

# Deploying Applications in Kubernetes

**Anthony E. Nocentino**  
[aen@centinosystems.com](mailto:aen@centinosystems.com)



# Course Overview

- **Module 0** - Introduction
- **Module 1** - Container Fundamentals and Introducing Kubernetes
- **Module 2** - Kubernetes Architecture and API Objects
- **Lunch @ 12:00-12:45**
- **Module 3** - Interacting With Your Cluster
- **Module 4** - Deploying Applications in Kubernetes
- **Module 5** - Building and Deploying Container-based Applications in Kubernetes

# Agenda

- **Deploying Applications in Kubernetes**
  - Namespaces
  - Labels
  - Services
  - Deployments

# Organizing Objects in Kubernetes

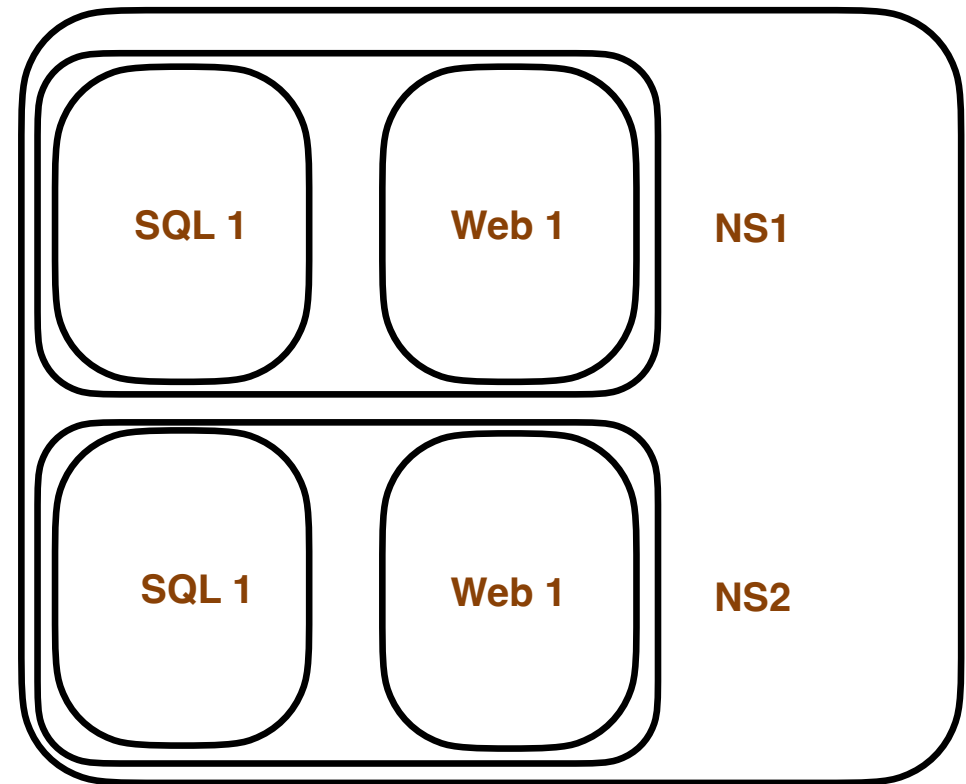
- **Namespaces**
- **Labels**
- **Annotations**

# Namespaces

- Ability to subdivide a cluster and its resources
- Conceptually a “virtual cluster”
  - Resource isolation/organization
  - Security boundary for Role Based Access Controls
  - Naming boundary
- A resource can be in only one namespace
- Has nothing to do with the concept of a Linux namespace

# Working with Namespaces

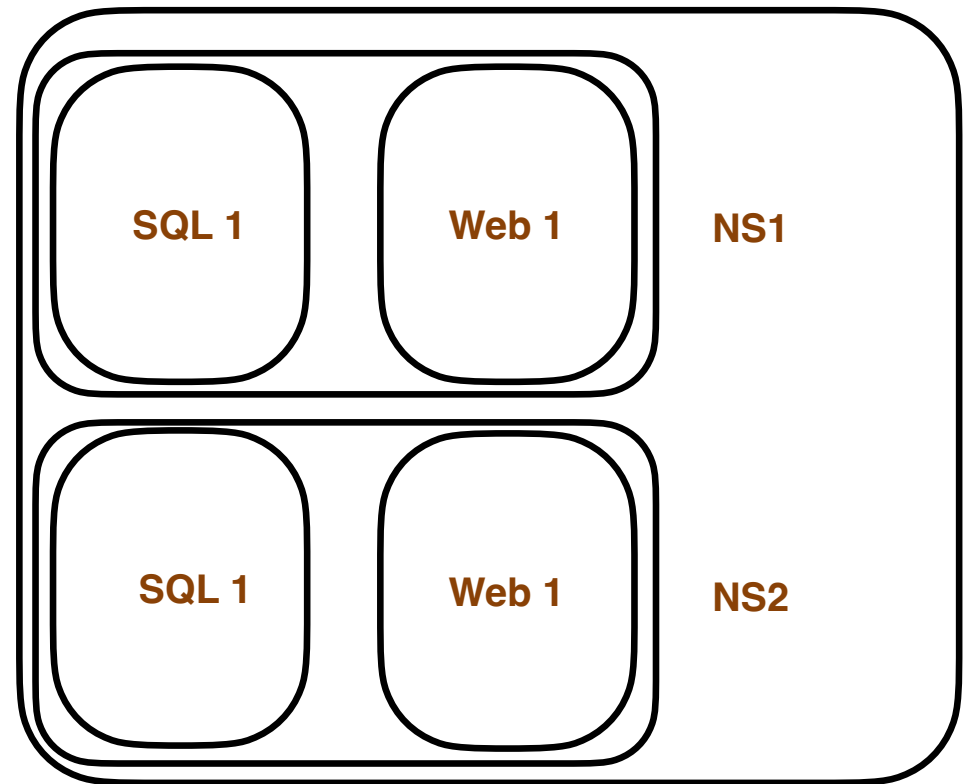
- Create/Query/Delete
- Operate on objects in a Namespace
- Some objects are Namespaced...  
some aren't
- Resources are Namespaces...  
Pods, Controllers, Services
- Physical things are not...  
PersistentVolumes, Nodes



Cluster

# Working with Namespaces

- **kube-public**
- **kube-system**
- User Defined
  - Imperatively with **kubectl**
  - Declaratively in a Manifest in YAML



Cluster

# Labels

- Used to organize resources - Pods, Nodes and more
- Enables you to perform operations on a collection of resources...like Pods
- Influence internal operations of Kubernetes
- Non-hierarchical, key/value pair
- Have more than one label per resource
- Label Selectors
  - Used to select/query objects
  - Return collections of objects that satisfy search conditions



# Working With Labels

- Creating resources with Labels
  - Imperatively with **kubect1**
  - Declaratively in a Manifest in YAML
- Editing existing resources' Labels
  - Assign a new Label
  - Overwriting an existing Label

## Adding and Editing Labels

```
kubectl label pod nginx tier=PROD app=v1
```

```
kubectl label pod nginx tier=DEBUG app=v1 --overwrite
```

```
kubectl label pod nginx app-
```

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
  name: nginx-pod
```

```
  labels:
```

```
    tier: PROD
```

```
    app: v1
```

```
spec:
```

```
  containers:
```

```
  - name: nginx
```

```
...
```



# Querying Using Labels and Selectors

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

```
kubectl get pods --selector tier=prod
```

```
kubectl get pods -l tier=prod
```

```
kubectl get pods -l 'tier in (prod,qa)'
```

```
kubectl get pods -l 'tier notin (prod,qa)'
```

```
kubectl get nodes --show-labels
```

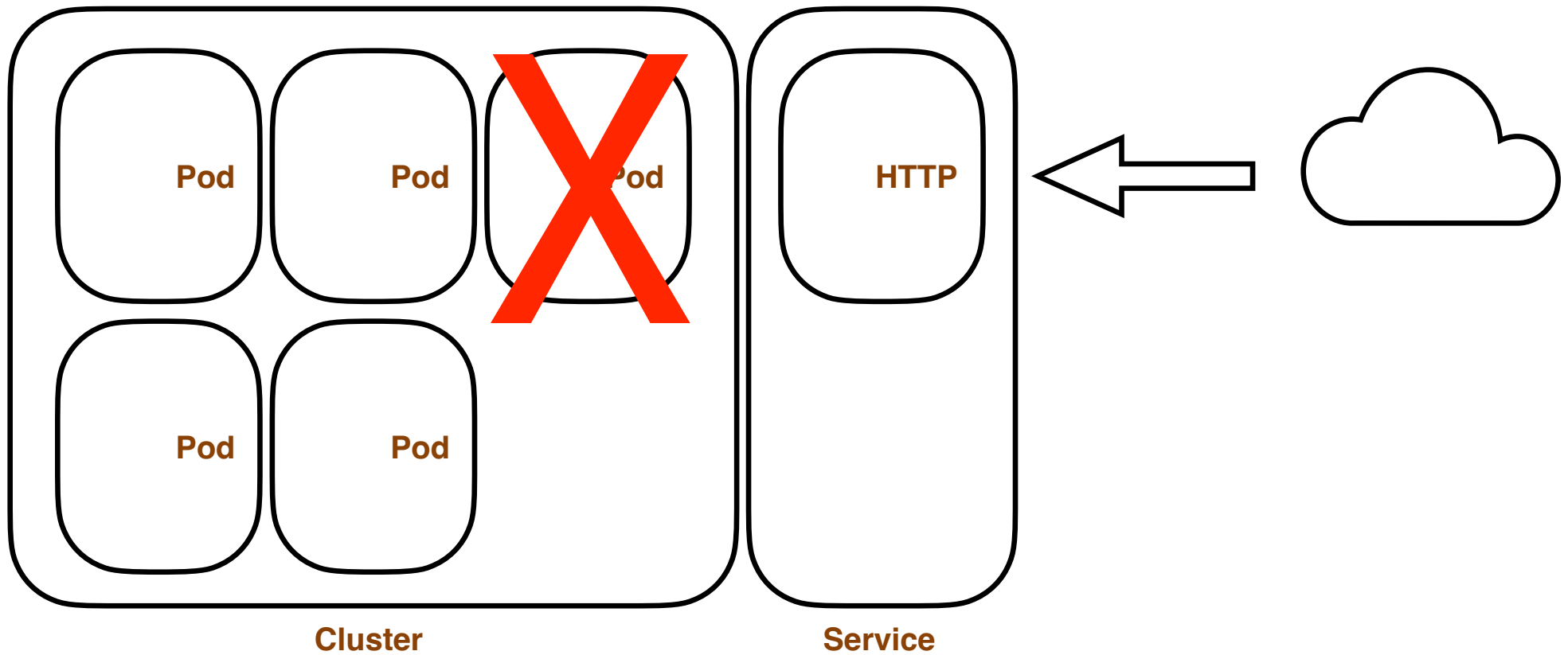
# How Kubernetes Uses Labels

- **Deployments, ReplicaSets** and **Services** match Pods using selector labels
  - Pod Selector - **matchLabels** and **matchExpressions**
- Influencing Scheduling Pods based on labels
- Nodes (specifically for access to hardware, perhaps high speed disk or GPU)
  - Specified Node name with **nodeAffinity**
  - Using a label selector with **nodeSelector**

# Using Services

- Due to the ephemerality of Pods, ReplicaSets ability to add/remove Pods
- Provide persistent IP and DNS to Pod based applications
- Services use Label Selectors to “select” the Pods in a Service,
  - Registering the Endpoints in the Service (Pod IP and Port pair)
- Implemented in the kube-proxy on the Node in iptables (kernel mode)
- kube-proxy watches the API Server and the Endpoints

## Using Services



# Types of Services

- **NodePort** - available on the internal Cluster's IP/Port and on each Node
- **ClusterIP** - (default) - IP accessible only inside the cluster
- **LoadBalancer** - Cloud specific load balancer, redirects to Node services

# Defining Services

```
kubectl expose deployment hello-world --port=80 --target-port=8080
```

```
apiVersion: v1
kind: Service
metadata:
  name: hello-world
spec:
  selector:
    app: hello-world
  ports:
    - port: 80
      protocol: TCP
      targetPort: 8080
```



# Monitoring Application Health

- How do we know a Pod is up and ready to get new application connections?
- How do we know if an application has crashed and stopped responding?
- How do we know if a newly deployed Pod has started its application properly and is ready for traffic?

# Liveness and Readiness Probes

- **Liveness** - used to know when to restart a Container
  - Containers that are NOT live are **restarted** by the Container Runtime
- **Readiness** - use to know when a Container is ready to start accepting traffic
  - Endpoint Objects (Pod IP/Port) that are NOT ready are **removed** from Service load balancer
- Types of Probes
  - **Exec** - execute a command and look for an exit code of 0
  - **TCP** - completes a TCP connection to the defined port
  - **HTTP GET** - Retrieve a defined URL
    - looking for a Response Code  $\geq 200$  and  $< 400$

# Defining Liveness and Readiness Probes

```
kind: Pod
```

```
...
```

```
spec:
```

```
  containers:
```

```
    - name: hello-world
```

```
      image: psk8s.azurecr.io/hello-app:1.0
```

```
  ports:
```

```
    - containerPort: 8080
```

```
  livenessProbe:
```

```
    httpGet:
```

```
      path: /
```

```
      port: 8080
```

```
      initialDelaySeconds: 10
```

```
      periodSeconds: 10
```

```
  readinessProbe:
```

```
    httpGet:
```

```
      path: /
```

```
      port: 8080
```

```
      initialDelaySeconds: 10
```

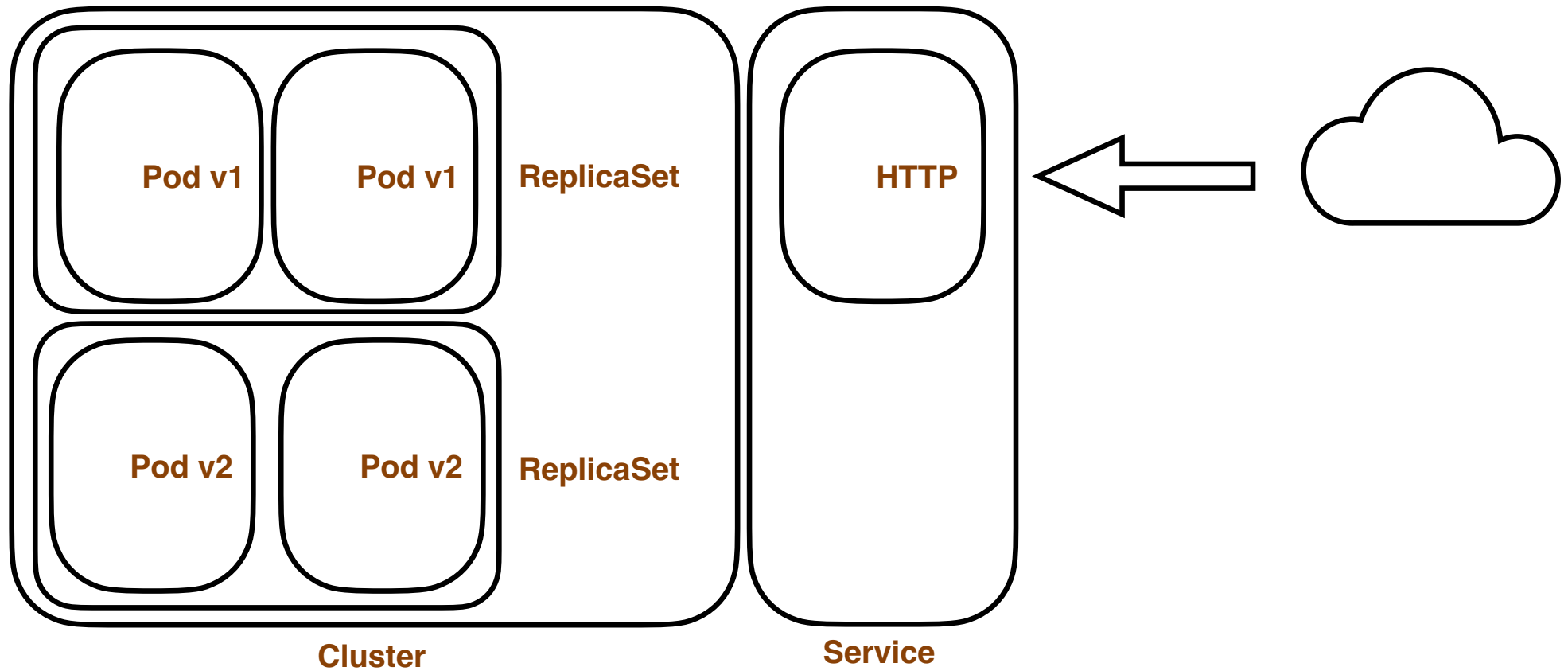
```
      periodSeconds: 10
```



# Using Deployments

- **Deployments** are used to provide declarative updates to Pods and **ReplicaSets**
- We define the state and use the Deployment Controller to move towards that state
- **Deployments** are made of **ReplicaSets** and manage the transition between the **ReplicaSets**
- The pod-template hash Label is used to differentiate between versions of **ReplicaSets**

# Controller Operations - Deployment



# Using Deployments

- Give you the controlled rollout of a new version of your application
  - Control the rate of deploying new Pods
    - **Recreate** vs. **RollingUpdate**
  - Control rollout with health checks
    - **maxSurge**
    - **maxUnavailable**
- Rollback to an earlier version
- Scale based on load
- Pause to make corrections

# Defining a Deployment



# Hands on Lab

- Working with Namespaces and Labels
- Using Services and Endpoints
- Using Deployments
  - Rollout
  - Changing an image
  - Rolling back
- Understanding how Kubernetes uses Labels in Controllers and Services
  - **Deployments**
  - **ReplicaSets**



# Review

- **Deploying Applications in Kubernetes**
  - Namespaces
  - Labels
  - Services
  - Deployments