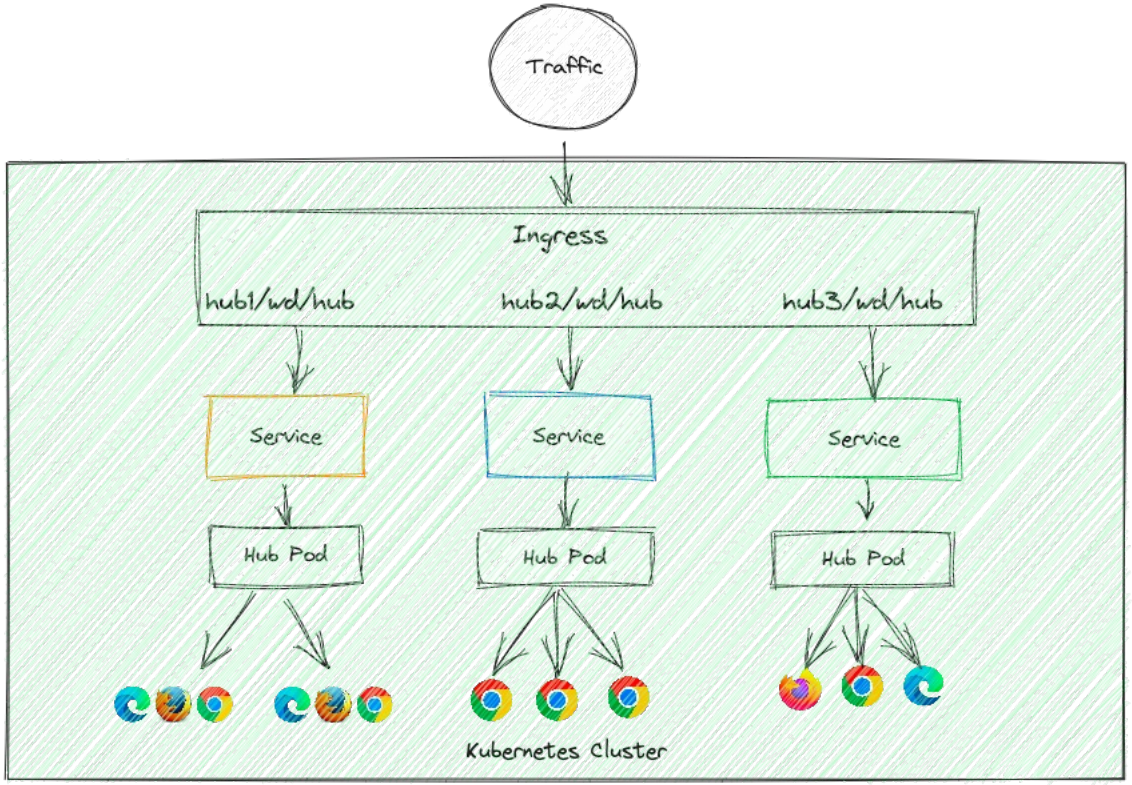
Futuristic Micro-Frontends

- Create & Destroy : Branch based Selenium Grid ?
- Dynamic Grid for every test run

Cloud Agnostic



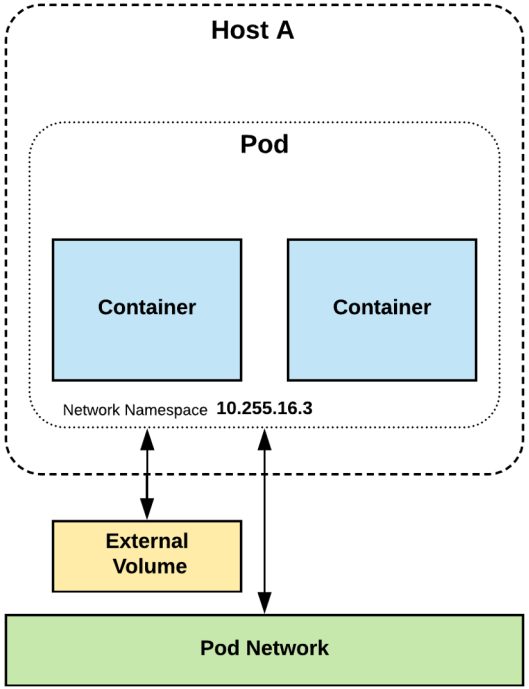
Kubernetes Cluster



A Couple Key Concepts...

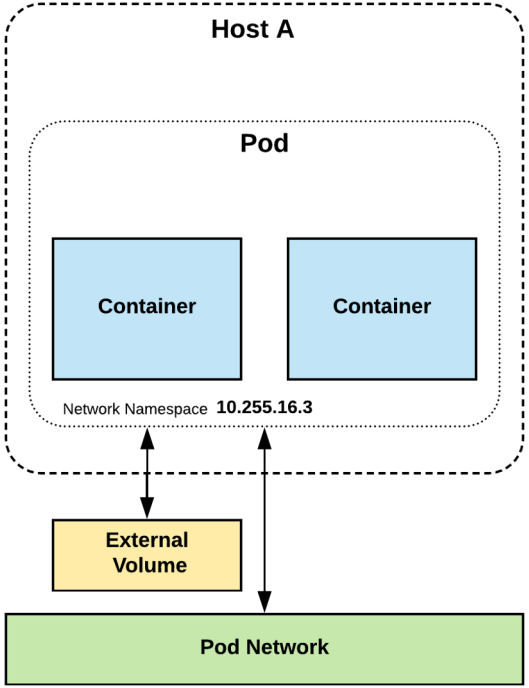
Pods

- **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**.



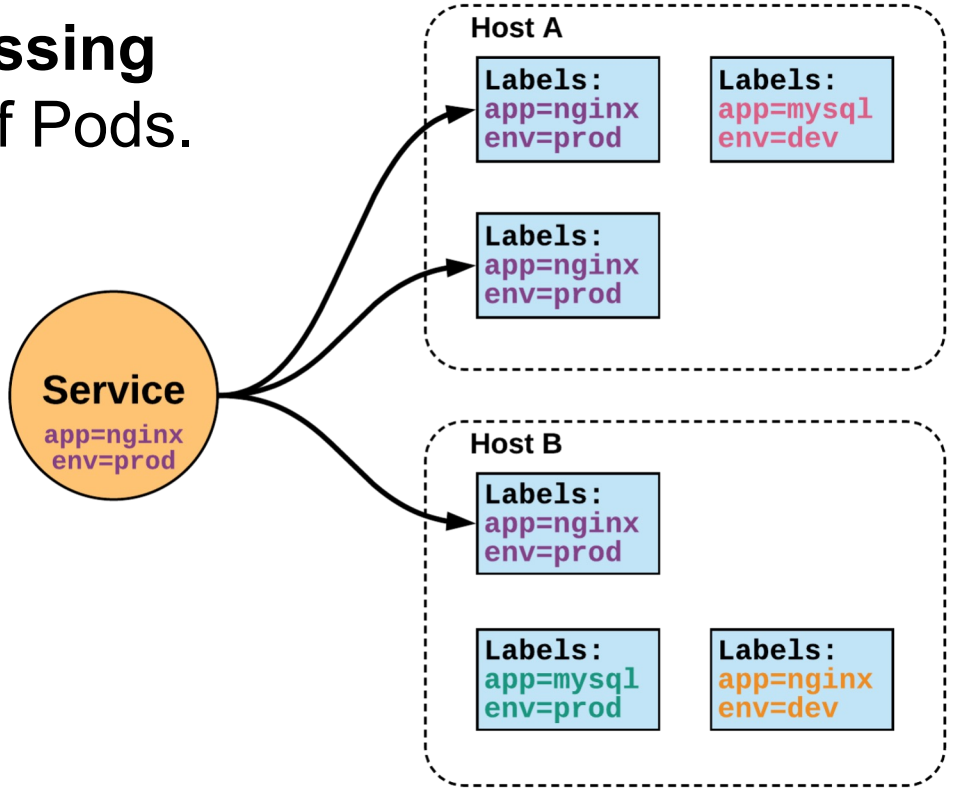
Pods

They are also Ephemeral!



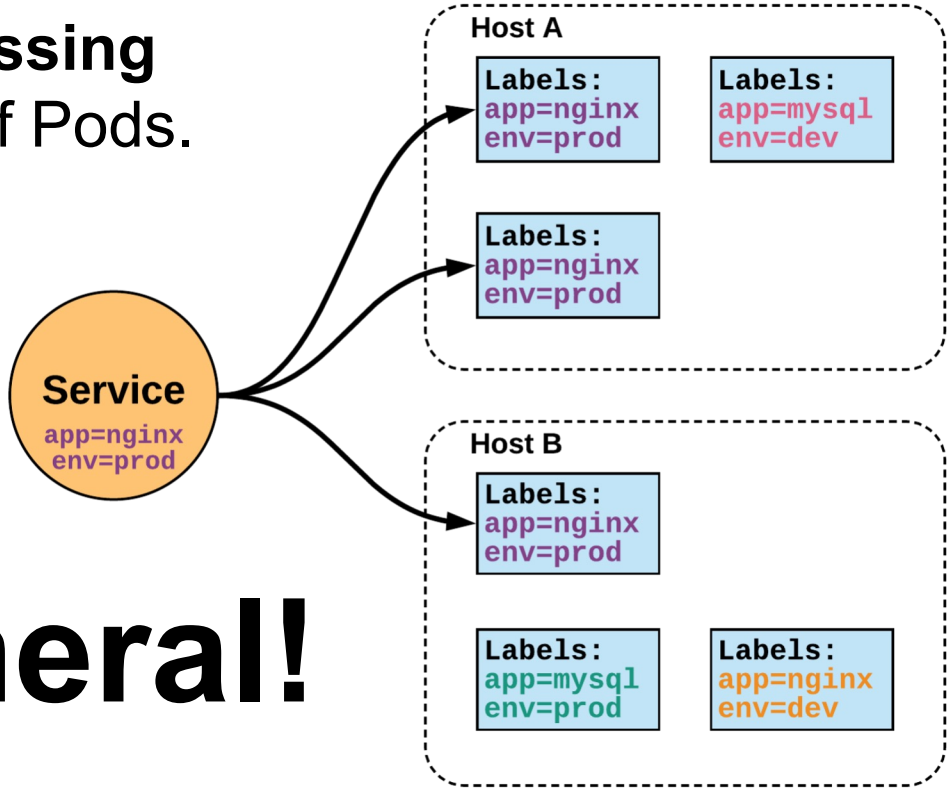
Services

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



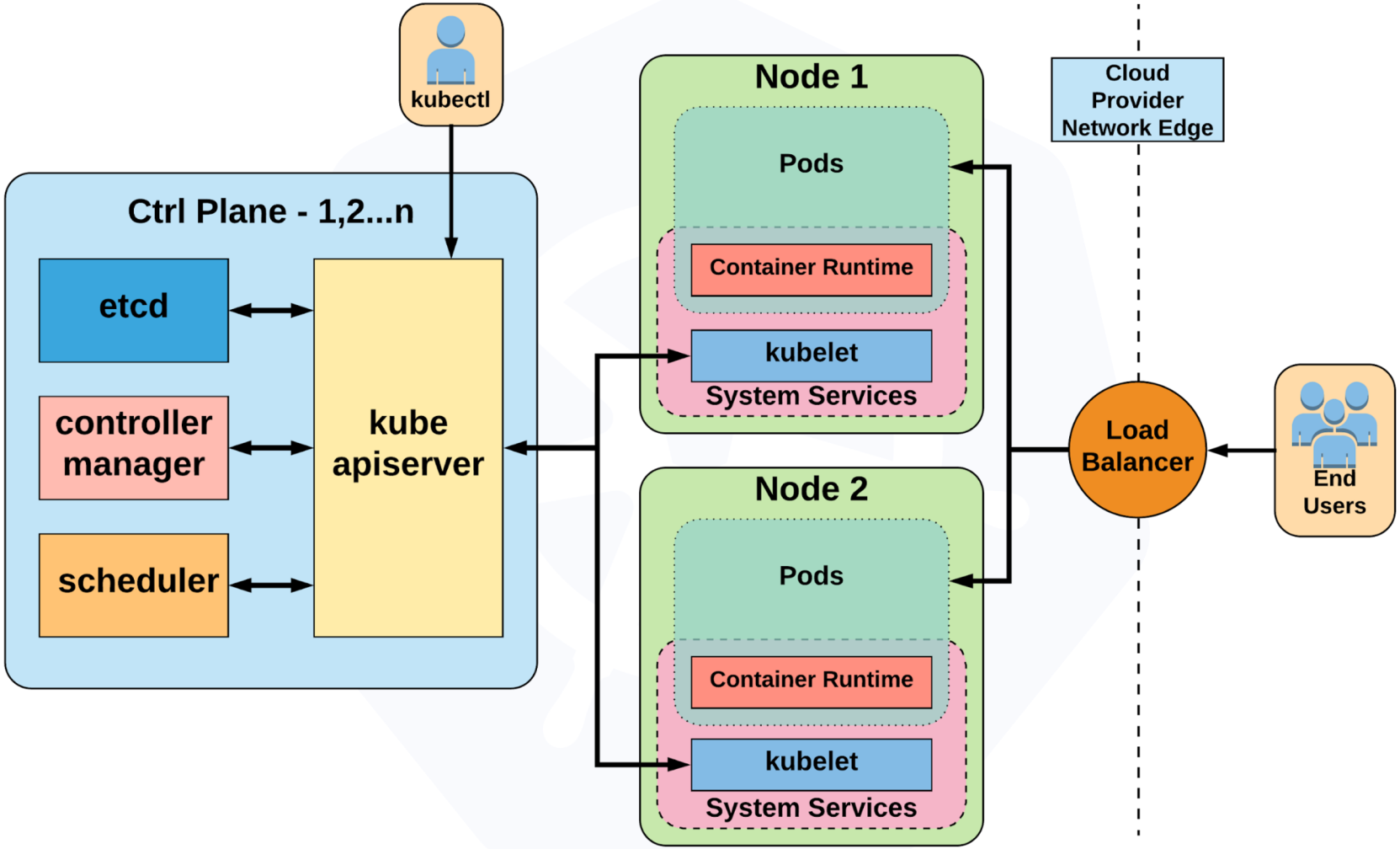
Services

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



NOT Ephemeral!

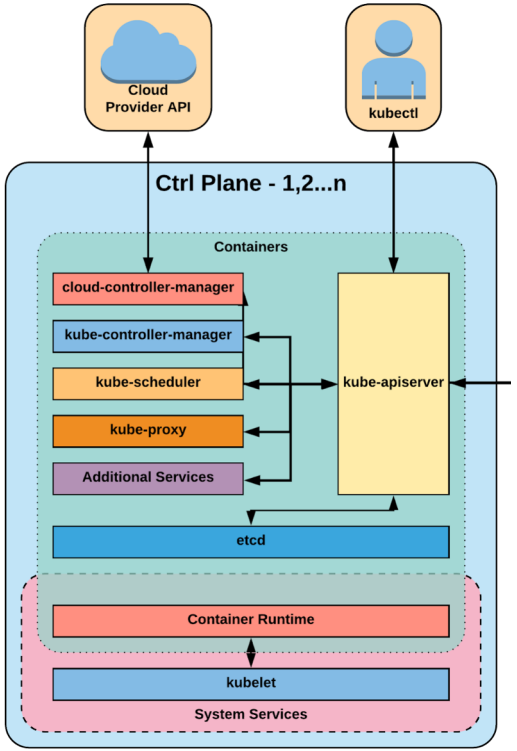
Architecture Overview


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.

etcd

Uses *“Raft Consensus”* among a quorum of systems to create a fault-tolerant consistent *“view”* of the cluster.

https://raft.github.io/

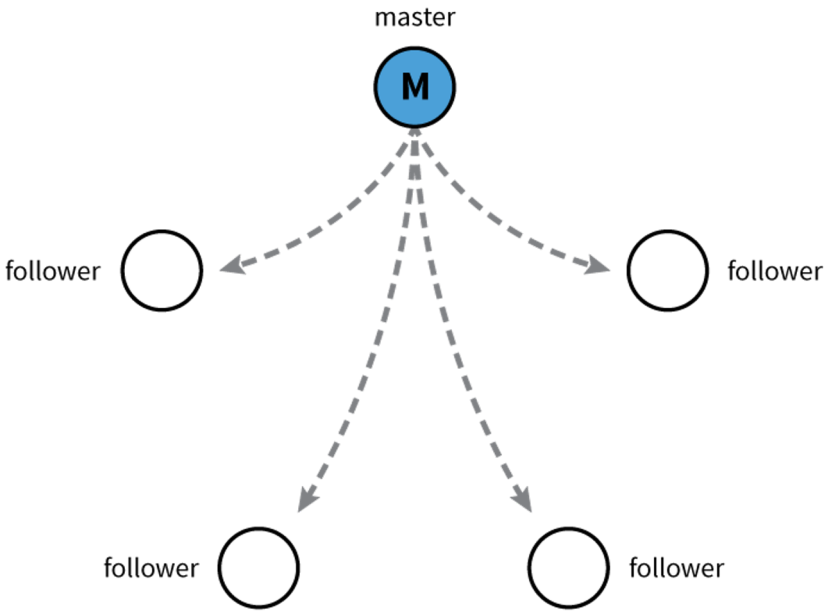


Image Source

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#L344

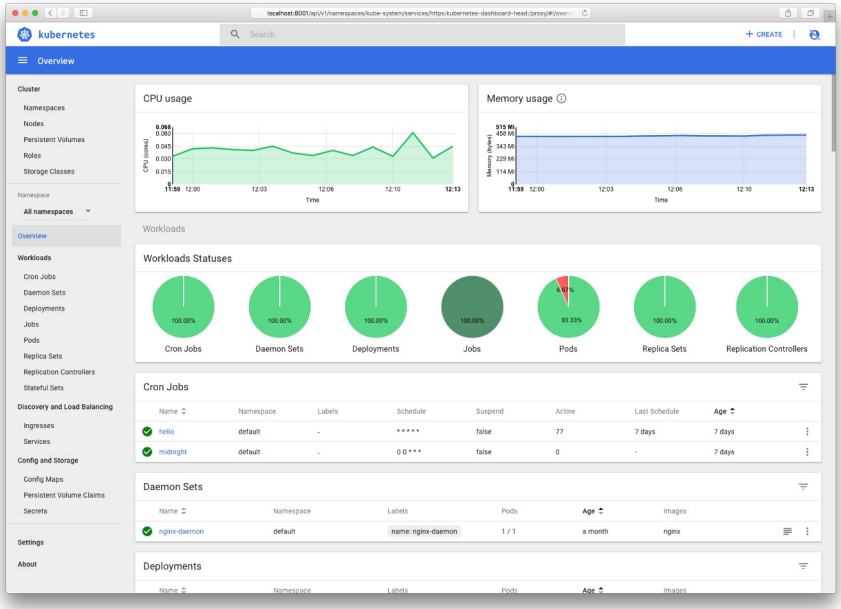
kube-scheduler

- 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.

Optional Services

Kube Dashboard

A limited, general purpose web front end for the Kubernetes Cluster.



Heapster / Metrics API Server

- Provides metrics for use with other Kubernetes Components.
  - Heapster (deprecated, removed last Dec)
  - Metrics API (current)

Concepts and Resources

The API and Object Model

Concepts and Resources

API Overview

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

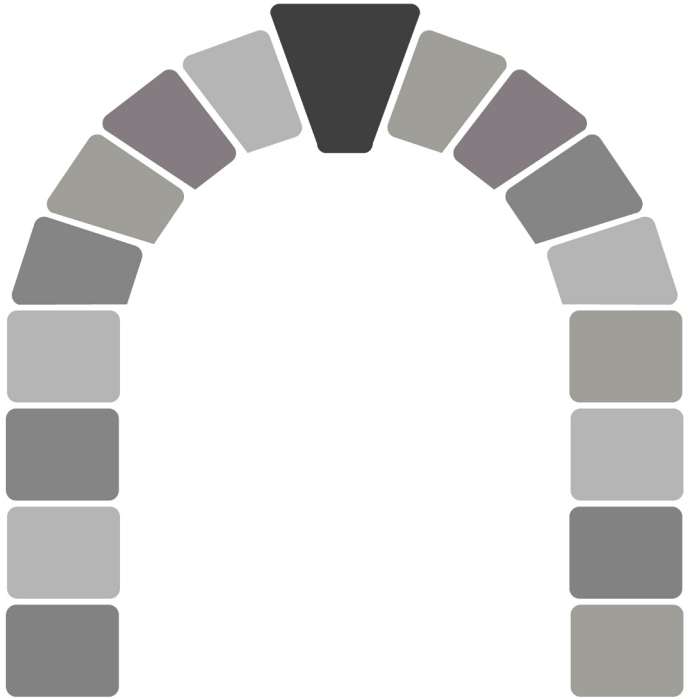
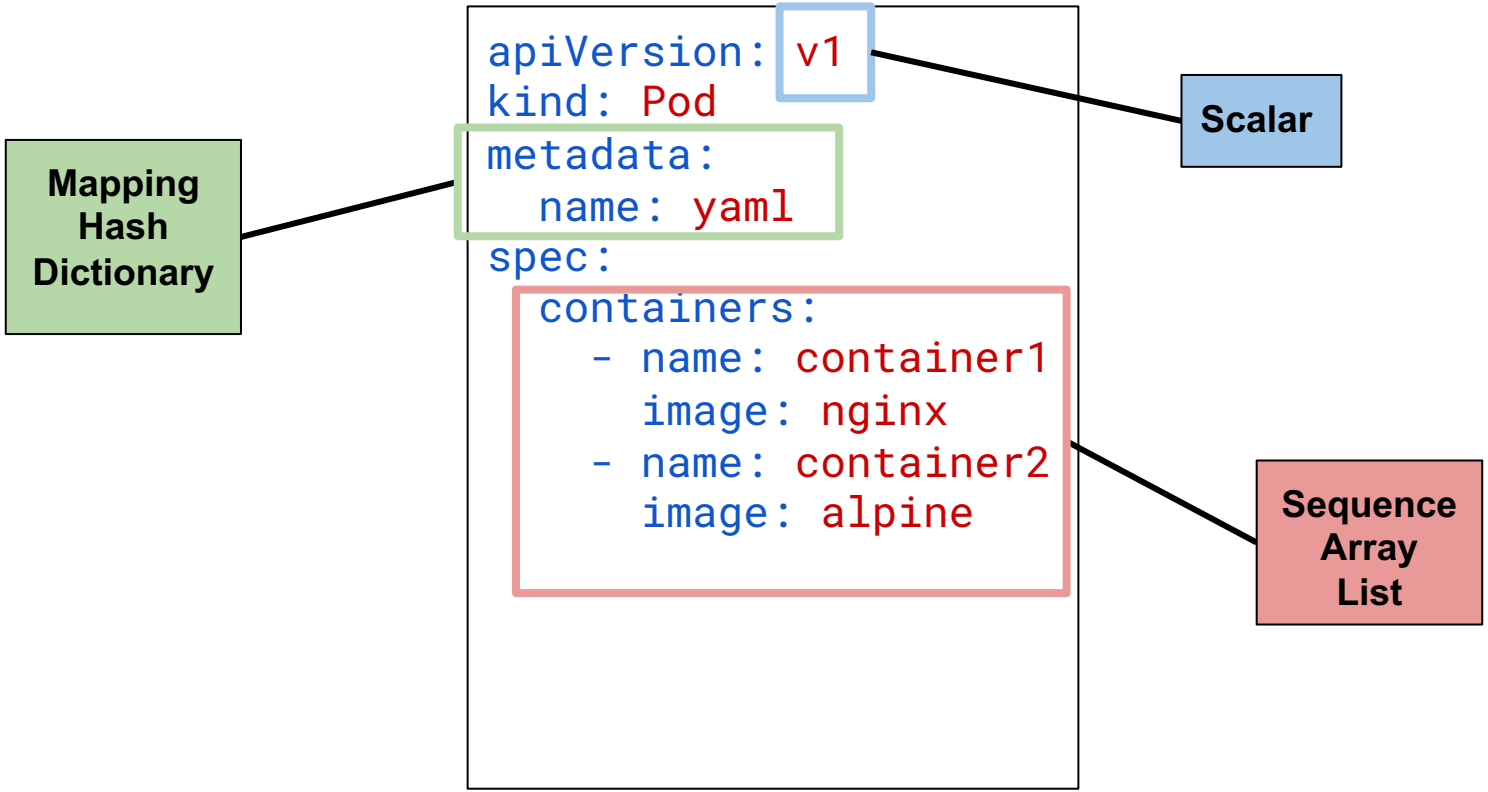


Image Source

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
```

Lab - github.com/mrbobbytables/k8s-intro-tutorials/blob/master/cli

Using the API

(aka, using the CLI)

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.

```yaml
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
```

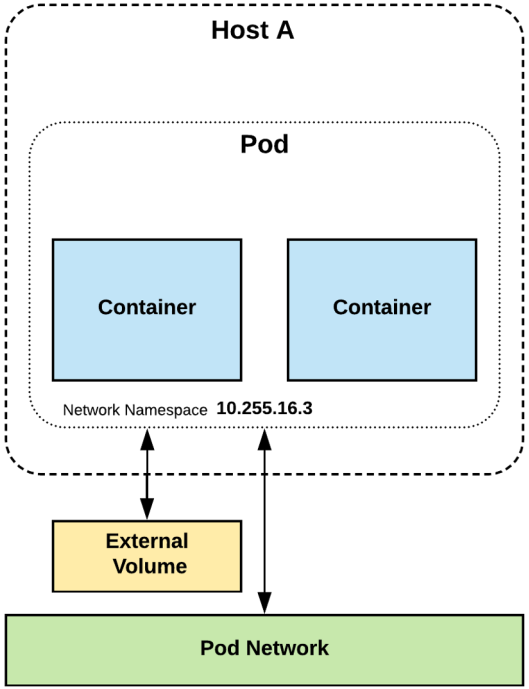
Default Namespaces

- **`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
    ports:
    - containerPort: 80
    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

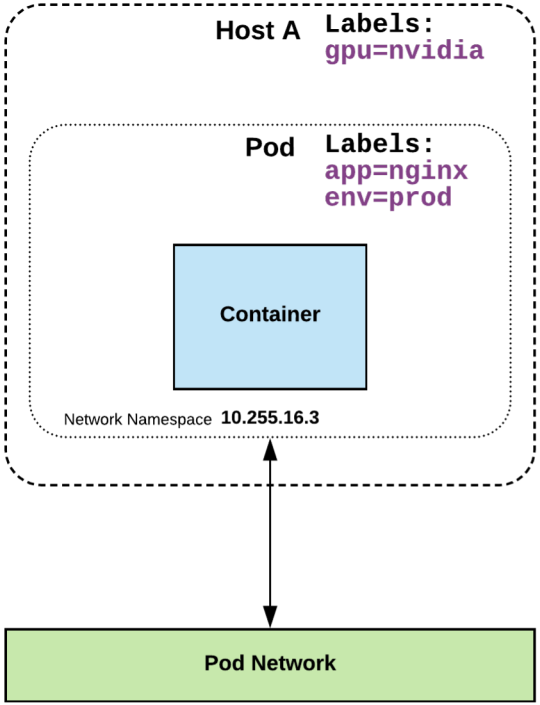
- `name` - The name of the container
- `image` - The container image
- `ports` - array of ports to expose. Can be granted a friendly name and protocol may be specified
- `env` - array of environment variables
- `command` – Entrypoint array (equiv to Docker `ENTRYPOINT`)
- `args` - Arguments to pass to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

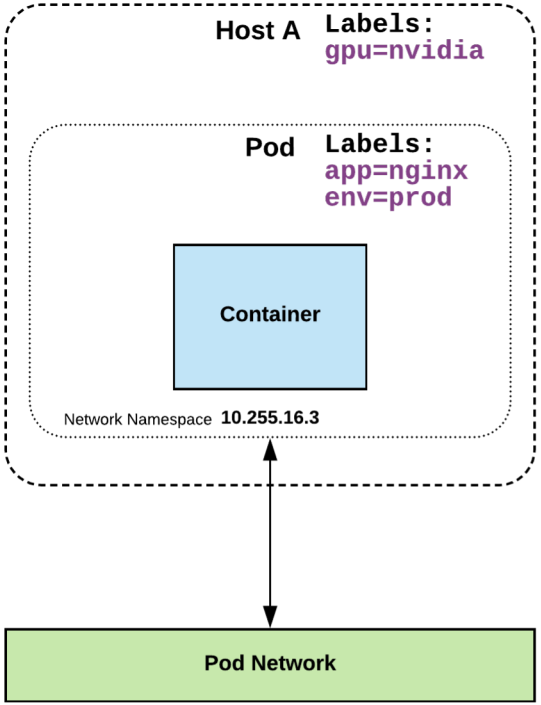
- key-value pairs that are used to identify, describe and group together related sets of objects or resources.
- **NOT** characteristic of uniqueness.
- Have a strict syntax with a slightly limited character set*.



* 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
```



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
```

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.

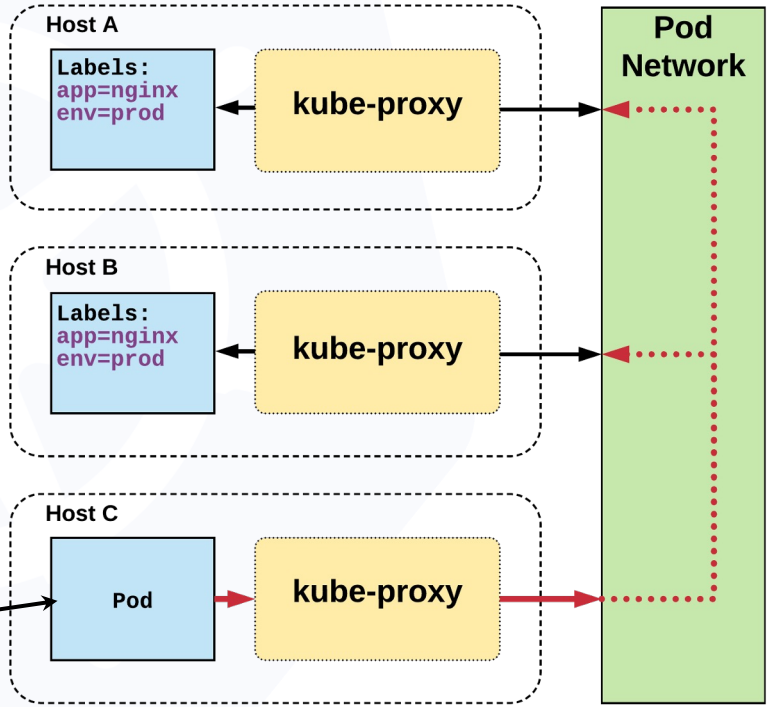
```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  selector:
    app: nginx
    env: prod
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

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
```

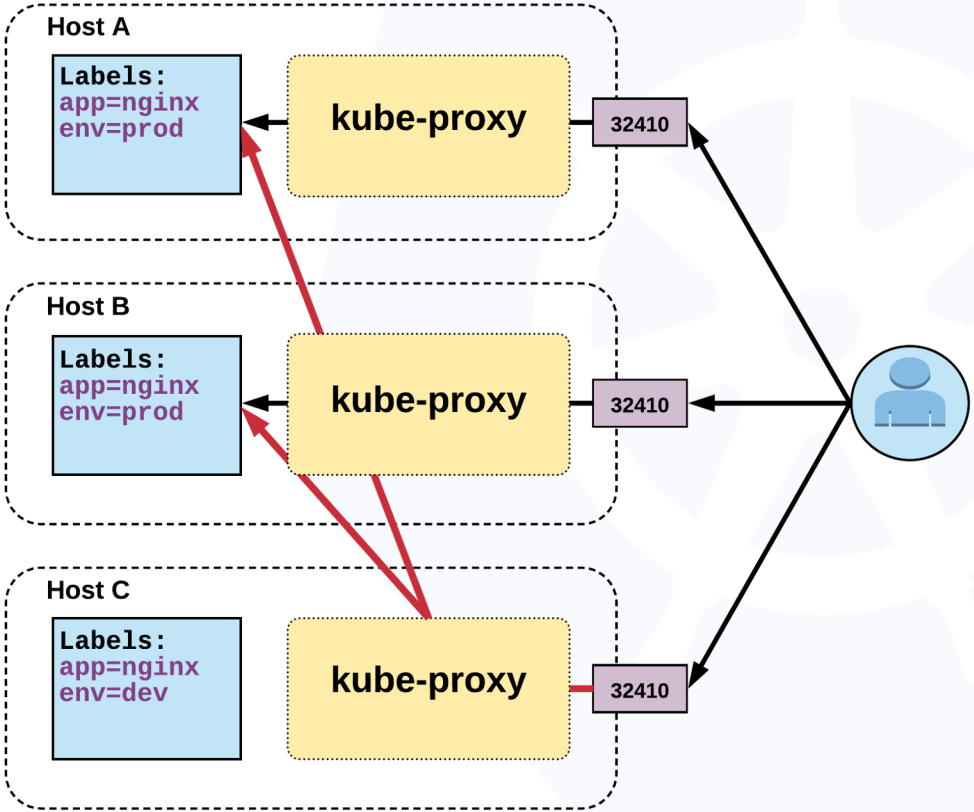


NodePort Service

- **NodePort** services extend the **ClusterIP** service.
- Exposes a port on every node's IP.
- Port can either be statically defined, or dynamically taken from a range between 30000-32767.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: NodePort
  selector:
    app: nginx
    env: prod
  ports:
  - nodePort: 32410
    protocol: TCP
    port: 80
    targetPort: 80
```

NodePort Service



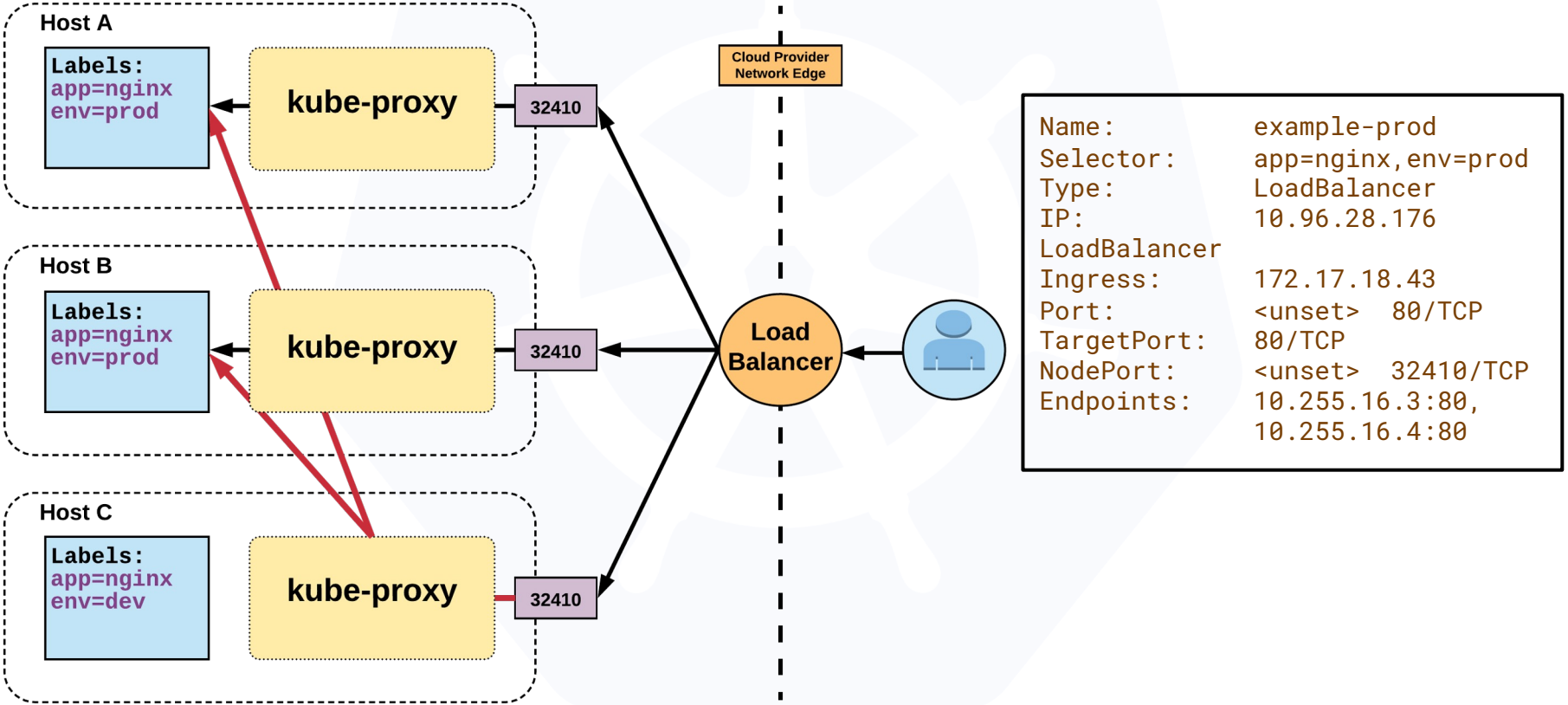

```
Name:           example-prod
Selector:       app=nginx,env=prod
Type:           NodePort
IP:             10.96.28.176
Port:           <unset>  80/TCP
TargetPort:     80/TCP
NodePort:       <unset>  32410/TCP
Endpoints:      10.255.16.3:80,
                10.255.16.4:80
```

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



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
```

Lab - github.com/mrbobbytables/k8s-intro-tutorials/blob/master/core

Exploring the Core

Workloads

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

Lab - github.com/mrbobbytables/k8s-intro-tutorials/blob/master/workloads

Using Workloads

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 `Recreate` or `RollingUpdate`.
  - `Recreate:` All existing Pods are killed before the new ones are created.
  - `RollingUpdate:` Cycles through updating the Pods according to the parameters: `maxSurge` and `maxUnavailable`.

```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>
```

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>
```

CronJob

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-example
spec:
  schedule: "*/1 * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      completions: 4
      parallelism: 2
      template:
        spec:
          containers:
          - name: hello
            image: alpine:latest
            command: ["/bin/sh", "-c"]
            args: ["echo hello from $HOSTNAME!"]
          restartPolicy: Never
```

```
$ kubectl get jobs
NAME                           DESIRED   SUCCESSFUL   AGE
cronjob-example-1519053240     4         4            2m
cronjob-example-1519053300     4         4            1m
cronjob-example-1519053360     4         4            26s
```

```
$ kubectl describe cronjob cronjob-example
Name:                          cronjob-example
Namespace:                     default
Labels:                        <none>
Annotations:                   <none>
Schedule:                      */1 * * * *
Concurrency Policy:            Allow
Suspend:                       False
Starting Deadline Seconds:     <unset>
Selector:                      <unset>
Parallelism:                   2
Completions:                   4
Pod Template:
  Labels:  <none>
  Containers:
   hello:
    Image:  alpine:latest
    Port:   <none>
    Command:
      /bin/sh
      -c
    Args:
      echo hello from $HOSTNAME!
    Environment:     <none>
    Mounts:          <none>
  Volumes:           <none>
Last Schedule Time:  Mon, 19 Feb 2018 09:54:00 -0500
Active Jobs:         cronjob-example-1519052040
Events:
  Type    Reason            Age   From                Message
  ----    ------            ----  ----                -------
  Normal  SuccessfulCreate  3m    cronjob-controller  Created job cronjob-example-1519051860
  Normal  SawCompletedJob   2m    cronjob-controller  Saw completed job: cronjob-example-1519051860
  Normal  SuccessfulCreate  2m    cronjob-controller  Created job cronjob-example-1519051920
  Normal  SawCompletedJob   1m    cronjob-controller  Saw completed job: cronjob-example-1519051920
  Normal  SuccessfulCreate  1m    cronjob-controller  Created job cronjob-example-1519051980
```

Using Workloads

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: {}
```

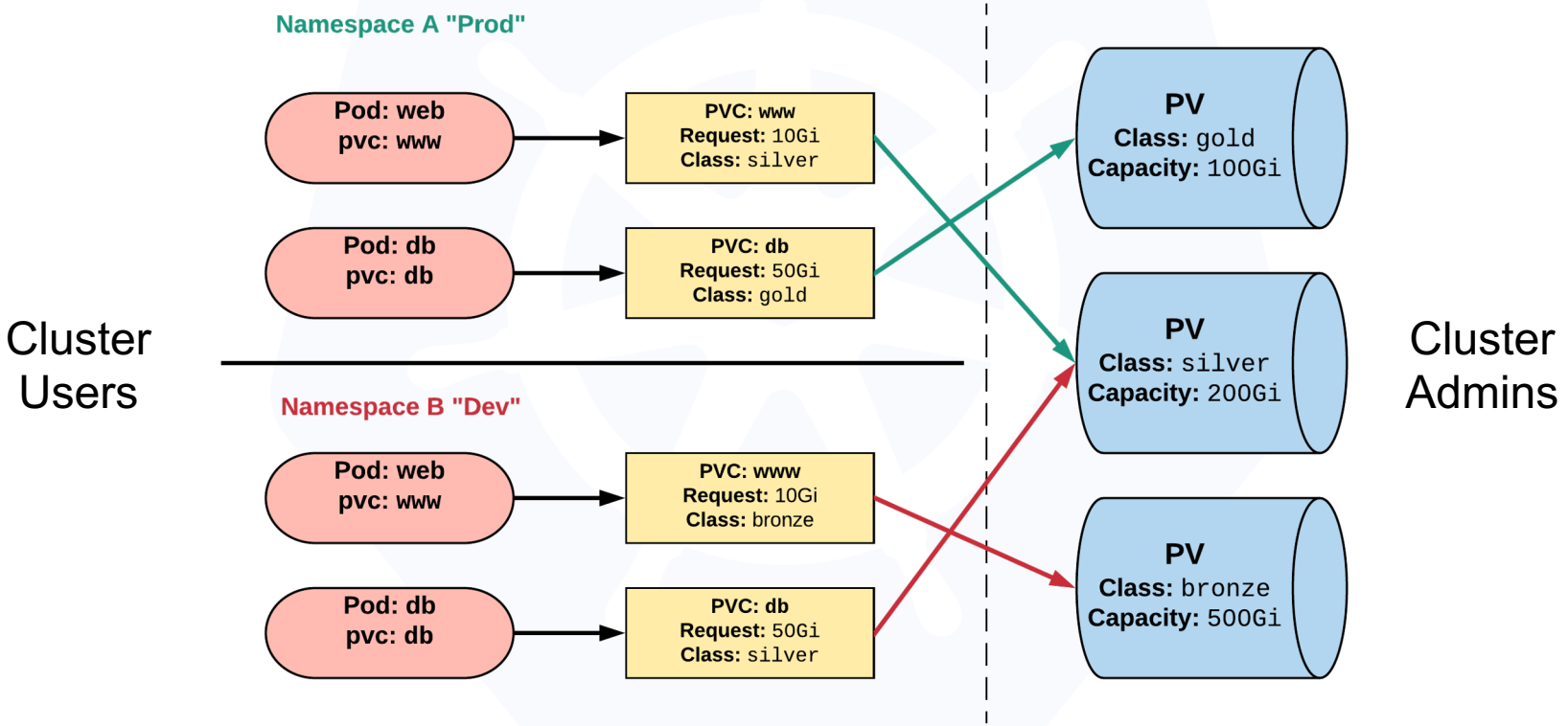
Persistent Volumes

- A **PersistentVolume** (PV) represents a storage resource.
- PVs are a **cluster wide resource** linked to a backing storage provider: NFS, GCEPersistentDisk, 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



PersistentVolume

- `capacity.storage:` The total amount of available storage.
- `volumeMode:` The type of volume, this can be either `Filesystem` or `Block`.
- `accessModes:` A list of the supported methods of accessing the volume. Options include:
  - `ReadWriteOnce`
  - `ReadOnlyMany`
  - `ReadWriteMany`

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfsserver
spec:
  capacity:
    storage: 50Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /exports
    server: 172.22.0.42
```

PersistentVolume

- `persistentVolumeReclaimPolicy:` The behaviour for PVC's that have been deleted. Options include:
  - `Retain` - manual clean-up
  - `Delete` - storage asset deleted by provider.
- `storageClassName:` Optional name of the storage class that PVC's can reference. If provided, **ONLY** PVC's referencing the name consume use it.
- `mountOptions:` Optional mount options for the PV.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfsserver
spec:
  capacity:
    storage: 50Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /exports
    server: 172.22.0.42
```

PersistentVolumeClaim

- `accessModes:` The selected method of accessing the storage. This **MUST** be a subset of what is defined on the target PV or Storage Class.
  - `ReadWriteOnce`
  - `ReadOnlyMany`
  - `ReadWriteMany`
- `resources.requests.storage:` The desired amount of storage for the claim
- `storageClassName:` The name of the desired Storage Class

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-sc-example
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: slow
```

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
```

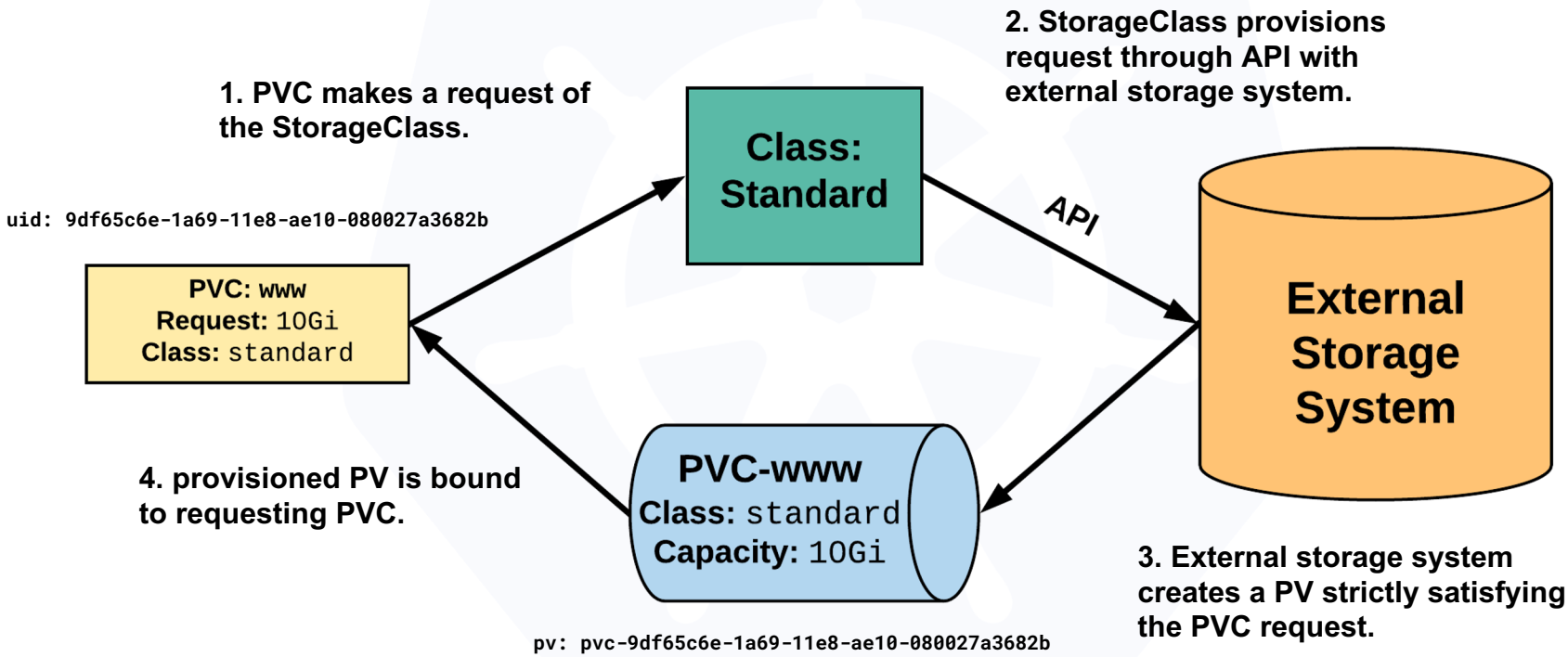
PV Phases

Available	Bound	Released	Failed
PV is ready and available to be consumed.	The PV has been bound to a claim.	The binding PVC has been deleted, and the PV is pending reclamation.	An error has been encountered attempting to reclaim the PV.

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



StorageClass

- `provisioner:` Defines the '*driver*' to be used for provisioning of the external storage.
- `parameters:` A hash of the various configuration parameters for the provisioner.
- `reclaimPolicy:` The behaviour for the backing storage when the PVC is deleted.
  - `Retain` - manual clean-up
  - `Delete` - storage asset deleted by provider

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  zones: us-central1-a, us-central1-b
reclaimPolicy: Delete
```

Available StorageClasses

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

Internal Provisioner

Lab - github.com/mrbobbytables/k8s-intro-tutorials/blob/master/storage

Working with Volumes

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

- 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!
```

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.

Injecting as Environment Variable

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: cm-env-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["printenv CITY"]
        env:
        - name: CITY
          valueFrom:
            configMapKeyRef:
              name: manifest-example
              key: city
      restartPolicy: Never
```

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: secret-env-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["printenv USERNAME"]
        env:
        - name: USERNAME
          valueFrom:
            secretKeyRef:
              name: manifest-example
              key: username
      restartPolicy: Never
```

Injecting as Environment Variable

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: cm-env-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["printenv CITY"]
        env:
        - name: CITY
          valueFrom:
            configMapKeyRef:
              name: manifest-example
              key: city
      restartPolicy: Never
```

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: secret-env-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["printenv USERNAME"]
        env:
        - name: USERNAME
          valueFrom:
            secretKeyRef:
              name: manifest-example
              key: username
      restartPolicy: Never
```

Injecting as a Volume

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: cm-vol-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["cat /myconfig/city"]
        volumeMounts:
        - name: config-volume
          mountPath: /myconfig
      restartPolicy: Never
      volumes:
      - name: config-volume
        configMap:
          name: manifest-example
```

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: secret-vol-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["cat /mysecret/username"]
        volumeMounts:
        - name: secret-volume
          mountPath: /mysecret
      restartPolicy: Never
      volumes:
      - name: secret-volume
        secret:
          secretName: manifest-example
```

Injecting as a Volume

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: cm-vol-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["cat /myconfig/city"]
        volumeMounts:
        - name: config-volume
          mountPath: /myconfig
      restartPolicy: Never
      volumes:
      - name: config-volume
        configMap:
          name: manifest-example
```

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: secret-vol-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["cat /mysecret/username"]
        volumeMounts:
        - name: secret-volume
          mountPath: /mysecret
      restartPolicy: Never
      volumes:
      - name: secret-volume
        secret:
          secretName: manifest-example
```

Putting it all Together

Questions?