

# Creating Pods

---



**Dan Wahlin**

WAHLIN CONSULTING

@danwahlin [www.codewithdan.com](http://www.codewithdan.com)



# Module Overview

Pod Core Concepts

Creating a Pod

kubectl and Pods

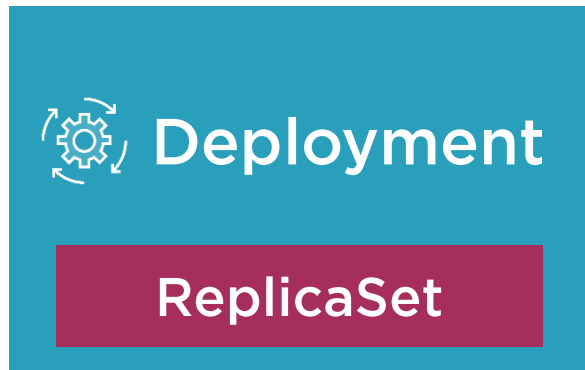
YAML Fundamentals

Defining a Pod with YAML

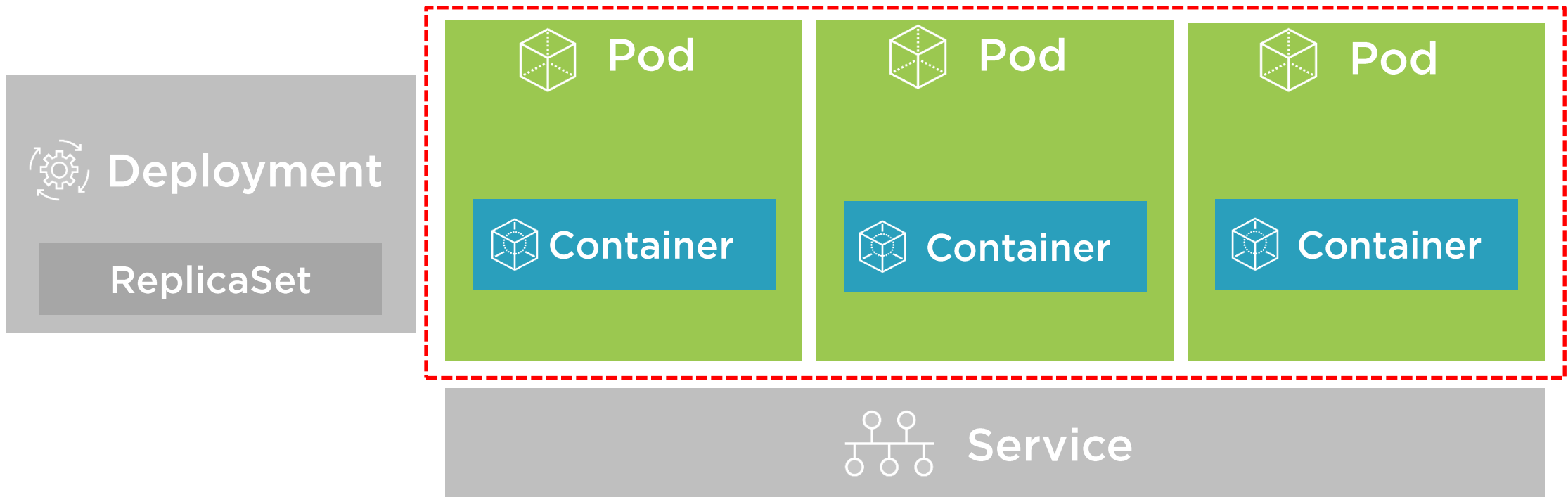
Pod Health



# You Are Here



# You Are Here



# Pod Core Concepts

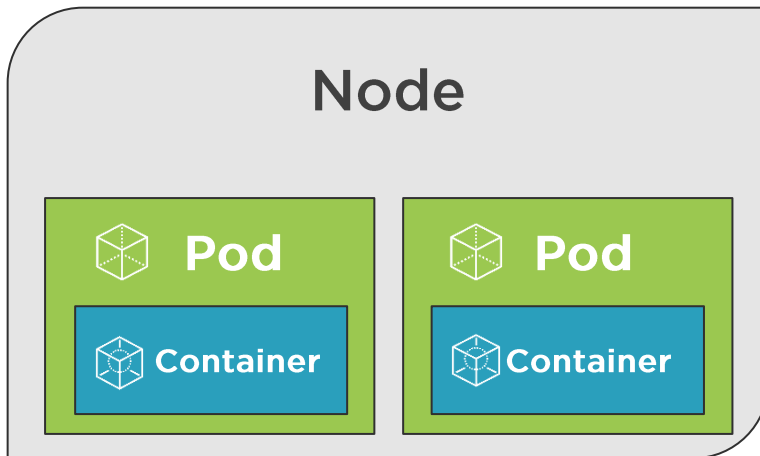
---



A Pod is the basic execution unit of a Kubernetes application—the smallest and simplest unit in the Kubernetes object model that you create or deploy.



# Kubernetes Pods



Smallest object of the Kubernetes object model

Environment for containers

Organize application "parts" into Pods (server, caching, APIs, database, etc.)

Pod IP, memory, volumes, etc. shared across containers

Scale horizontally by adding Pod replicas

Pods live and die but never come back to life



# The Role of Pods



Master



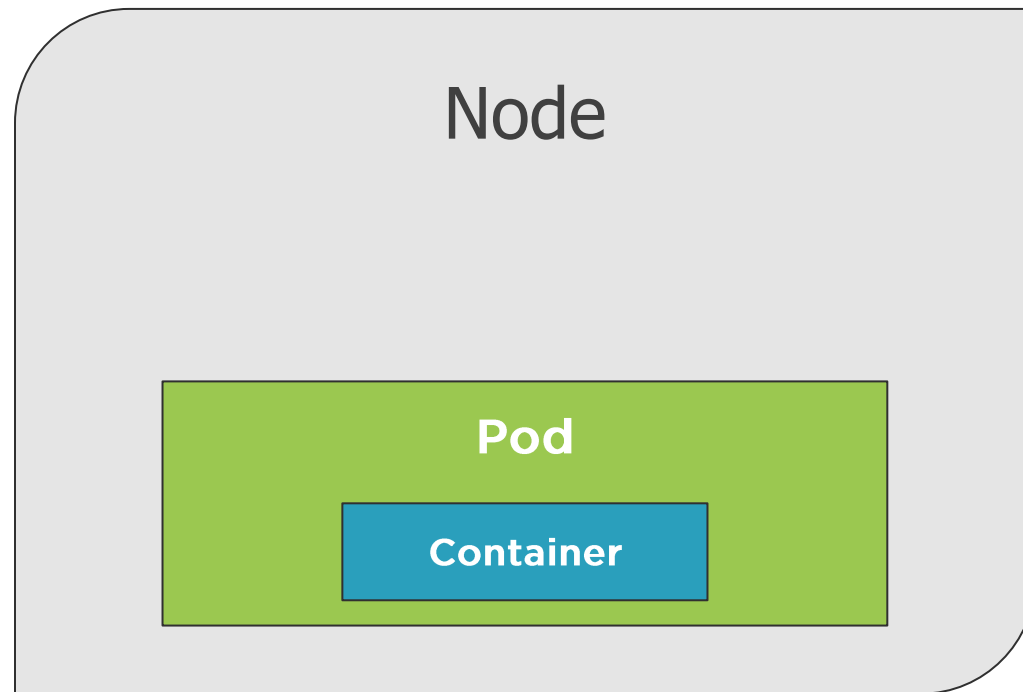
Node

Pod

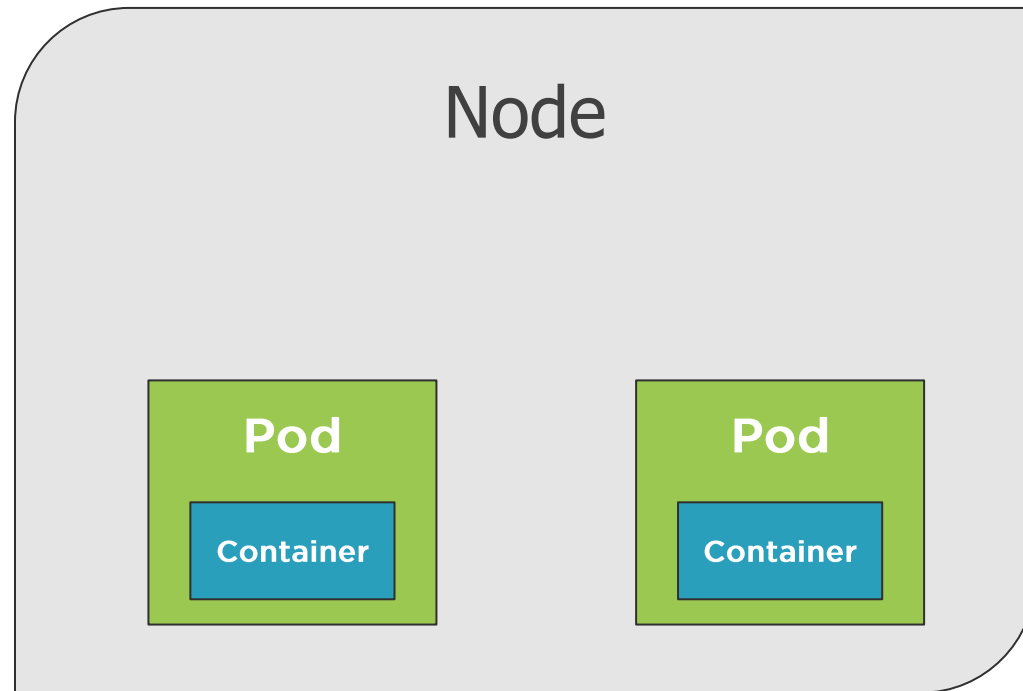




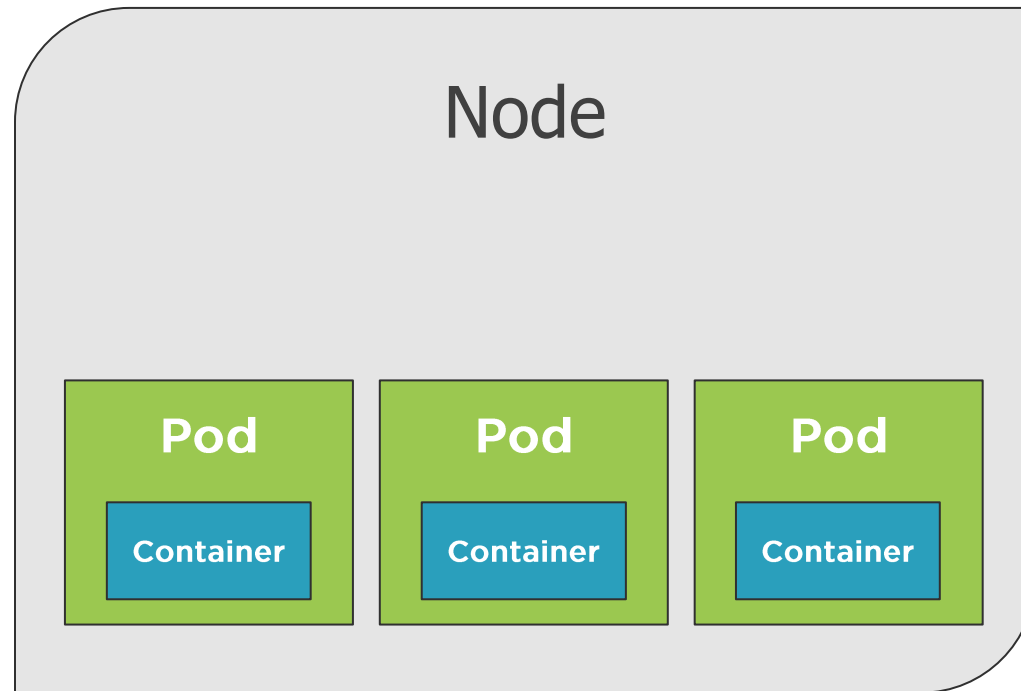
# The Role of Pods



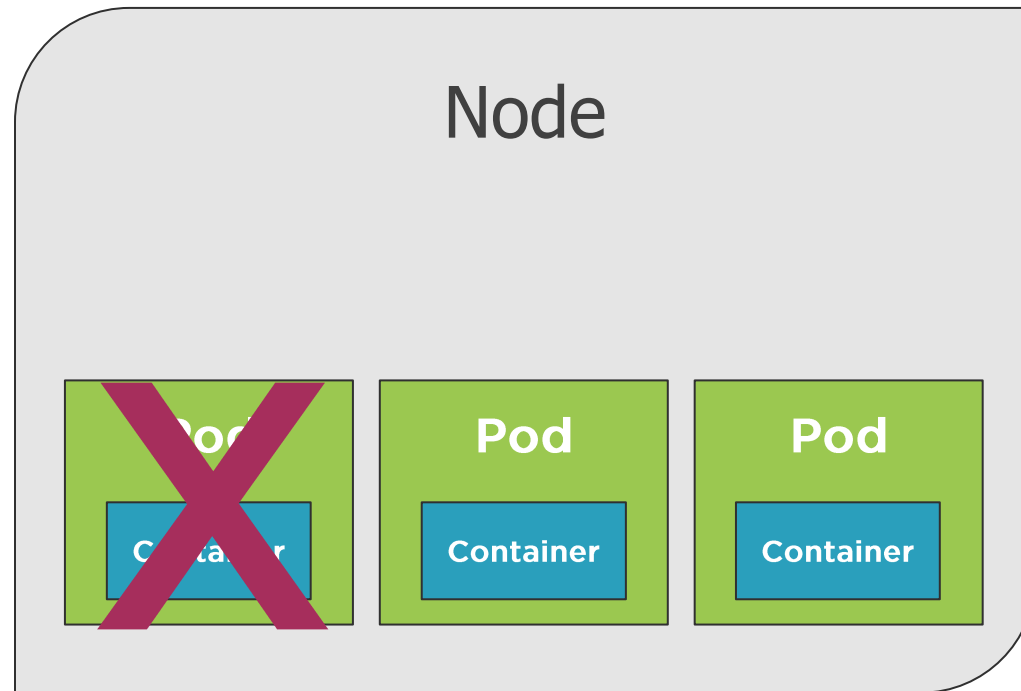
# The Role of Pods



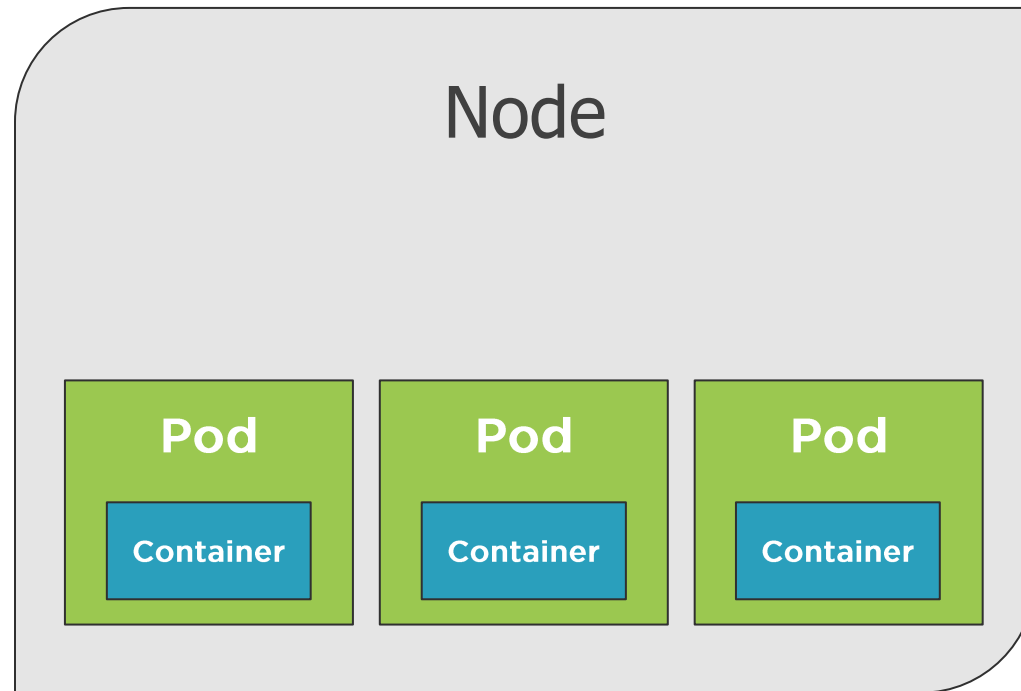
# The Role of Pods



# The Role of Pods



# The Role of Pods



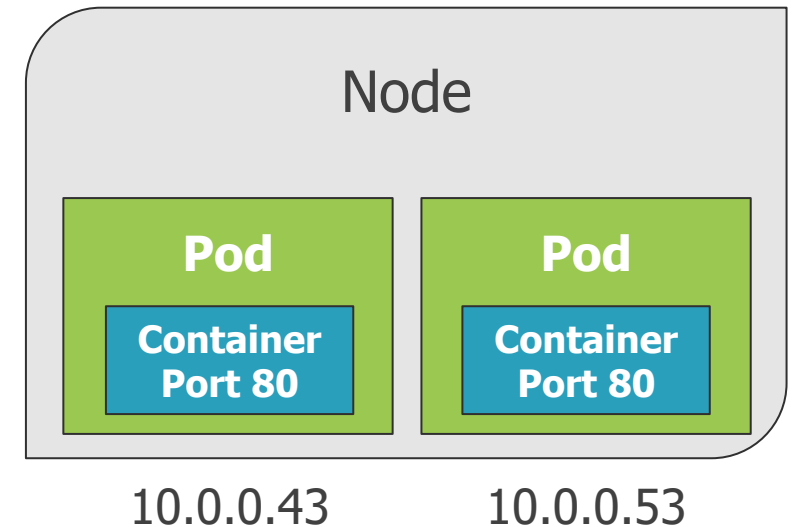
Pod containers share the same Network namespace (share IP/port)

Pod containers have the same loopback network interface (localhost)

Container processes need to bind to different ports within a Pod

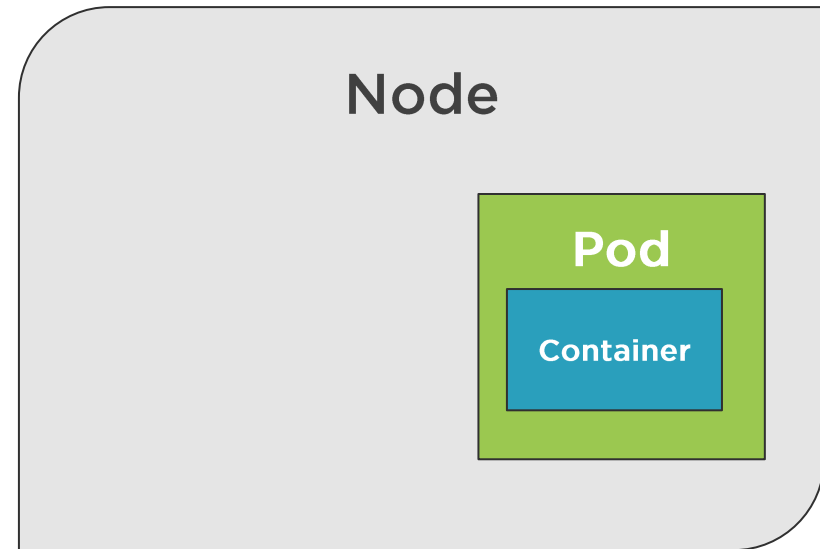
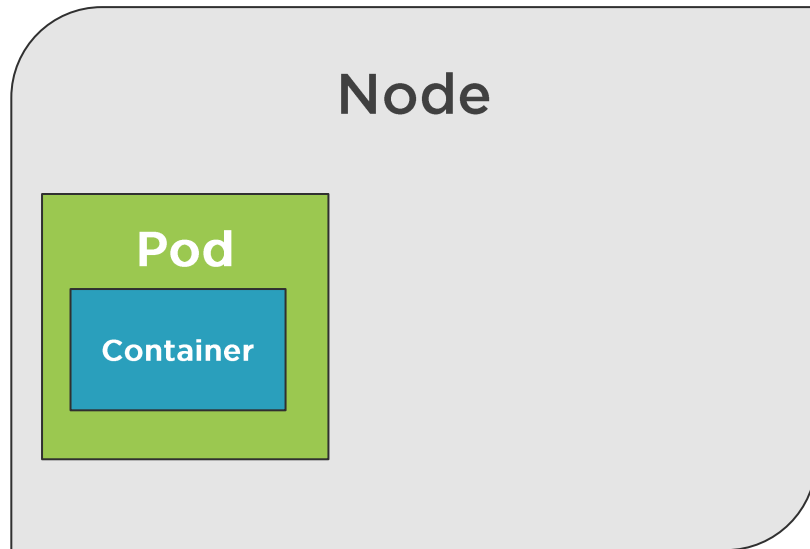
Ports can be reused by containers in separate Pods

## Pods, IPs, and Ports



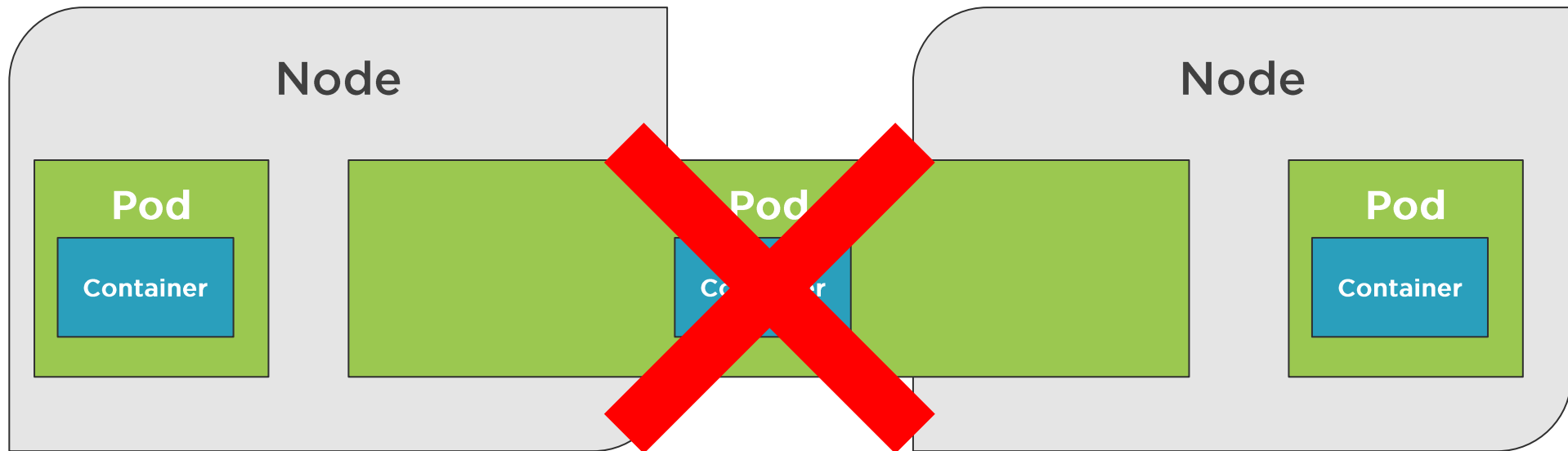
# Nodes and Pods

**Pods do not span nodes**



# Nodes and Pods

**Pods do not span nodes**





# Creating a Pod

---



# Running a Pod

There are several different ways to schedule a Pod:

**kubectl run** command

**kubectl create/apply** command with a yaml file

# Run the nginx:alpine container in a Pod

```
kubectl run [podname] --image=nginx:alpine
```

```
# List only Pods
```

```
kubectl get pods
```

```
# List all resources
```

```
kubectl get all
```

Get Information about a Pod

The **kubectl get** command can be used to retrieve information about Pods and many other Kubernetes objects



# Expose a Pod Port

Pods and containers are only accessible within the Kubernetes cluster by default

One way to expose a container port externally: **kubectl port-forward**

```
# Enable Pod container to be  
# called externally
```

```
kubectl port-forward [name-of-pod] 8080:80
```

Internal port



External port

# Will cause pod to be recreated

```
kubectl delete pod [name-of-pod]
```

# Delete Deployment that manages the Pod

```
kubectl delete deployment [name-of-deployment]
```

## Deleting a Pod

Running a Pod will cause a Deployment to be created

To delete a Pod use **kubectl delete pod** or find the deployment and use **kubectl delete deployment**



# kubectl and Pods

---



```
kubectl run [pod-name] --image=nginx:alpine
```

```
kubectl get pods
```

```
kubectl port-forward [pod-name] 8080:80
```

```
kubectl delete pod [pod-name]
```

## Working with Pods Using kubectl

**Different kubectl commands can be used to run, view, and delete Pods**



# YAML Fundamentals

---





# YAML Review



YAML files are composed of maps and lists

Indentation matters (be consistent!)

Always use spaces

**Maps:**

- name: value pairs
- Maps can contain other maps for more complex data structures

**Lists:**

- Sequence of items
- Multiple maps can be defined in a list

```
key: value
complexMap:
  key1: value
  key2:
    subKey: value
items:
  - item1
  - item2
itemsMap:
  - map1: value
    map1Prop: value
  - map2: value
    map2Prop: value
```

- ◀ YAML maps define a key and value
- ◀ More complicated map structures can be defined using a key that references another map
- ◀ YAML lists can be used to define a sequence of items
- ◀ YAML lists can define a sequence maps

Note:

- Indentation matters
- Use spaces NOT tabs



# Defining a Pod with YAML

---



# Defining a Pod with YAML



Pod

+

**kubectl**

=



```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx
spec:
  containers:
  - name: my-nginx
    image: nginx:alpine
```

- ◀ Kubernetes API version
- ◀ Type of Kubernetes resource
- ◀ Metadata about the Pod
- ◀ The spec/blueprint for the Pod
- ◀ Information about the containers that will run in the pod



# Creating a Pod Using YAML

To create a pod using YAML use the **kubectl create** command along with the **--filename** or **-f** switch

```
# Perform a "trial" create and also validate the YAML
kubectl create -f file.pod.yml --dry-run --validate=true
```

```
# Create a Pod from YAML
# Will error if Pod already exists
kubectl create -f file.pod.yml
```



Default value

# Creating or Applying Changes to a Pod

To create or apply changes to a pod using YAML use the **kubectl apply** command along with the **--filename** or **-f** switch

```
# Alternate way to create or apply changes to a  
# Pod from YAML
```

```
kubectl apply -f file.pod.yml
```

```
# Use --save-config when you want to use  
# kubectl apply in the future
```

```
kubectl create -f file.pod.yml --save-config
```

Store current  
properties in  
resource's annotations



```
apiVersion: v1

kind: Pod

metadata:
  annotations:
    kubectl.kubernetes.io/
    last-applied-configuration:
    {"apiVersion":"v1","kind":"Pod",
     "metadata":{"
       "name": "my-nginx"
     ...
    }}
  ...
```

- ◀ **--save-config** causes the resource's configuration settings to be saved in the annotations
- ◀ Example of saved configuration
- ◀ Having this allows in-place changes to be made to a Pod in the future using **kubectl apply**

In-place/non-disruptive changes can also be made to a Pod using **kubectl edit** or **kubectl patch**.





# Deleting a Pod

To delete a Pod use **kubectl delete**

```
# Delete Pod
```

```
kubectl delete pod [name-of-pod]
```

```
# Delete Pod using YAML file that created it
```

```
kubectl delete -f file.pod.yml
```

# kubectl and YAML

---



```
kubectl create -f nginx.pod.yml --save-config  
kubectl describe pod [pod-name]  
kubectl apply -f nginx.pod.yml  
kubectl exec [pod-name] -it sh  
kubectl edit -f nginx.pod.yml  
kubectl delete -f nginx.pod.yml
```

## Creating and Inspecting Pods with kubectl

**Several different commands can be used to create and modify Pods**



# Pod Health

---



Kubernetes relies on Probes to  
determine the health of a  
Pod container.



A Probe is a diagnostic performed periodically by the kubelet on a Container.



# Types of Probes



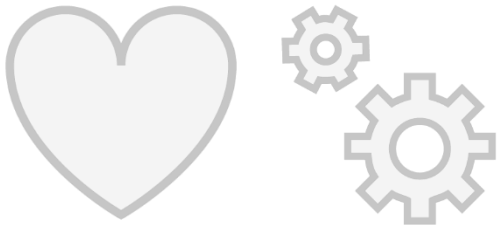
**Liveness  
Probe**



**Readiness  
Probe**



# Types of Probes



**Liveness probes** determine if a Pod is healthy and running as expected

**Readiness probes** determine if a Pod should receive requests

**Failed Pod containers are recreated by default (restartPolicy defaults to Always)**



**ExecAction** – Executes an action inside the container

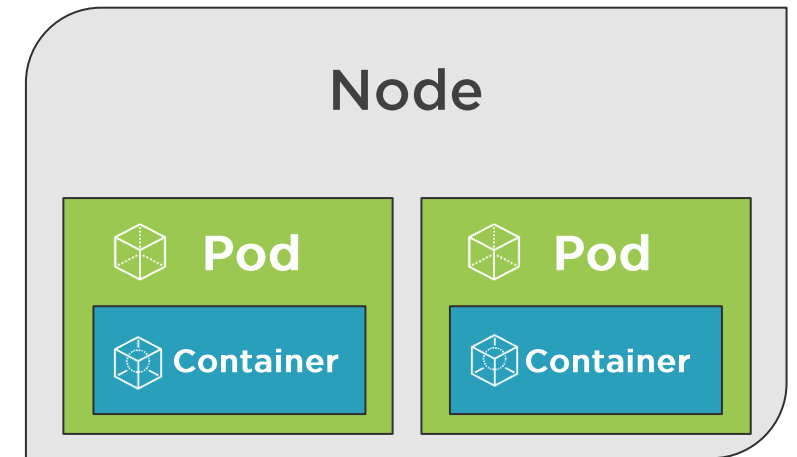
**TCPSocketAction** – TCP check against the container's IP address on a specified port

**HTTPGetAction** – HTTP GET request against container

Probes can have the following results:

- Success
- Failure
- Unknown

## Probe Types



```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: my-nginx
    image: nginx:alpine
    livenessProbe:
      httpGet:
        path: /index.html
        port: 80
      initialDelaySeconds: 15
      timeoutSeconds: 2
      periodSeconds: 5
      failureThreshold: 1
```

- ◀ Define liveness probe
- ◀ Check /index.html on port 80
- ◀ Wait 15 seconds
- ◀ Timeout after 2 seconds
- ◀ Check every 5 seconds
- ◀ Allow 1 failure before failing Pod



# Defining an ExecAction Liveness Probe

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox

    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30;
      rm -rf /tmp/healthy; sleep 600

    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

◀ Define args for container

◀ Define liveness probe

◀ Define action/command to execute



```
apiVersion: v1

kind: Pod

...

spec:
  containers:
  - name: my-nginx
    image: nginx:alpine
    readinessProbe:
      httpGet:
        path: /index.html
        port: 80
      initialDelaySeconds: 2
      periodSeconds: 5
```

- ◀ Define readiness probe
- ◀ Check /index.html on port 80
- ◀ Wait 2 seconds
- ◀ Check every 5 seconds



Readiness Probe:

When should a container start  
receiving traffic?

Liveness Probe:

When should a container restart?



# Pod Health in Action

---



# Summary



Pods are the smallest unit of Kubernetes

Containers run within Pods and share a Pod's memory, IP, volumes, and more

Pods can be started using different kubectl commands

YAML can be used to create a Pod

Health checks provide a way to notify Kubernetes when a Pod has a problem

