Kubernetes

A brief(?!) introduction

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.

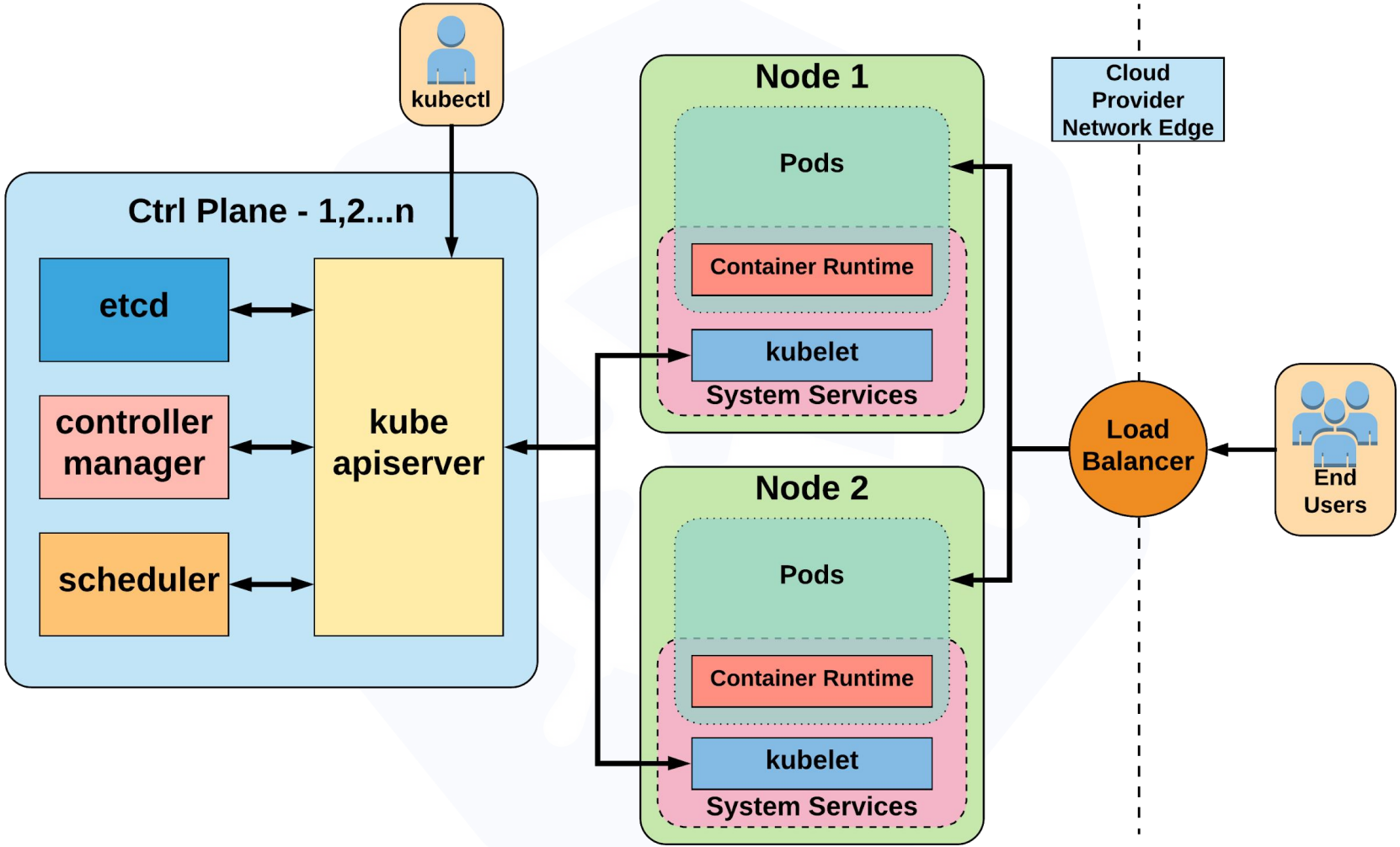
What does Kubernetes do?

- **Abstracts away the underlying hardware** of nodes and provides a uniform interface for workloads to be both deployed and consume the shared pool of resources.
- Works as an engine for resolving state by converging actual and the **desired state** of the system.

What are its capabilities?

- Autoscale Workloads
- Persistent services, cron jobs, and (to some extent) batch workloads
- Manage Stateless and Stateful Applications
- Provide native methods of service discovery
- Self healing

Architecture Overview

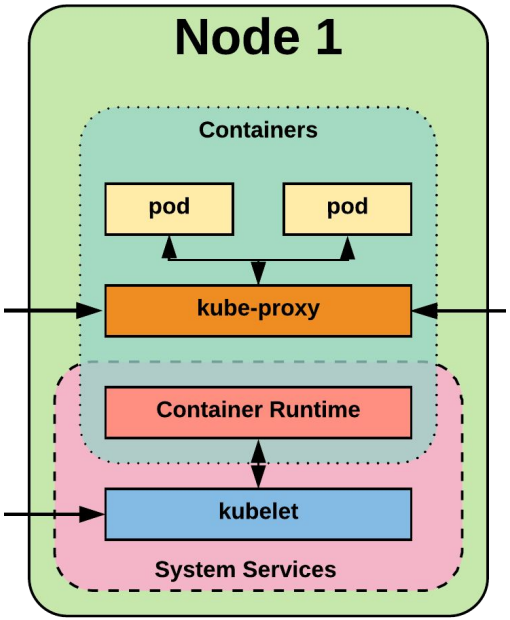

kubectl
Ctrl Plane - 1,2...n
etcd
controller manager
scheduler
kube apiserver
Node 1
Pods
Container Runtime
kubelet
System Services
Node 2
Pods
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users

The Control Plane

- The Kubernetes "master"
- Manages the cluster configuration & state via distributed datastore
- Responds to state changes, provides API (CLI + RESTful interfaces), scheduling, etc

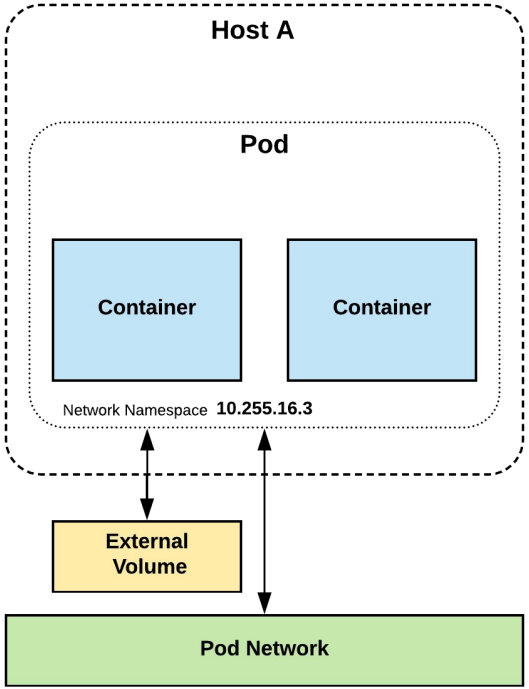
Nodes

- Pods
  - One or more containers in a single namespace
- kubelet
  - Manages Pods
- kube-proxy
  - Manages Pod networking
- Container Runtime Engine
  - Manages Containers



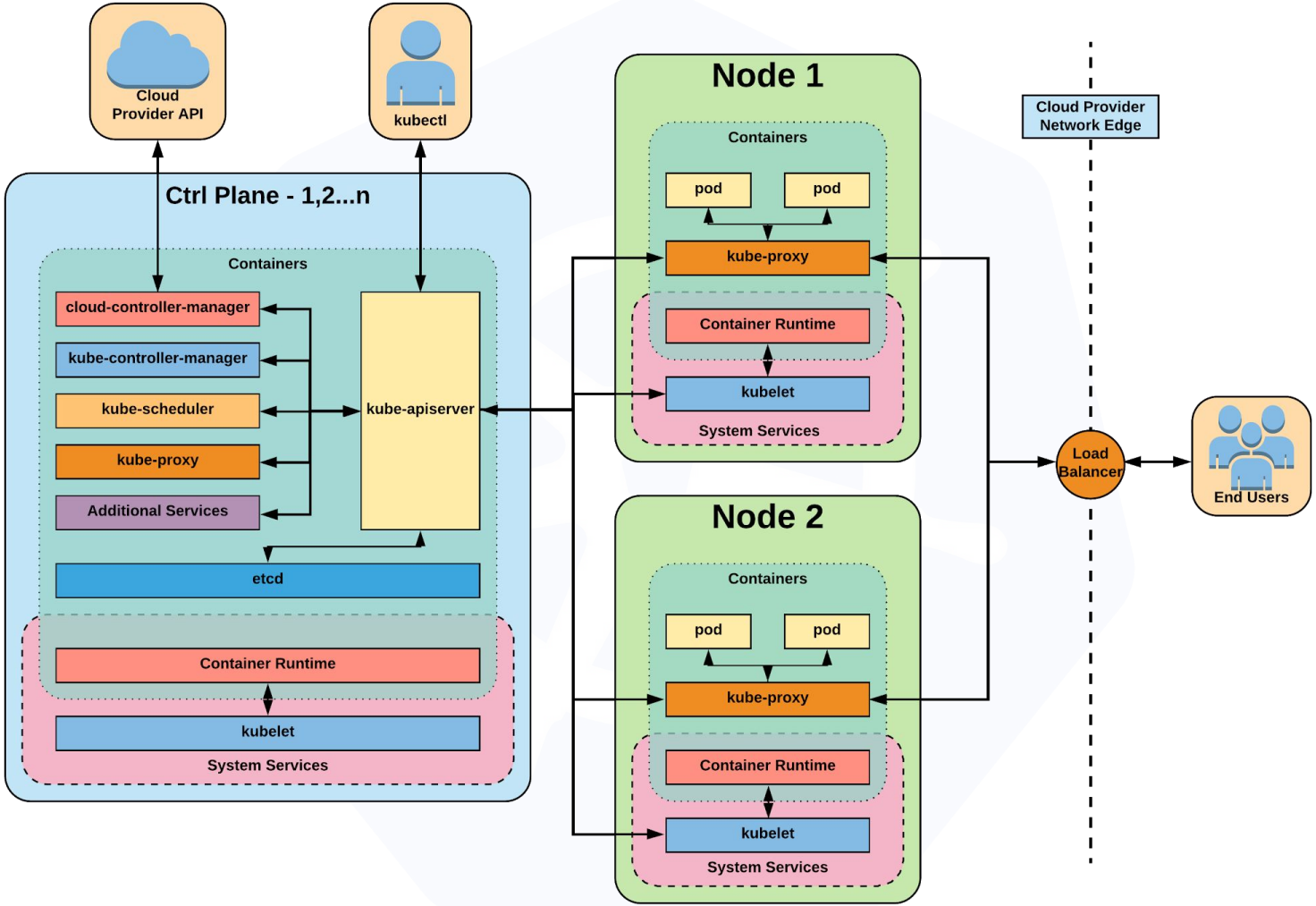
Pods

- The smallest deployable object in Kubernetes
- **Pods are not restarted when they die.**
- Higher level objects manage replicas, fault tolerance, etc
- State must be stored externally!



kubelet

- Daemon on every node that manages pod lifecycle
- Communicates with the Container Runtime Interface (CRI) to deploy containers
  - Typically Docker
- Listens over HTTP(S) for Pod specification files
  - Typically from API Server


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

Concepts and Resources

The API and Object Model

Concepts and Resources

API Overview

- Kubernetes provides a RESTful API for all interactions with a cluster
- The `kubectl` client wraps around this API
- **Everything** within Kubernetes is an **API Object**.

API Groups

- An API Group is a **REST compatible path** that acts as the type descriptor for a Kubernetes object.
- Referenced within an object as the `apiVersion` and `kind`.

Format:

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

Examples:

`/apis/apps/v1/deployments`

`/apis/batch/v1beta1/cronjobs`

Object Model

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

Object Model Requirements

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

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

Object 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 Objects

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

Namespaces

Pods **Services**

Labels **Selectors**

Namespace

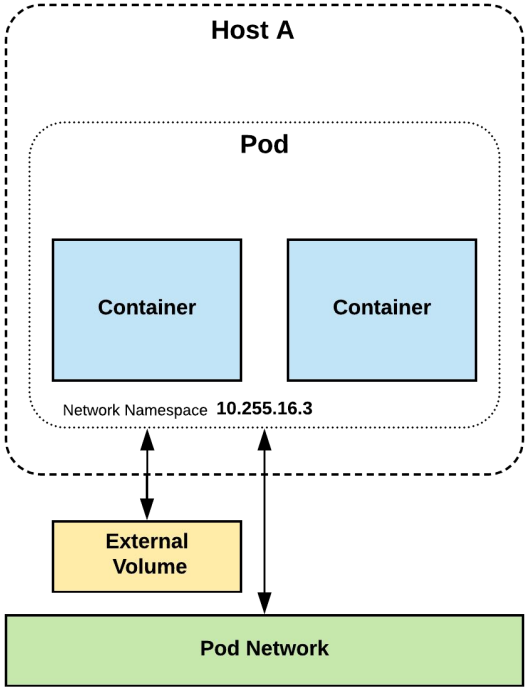
Namespaces are a logical cluster or environment, and are the primary method of partitioning a cluster or scoping access.

```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
```

Pod

- Foundational building block of Kubernetes Workloads.
- Pods are one or more containers that share volumes and a network namespace



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

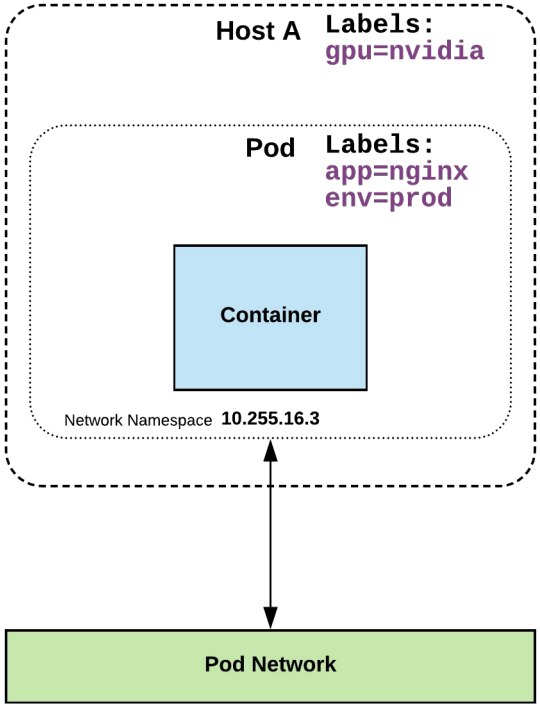
- `name` - The name of the container
- `image` - The container image
- `ports` - array of ports to expose. Can be granted a friendly name and protocol may be specified
- `env` - array of environment variables
- `command` – Entrypoint array (equiv to Docker `ENTRYPOINT`)
- `args` - Arguments to pass to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

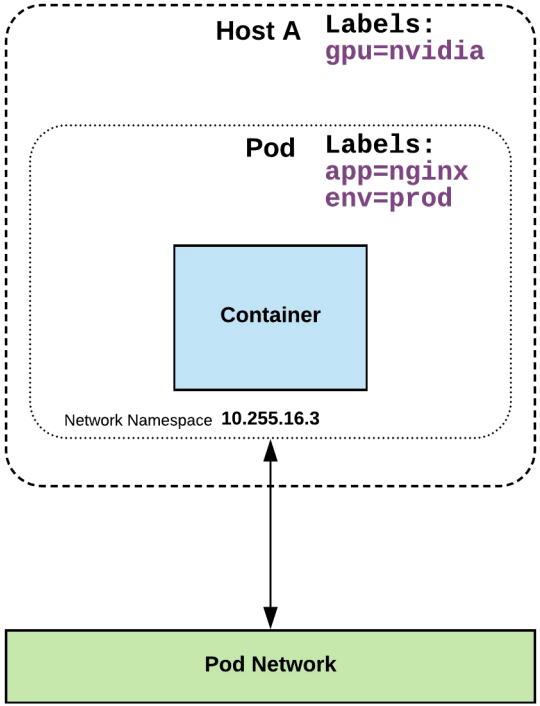
Labels

- key-value pairs that are used to identify, describe and group together related sets of objects or resources.
- Can be applied to nodes, pods, services, etc etc.



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
```



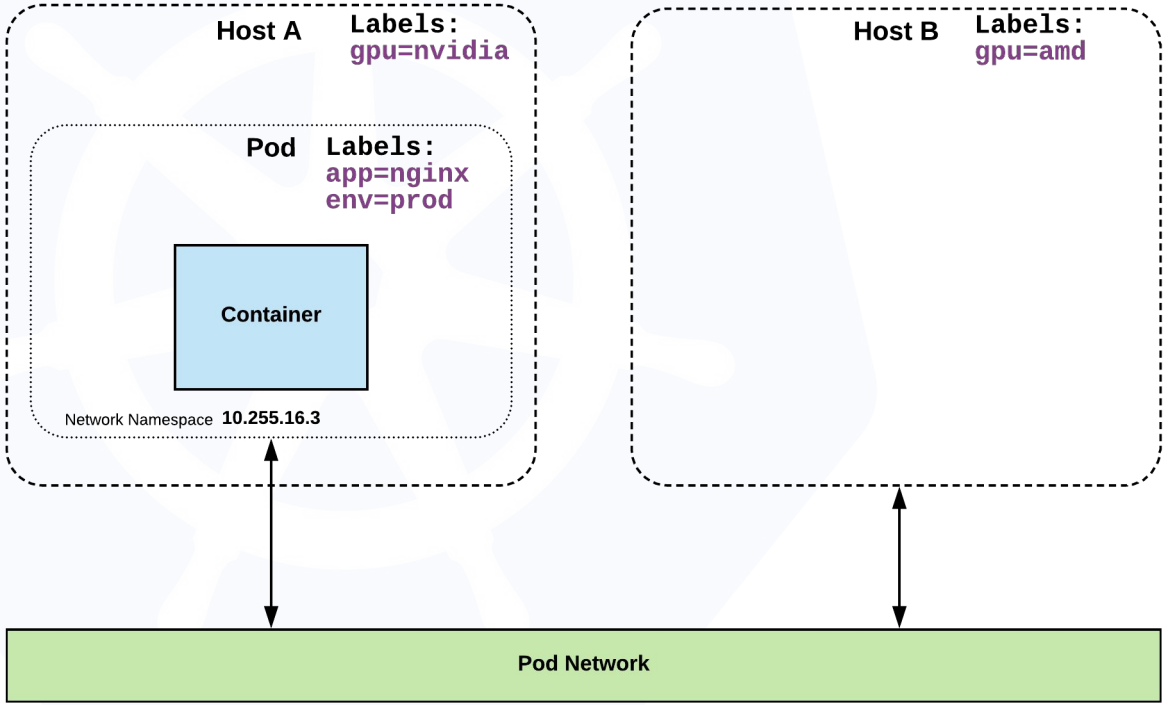
Selectors

- Use labels to filter or select objects, and are used throughout Kubernetes.
- Supports equality-based selection and "set-based" selection
  - item != foo
  - item in (bar,baz)

```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
```



Services

- An abstraction to expose Pods to the network
- Attached to pods via Selectors
- Defines a network policy by which to access the Pods attached
- **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)
  - Internal cluster-only networking
- **NodePort**
  - Maps to a port on the "external" network of the host
- **LoadBalancer**
  - Uses an external system to map an IP to the service
- **ExternalName**
  - Maps the service IP to the value of a DNS lookup

ClusterIP Service

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

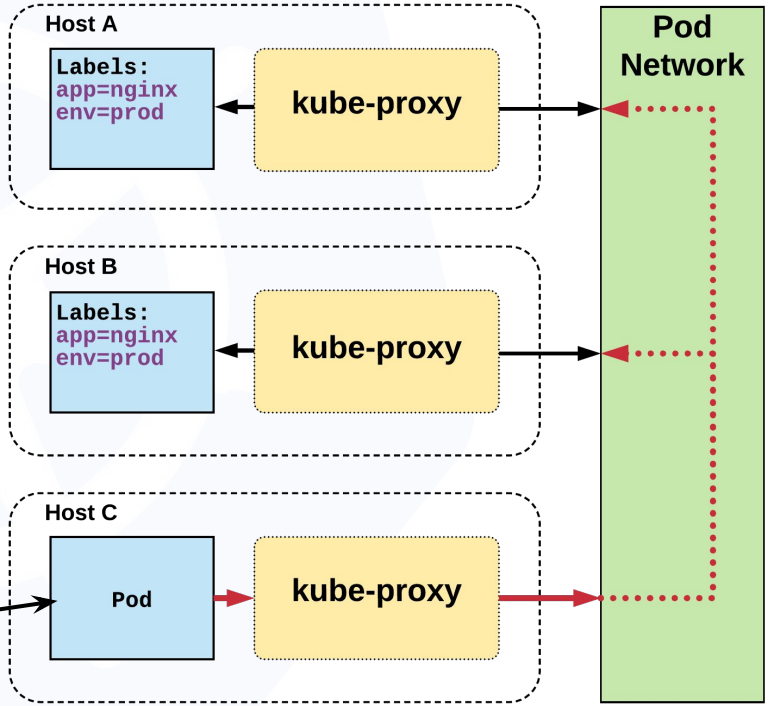
```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  selector:
    app: nginx
    env: prod
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

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
```

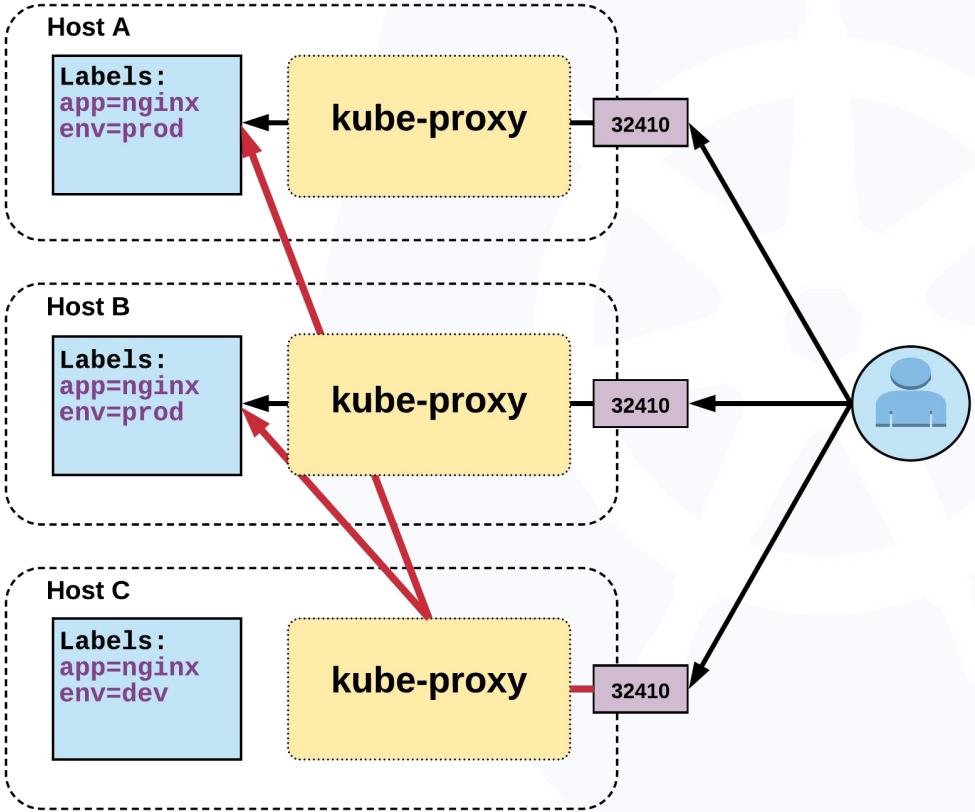


NodePort Service

- **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.

```
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



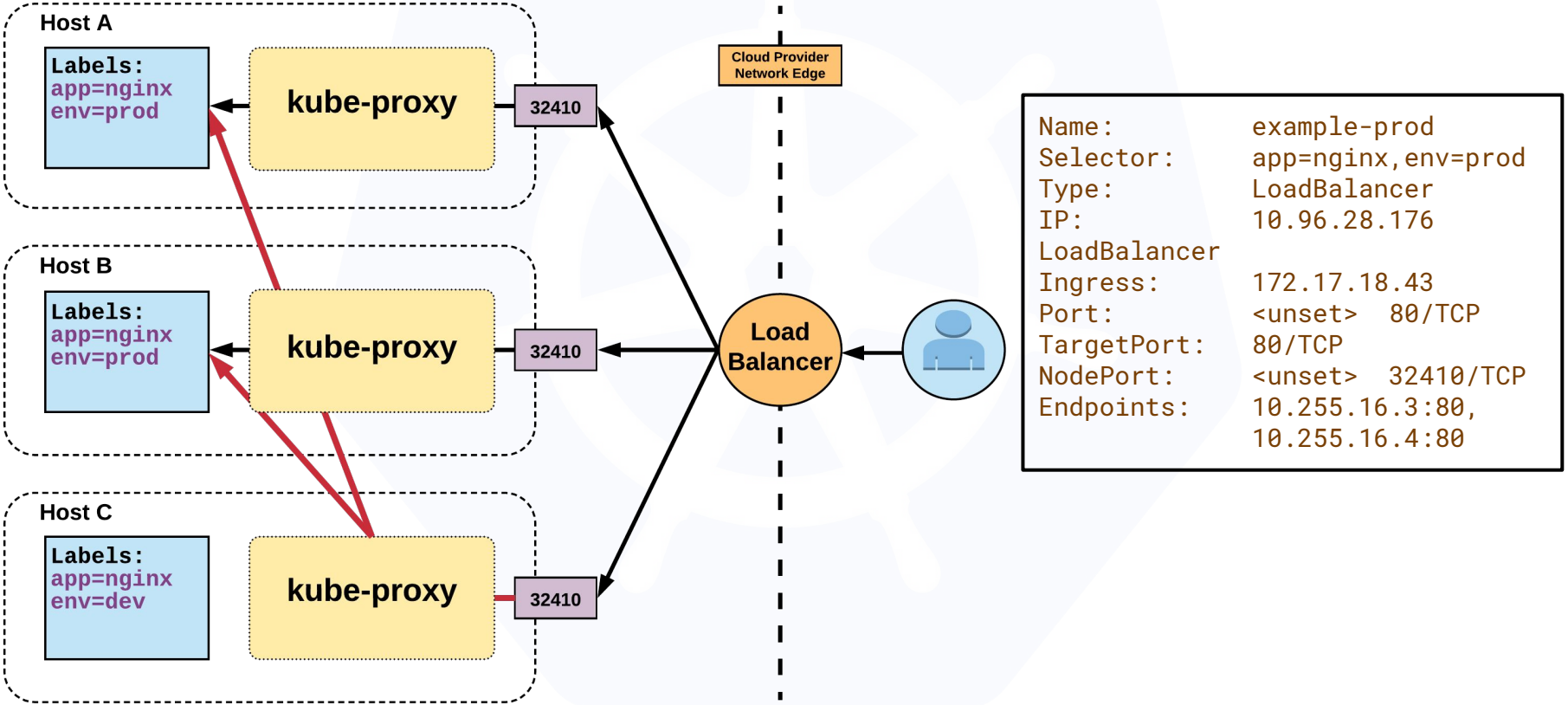

```
Name:          example-prod
Selector:      app=nginx,env=prod
Type:          NodePort
IP:            10.96.28.176
Port:          <unset>  80/TCP
TargetPort:    80/TCP
NodePort:      <unset>  32410/TCP
Endpoints:     10.255.16.3:80,
               10.255.16.4:80
```

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



Break time!

Workloads

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

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

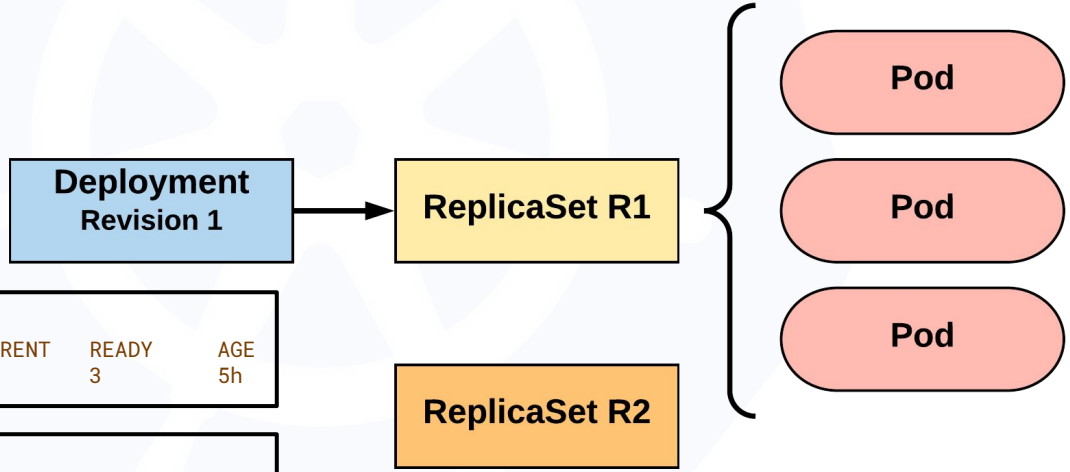
- `revisionHistoryLimit:` The number of previous iterations of the Deployment to retain.
- `strategy:` Describes the method of updating the Pods based on the `type`. Valid options are `Recreate` or `RollingUpdate`.
  - `Recreate:` All existing Pods are killed before the new ones are created.
  - `RollingUpdate:`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

```
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>
```

RollingUpdate Deployment

Updating pod template generates a new **ReplicaSet** revision.

R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



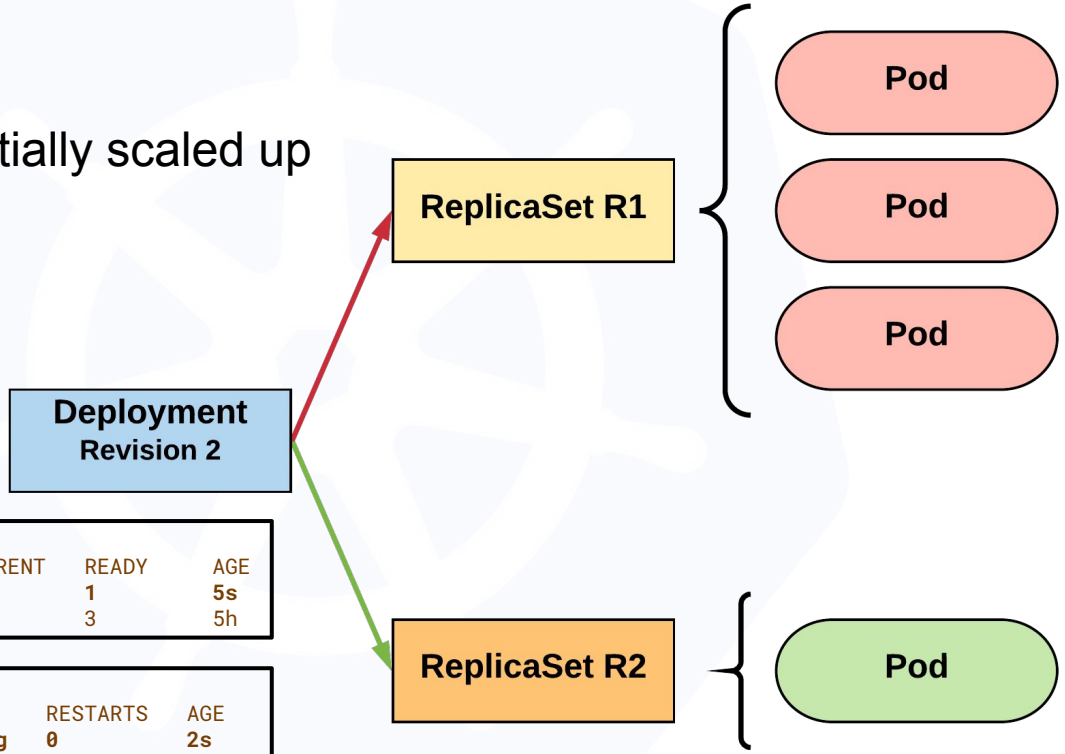

```
$ kubectl get replicaset
NAME                  DESIRED   CURRENT   READY   AGE
mydep-6766777fff      3         3         3       5h
```

```
$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
mydep-6766777fff-9r2zn      1/1     Running   0          5h
mydep-6766777fff-hsfz9      1/1     Running   0          5h
mydep-6766777fff-sjxhf      1/1     Running   0          5h
```

RollingUpdate Deployment

New **ReplicaSet** is initially scaled up based on `maxSurge`.

R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



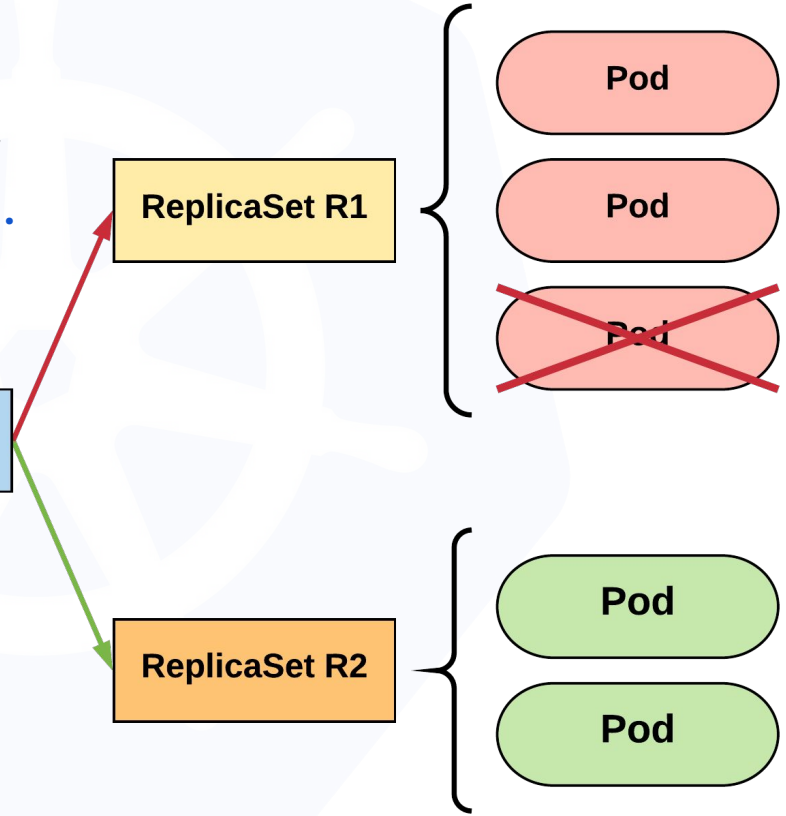

```
$ kubectl get replicaset
NAME                  DESIRED   CURRENT   READY   AGE
mydep-54f7ff7d6d      1         1         1       5s
mydep-6766777fff      2         3         3       5h
```

```
$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
mydep-54f7ff7d6d-9gvll      1/1     Running   0          2s
mydep-6766777fff-9r2zn      1/1     Running   0          5h
mydep-6766777fff-hsfz9      1/1     Running   0          5h
mydep-6766777fff-sjxhf      1/1     Running   0          5h
```

RollingUpdate Deployment

Phase out of old Pods managed by `maxSurge` and `maxUnavailable`.

R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



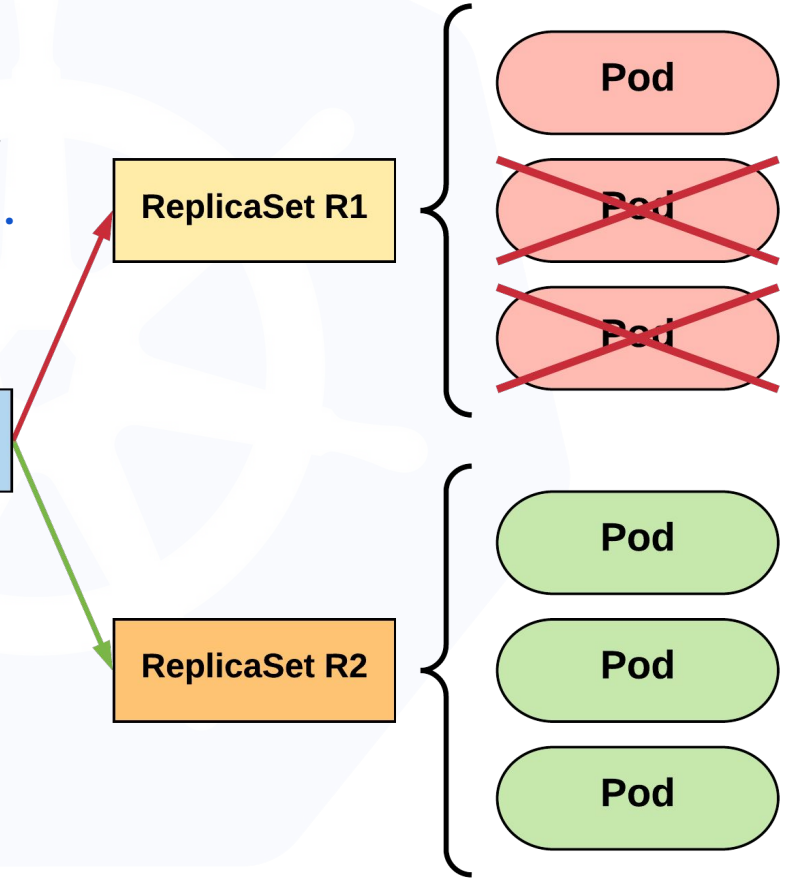

```
$ kubectl get replicaset
NAME                  DESIRED   CURRENT   READY   AGE
mydep-54f7ff7d6d      2         2         2       8s
mydep-6766777fff      2         2         2       5h
```

```
$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
mydep-54f7ff7d6d-9gvll      1/1     Running   0          5s
mydep-54f7ff7d6d-cqvlq      1/1     Running   0          2s
mydep-6766777fff-9r2zn      1/1     Running   0          5h
mydep-6766777fff-hsfz9      1/1     Running   0          5h
```

RollingUpdate Deployment

Phase out of old Pods managed by `maxSurge` and `maxUnavailable`.

R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



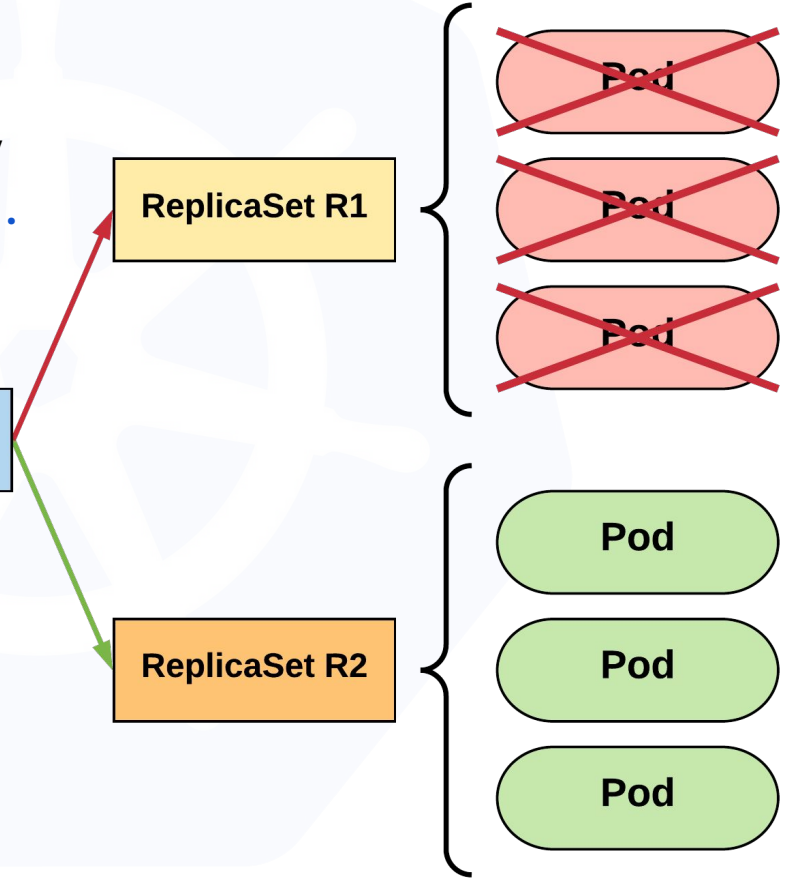

```
$ kubectl get replicaset
NAME                  DESIRED   CURRENT   READY   AGE
mydep-54f7ff7d6d      3         3         3       10s
mydep-6766777fff      0         1         1       5h
```

```
$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
mydep-54f7ff7d6d-9gvll      1/1     Running   0          7s
mydep-54f7ff7d6d-cqvlq      1/1     Running   0          5s
mydep-54f7ff7d6d-gccr6      1/1     Running   0          2s
mydep-6766777fff-9r2zn      1/1     Running   0          5h
```

RollingUpdate Deployment

Phase out of old Pods managed by `maxSurge` and `maxUnavailable`.

R1 pod-template-hash:
676677fff
R2 pod-template-hash:
54f7ff7d6d



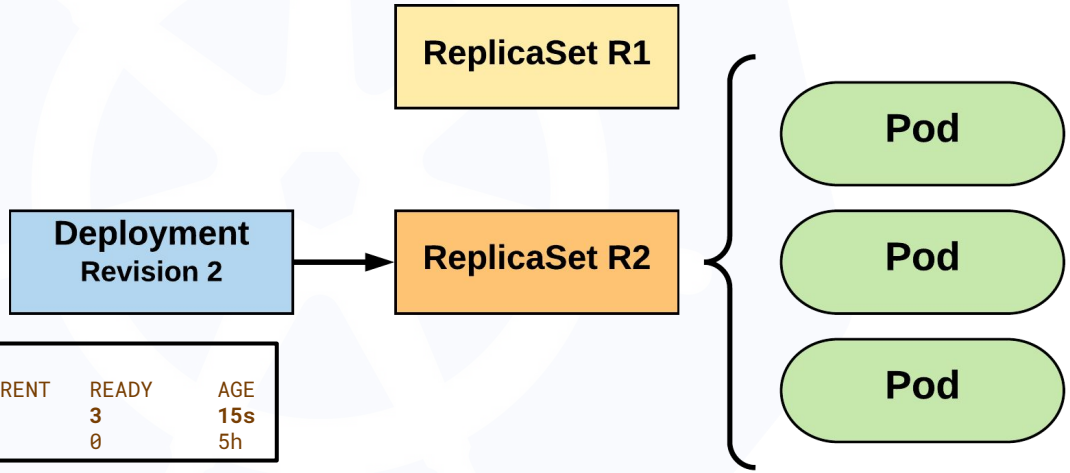

```
$ kubectl get replicaset
NAME                  DESIRED   CURRENT   READY   AGE
mydep-54f7ff7d6d      3         3         3       13s
mydep-6766777fff      0         0         0       5h
```

```
$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
mydep-54f7ff7d6d-9gvll      1/1     Running   0          10s
mydep-54f7ff7d6d-cqvlq      1/1     Running   0          8s
mydep-54f7ff7d6d-gccr6      1/1     Running   0          5s
```

RollingUpdate Deployment

Updated to new deployment revision completed.

R1 pod-template-hash:
676677fff
R2 pod-template-hash:
54f7ff7d6d




```
$ kubectl get replicaset
NAME                  DESIRED   CURRENT   READY   AGE
mydep-54f7ff7d6d      3         3         3       15s
mydep-6766777fff      0         0         0       5h
```

```
$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
mydep-54f7ff7d6d-9gvll   1/1     Running   0          12s
mydep-54f7ff7d6d-cqvlq   1/1     Running   0          10s
mydep-54f7ff7d6d-gccr6   1/1     Running   0          7s
```

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.

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.

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>
```

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: {}
```

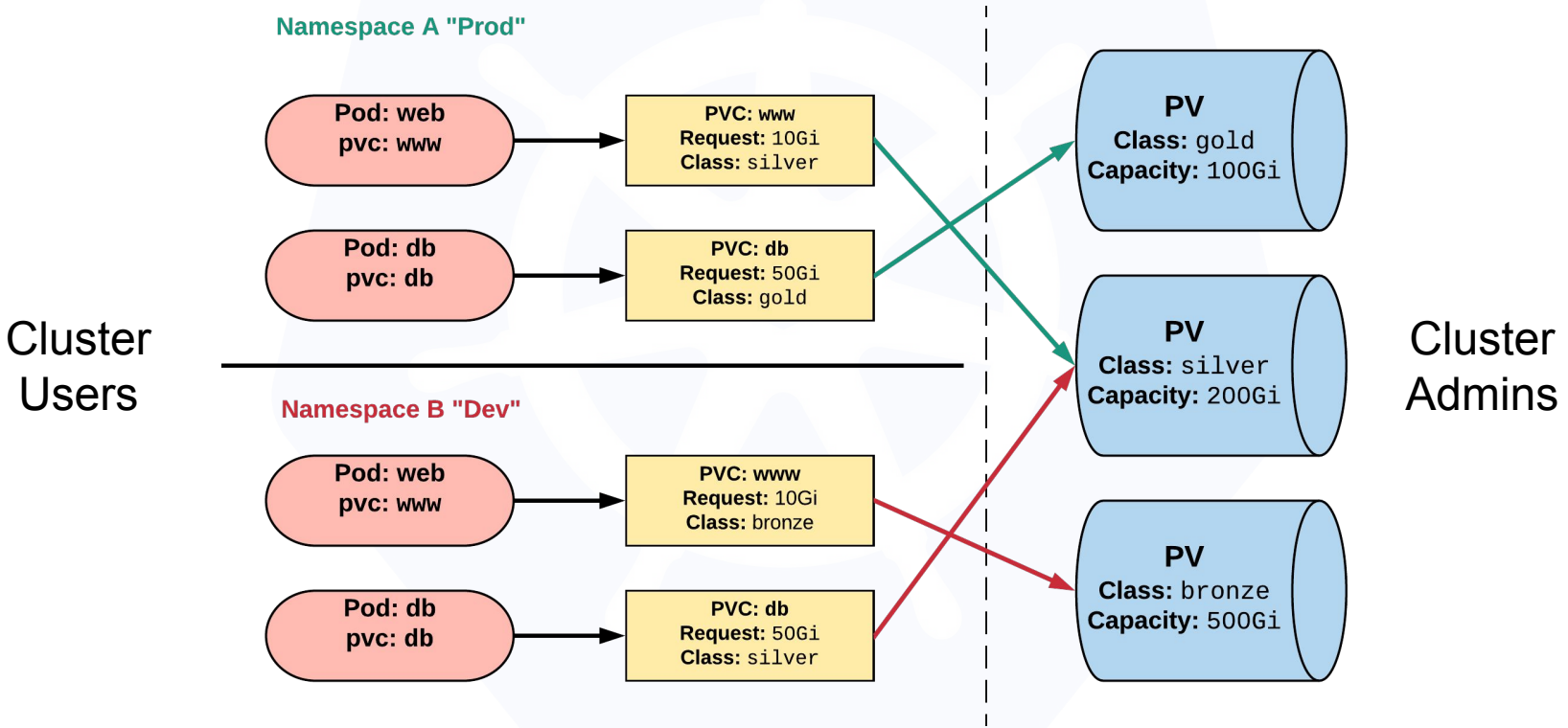
Persistent Volumes

- 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.

```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
```

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

```
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
```

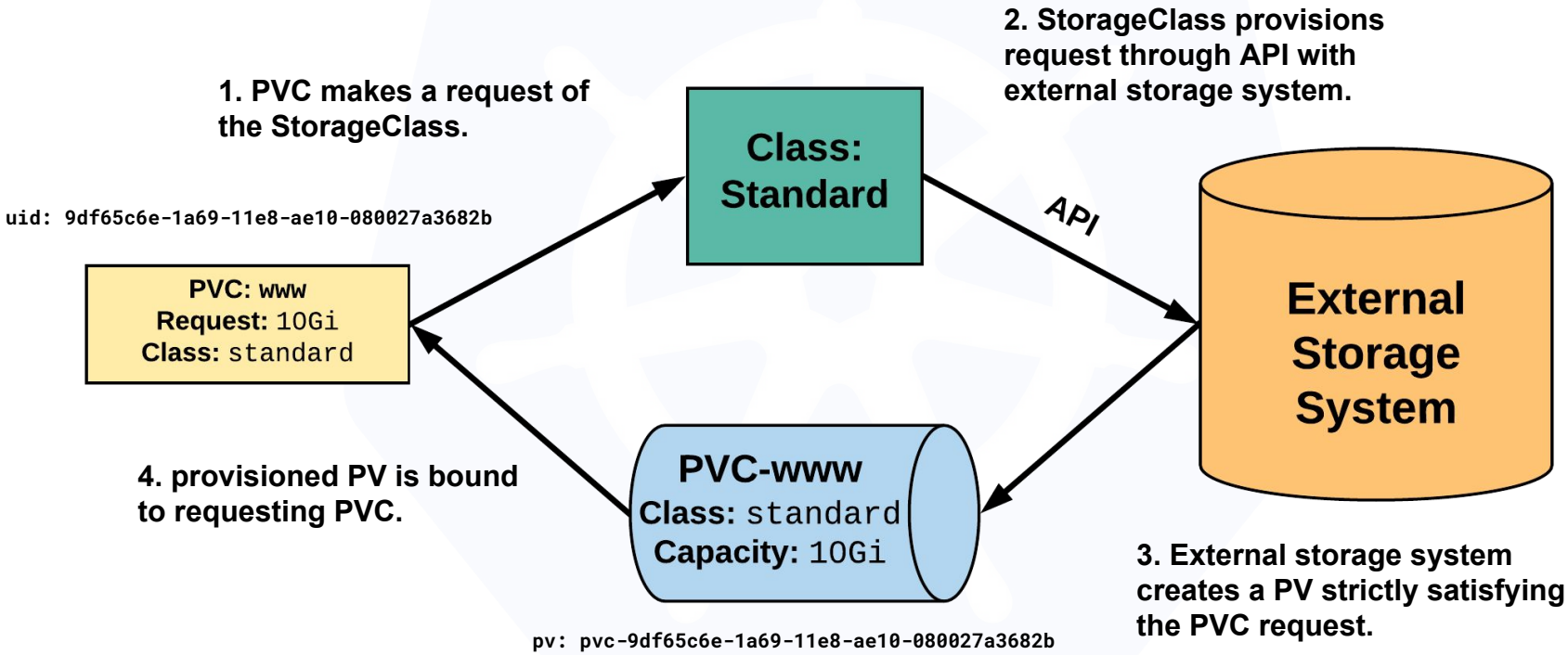
PV Phases

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

- 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!
```

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==
```

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 in a Command

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

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

Injecting in a Command

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

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: secret-cmd-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["echo Hello ${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
```

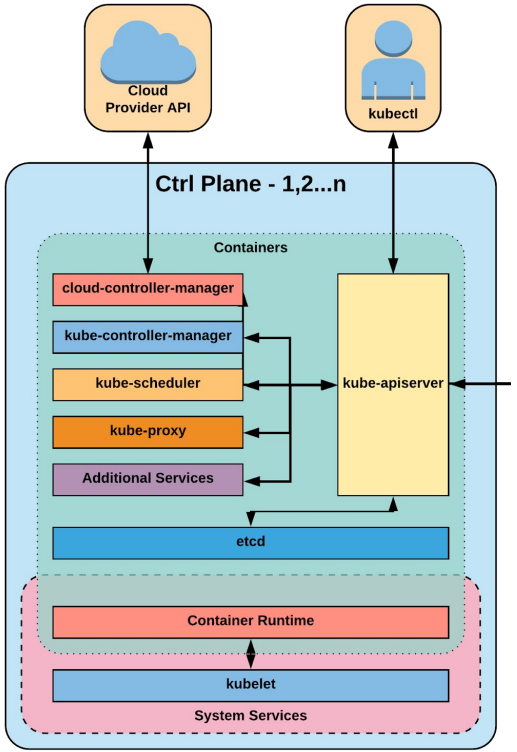
```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: secret-vol-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["cat /mysecret/username"]
        volumeMounts:
        - name: secret-volume
 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

Questions?

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**.

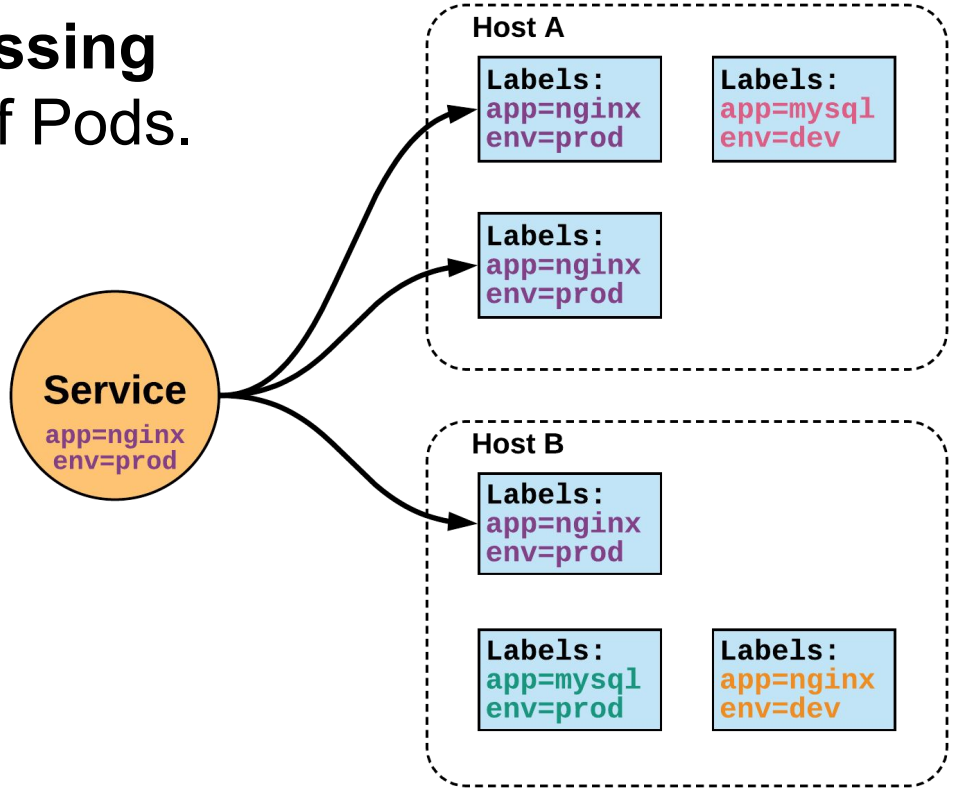
List of core controllers:
https://github.com/kubernetes/kubernetes/blob/master/cmd/kube-controller-manager/app/controllermanager.go#L344

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.

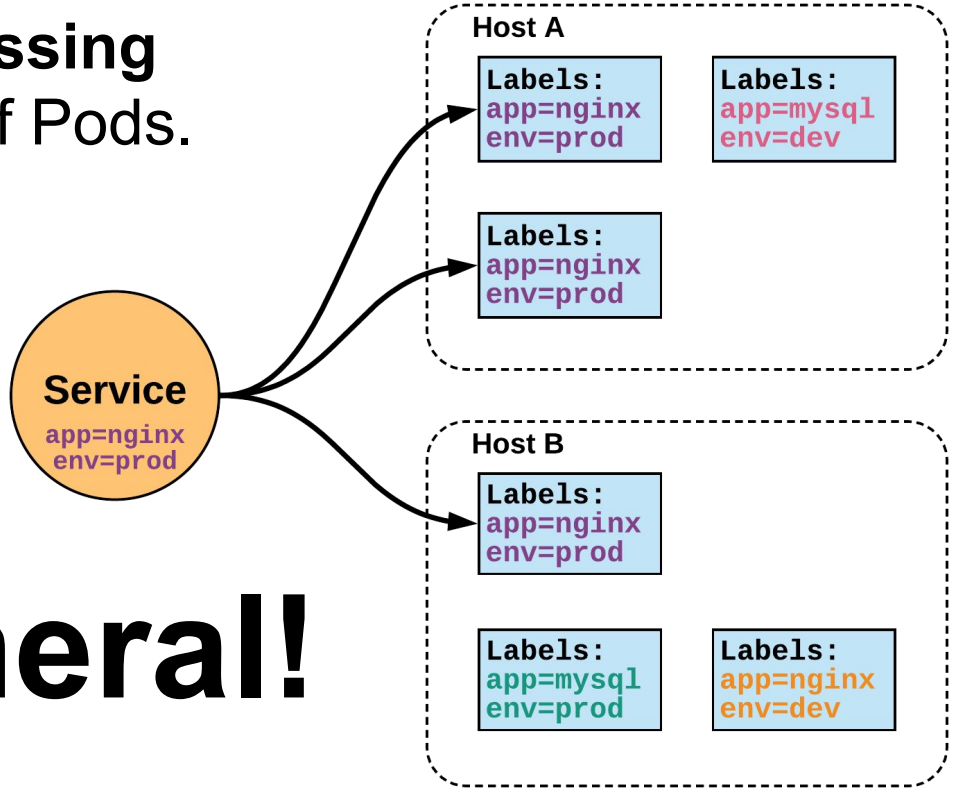
Services

- **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!

Networking

Architecture Overview

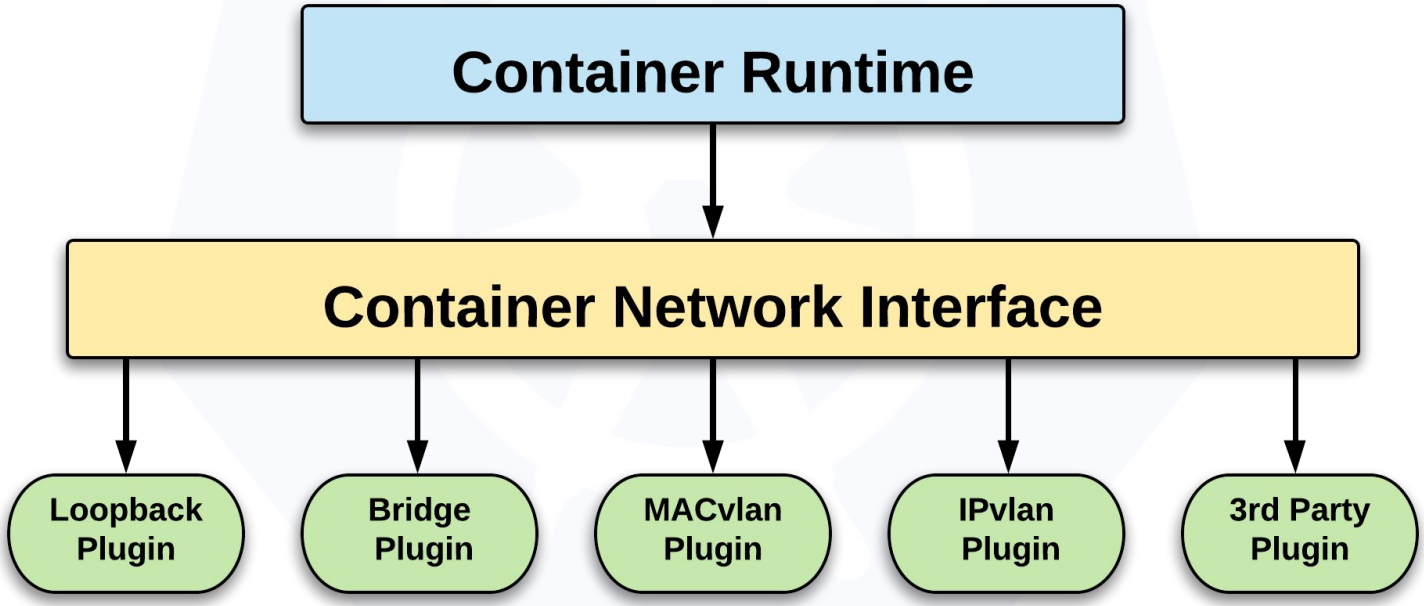
Kubernetes Networking

- **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.

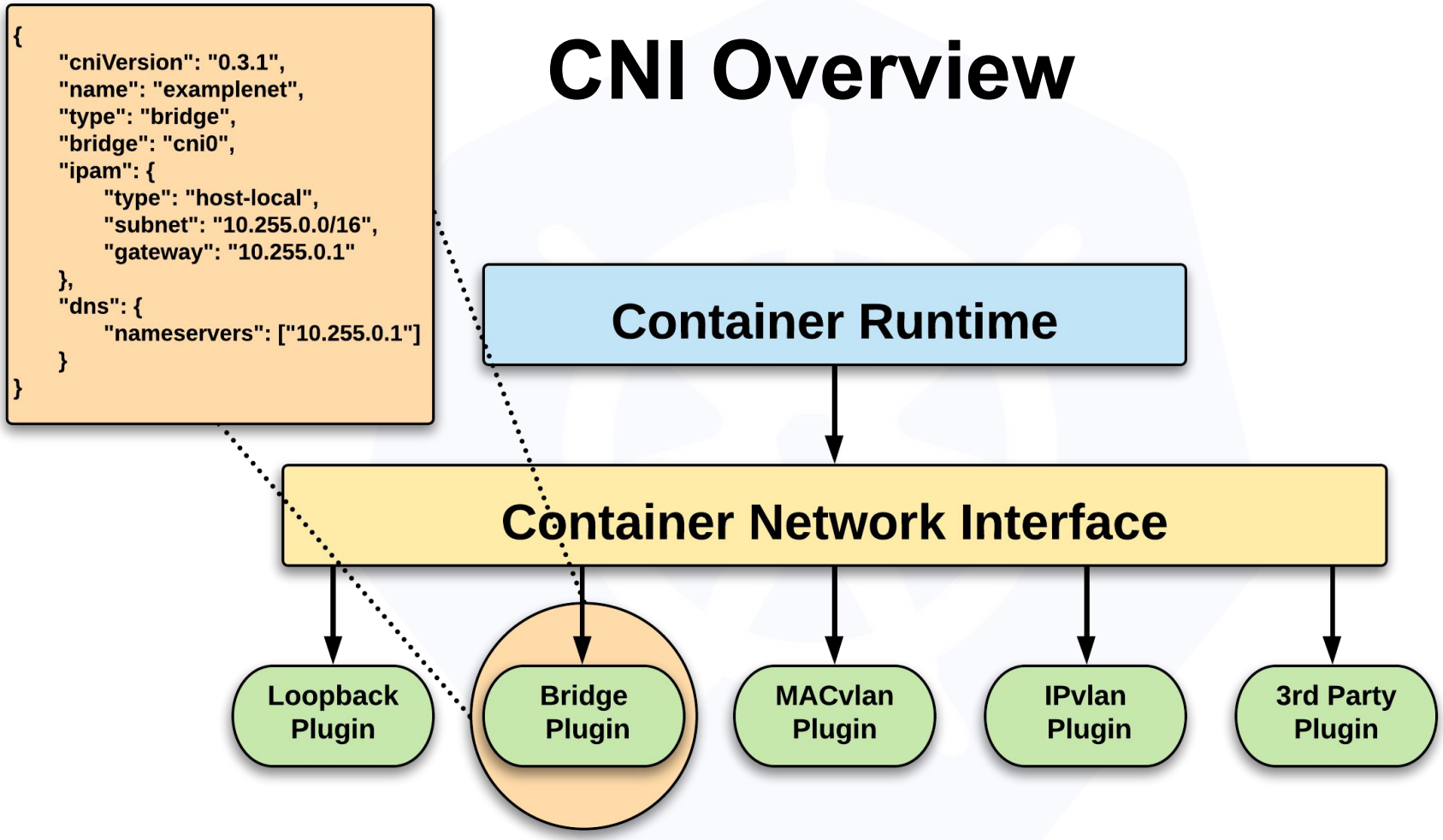
Container Network Interface (CNI)

- 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**.

CNI Overview



CNI Overview



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.

Fundamentals Applied

- **Pod-to-Service**
  - managed by **kube-proxy** and given a **persistent cluster unique IP**
  - exists beyond a Pod's lifecycle.
- **External-to-Service**
  - Handled by **kube-proxy.**
  - Works in cooperation with a cloud provider or other external entity (load balancer).

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**.

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”]
```