redhat

Kubernetes Overview

redhat

4

1. The Project

`github.com/openshift/origin`

`github.com/kubernetes`

2. Community Deployment

`kops`

`kubeadm`

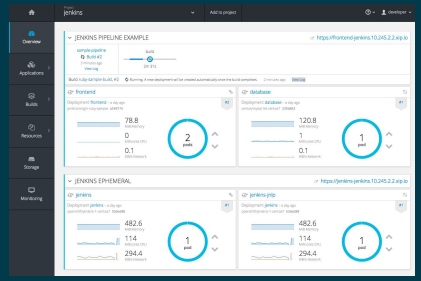
`minikube/minishift/crc (codeready containers)`

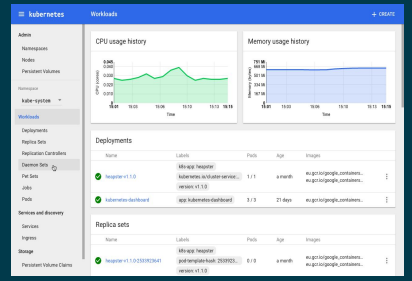
3. Product

4. Service

4

1. Dashboard





2. Command Line Interface

```
$ oc/kubectl apply -f my-new-deployment.yaml
```

3. SDK/Client Libraries

```
pod, err :=
c.Pods(v1.NamespaceDefault).Get("my-pod")
    if err != nil {
        fmt.Println(err)
        return
```

4. Helm

```
$ helm install stable/mariadb
```

API

redhat.

oc/kubectl

A command line interface for running commands against Kubernetes clusters.

Installing kubectl

Homebrew on macOS

```
$ brew install kubectl
```

Chocolatey on Windows

```
$ choco install kubectl
```

Linux/macOS/Windows

```
$ curl -LO https://storage.googleapis.com/kubernetes-release/release/
```

How to Provide Your K8s Auth Info

1 - argument

```
oc --kubeconfig config cluster-info
```

2 - variable

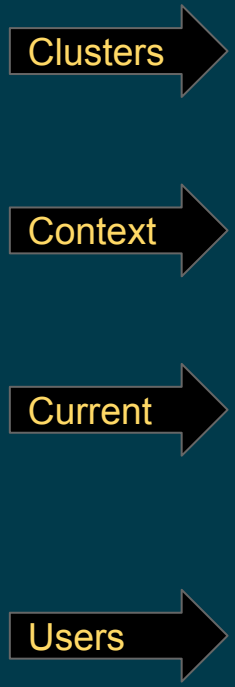
```
export KUBECONFIG=config
oc cluster-info
```

3 - .kube/config

```
oc cluster-info
```

```
cat .kube/config
```

Anatomy of a `kubeconfig`




```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: <cacert>
    server: https://192.168.56.60:6443
  name: my-first-cluster
contexts:
- context:
    cluster: my-first-cluster
    user: kubelet
  name: default
current-context: default
kind: Config
preferences: {}
users:
- name: kubelet
  user:
    client-certificate-data: <yourclientcert>
    client-key-data: <yourclientkey>
```

About Client-Go

A collection of tools/frameworks (in the form of Go packages) for all your Kubernetes programming needs.

Contents of Client-Go

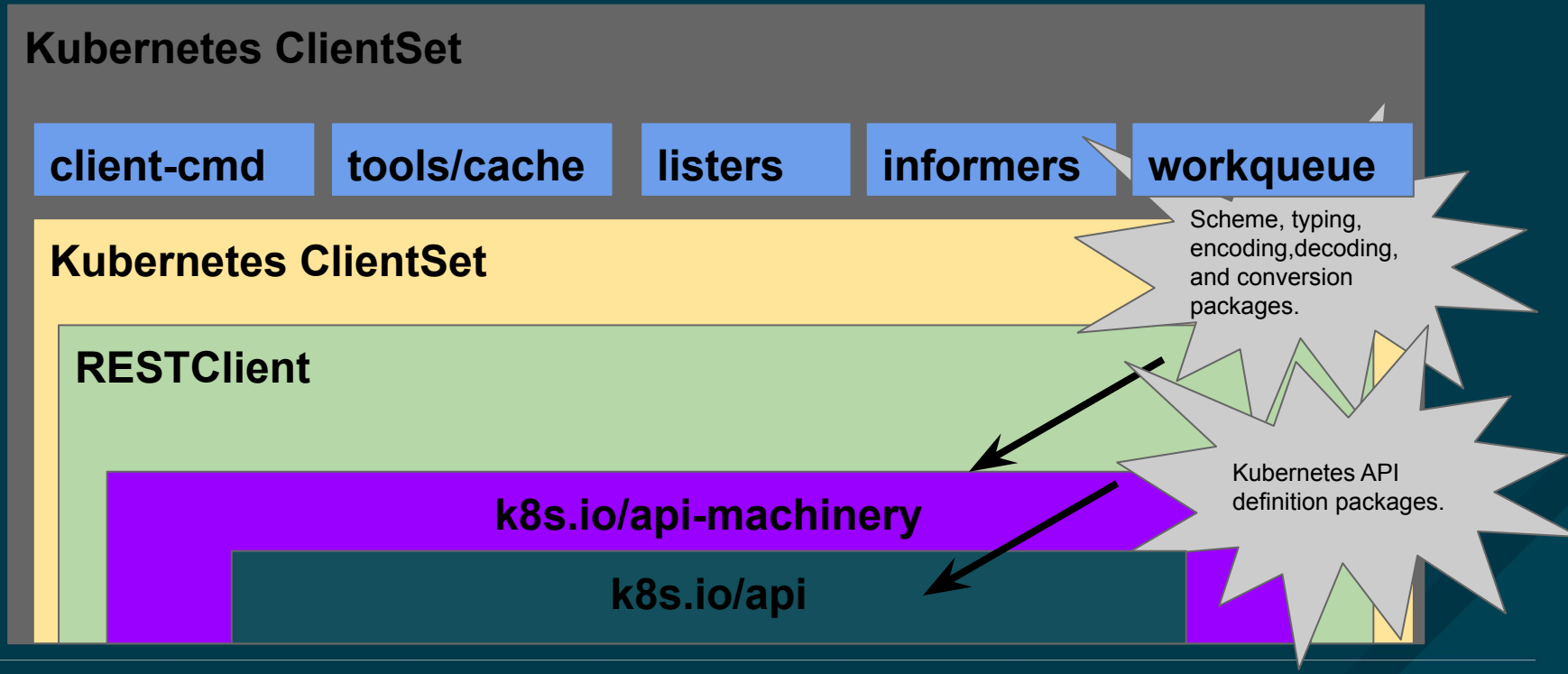
Clients

- Clientset
- Dynamic Client
- RESTclient

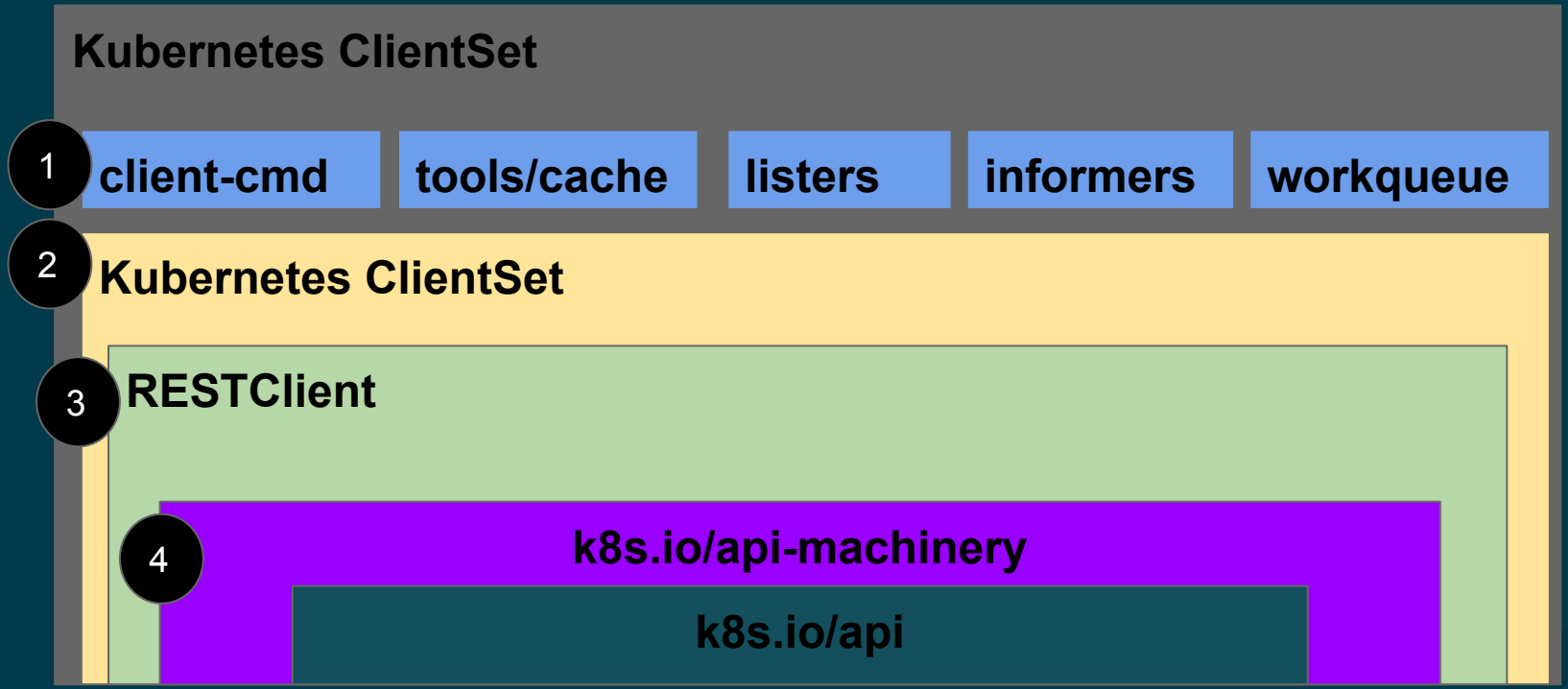
Utilities for Writing Controllers

- Workqueue
- Informers/Shared Informers

Client-Go Implementation



Out-of-Cluster Interaction With Kubernetes API



1 client-cmd

Fetch the kube-config file and use current-context

```
func main() {
	var kubeconfig *string
	if home := homeDir(); home != "" {
		kubeconfig = flag.String("kubeconfig", filepath.Join(home, ".kube", "config"), "(optional) absolute path to the kubeconfig file")
	} else {
		kubeconfig = flag.String("kubeconfig", "", "absolute path to the kubeconfig file")
	}
	flag.Parse()

config, err := clientcmd.BuildConfigFromFlags("", *kubeconfig)
	if err != nil {
		panic(err.Error())
	}
```

"k8s.io/client-go/tools/clientcmd"

Create the client-set

```
clientset, err := kubernetes.NewForConfig(config)
        if err != nil {
                panic(err.Error())
        }
```

"k8s.io/client-go/kubernetes"

2 Kubernetes ClientSet

3 RESTClient

4 k8s.io/api-machinery

Retrieve the Corev1 Client via clientset and list all pods in the cluster (across all namespaces)

```
for {
        pods, err := clientset.CoreV1().Pods("").List(metav1.ListOptions{})
        if err != nil {
                panic(err.Error())
        }
        fmt.Printf("There are %d pods in the cluster\n", len(pods.Items))
}
```

```
Verb(verb string) *Request
Post() *Request
Put() *Request
Patch(pt types.PatchType) *Request
Get() *Request
Delete() *Request
APIVersion() schema.GroupVersion
```

```
type ListOptions struct {
    TypeMeta      `json:",inline"`
    LabelSelector    string
`json:"labelSelector,omitempty"`
    FieldSelector    string
`json:"fieldSelector,omitempty"`
protobuf:"bytes,2,opt,name=fieldSele
```

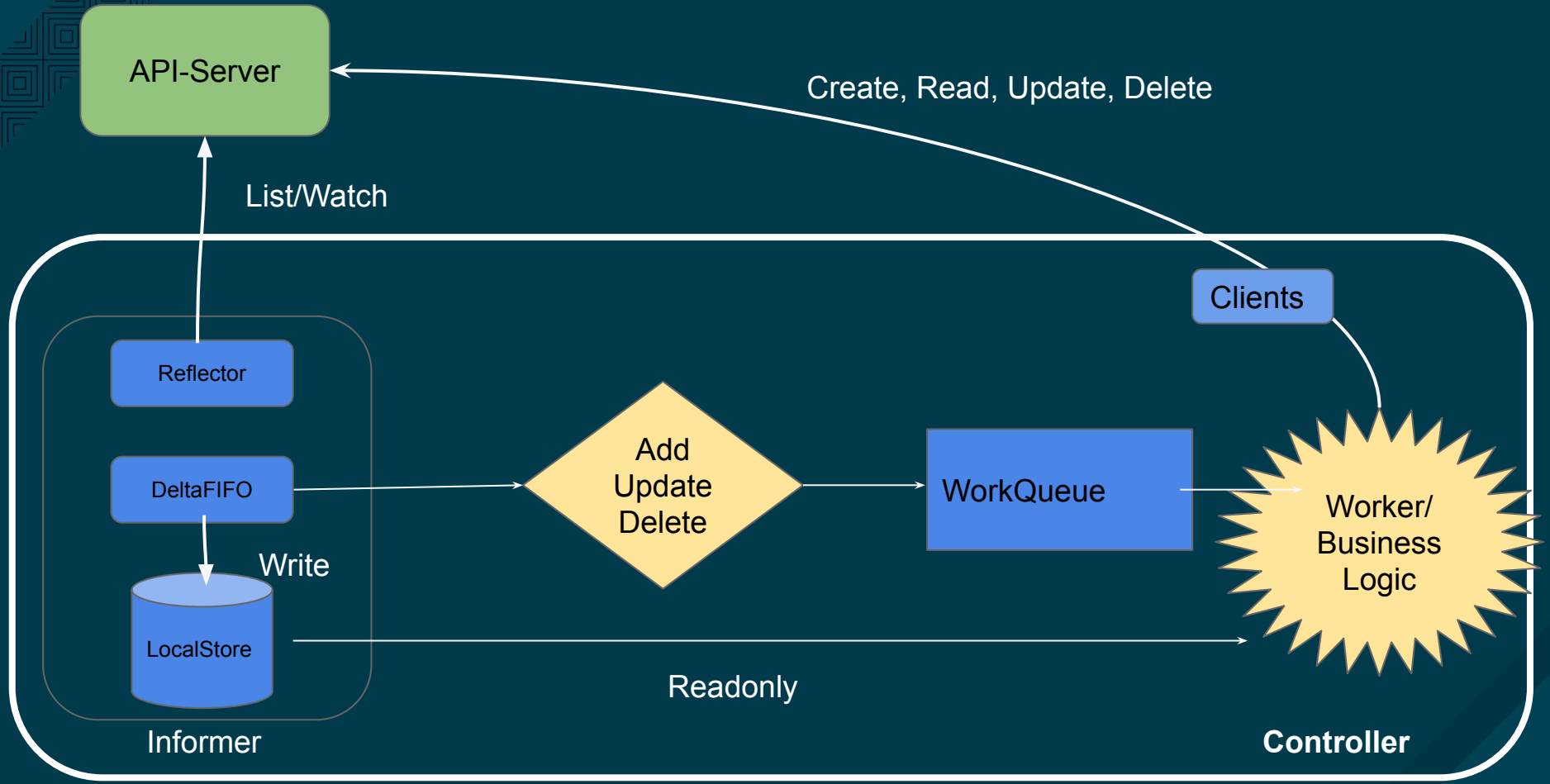
2 Kubernetes ClientSet

3 RESTClient

4 k8s.io/api-machinery

Retrieve the Corev1 Client via clientset and get **spec** for **individual pod** in the **default** namespace.

```
for {
        pod,  err := clientset.CoreV1().Pods("default").Get(“my-pod”, metav1.GetOptions{})
        if err != nil {
                panic(err.Error())
        }
        fmt.Printf("%v\n\n\n\n", pod.spec)
```


API-Server
Create, Read, Update, Delete
List/Watch
Clients
Reflector
DeltaFIFO
Write
LocalStore
Add
Update
Delete
WorkQueue
Worker/
Business
Logic
Readonly
Informer
Controller

```
1    while true {
2    receiveInfoAboutAPIObjects()
3    synchronizeRealStateToMatchFetchedInfo()
4    }
```

redhat

Kubernetes Concepts

redhat

What is a Kubernetes Resource?

Most Common Definition...

Any individual Kubernetes item such as a deployment, pod, service, or secret, etc.

Kubernetes Resources

- Nodes
- Namespaces
- Pods
- Endpoints
- Services
- Deployments
- ReplicaSets
- Persistent Volumes
- PersistentVolumeClaims
- ConfigMaps
- DaemonSets
- StatefulSets
- Events
- PodDisruptionBudgets
- PodSecurityPolicies
- ResourceQuotas
- Service Accounts
- HorizontalPodAutoScalers

A Better Definition...

A Kubernetes Resource is a **declarative API** with well defined Schema structure and endpoints.*

*Because the structure of the Schema and Endpoints are predictable and structured, most Kubernetes tools work with any Kubernetes API even if they are not part of the core (e.g. extensions through CRDs).

```
oc proxy
```

```
curl localhost:8001
```

What is a Declarative API?

Declarative vs. Imperative API

- Declarative expresses a fixed state that the cluster must continually work towards.
- “What to Do”
  - Example: `$ replicas 3`
- Imperative API expresses an operation that may change state but does not define an absolute state that must be maintained.
- “How to Do It”
  - Example: `$ add-pods 2`

ReplicaSet Manifest

Resource Schema Components

GVK aka TypeMeta

Metadata aka ObjectMeta

Spec

Status

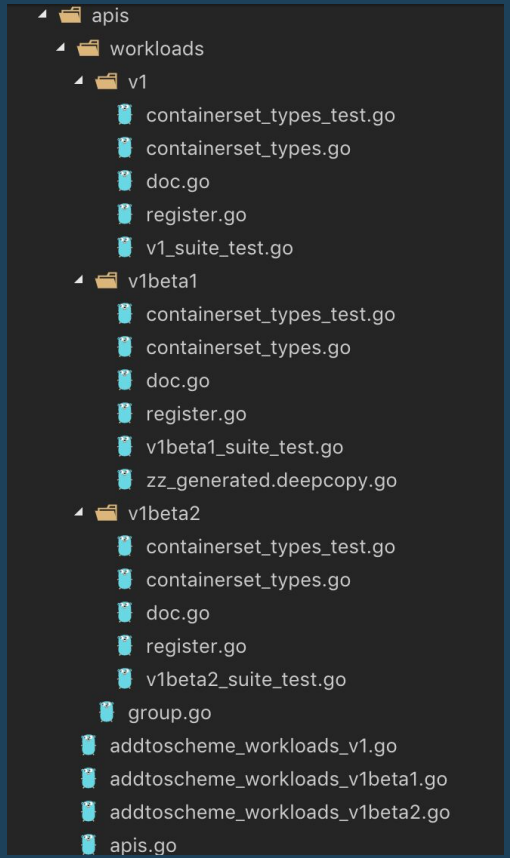
```
apiVersion: extensions/v1beta1
kind: ReplicaSet

metadata:
  name: my-first-replica-set
  namespace: myproject
spec:
  selector:
    matchLabels:
     app: nginx
  replicas: 5
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
status:
  availableReplicas: 1
  fullyLabeledReplicas: 1
  observedGeneration: 1
  readyReplicas: 1
  replicas: 1
```

Resource Schema: Group, Version, Kind (GVK)

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
```

- The resource ***Group*** is similar to package in a language. It disambiguates different APIs that may happen to have identically named Kinds. Groups often contain a domain name, such as redhat.com.
- The resource ***Version*** defines the stability of the API and backward compatibility guarantees - such as v1beta1 or v1.
- The resource ***Kind*** is the name of the API - such as Deployment or Service.

A Note About API Versions



Difference between API Version Numbers

i.e. apps/v1beta1, apps/v1beta2

- Unspecified fields may have different defaults.
- The same logical fields may have different names.

```
kubectl explain deployments.spec --api-version="apps/v1beta1"
```

```
revisionHistoryLimit   <integer>
The number of old ReplicaSets to retain to allow rollback. This is a pointer to distinguish between explicit zero and not specified.
Defaults to 2.
```

```
kubectl explain deployments.spec --api-version="apps/v1beta2"
```

```
revisionHistoryLimit   <integer>
The number of old ReplicaSets to retain to allow rollback. This is a pointer to distinguish between explicit zero and not specified.
Defaults to 10.
```

redhat.

API Versions

Alpha (i.e. v1alpha1)

- Disabled by default. Must be enabled via API.
- May contain bugs. Features may be changed or removed. Field names may also be changed and not supported in the future.
- Only use for short-lived testing clusters.

Beta (i.e. v1beta1)

- Enabled by default.
- Considered safe. Backwards compatibility on field names.
- Support for the feature will not be dropped, though details may change.

Stable (i.e. v1,v2)

- Stable versions of features will appear in many subsequent versions.

Not Flexible

```
http://kubernetes:6443/api/v1/pods

http://kubernetes:6443/api/v1/replicasets

http://kubernetes:6443/api/v1/services

http://kubernetes:6443/api/v1/deployments
```

Flexible

```
curl kubernetes:6443
```

```
"/api/v1"
"/apis/authentication.k8s.io/v1"
"/apis/authentication.k8s.io/v1beta1"
"/apis/authorization.k8s.io/v1"
"/apis/authorization.k8s.io/v1beta1"
"/apis/certificates.k8s.io/v1beta1"
"/apis/certificates.k8s.io"
"/apis/extensions/v1beta1"
"/apis/policy/v1beta1"
"/apis/rbac.authorization.k8s.io/v1beta1"
"/apis/rbac.authorization.k8s.io/v1alpha1"
"/apis/storage.k8s.io/v1"
"/apis/storage.k8s.io/v1beta1"
```

Allows the program to move, change, and grow over time.

Engineers can advertise to support older API versions, and offer backward-compatibility guarantees.

See Current API-Versions

```
oc api-versions
```

```
"/api/v1"
"/apis/authentication.k8s.io/v1"
"/apis/authentication.k8s.io/v1beta1"
"/apis/authorization.k8s.io/v1"
"/apis/authorization.k8s.io/v1beta1"
"/apis/certificates.k8s.io/v1beta1"
"/apis/certificates.k8s.io"
"/apis/extensions/v1beta1"
"/apis/policy/v1beta1"
"/apis/rbac.authorization.k8s.io/v1beta1"
"/apis/rbac.authorization.k8s.io/v1alpha1"
"/apis/storage.k8s.io/v1"
"/apis/storage.k8s.io/v1beta1"
```

News Snippet About Introduction of v1 NetworkPolicy

Two of the changes you need to be aware of are:

» The v1beta1 NetworkPolicy API Has Been Deprecated

The v1beta1 version of the NetworkPolicy API has been deprecated in favor of moving forward with the new behaviors and updating the behavior of the *extensions* API to allow for future expansion and development.Keep in mind that while the v1 NetworkPolicy API eclipses the existing beta, the new API endpoint will only be available on Kubernetes 1.7+ (as older versions do not include the v1 API code). As such, as you work towards upgrading, you'll want to ensure that you are using the correct version of Project Calico for the NetworkPolicy behavior you want.

» The DefaultDeny Annotation Has Been Removed

One of the bigger changes in Kubernetes 1.7 is the removal of the DefaultDeny annotation. This means that when upgrading, you should **first delete any existing NetworkPolicy** objects in namespaces that previously **did not have** the "DefaultDeny" annotation (as this may cause Kubernetes to unintentionally block traffic now).

Kubernetes API Actions and HTTP Method

Verb (API)	HTTP Method
Get	GET
List	GET
Watch	GET
Create	POST
Update	PUT
Patch	PATCH
Delete	DELETE

Kubernetes Subcommand & HTTP Method

Subcommand	Object does not exist	Object exists
apply	POST	PATCH/DELETE
create	POST	error!
replace	error!	PUT
delete	error!	DELETE
patch	error!	PATCH

Interacting with the API


```
kubectl/oc create -f podmanifest.json
```

```
curl -X POST http://localhost:8001/api/v1/namespaces/myproject/pods/ -H
"Content-type: application/json" -d @podmanifest.json
```


```
kubectl/oc replace -f podmanifest.json
```

```
curl -X PUT http://localhost:8001/api/v1/namespaces/myproject/pods/mypod -H
"Content-type: application/json" -d @newpodmanifest.json
```


```
kubectl/oc patch -f patch.json
```

```
kubectl patch etcdcluster example-etcd-cluster --type='json' -p '[{"op":
"replace", "path": "/spec/size", "value":5}]'
```

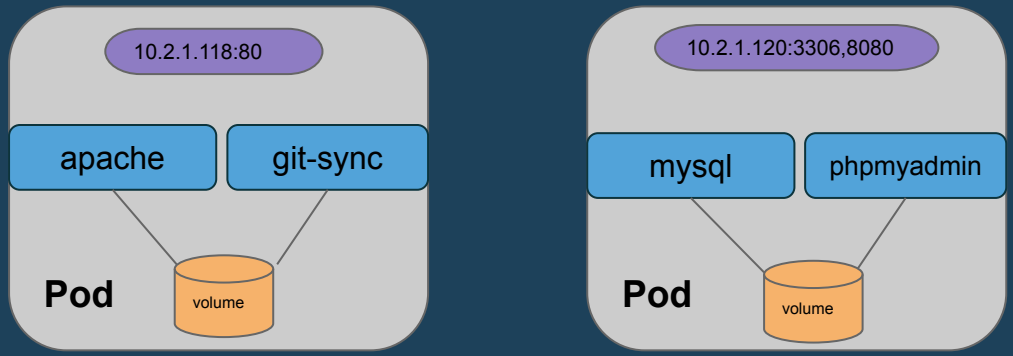
redhat.

Labels/Selectors

Key/value pairs attached to resources.

Used for grouping, viewing, and operating.

Labels: Grouping




```
labels:
  name: apache
  app: mynewapp
  role: frontend
```

```
labels:
  name: mysql
  app: mynewapp
  role: db
```

Labels: Viewing

```
kubectl get pods --show-labels
```

```
db-dev                          1/1       Running             0          6s        app=my-app,environment=dev,tier=backend
www-dev                         1/1       Running             0          6s        app=my-app,environment=dev,tier=frontend
www-prod                        1/1       Running             0          6s        app=my-app,environment=production,tier=frontend
```

```
kubectl get pods -L app,environment,tier -l environment!=dev
```

```
www-prod                        1/1       Running             0          4m        my-app    production    frontend
```

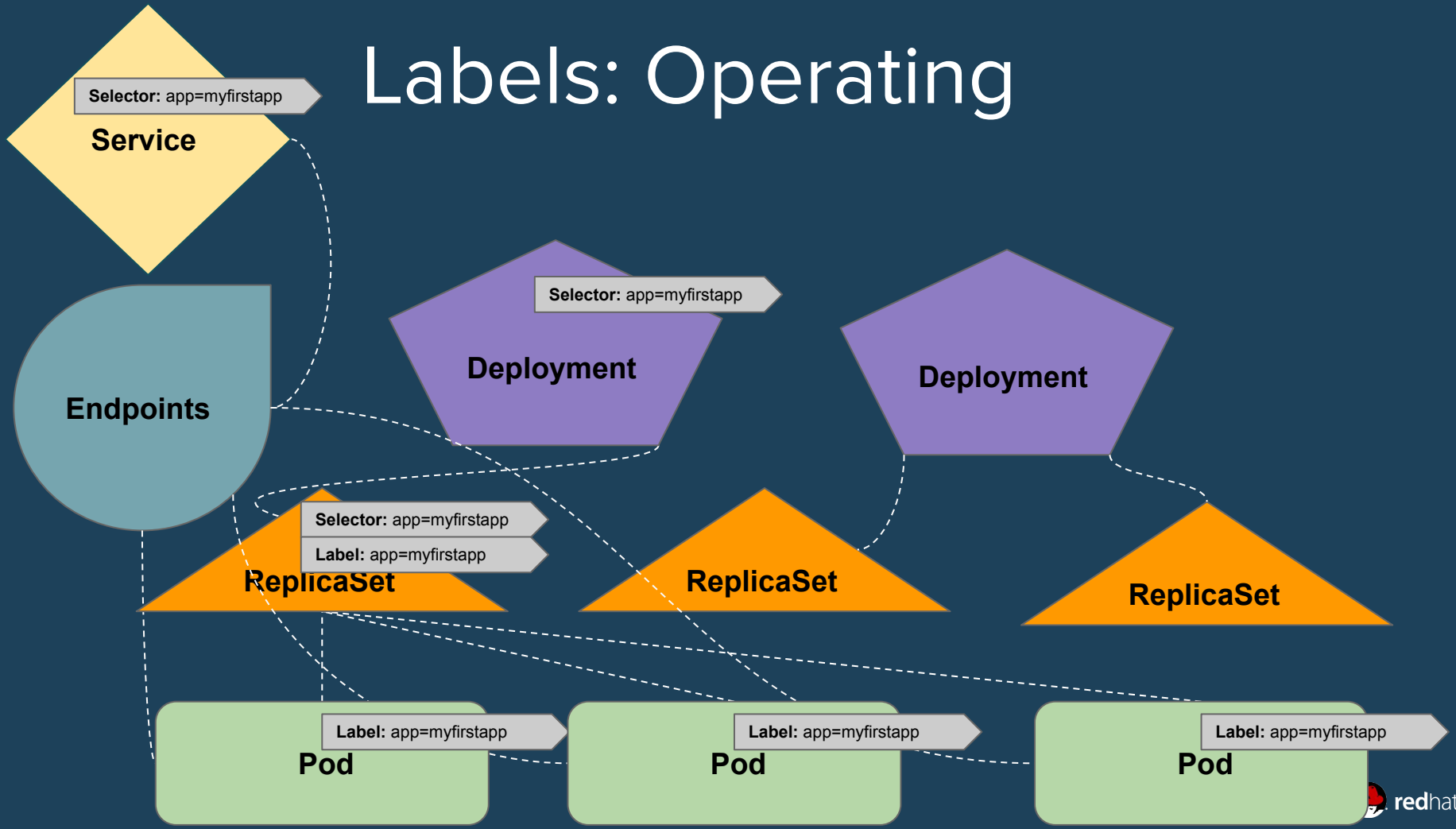
```
kubectl get pods -l "tier notin (backend,cache),environment in (dev)"
```

```
www-dev                         1/1       Running   0          6m
```

We will Return at

12:00pm CT
11:00am UTC

Labels: Operating



Using Labels

Labels can be displayed as a column in output with -L option:

```
$ kubectl get pods -L tier
NAME                          READY     STATUS    RESTARTS   AGE       TIER
my-nginx-3800858182-1v53o     1/1       Running   0          46s       backend
my-nginx-3800858182-2ds1q     1/1       Running   0          46s       backend
```

http://kubernetes.io/docs/user-guide/labels/

Updating Labels

Sometimes existing pods and other resources need to be relabeled before creating new resources

```
$ kubectl label pods -l app=nginx,tier=fe # select by label app=nginx, apply tier=fe
pod "my-nginx-v4-9gw19" labeled
pod "my-nginx-v4-hayza" labeled
pod "my-nginx-v4-mde6m" labeled
pod "my-nginx-v4-sh6m8" labeled
pod "my-nginx-v4-wfof4" labeled

$ kubectl get pods -l app=nginx -L tier
NAME                READY     STATUS    RESTARTS   AGE       TIER
my-nginx-v4-9gw19   1/1       Running   0          15m       fe
my-nginx-v4-hayza   1/1       Running   0          14m       fe
my-nginx-v4-mde6m   1/1       Running   0          18m       fe
my-nginx-v4-sh6m8   1/1       Running   0          19m       fe
my-nginx-v4-wfof4   1/1       Running   0          16m       fe
```

Using Labels Effectively

Examples of multiple labels for app, tier and role:

```
labels:
        app: guestbook
        tier: frontend
```

```
labels:
        app: guestbook
        tier: backend
        role: master
```

```
labels:
        app: guestbook
        tier: backend
        role: slave
```

Other example labels:

- "release" : "stable" or "canary"
- "partition" : "customerA" or "customerB"
- "track" : "daily" or "weekly"

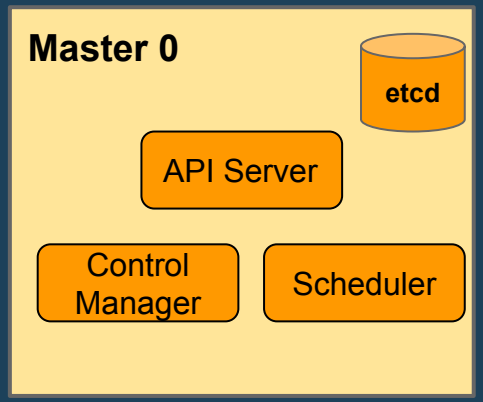
What is a Declarative API?

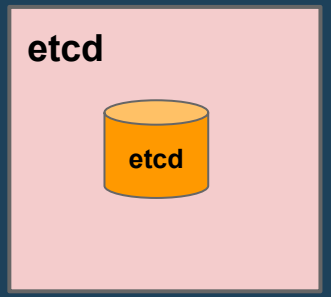
Cluster

A group of servers (or virtual machines) configured to run a functioning Kubernetes system.

Kubernetes Cluster

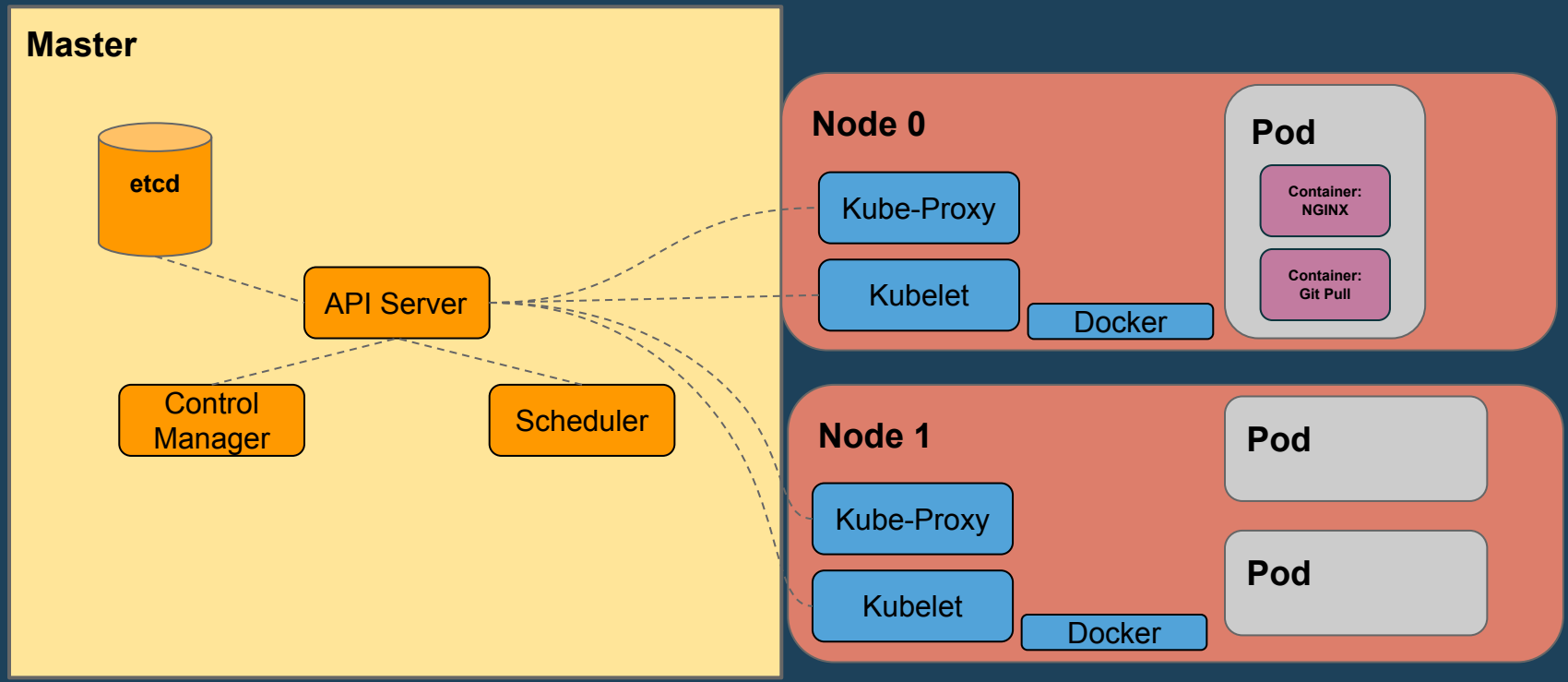
Kubernetes Master





Also known as a *controller* or *head* node.

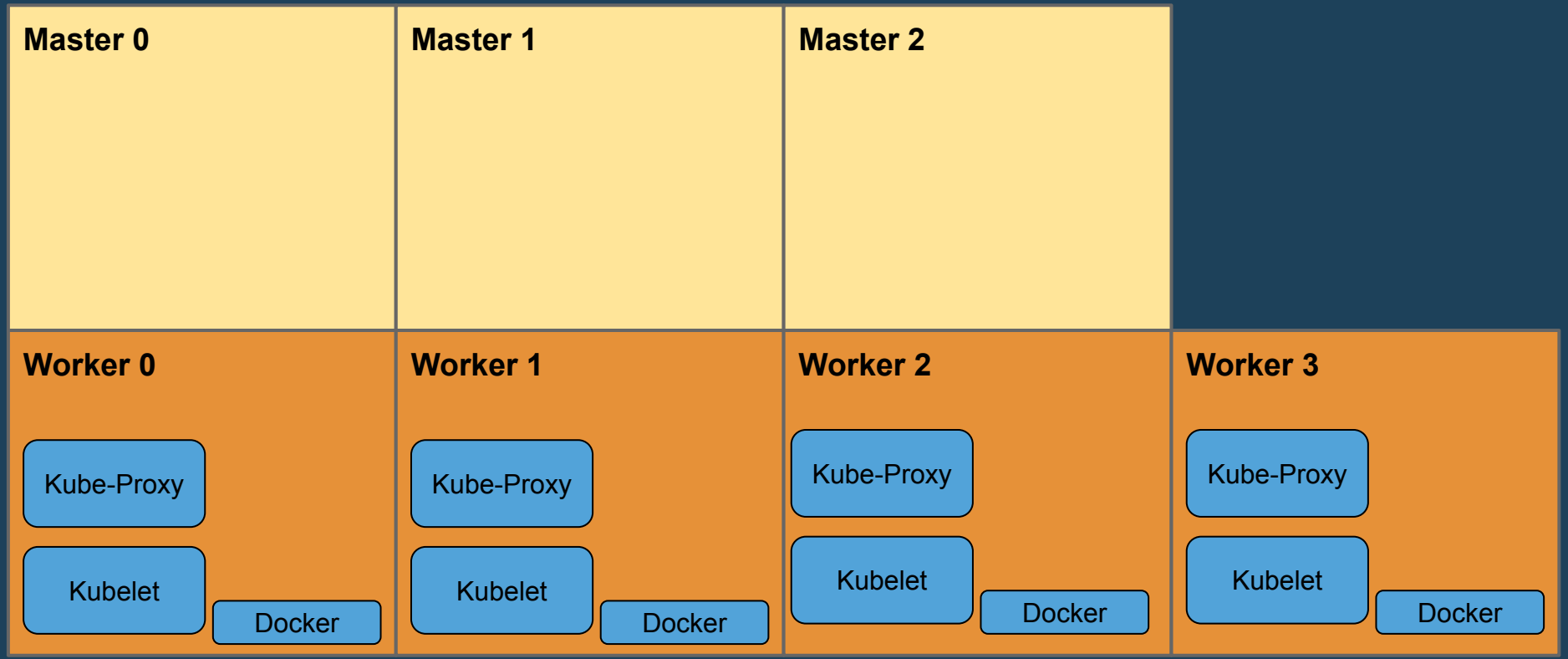
Kubernetes Cluster



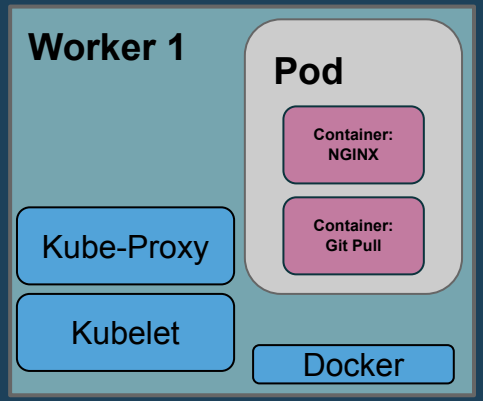
Node

A worker machine (previously known as a *minion*). Contains the services necessary to run pods.

Kubernetes Node/Worker



Kubernetes Node/Worker



Previously known as minion.

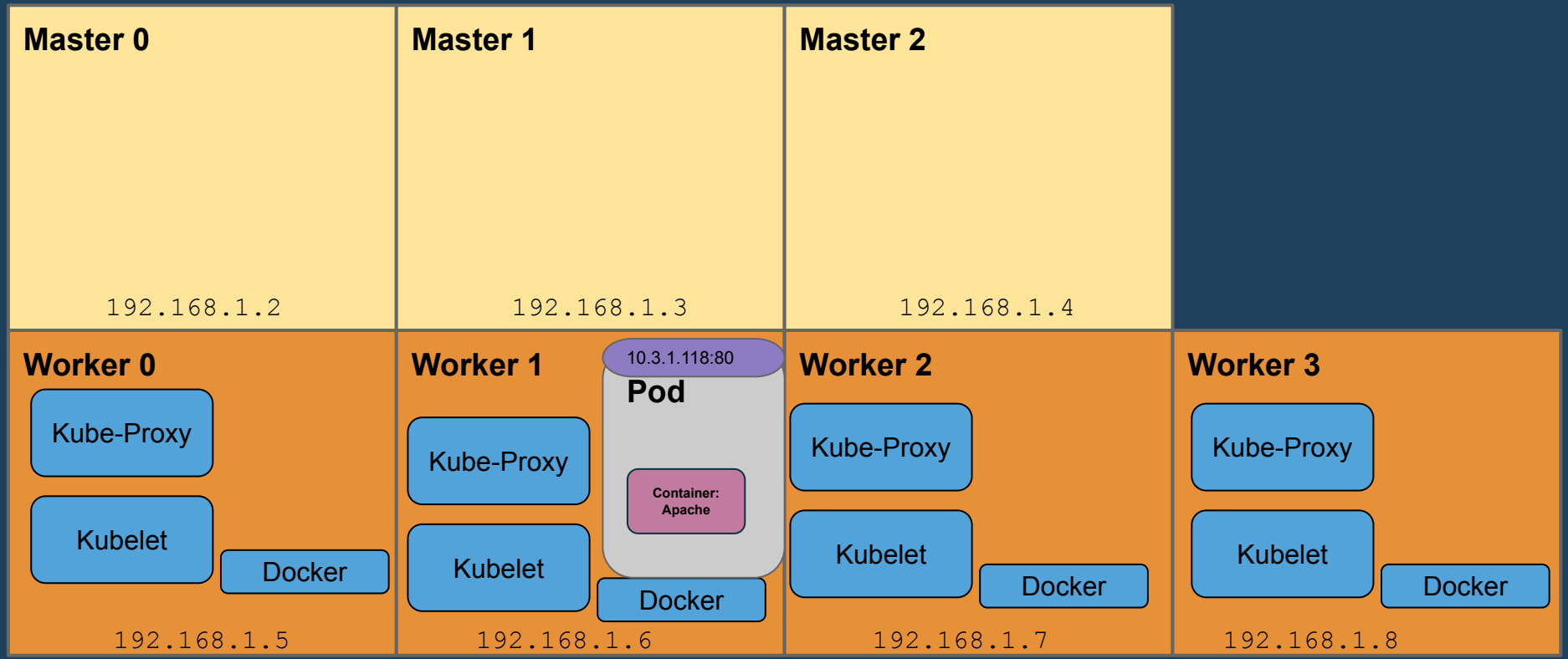
Pod

A group of one or more containers running on a single node.

Pods are the smallest deployable units of computing that can be created and managed in Kubernetes.

```
kubectl run my-first-pod --image=apache --restart=Never --port=80
```

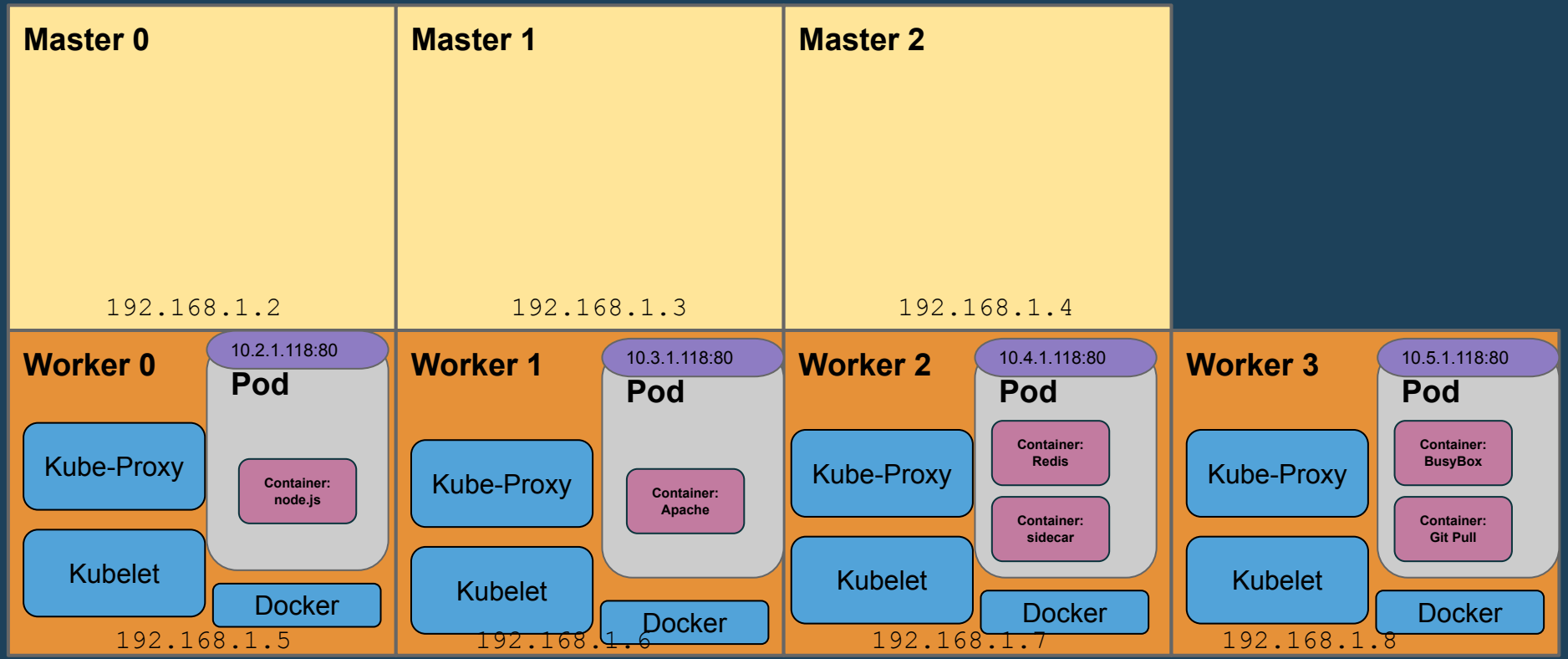
Kubernetes Pod



redhat.

```
kubectl run my-first-pod --image=apache --restart=Always --port=80
```

Kubernetes Pods



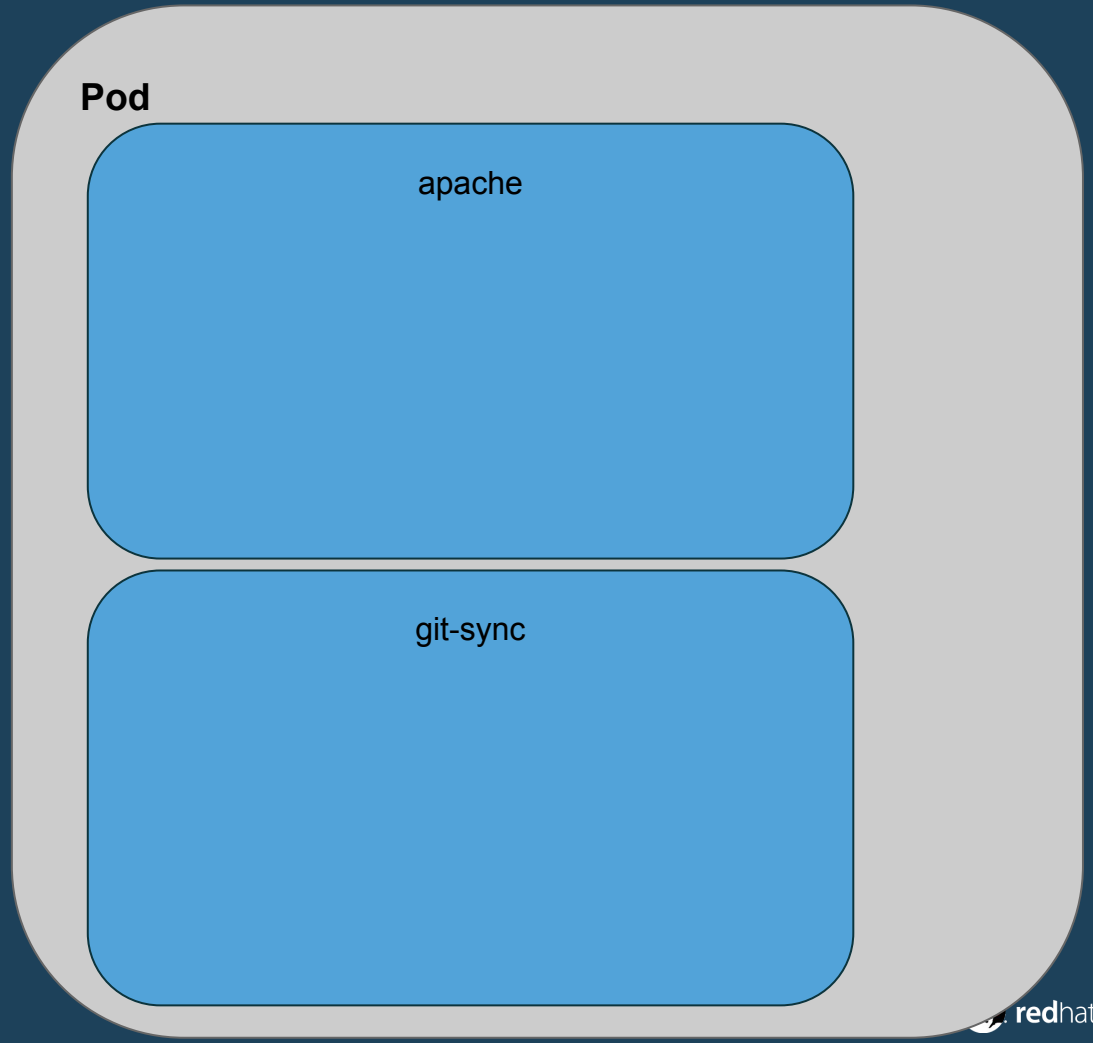
redhat.

Let’s take a closer look at a pod.

About Pods

Contain one or more containers.

These “extra” containers are commonly called “side-cars”.



What can a sidecar container do?

Data puller.

Data watcher.

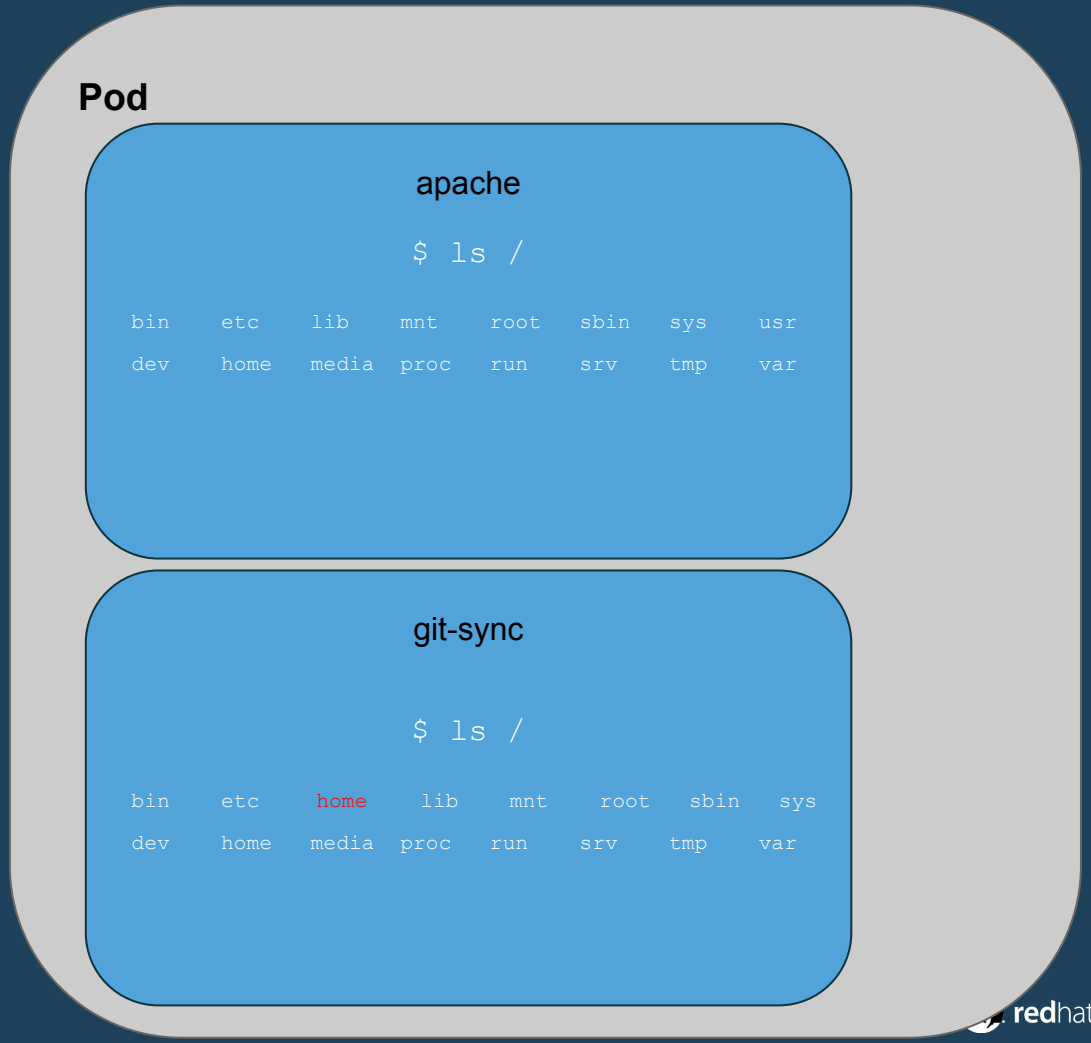
Log watcher.

Reverse Proxy (Log/Modify Requests)

What is separated/isolated?

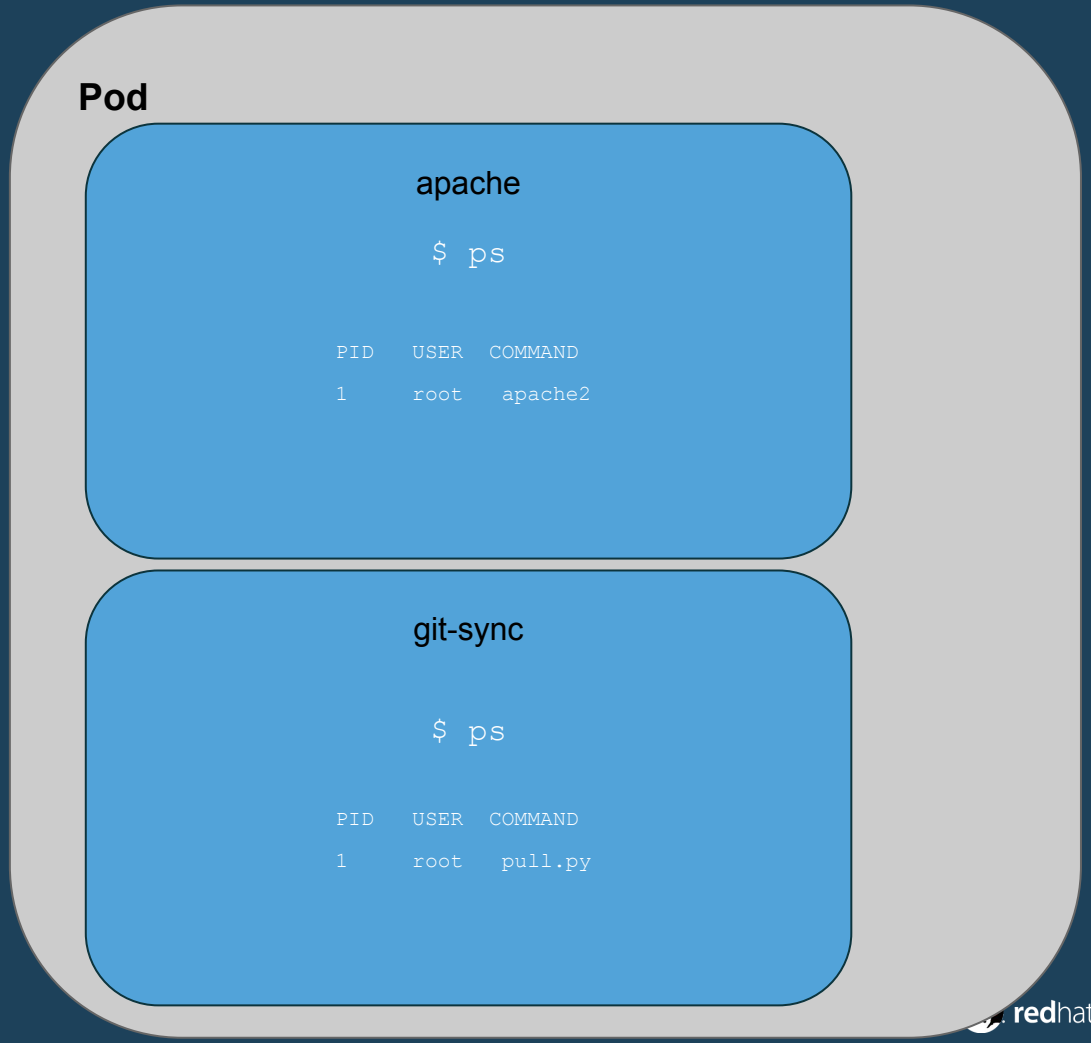
Mnt Namespaces

Separate filesystems.



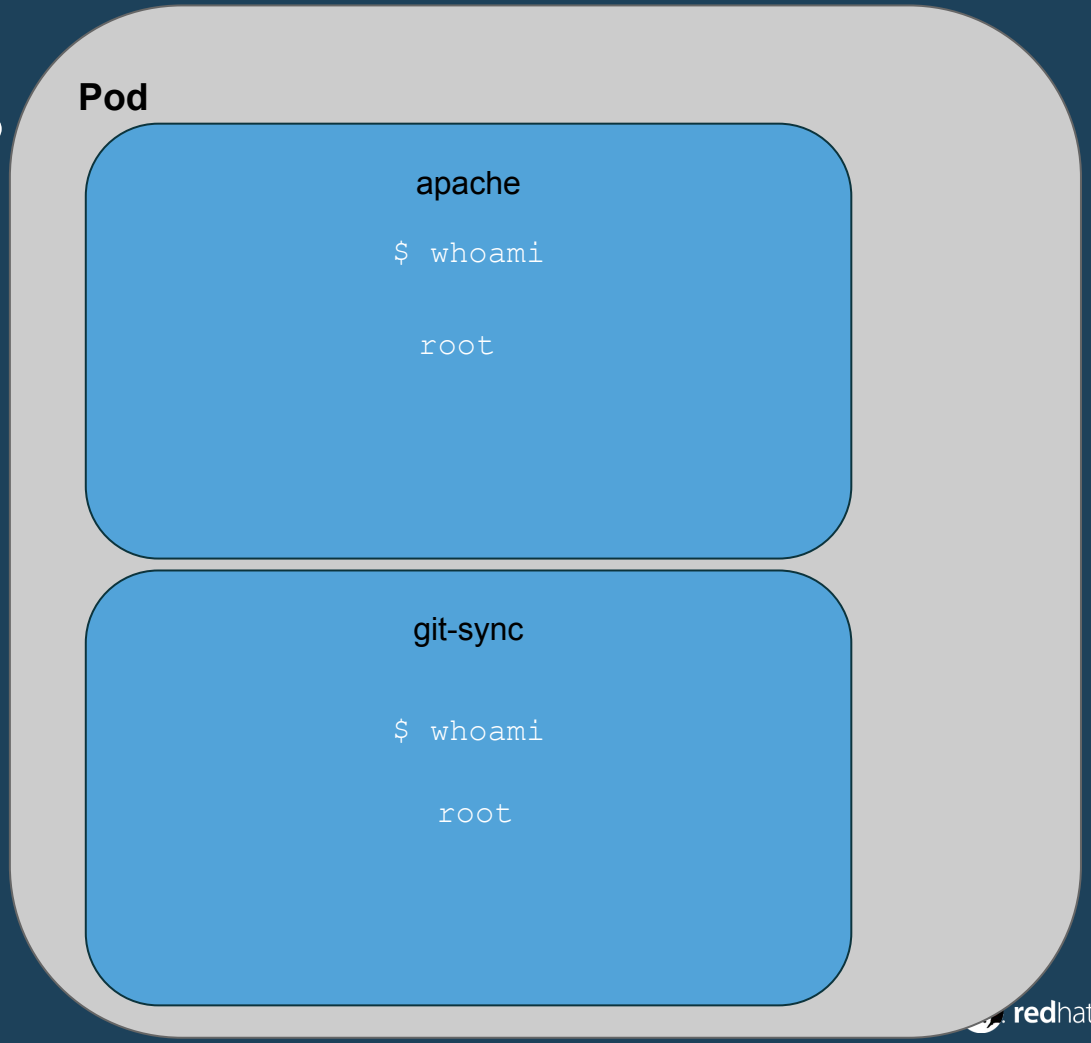
PID Namespaces

Separate processes.



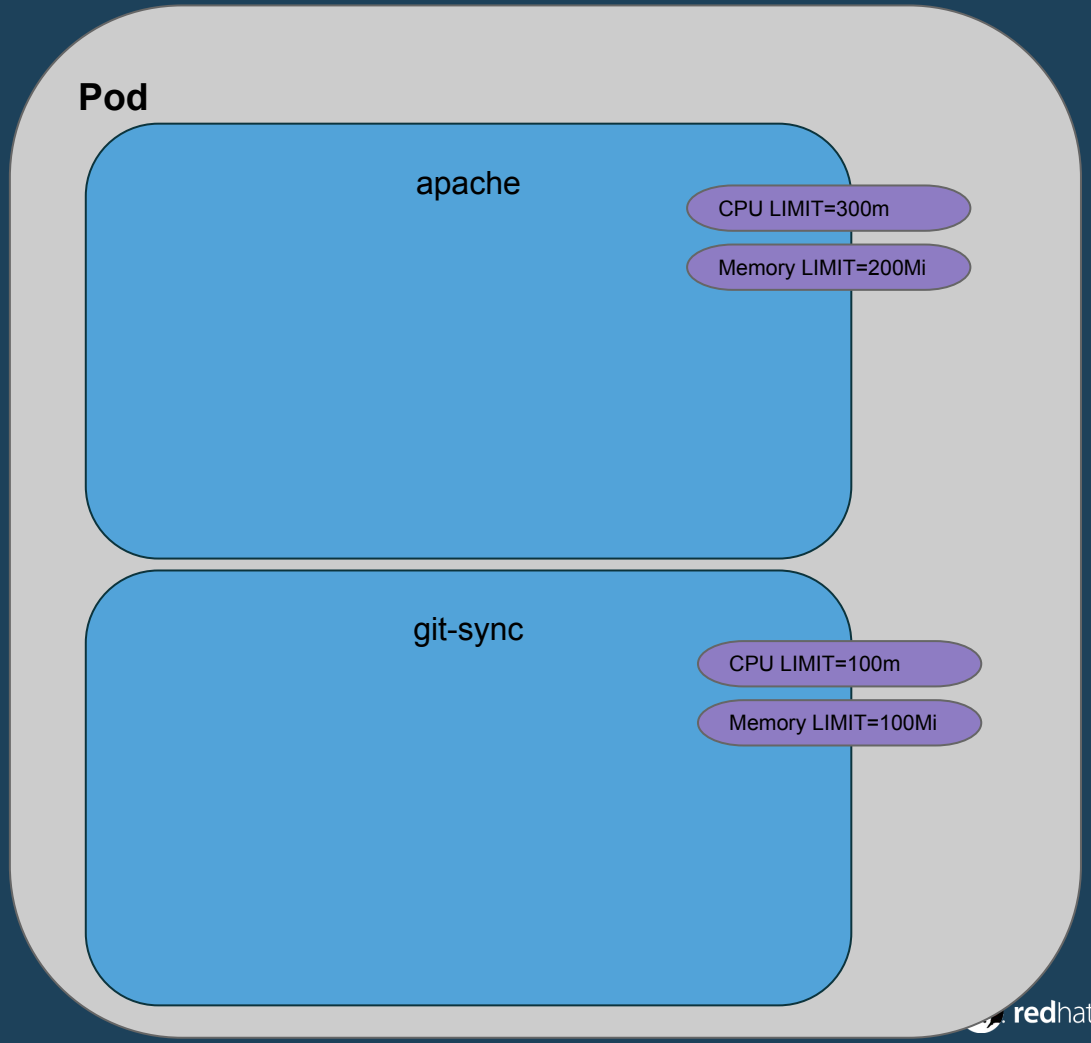
User Namespaces

Separate user namespaces.



Cgroups

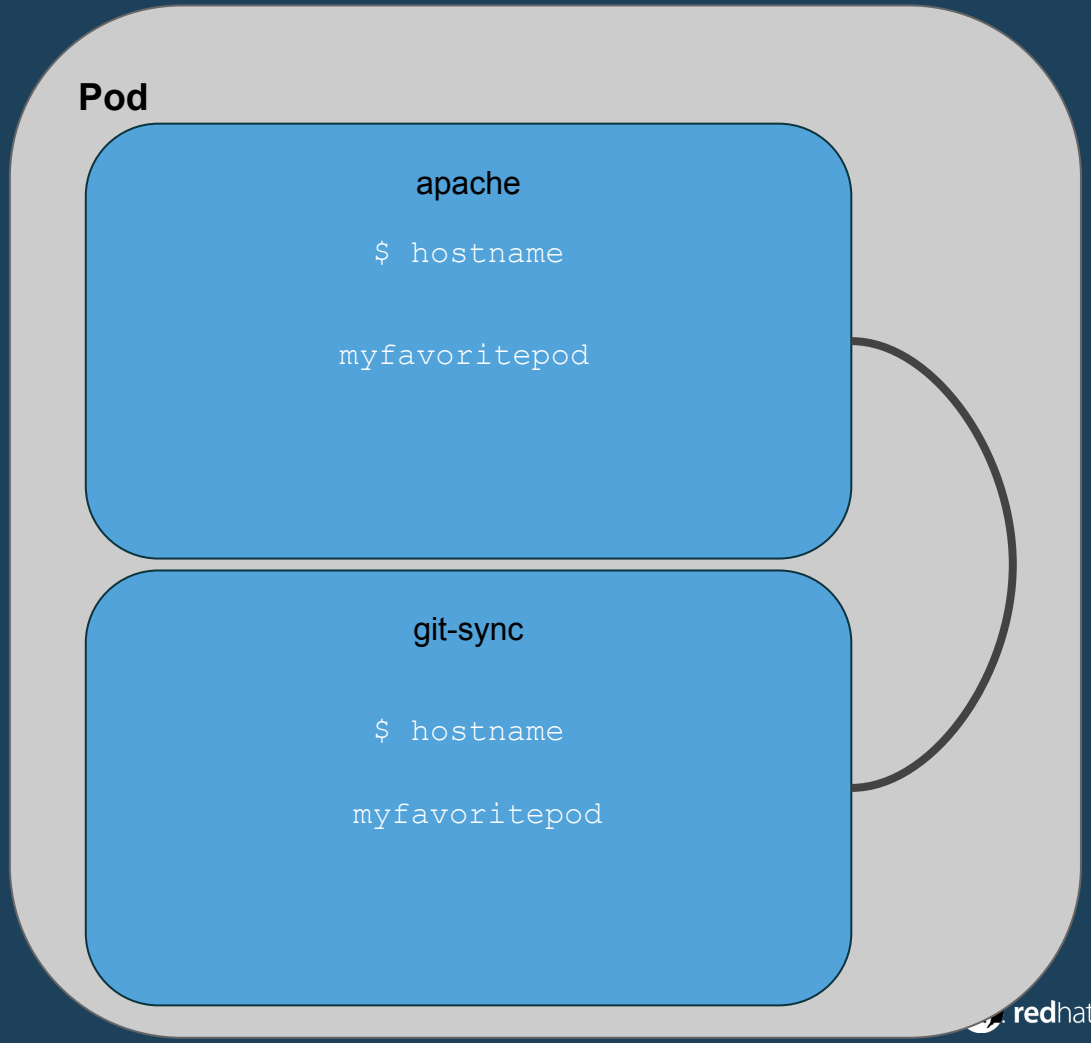
Each container has its own cpu/memory cgroup.



What is shared?

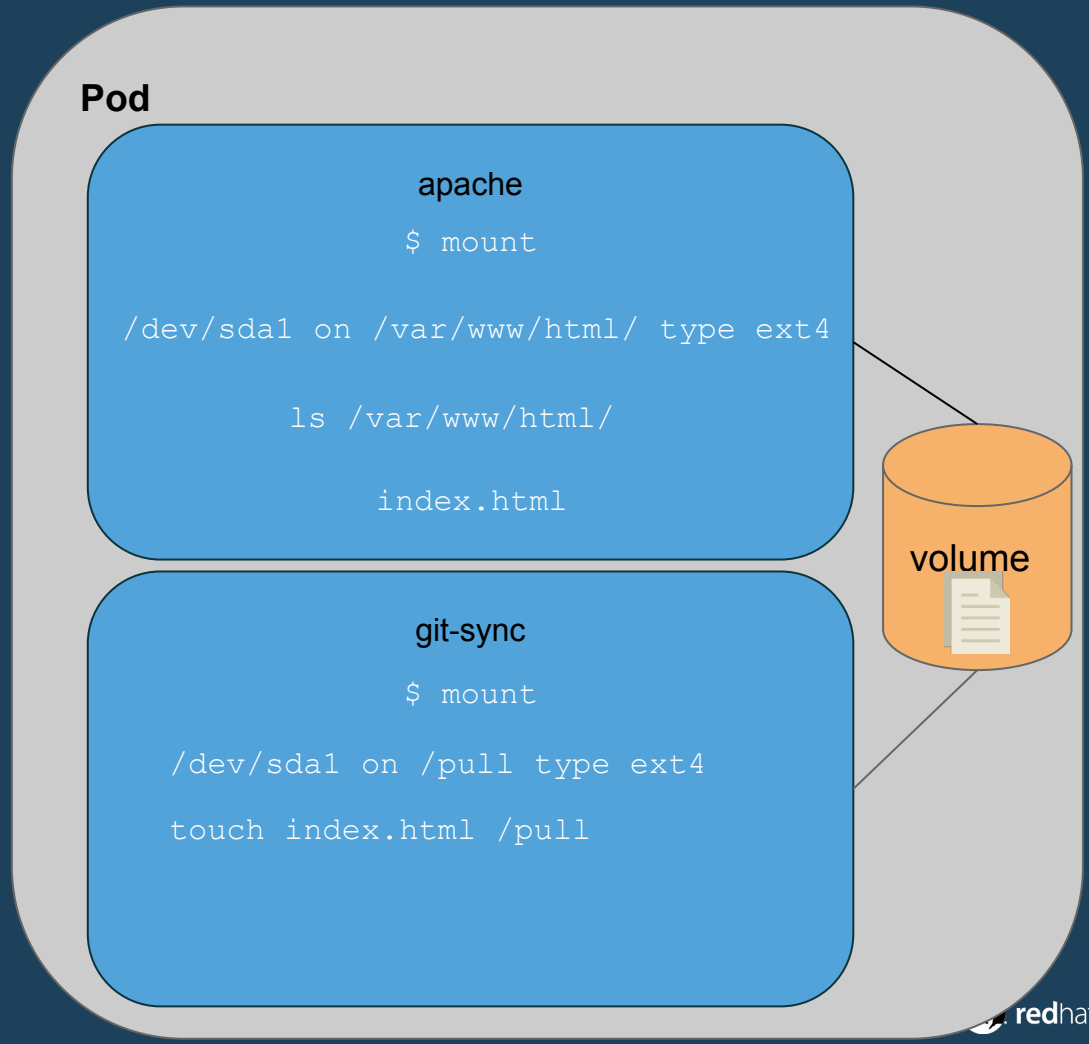
UTS Namespaces

Same hostname.



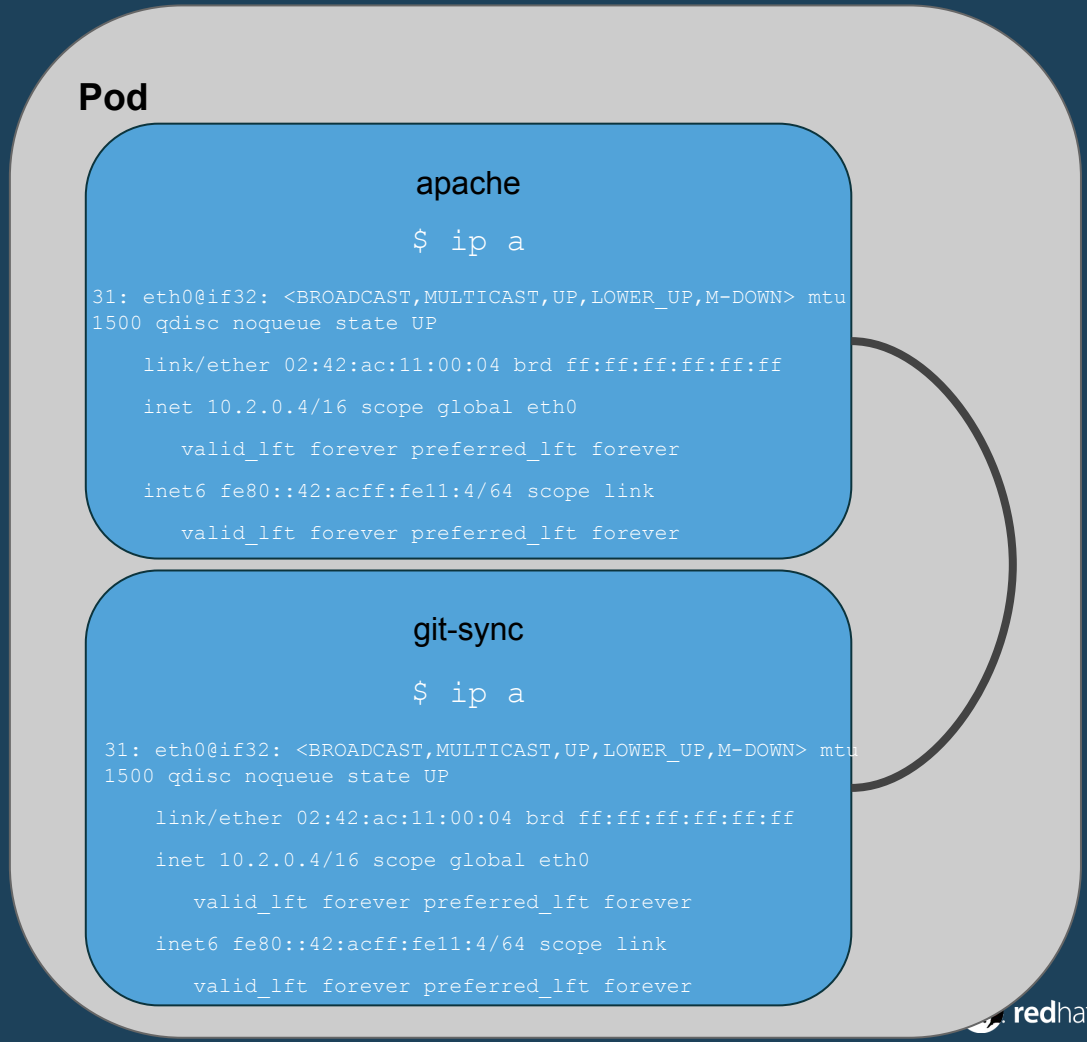
Volumes

Share the same volume but mounted as different directories.



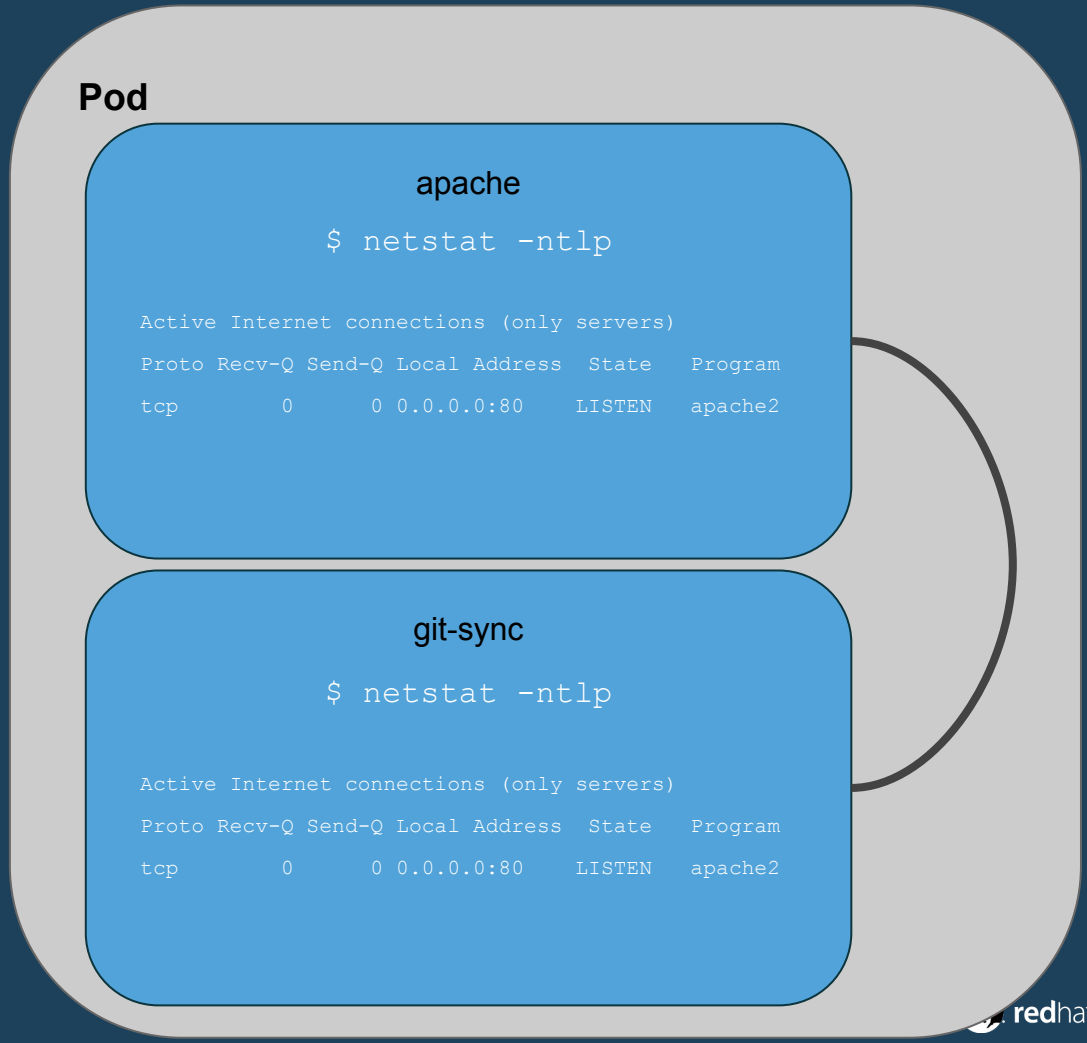
Net Namespaces

Same IP address.



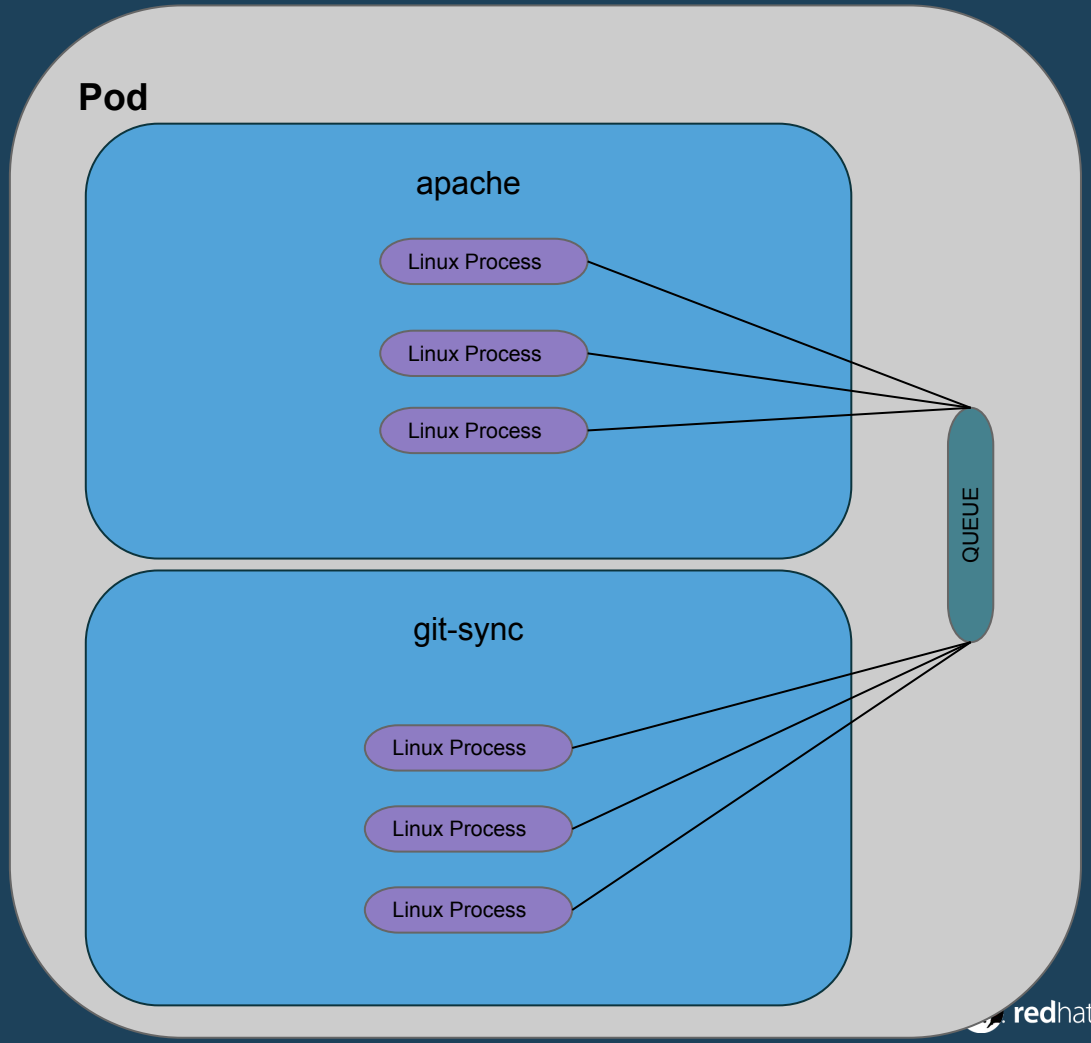
Net Namespaces

Same network sockets.



IPC Namespace

Containers can communicate via SystemV or POSIX shared memory, semaphores, or messages.



`kubectl run`

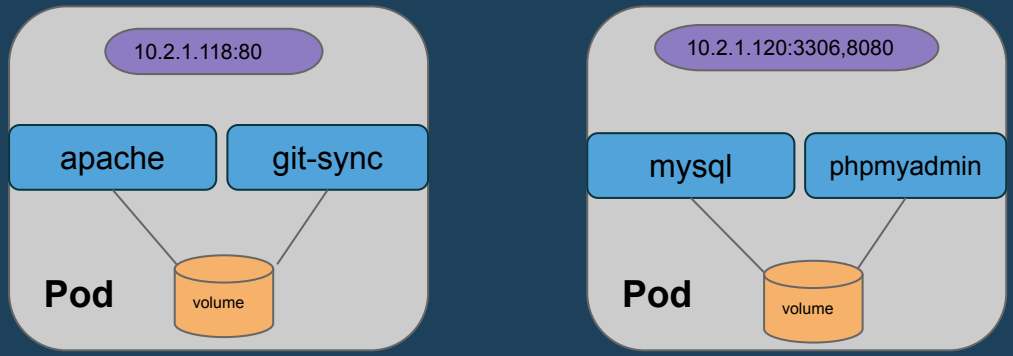
- ‘Off the cuff’ command to create **pods**, **deployments**, or **jobs**.
- Great for testing and troubleshooting.
- Similar to “docker run” in usage.

Labels/Selectors

Key/value pairs attached to resources.

Used for grouping, viewing, and operating.

Labels: Grouping




```
labels:
  name: apache
  app: mynewapp
  role: frontend
```

```
labels:
  name: mysql
  app: mynewapp
  role: db
```

Labels: Viewing

```
kubectl get pods --show-labels
```

```
db-dev                          1/1       Running             0          6s        app=my-app,environment=dev,tier=backend
www-dev                         1/1       Running             0          6s        app=my-app,environment=dev,tier=frontend
www-prod                        1/1       Running             0          6s        app=my-app,environment=production,tier=frontend
```

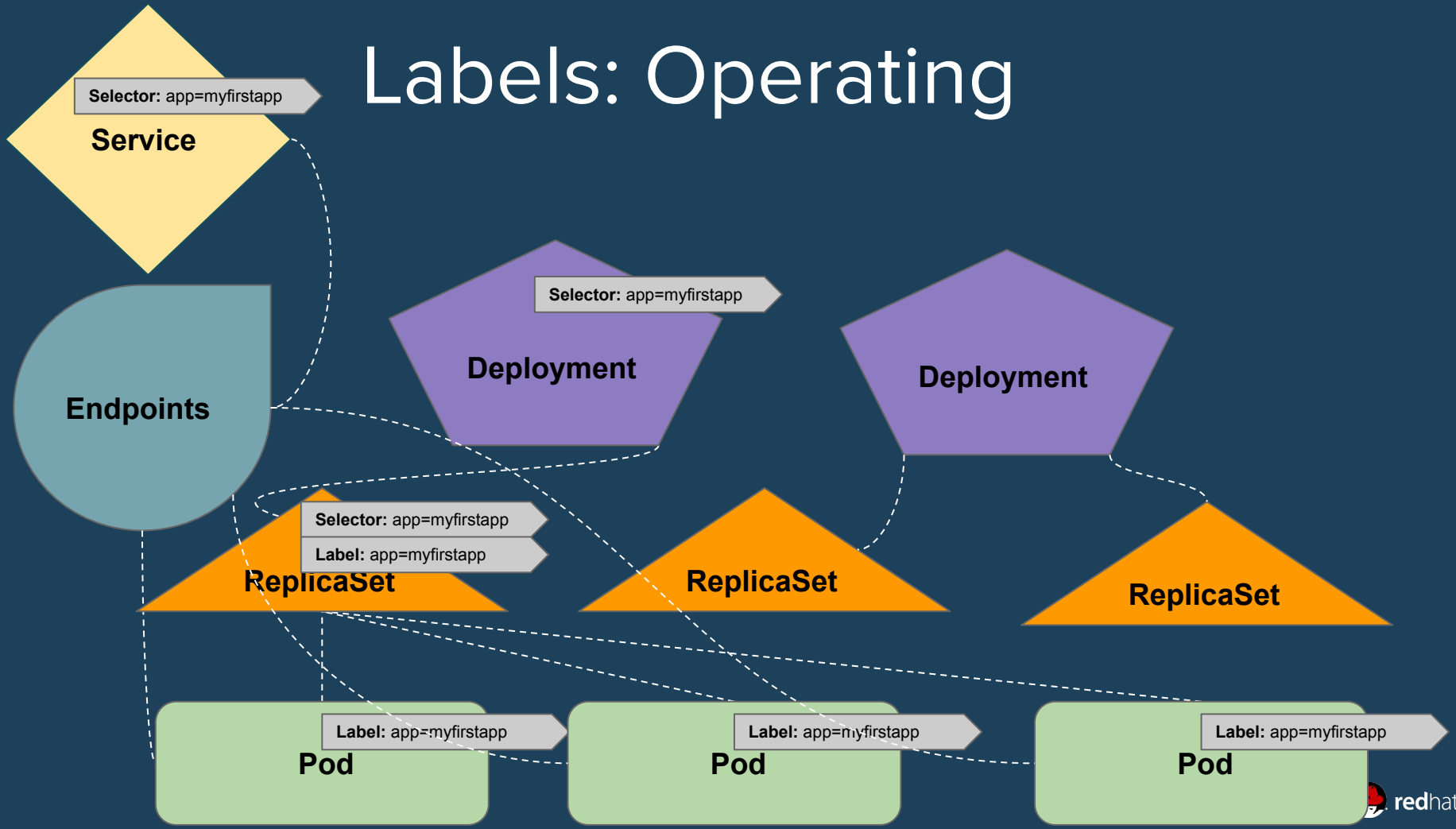
```
kubectl get pods -L app,environment,tier -l environment!=dev
```

```
www-prod                        1/1       Running             0          4m        my-app    production    frontend
```

```
kubectl get pods -l "tier notin (backend,cache),environment in (dev)"
```

```
www-dev                         1/1       Running   0          6m
```

Labels: Operating



Using Labels

Labels can be displayed as a column in output with -L option:

```
$ kubectl get pods -L tier
NAME                          READY     STATUS    RESTARTS   AGE       TIER
my-nginx-3800858182-1v53o     1/1       Running   0          46s       backend
my-nginx-3800858182-2ds1q     1/1       Running   0          46s       backend
```

http://kubernetes.io/docs/user-guide/labels/

Updating Labels

Sometimes existing pods and other resources need to be relabeled before creating new resources

```
$ kubectl label pods -l app=nginx,tier=fe # select by label app=nginx, apply tier=fe
pod "my-nginx-v4-9gw19" labeled
pod "my-nginx-v4-hayza" labeled
pod "my-nginx-v4-mde6m" labeled
pod "my-nginx-v4-sh6m8" labeled
pod "my-nginx-v4-wfof4" labeled

$ kubectl get pods -l app=nginx -L tier
NAME                READY     STATUS    RESTARTS   AGE       TIER
my-nginx-v4-9gw19   1/1       Running   0          15m       fe
my-nginx-v4-hayza   1/1       Running   0          14m       fe
my-nginx-v4-mde6m   1/1       Running   0          18m       fe
my-nginx-v4-sh6m8   1/1       Running   0          19m       fe
my-nginx-v4-wfof4   1/1       Running   0          16m       fe
```

Using Labels Effectively

Examples of multiple labels for app, tier and role:

```
labels:
        app: guestbook
        tier: frontend
```

```
labels:
        app: guestbook
        tier: backend
        role: master
```

```
labels:
        app: guestbook
        tier: backend
        role: slave
```

Other example labels:

- "release" : "stable" or "canary"
- "partition" : "customerA" or "customerB"
- "track" : "daily" or "weekly"

Pods using a Configuration File

- Instead of using the **`kubectl run`** command to create a deployment object, you can specify a deployment YAML file
  - kubectl converts the YAML to JSON before sending it to the API Server
- Advantages over **`kubectl run`**:
  - Declarative (what to do) instead of imperative (how to do it)
  - Can track your changes in Git
  - Can have multiple containers in a pod
- Other Kubernetes objects (services, pods, etc.) can also be configured with YAML or JSON files

redhat

Kubernetes API

redhat

Pods using a Configuration File

- Instead of using the **`kubectl run`** command to create a deployment object, you can specify a deployment YAML file
  - kubectl converts the YAML to JSON before sending it to the API Server
- Advantages over **`kubectl run`**:
  - Declarative (what to do) instead of imperative (how to do it)
  - Can track your changes in Git
  - Can have multiple containers in a pod
- Other Kubernetes objects (services, pods, etc.) can also be configured with YAML or JSON files

Not Flexible

```
http://kubernetes:6443/api/v1/pods

http://kubernetes:6443/api/v1/replicasets

http://kubernetes:6443/api/v1/services

http://kubernetes:6443/api/v1/deployments
```

More Flexible

```
curl kubernetes:6443
```

```
"/api/v1"
"/apis/authentication.k8s.io/v1"
"/apis/authentication.k8s.io/v1beta1"
"/apis/authorization.k8s.io/v1"
"/apis/authorization.k8s.io/v1beta1"
"/apis/certificates.k8s.io/v1beta1"
"/apis/certificates.k8s.io"
"/apis/extensions/v1beta1"
"/apis/policy/v1beta1"
"/apis/rbac.authorization.k8s.io/v1beta1"
"/apis/rbac.authorization.k8s.io/v1alpha1"
"/apis/storage.k8s.io/v1"
"/apis/storage.k8s.io/v1beta1"
```

API Versions

Alpha (i.e. v1alpha1)

- Disabled by default. Must be enabled via API.
- May contain bugs. Features may be changed or removed.
- Only use for short-lived testing clusters.

Beta (i.e. v1beta1)

- Enabled by default.
- Considered safe.
- Support for the feature will not be dropped, though details may change.

Stable (i.e. v1,v2)

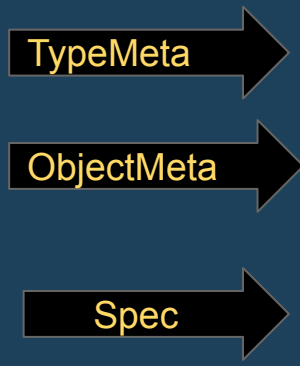
- Stable versions of features will appear in many subsequent versions.

See current Api Versions available.

```
kubectl api-versions
```

```
"/api/v1"
"/apis/authentication.k8s.io/v1"
"/apis/authentication.k8s.io/v1beta1"
"/apis/authorization.k8s.io/v1"
"/apis/authorization.k8s.io/v1beta1"
"/apis/certificates.k8s.io/v1beta1"
"/apis/certificates.k8s.io"
"/apis/extensions/v1beta1"
"/apis/policy/v1beta1"
"/apis/rbac.authorization.k8s.io/v1beta1"
"/apis/rbac.authorization.k8s.io/v1alpha1"
"/apis/storage.k8s.io/v1"
"/apis/storage.k8s.io/v1beta1"
```

TypeMeta, ObjectMeta, Spec




```
apiVersion: extensions/v1beta1
kind: ReplicaSet

metadata:
  name: my-first-replica-set
  namespace: myproject

spec:
  selector:
    matchLabels:
     app: nginx
  replicas: 5
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
```

News Snippet About Introduction of v1 NetworkPolicy

Two of the changes you need to be aware of are:

» The v1beta1 NetworkPolicy API Has Been Deprecated

The v1beta1 version of the NetworkPolicy API has been deprecated in favor of moving forward with the new behaviors and updating the behavior of the *extensions* API to allow for future expansion and development.Keep in mind that while the v1 NetworkPolicy API eclipses the existing beta, the new API endpoint will only be available on Kubernetes 1.7+ (as older versions do not include the v1 API code). As such, as you work towards upgrading, you'll want to ensure that you are using the correct version of Project Calico for the NetworkPolicy behavior you want.

» The DefaultDeny Annotation Has Been Removed

One of the bigger changes in Kubernetes 1.7 is the removal of the DefaultDeny annotation. This means that when upgrading, you should **first delete any existing NetworkPolicy** objects in namespaces that previously **did not have** the "DefaultDeny" annotation (as this may cause Kubernetes to unintentionally block traffic now).

Kubernetes API Actions and HTTP Method

Verb	HTTP Method
Get	GET
List	GET
Watch	GET
Create	POST
Update	PUT
Patch	PATCH
Delete	DELETE

Kubernetes Subcommand & HTTP Method

Subcommand	Object does not exist	Object exists
apply	POST	PATCH/DELETE
create	POST	error!
replace	error!	PUT
delete	error!	DELETE
Patch	PATCH	PATCH
Delete	DELETE	DELETE

Interacting with the API


```
kubectl/oc create -f podmanifest.json
```

```
curl -X POST http://localhost:8001/api/v1/namespaces/myproject/pods/ -H
"Content-type: application/json" -d @podmanifest.json
```


```
kubectl/oc replace -f podmanifest.json
```

```
curl -X PUT http://localhost:8001/api/v1/namespaces/myproject/pods/mypod -H
"Content-type: application/json" -d @newpodmanifest.json
```


```
kubectl/oc patch -f patch.json
```

```
kubectl patch etcdcluster example-etcd-cluster --type='json' -p '[{"op":
"replace", "path": "/spec/size", "value":5}]'
```

redhat

ReplicaSets

redhat

Redundancy

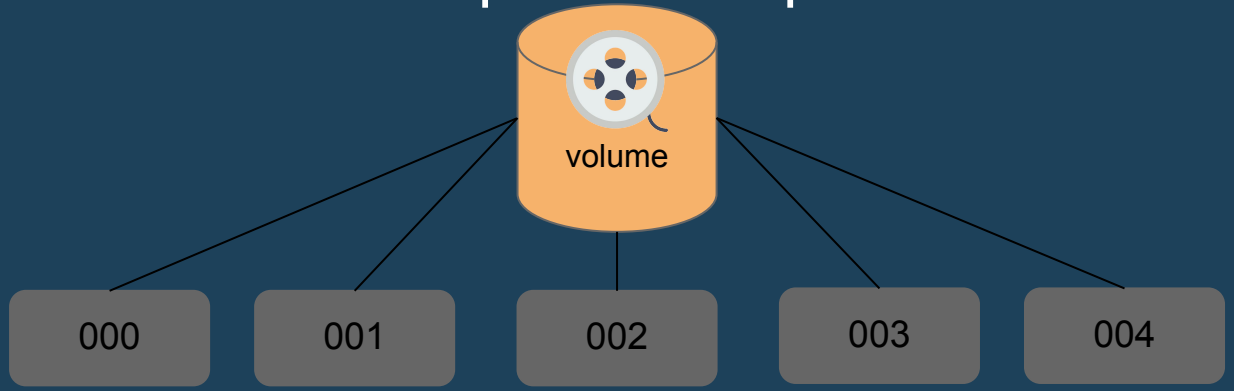
Multiple running instances means failure can be tolerated.

Scale

Multiple running instances mean more requests can be handled.

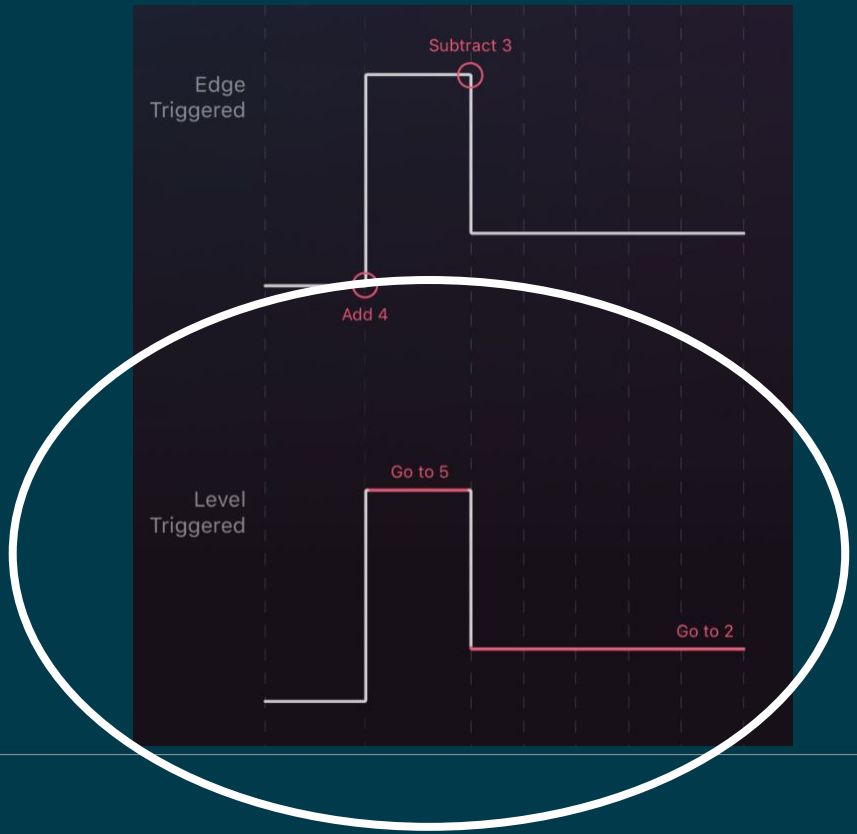
Sharding

Multiple running instances can handle different parts of a computation in parallel.



ReplicaSet is just a Reconcilation Loop.

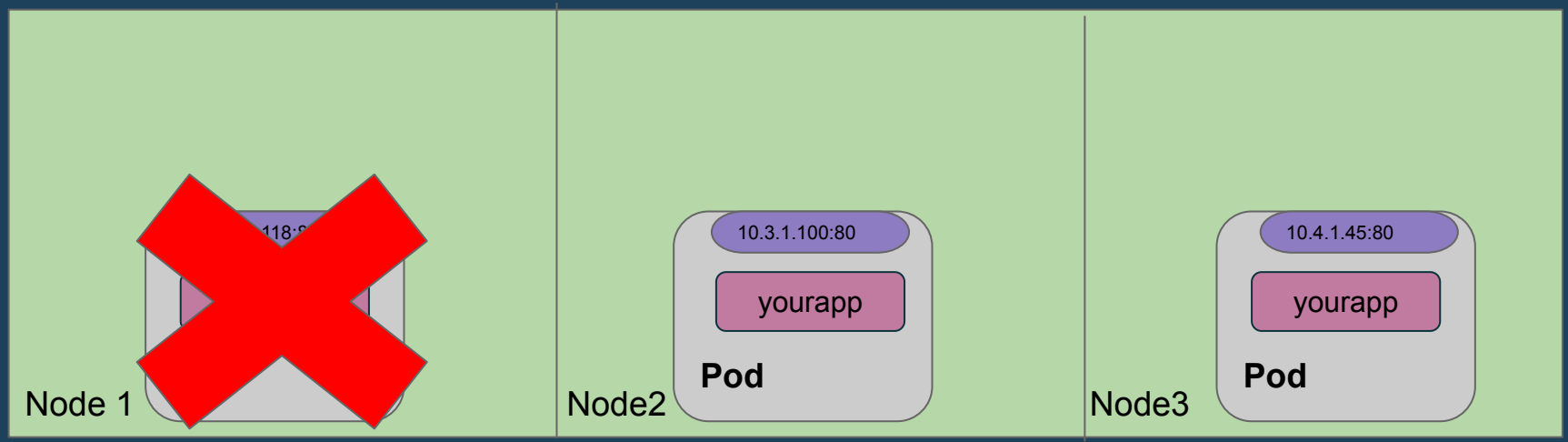
Edge Driven vs. Level Driven.



```
1    while true {
2    receiveInfoAboutAPIObjects()
3    synchronizeRealStateToMatchFetchedInfo()
4    }
```

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: my-first-replica-set
spec:
  selector:
    matchLabels:
     app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
```

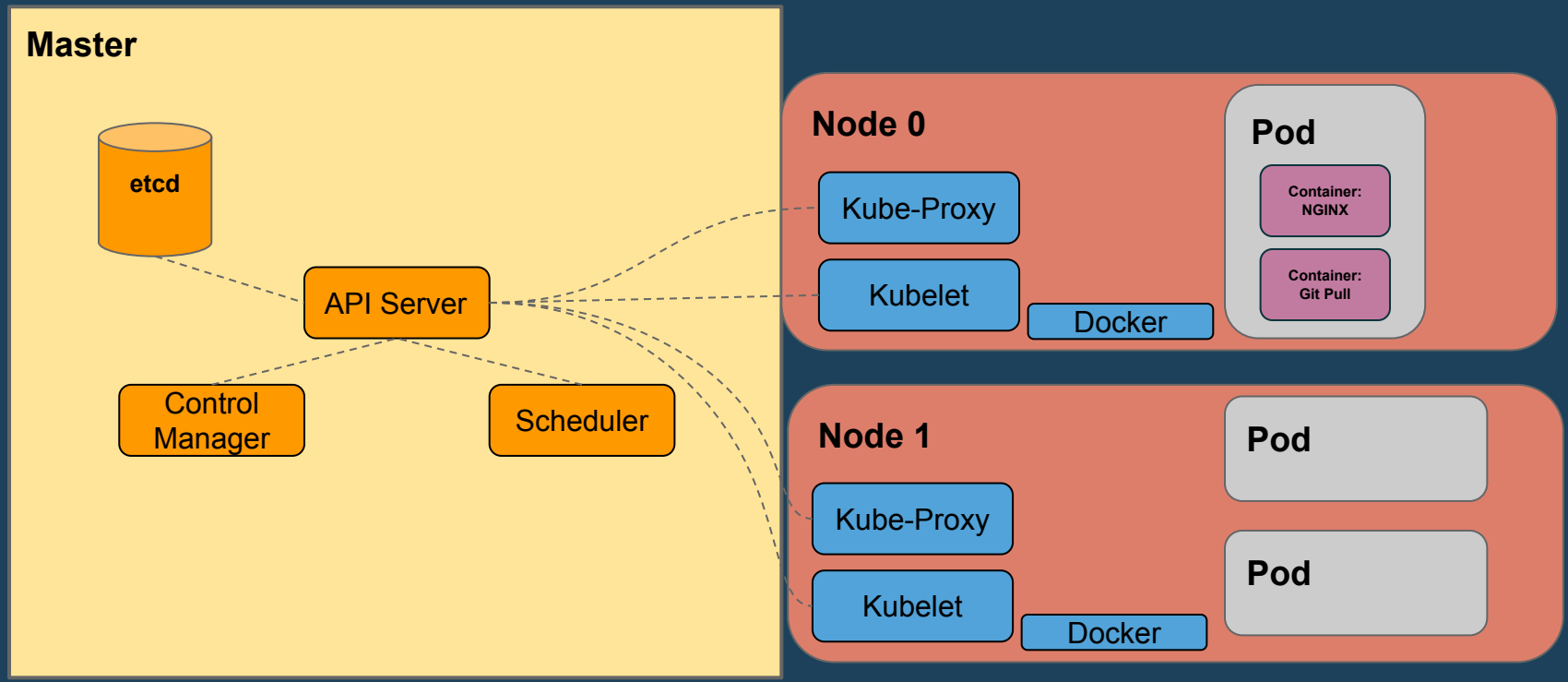
```
status:
  availableReplicas: 2
  fullyLabeledReplicas: 5
  observedGeneration: 724
  readyReplicas: 2
  replicas: 5
```



redhat.

```
1    while true {
2    receiveInfoAboutAPIObjects()
3    synchronizeRealStateToMatchFetchedInfo()
4    }
```

Kubernetes Cluster



Desired State and Actual State

- The replicaset object contains a desired state object (spec) from the user and the actual state from the ReplicaSet controller (status)

Desired State vs. Current State

Desired State:

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: my-first-replica-set
spec:
  selector:
    matchLabels:
     app: nginx
  replicas: 5
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
```

Current State:

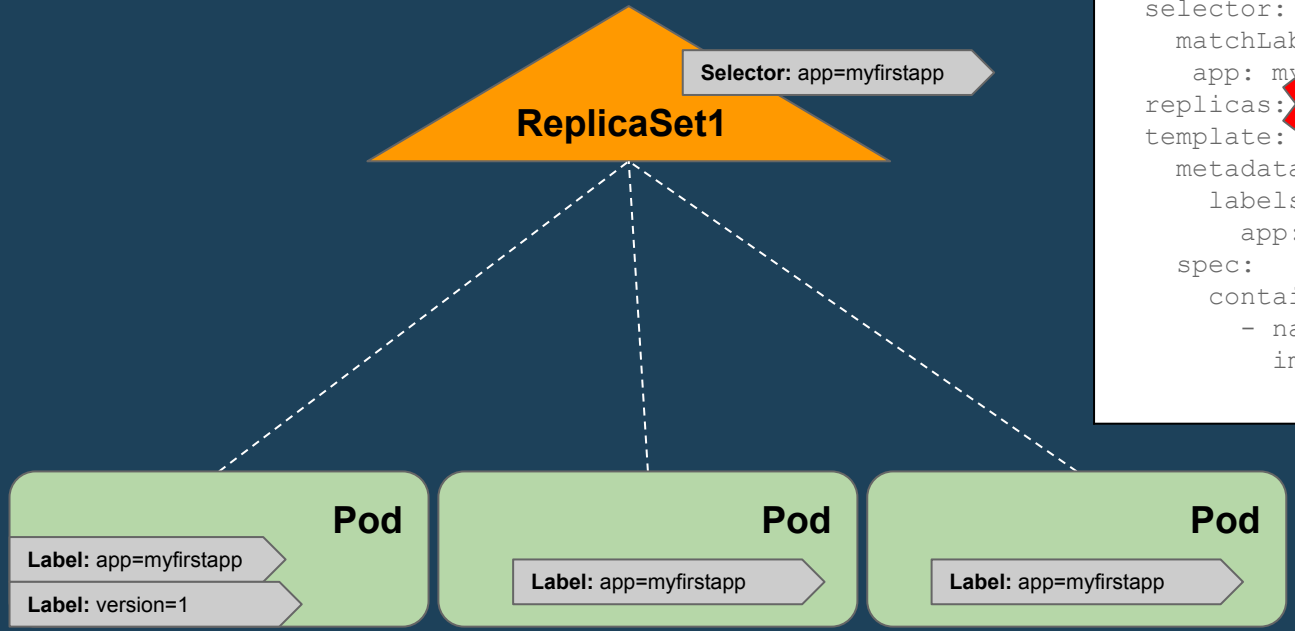
```
status:
  availableReplicas: 2
  fullyLabeledReplicas: 5
  observedGeneration: 724
  readyReplicas: 2
  replicas: 5
```

ReplicaSets in Action!

```
kubectl run myfirstapp --image quay.io/coreostrainme/hello-whoami:2.0.1  --restart=Never -l app=myfirstapp,version=1
```

```
kubectl create -f  myfirstreplicaset.yaml
kubectl scale replicaset myfirstreplicaset --replicas=3
```

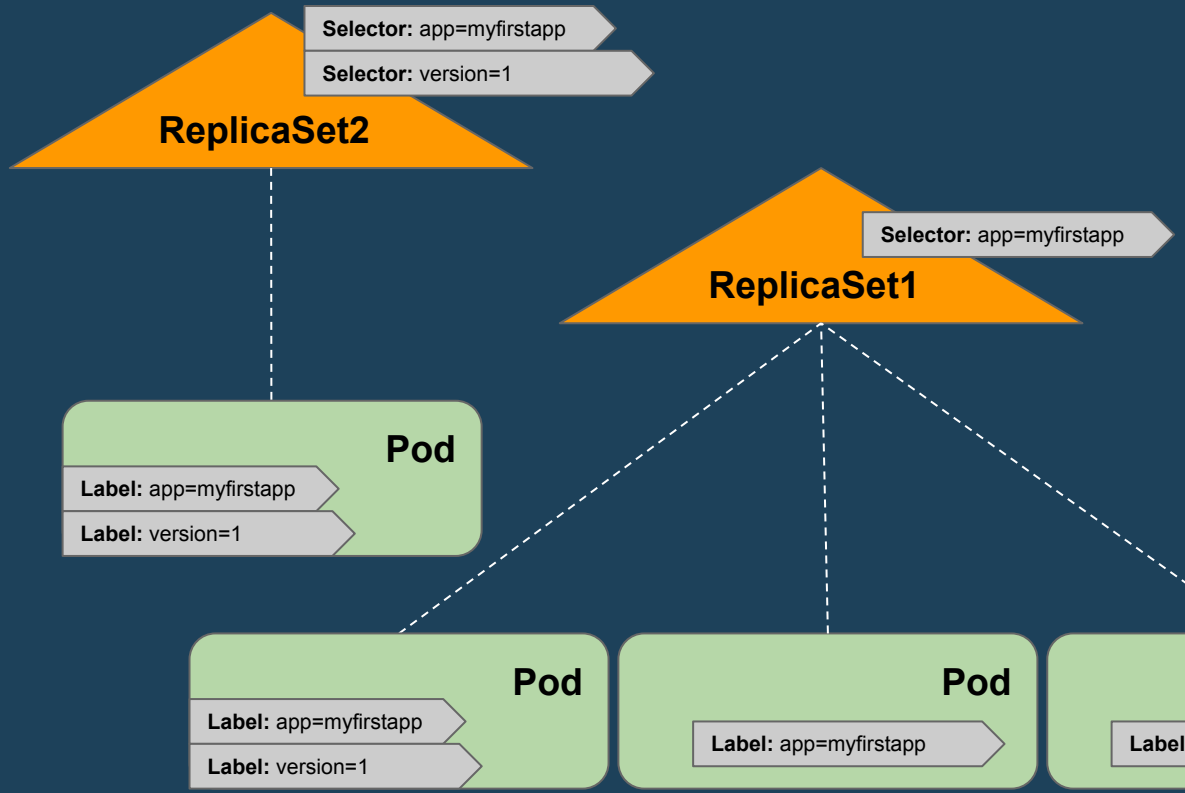
```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
```



redhat.

Creating another ReplicaSet

```
kubectl create -f  mysecondreplicaset.yaml
```

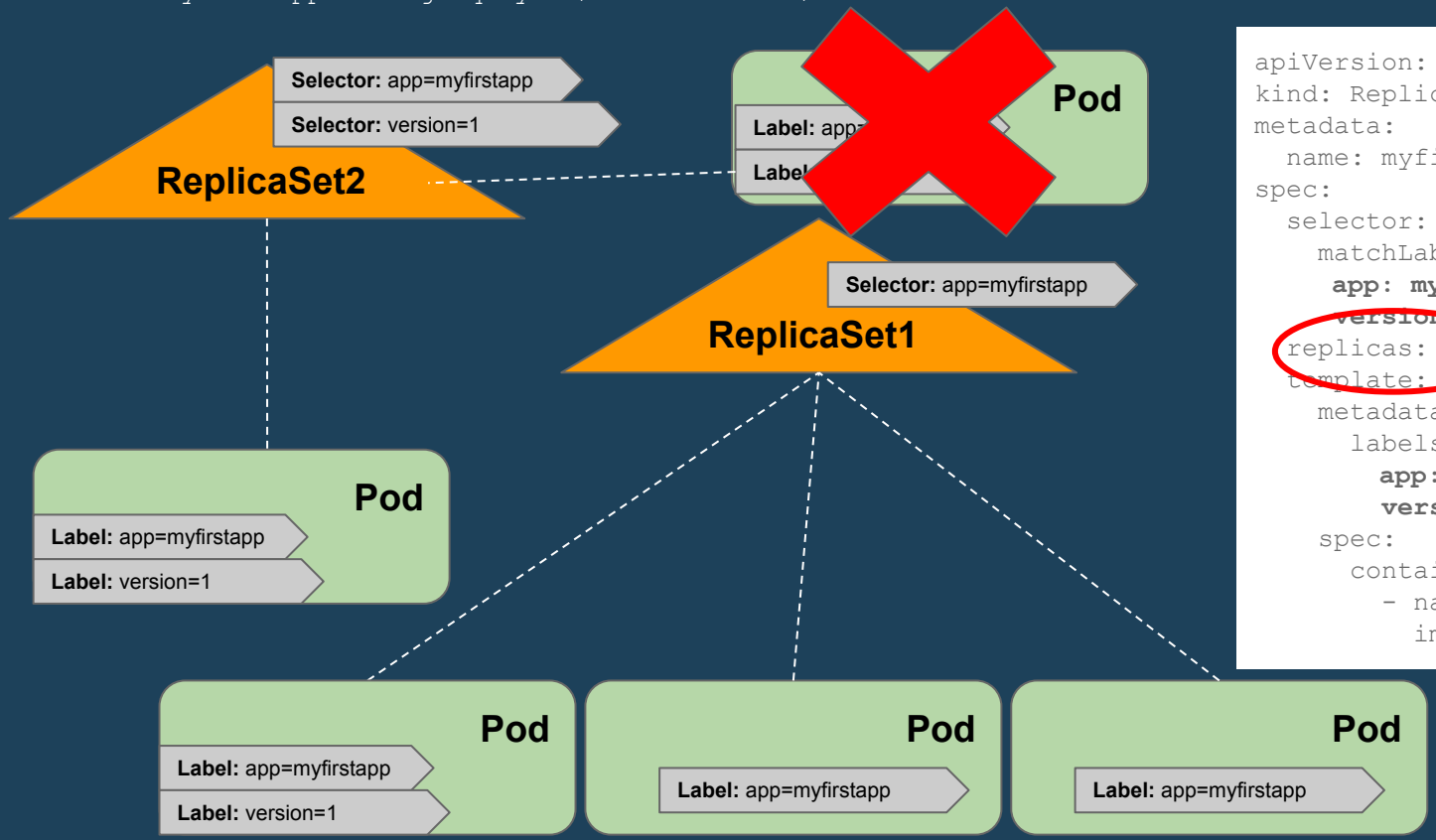



```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: mysecondreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
     version: 1
  replicas: 1
  template:
    metadata:
      labels:
        app: myfirstapp
        version: 1
    spec:
      containers:
        - name: nodejs
          image: myimage
```

redhat.

Creating Another Orphan Pod

```
kubectl run myfirstapp --image quay.io/coreostrainme/hello-whoami:2.0.1  --restart=Never -l app=myfirstapp,version=1
```




```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
      app: myfirstapp
      version: 1
  replicas: 1
  template:
    metadata:
      labels:
        app: myfirstapp
        version: 1
    spec:
      containers:
        - name: nodejs
          image: myimage
```

redhat.

ReplicaSets

Behind the scenes, replicas (pod copies) in a deployment are represented and managed by ReplicaSets (RS)

- RS's job is to create/recreate/destroy pods when needed from a template in the manifest
- Results in self-healing and application high availability

```
$ kubectl get replicasets
NAME                  DESIRED   CURRENT   AGE
nginx-684635458       2         2         31m
$ kubectl get pods
NAME                            READY     STATUS    RESTARTS   AGE
nginx-684635458-9p0kg   1/1       Running   0          31m
nginx-684635458-ibp6q   1/1       Running   0          31m
```

Scaling your Application: Autoscale

Kubernetes automatically chooses the number of nginx replicas as needed, from one to three in this example

```
$ kubectl autoscale replicaset my-nginx  --cpu-percent=80 --min=1 --max=3
replicaset "my-nginx" autoscaled
$ kubectl get pods -lapp=nginx
NAME              READY     STATUS    RESTARTS   AGE
my-nginx-1jgkf    1/1       Running   0          3m
my-nginx-divi2    1/1       Running   0          3m
$ kubectl get horizontalpodautoscaler
NAME      REFERENCE                          TARGET    CURRENT   MINPODS   MAXPODS   AGE
nginx     Replicaset/my-nginx                80%       39%       1         3         1m
```

kubectl scale

Update the size of the specified replication controller.

```
kubectl scale (-f FILENAME | TYPE NAME | TYPE/NAME) --replicas=COUNT
[--resource-version=version] [--current-replicas=count] [flags]
```

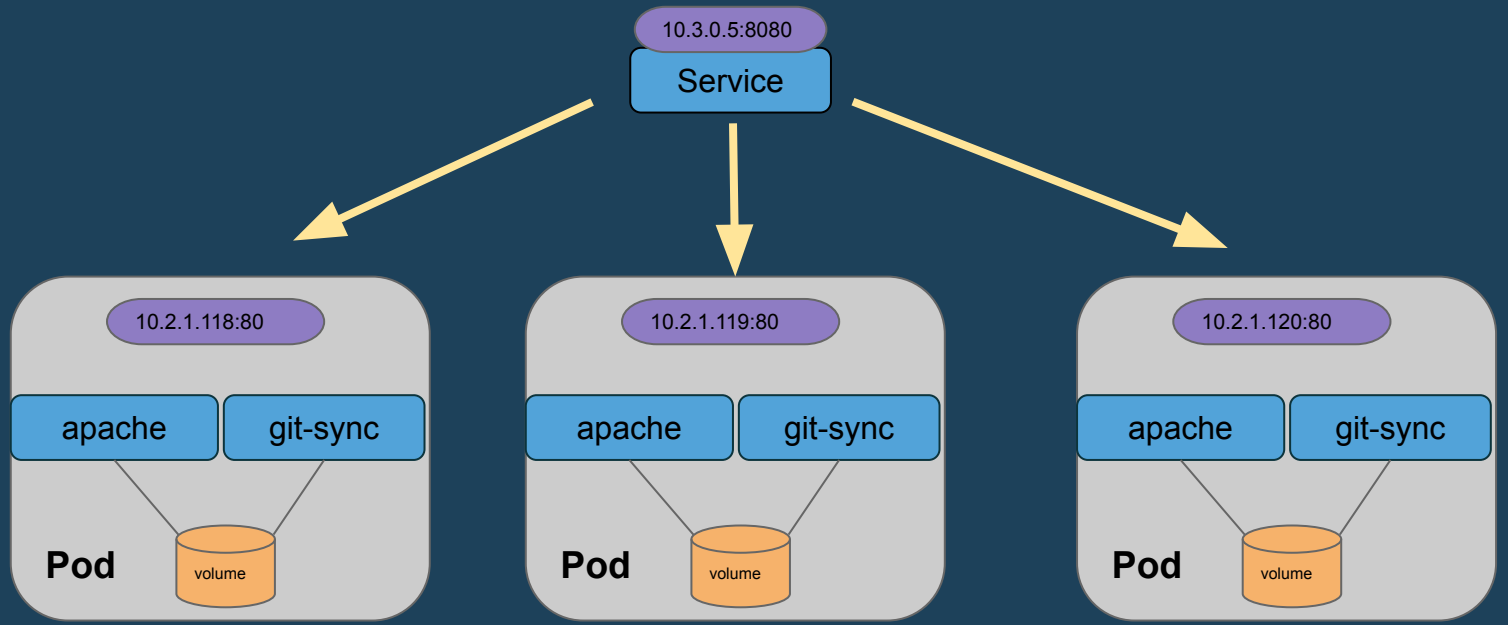
redhat

Services

redhat

Services

- Persistent IPs for Pods
- Load Balances Between Replicas

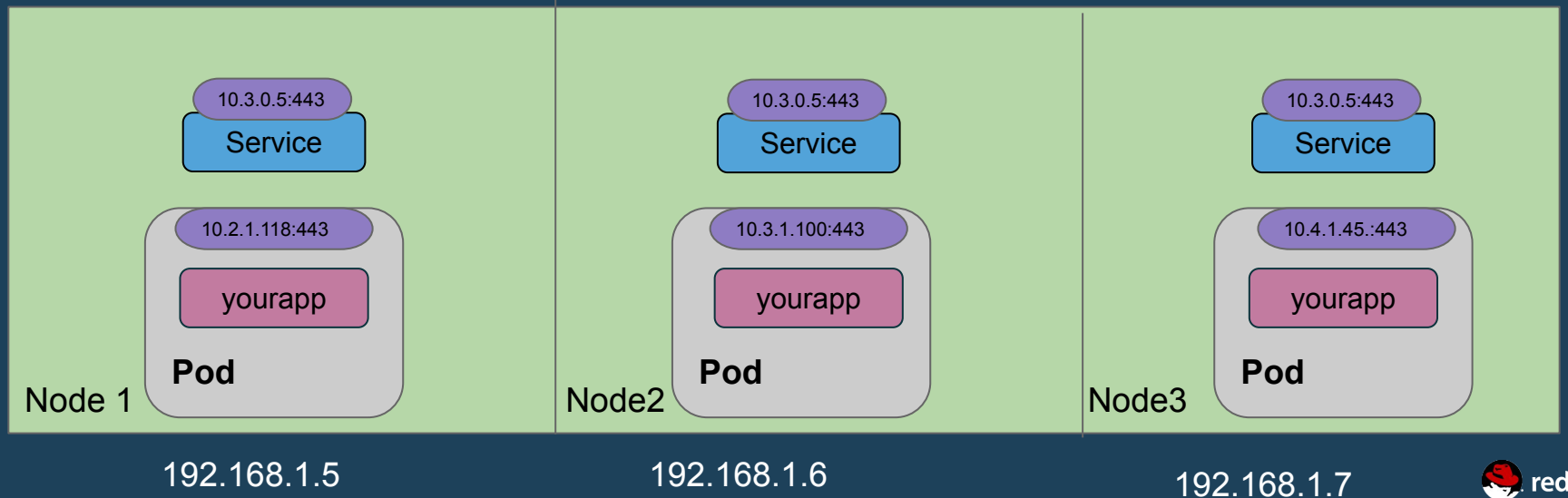


Service Types

- ClusterIP (shown previously)
  - Exposes a service using an internal IP accessible only in the Kubernetes cluster
- NodePort
  - Exposes a service at every node <NODE_IP>:<NODE_PORT>
- LoadBalancer
  - Works with a cloud provider to create a load balancer and rules to expose the service as a layer on top of NodePort

ClusterIP

```
Kubectl create -f deployment.yaml
Kubectl create -f myservice.yaml
```

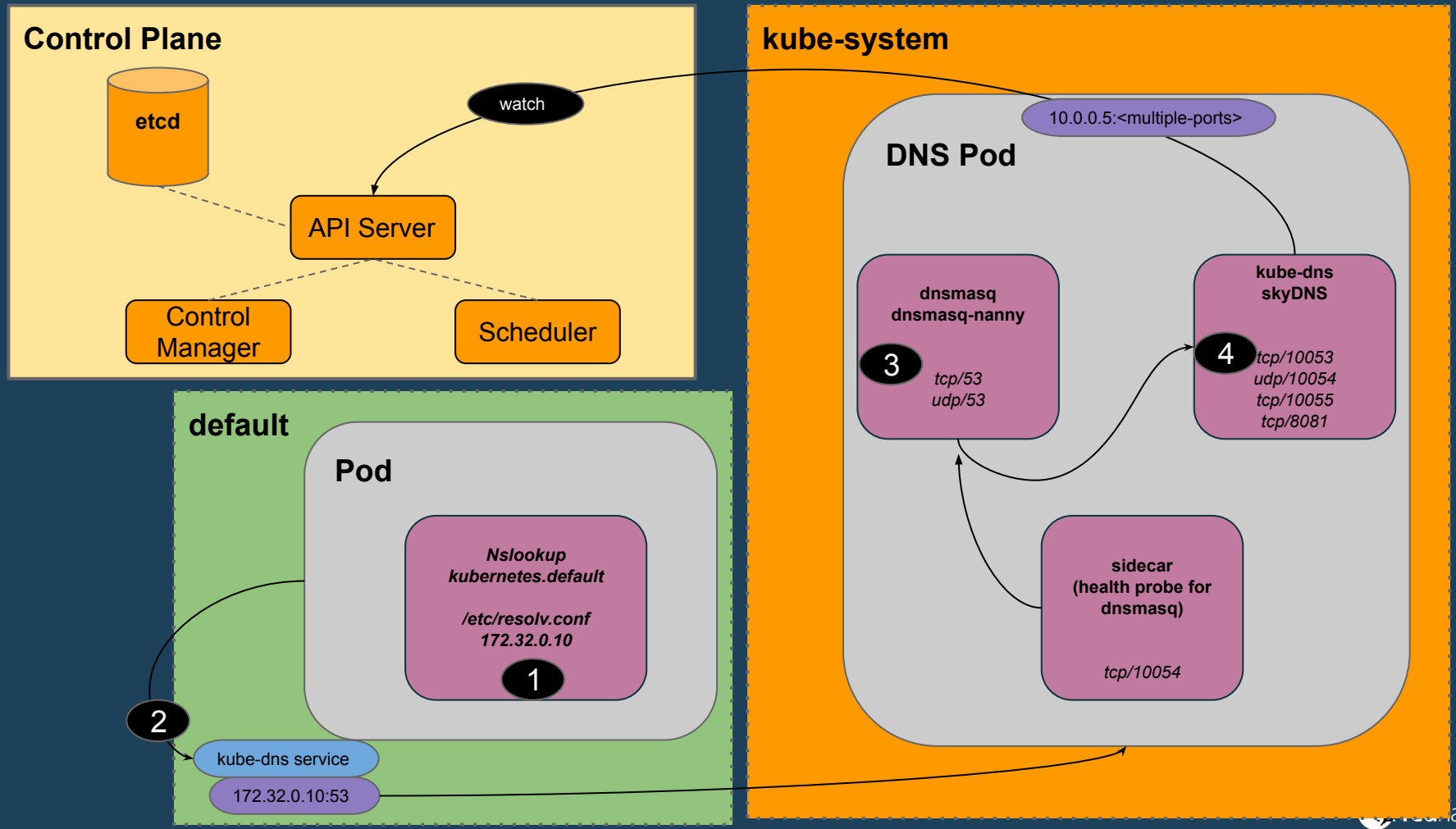


redhat.

Service Discovery

Kubernetes DNS

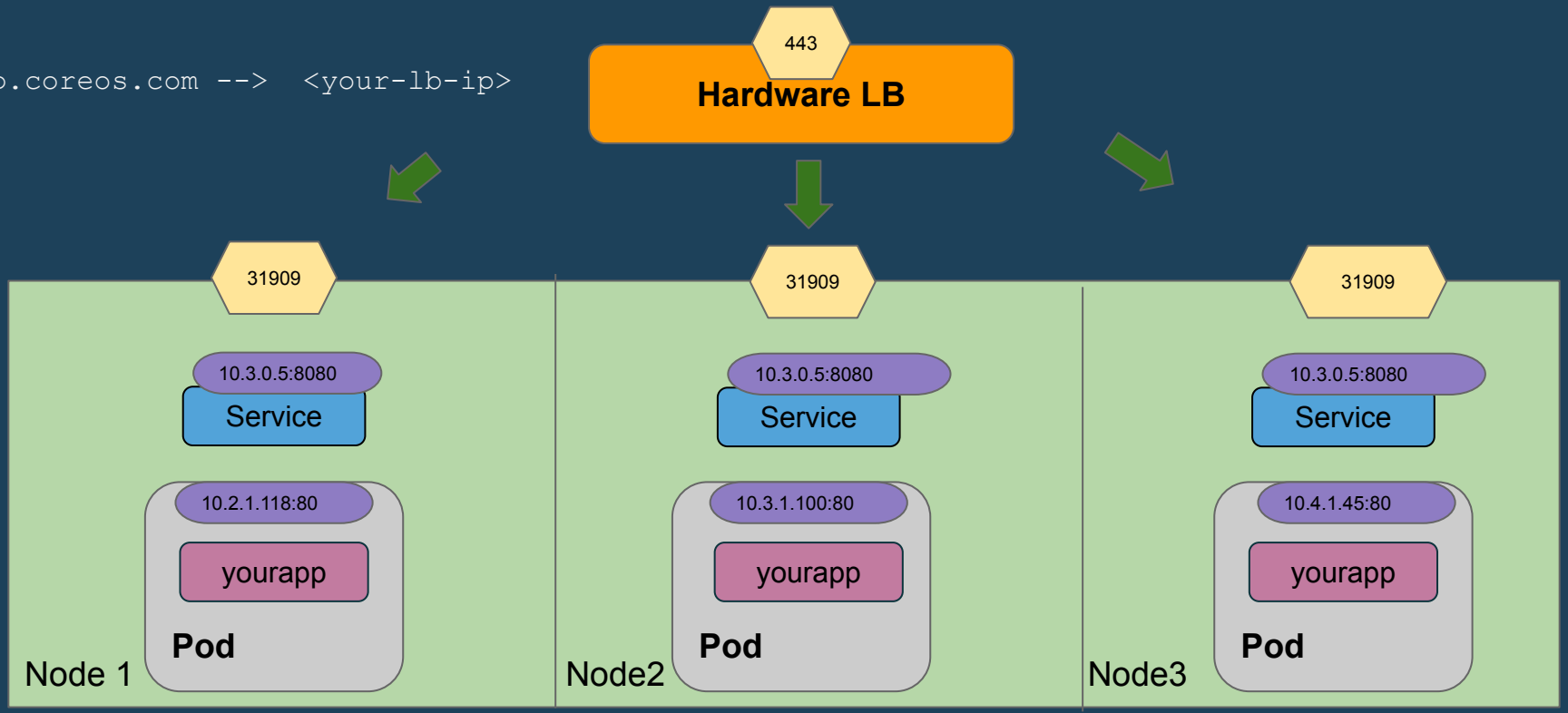
- Schedules a DNS Pod and Service on the cluster.
- Configures the kubelets to tell individual containers to use the DNS Service's IP to resolve DNS names.


Control Plane
etcd
watch
API Server
Control Manager
Scheduler
kube-system
10.0.0.5:<multiple-ports>
DNS Pod
dnsmasq
dnsmasq-nanny
3
tcp/53
udp/53
kube-dns
skyDNS
4
tcp/10053
udp/10054
tcp/10055
tcp/8081
sidecar
(health probe for
dnsmasq)
tcp/10054
default
Pod
Nslookup
kubernetes.default
/etc/resolv.conf
172.32.0.10
1
2
kube-dns service
172.32.0.10:53

NodePort

```
Kubectl create -f deployment.yaml
Kubectl create -f myservice.yaml
```

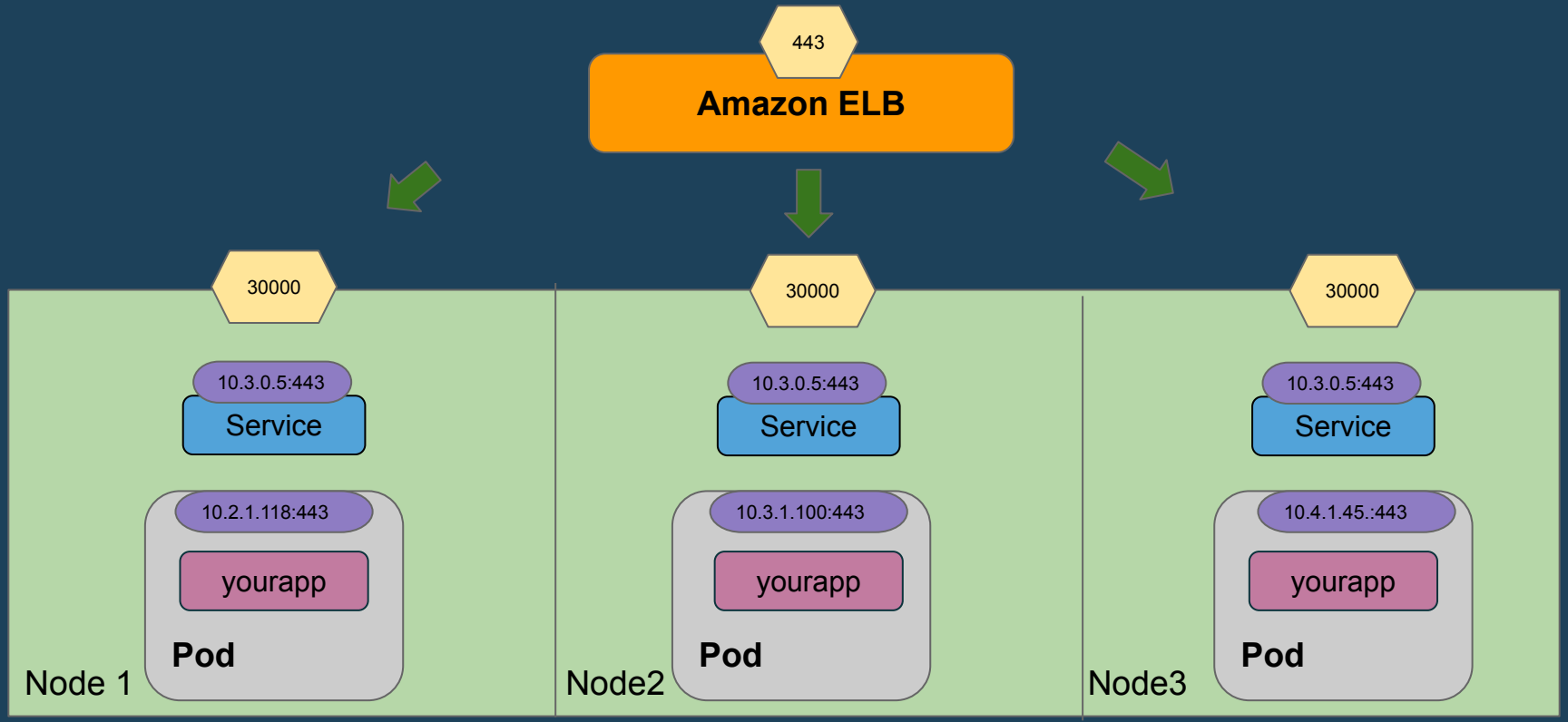
```
myapp.coreos.com -->  <your-lb-ip>
```



redhat.

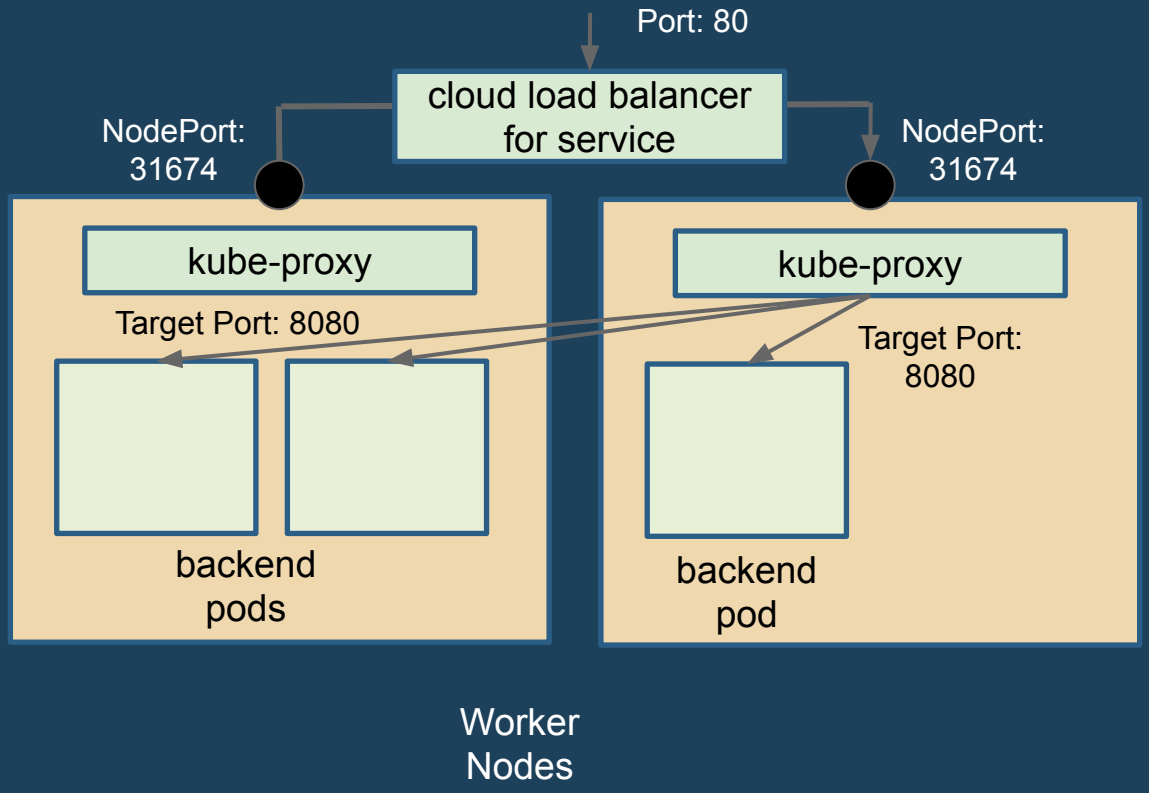
LoadBalancer

```
Kubectl create -f deployment.yaml
Kubectl expose… --port 443 --port 443
```



Example Service with LoadBalancer

- A cloud load balancer is created for every service
  - myservice.mydomain.com
- The load balancer picks a worker node and sends the request at NodePort 31674
  - This is configured in the load balancer
- kube-proxy identifies the service by port and load balances among all service pods



redhat

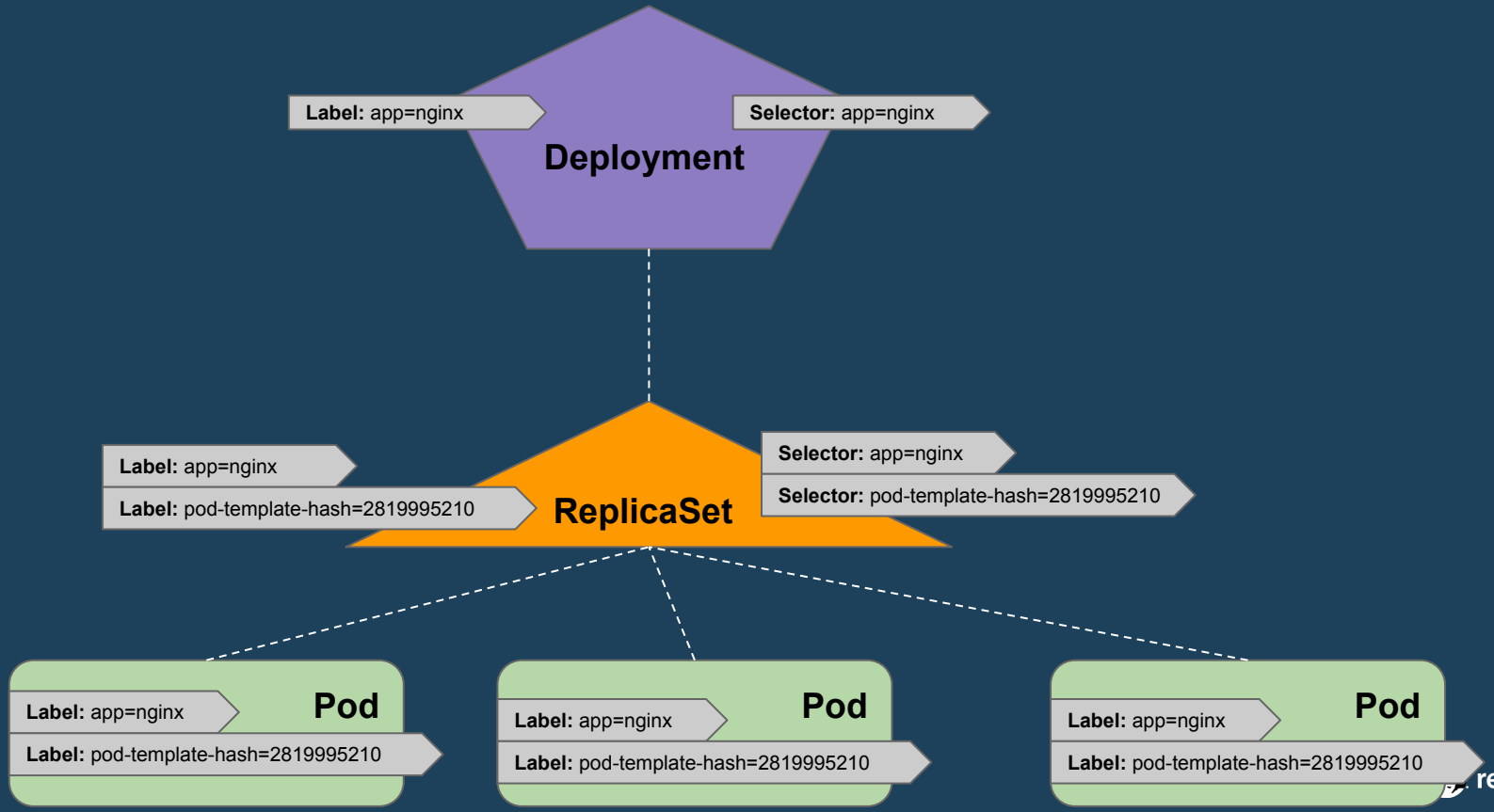
Deployments

redhat

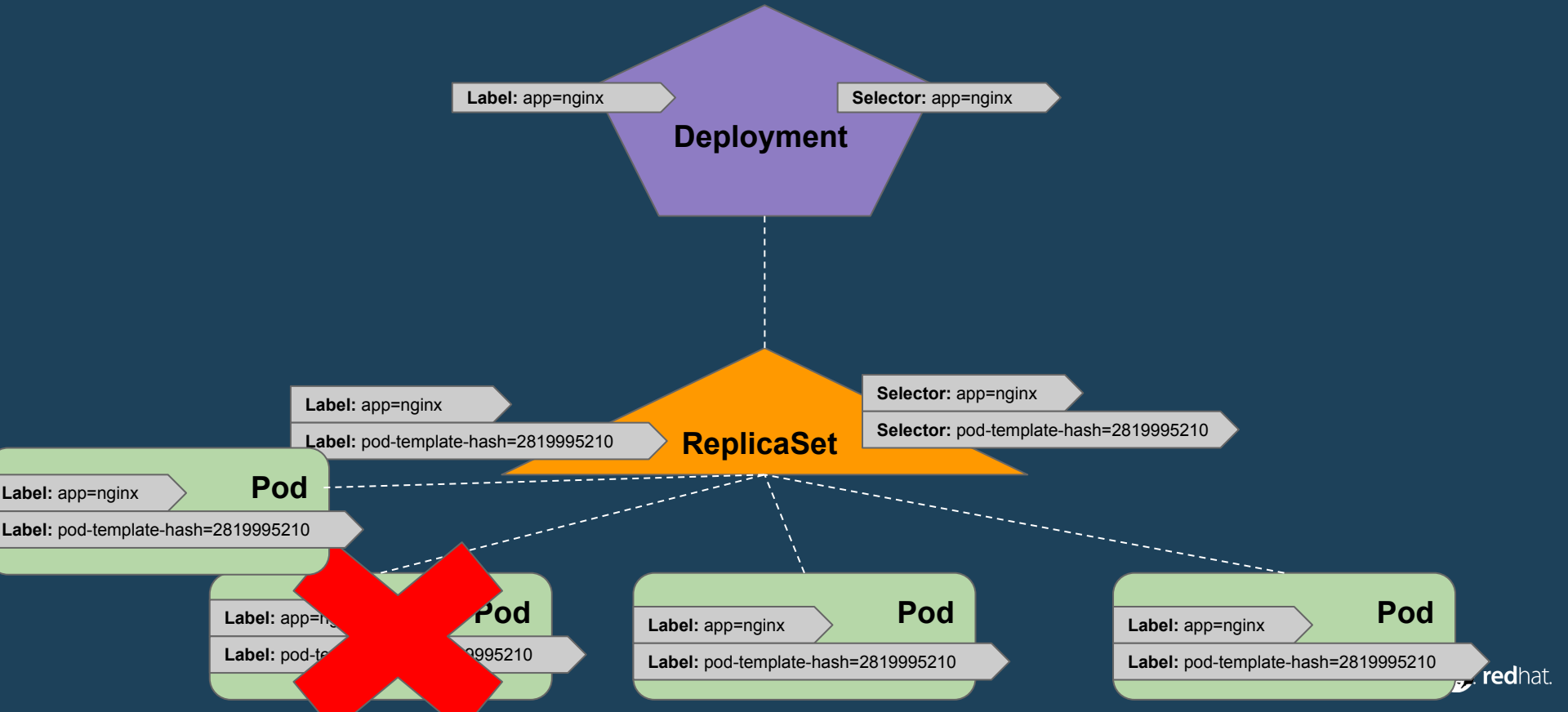
Sample Deployment Manifest file.

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

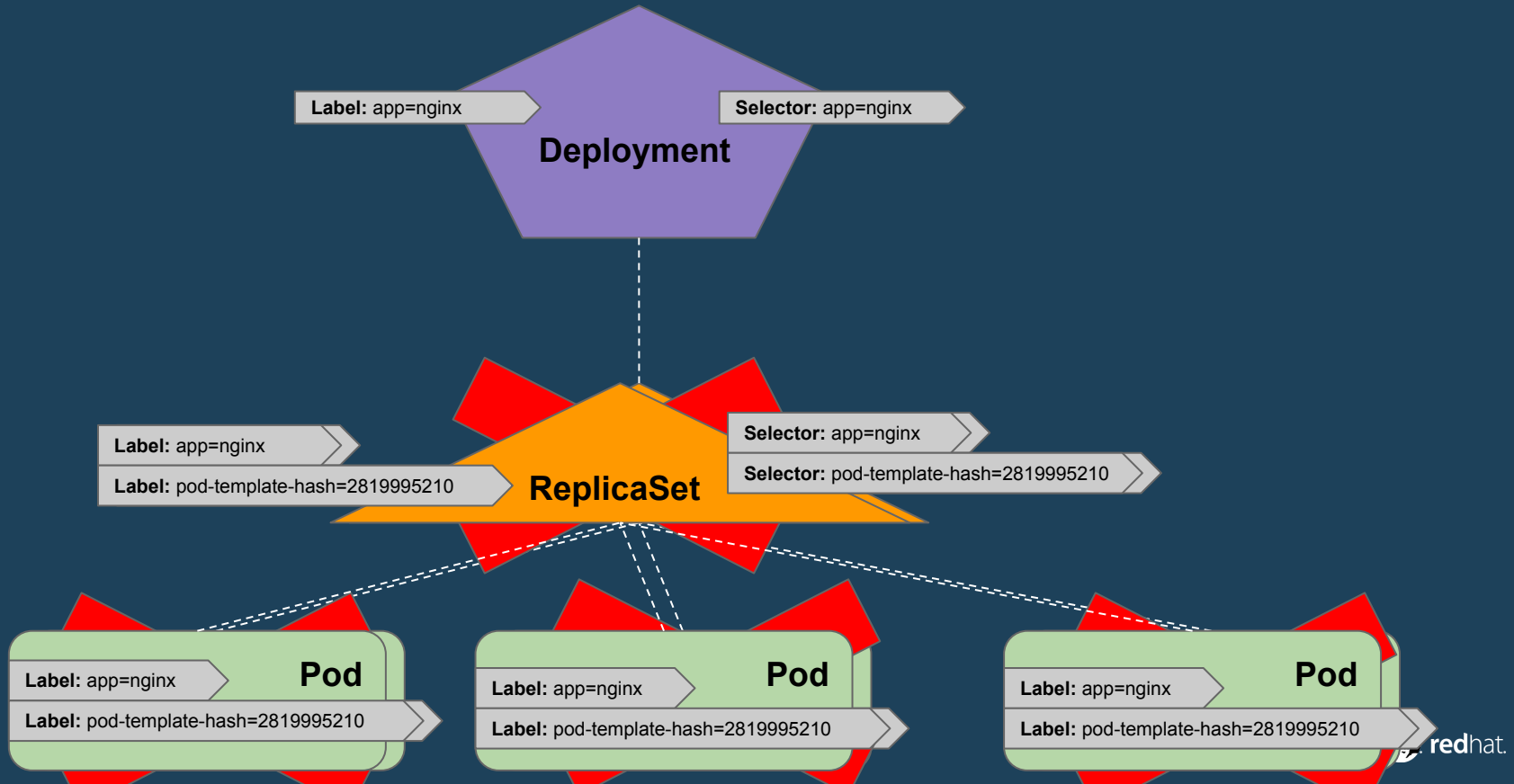
Deployments: Labels & Selectors



Deleting a Pod....



Deleting a ReplicaSet...

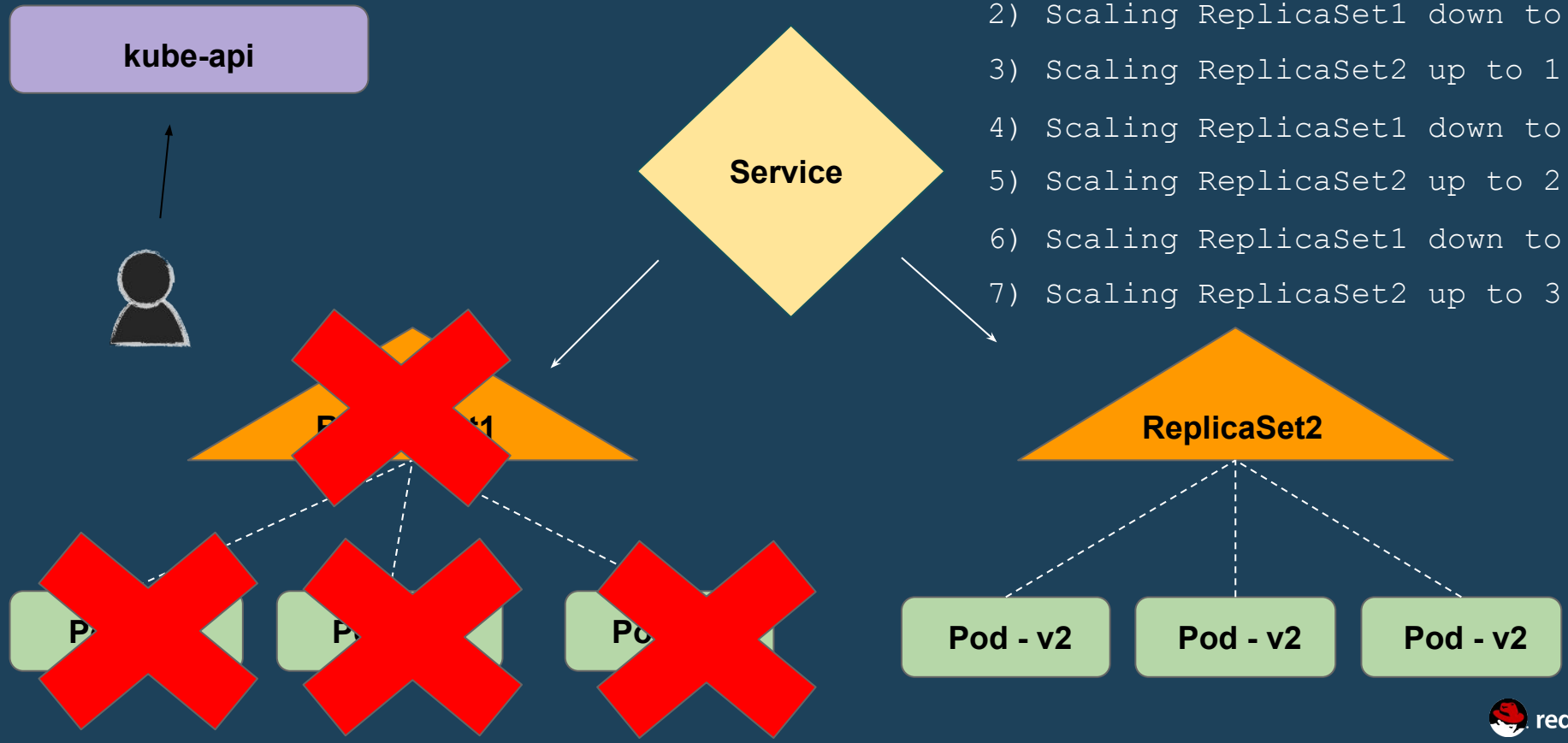


This is nice.

But to really grasp the power of Deployments, consider how we would “roll out” a new application to an existing ReplicaSet.

Rolling out a new app the “old way”.

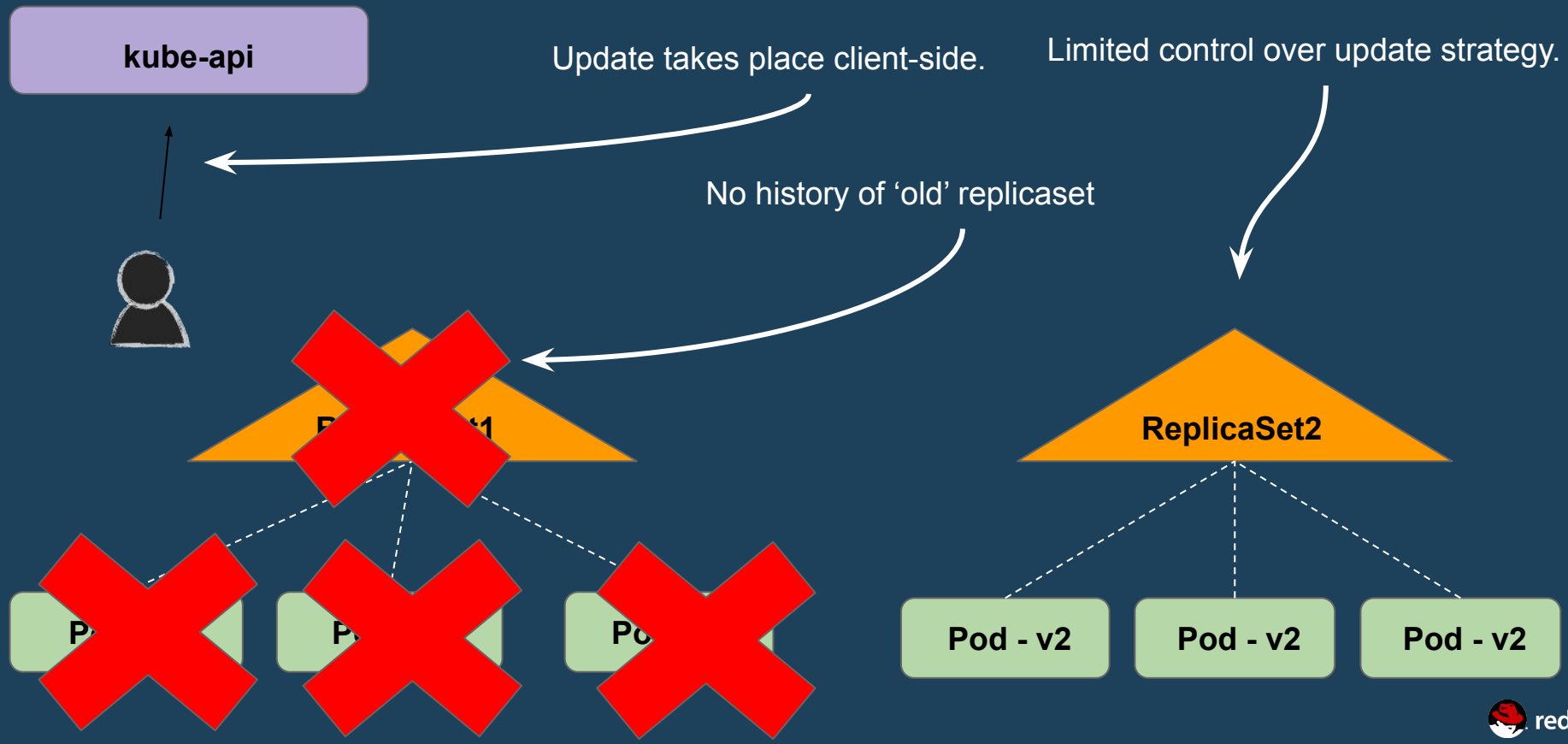
1) Created ReplicaSet2.
2) Scaling ReplicaSet1 down to 2.
3) Scaling ReplicaSet2 up to 1.
4) Scaling ReplicaSet1 down to 1.
5) Scaling ReplicaSet2 up to 2.
6) Scaling ReplicaSet1 down to 0.
7) Scaling ReplicaSet2 up to 3.



redhat.

What’s wrong with this picture?

Issues with “Old” Rolling Updates



redhat.

Deployments to the Rescue.

```
kubectl run nginx-deployment --image=nginx:1.7.9 --replicas=3
```

Deployment Rollout Strategies

2

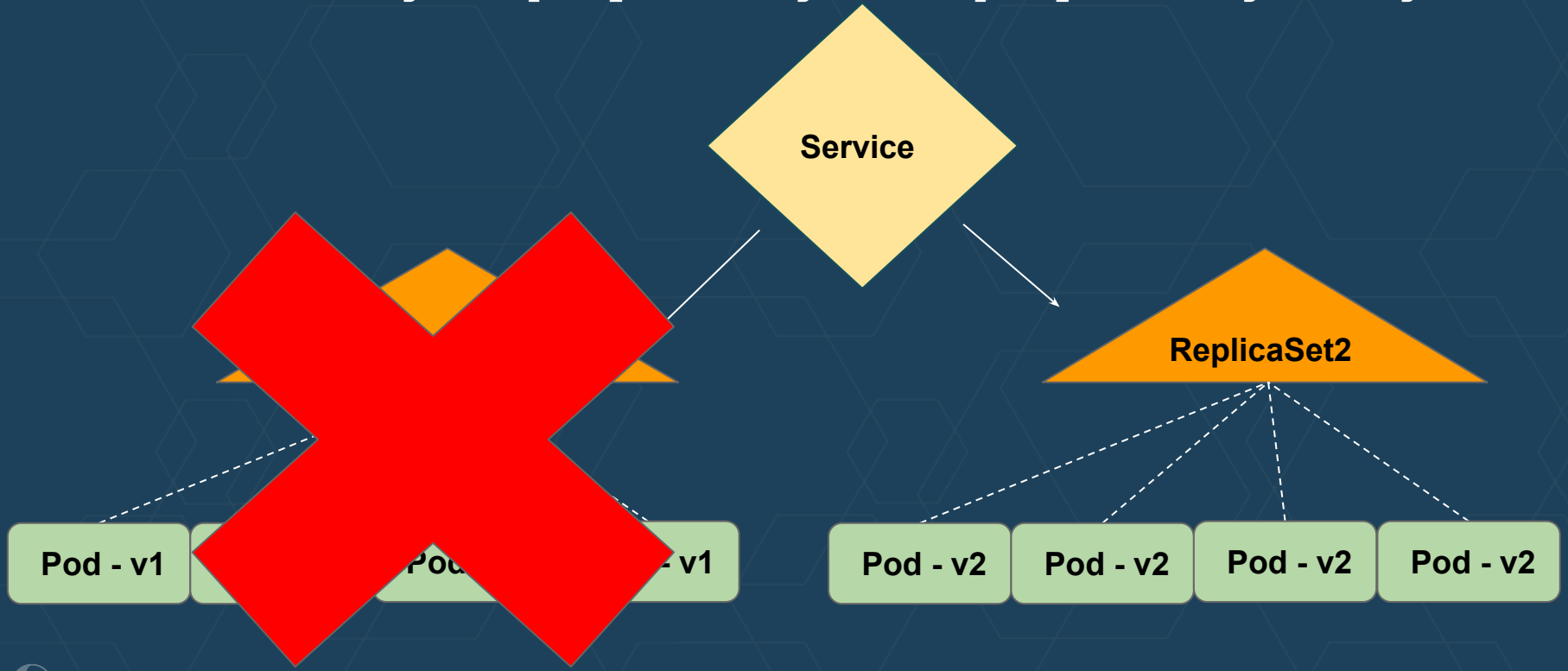
1) Recreate

2) RollingUpdate

If you don’t care about downtime...

Strategy: Recreate

```
kubectl set image deployment nginx-deployment nginx=nginx:1.12
```

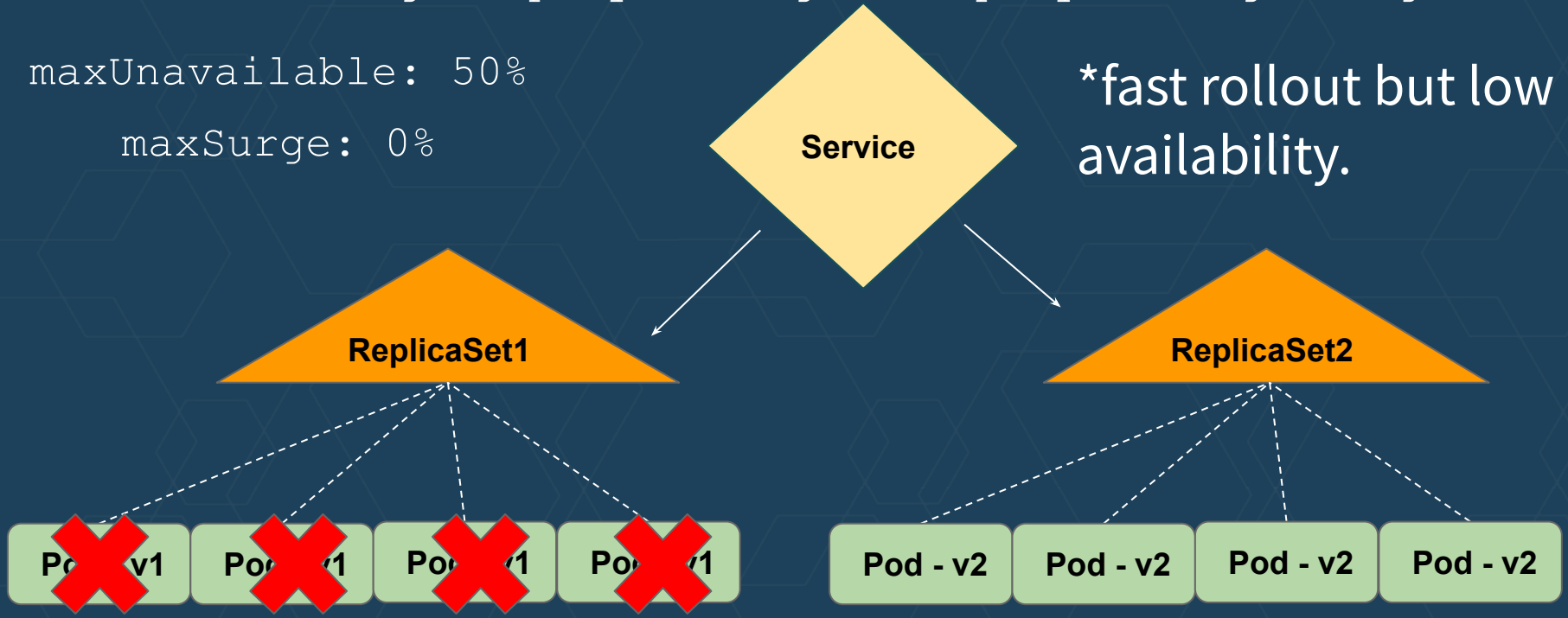


If you care about downtime...

Strategy: RollingUpdate

```
kubectl set image deployment nginx-deployment nginx=nginx:1.12
```

```
maxUnavailable: 50%
maxSurge: 0%
```



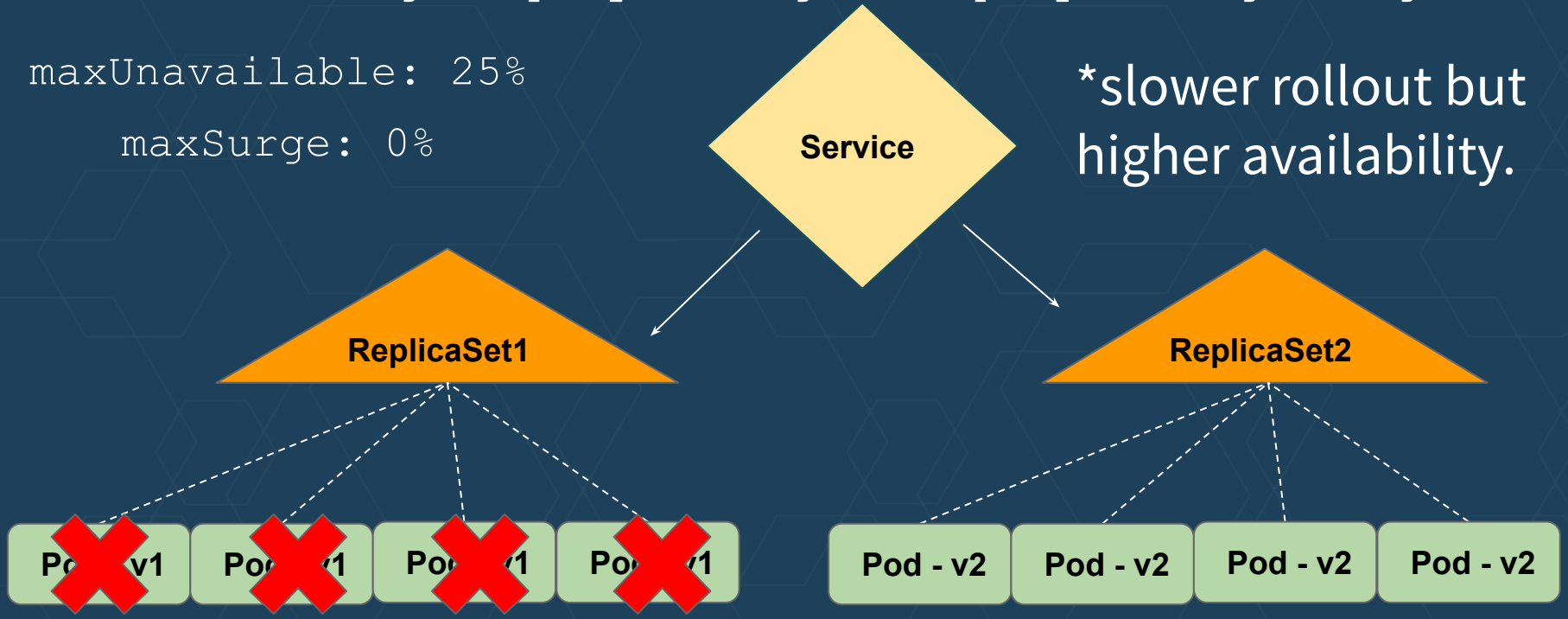
*fast rollout but low availability.

Strategy: RollingUpdate

```
kubectl set image deployment nginx-deployment nginx=nginx:1.12
```

```
maxUnavailable: 25%
maxSurge: 0%
```

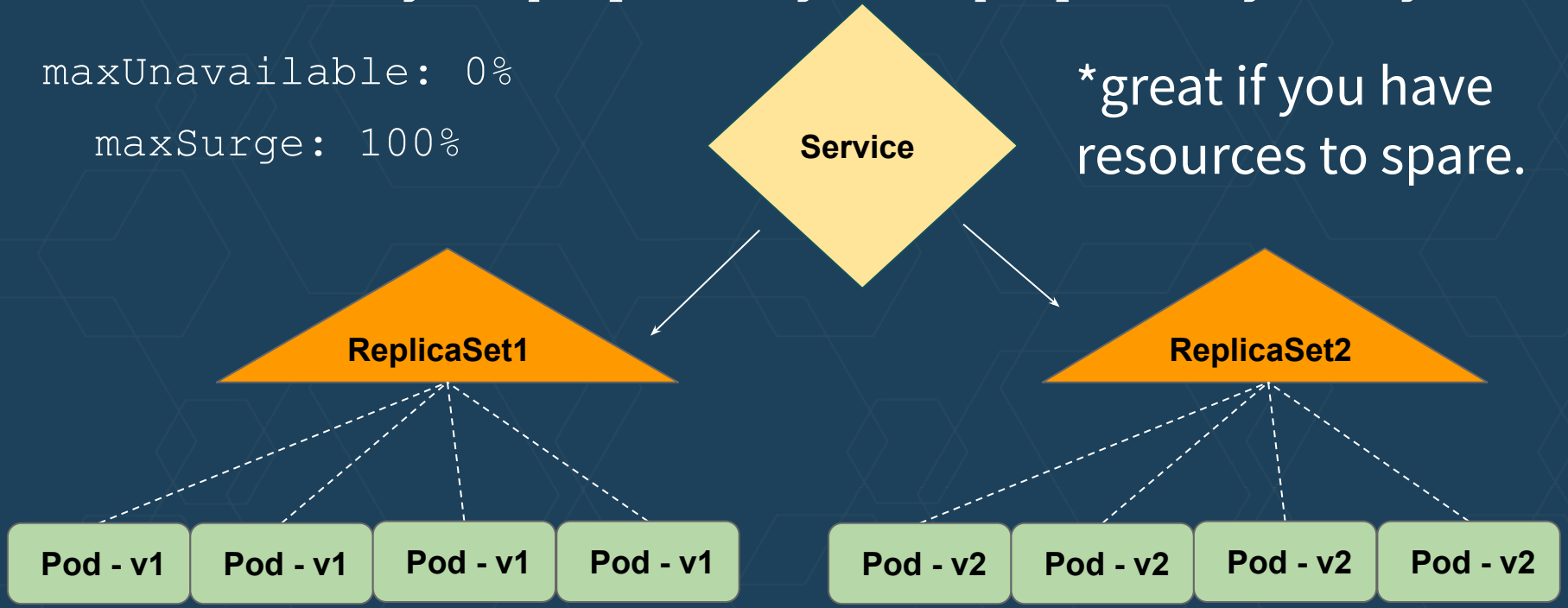
*slower rollout but higher availability.



Strategy: RollingUpdate

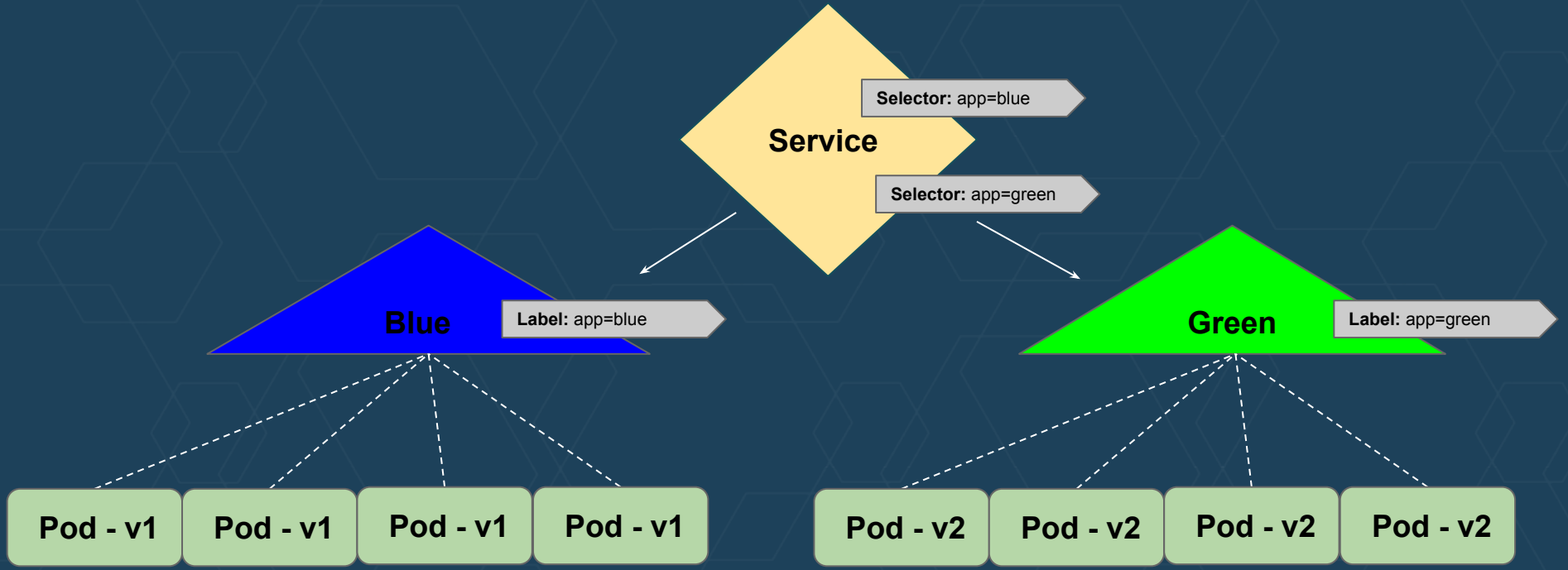
```
kubectl set image deployment nginx-deployment nginx=nginx:1.12
```

```
maxUnavailable: 0%
maxSurge: 100%
```



*great if you have resources to spare.

Blue-Green Deployments



Canary Deployments

original deployment:

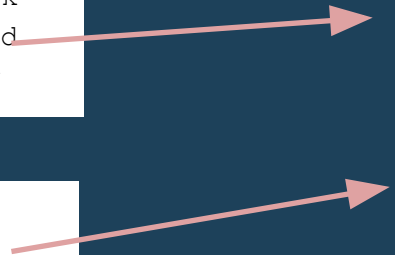
```
labels:
        app: guestbook
        tier: frontend
        track: stable
```

canary deployment:

```
labels:
        app: guestbook
        tier: frontend
        track: canary
```

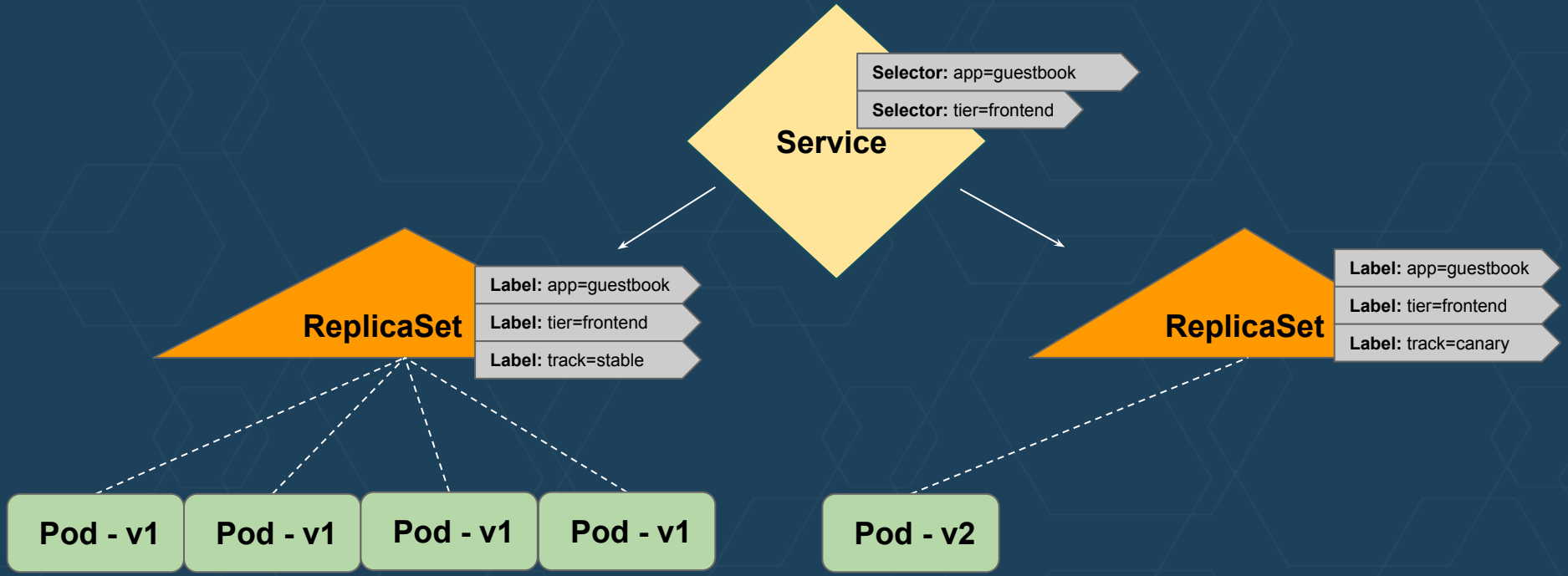
service:

```
selector:
     app: guestbook
     tier: frontend
```



- When updating an application, it is common to start with a single ”canary” to handle a small fraction of the existing traffic
- If the canary is successful, a full update can proceed
- Create a new deployment for the canary, with unique labels
- The frontend service can span both deployments by using a common label (“guestbook”)

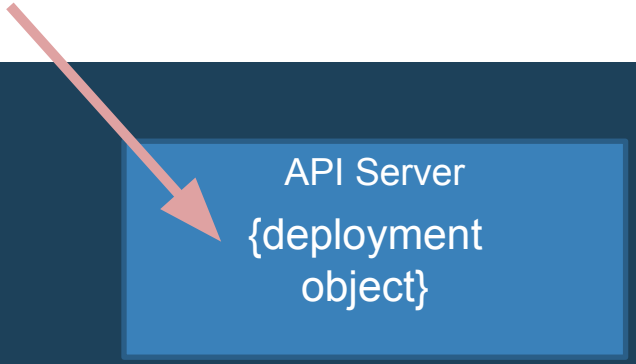
Canary Deployments



The Deployment Object

- The parameters of the **`kubectl run`** command provides the initial contents of a deployment object in the API Server

```
$ kubectl run hello-kube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
```



Desired State and Actual State

- The deployment object contains a desired state object (spec) from the user and the actual state from the deployment controller (status)

Deployment Example (1/2)

Run three replicas (copies) of pods containing an nginx web server/proxy/load balancer

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        tier: backend
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

Pod spec template defines the 'cookie cutter' used for creating new pods when necessary

Deployment Example (2/2)

```
$ kubectl create -f nginx-deployment.yaml
deployment "nginx-deployment" created

$ kubectl describe deployments/nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Wed, 24 Aug 2016 13:17:36 -0400
Labels:                 app=nginx
                tier=backend
Selector:       app=nginx,tier=backend
Replicas:       3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:      1 max unavailable, 1 max surge
OldReplicaSets:         <none>
NewReplicaSet:          nginx-deployment-4253660899 (3/3 replicas created)
Events:
[...]
```

You can also use '**`kubectl get deployment nginx-deployment -o json`**' to view the JSON

Deleting Deployments

kubectl delete deployment

```
$ kubectl delete deployment/my-nginx
deployment "my-nginx" deleted
```

If you try to delete the pods or ReplicaSets before deleting the deployments, Kubernetes will just replace them

Summary: Deployments, ReplicaSets and Pods

- A Deployment object results in the creation of a ReplicaSet (rs) object, which results in a number of Pod objects