

Running and Managing Pods



Anthony E. Nocentino

ENTERPRISE ARCHITECT @ CENTINO SYSTEMS

@nocentino www.centinosystems.com

Course Overview



Using the Kubernetes API

**Managing Objects with Labels, Annotations,
and Namespaces**

Running and Managing Pods

Overview

Understanding Pods

Controllers and Pods

Multi-container Pods

Managing Pod Health with Probes

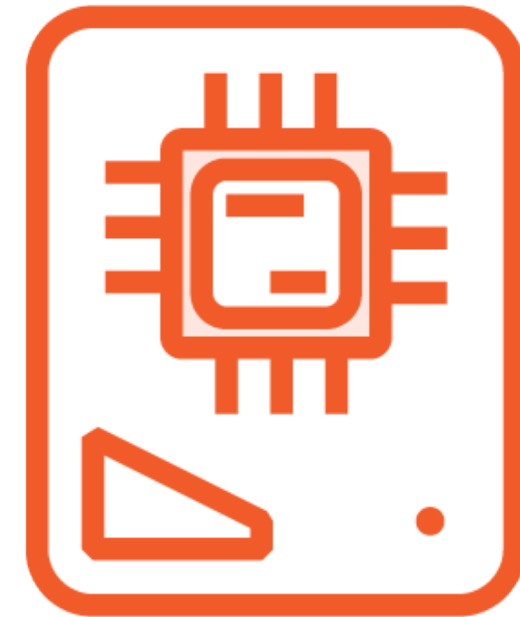
What Is a Pod?



**Wrapper around
your container
based application**



**One or more
containers**



Resources

What Is a Pod? (con't)



Unit of scheduling

Allocating work

A process that's running in your cluster

Unit of deployment

Your application configuration

Resources - Networking and Storage

Why do we need Pods?

Provide higher level abstraction
over a container for manageability

How Pods Manage Containers



Single Container Pods



Multi-Container Pods

Single Container Pods



Most common deployment scenario

Generally a single process running in a container

Often leads to easier application scaling

Controllers and Pods



Controllers keep your apps in the desired state

Responsible for starting and stopping Pods

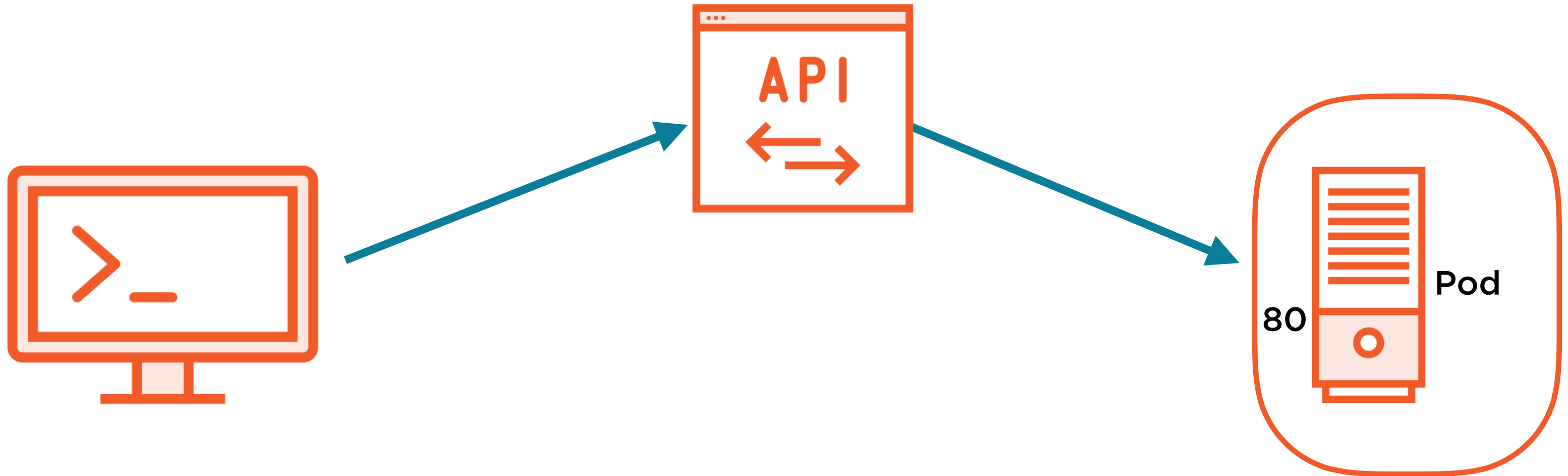
Application scaling

Application recovery

You don't want to run bare/naked Pods

They won't be recreated in the event of a failure

Working with Pods - kubectl



```
kubectl exec -it POD1 --container CONTAINER1 -- /bin/bash
```

```
kubectl logs POD1 --container CONTAINER1
```

```
kubectl port-forward pod POD1 LOCALPORT:CONTAINERPORT
```

Demo

Running Pods

- Bare Pods
- Creating Pods in a Deployment
- Using port-forward to access a Pod's application

Multi-container Pods



Tightly coupled applications

Scheduling processes together

Requirement on some shared resource

Usually something generating data while the other process consumes

Don't use this to influence scheduling, we use other techniques for that!

Multi-container Pods

```
apiVersion: v1
kind: Pod
metadata:
  name: multicontainer-pod
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    ...
  - name: alpine
    image: alpine
```

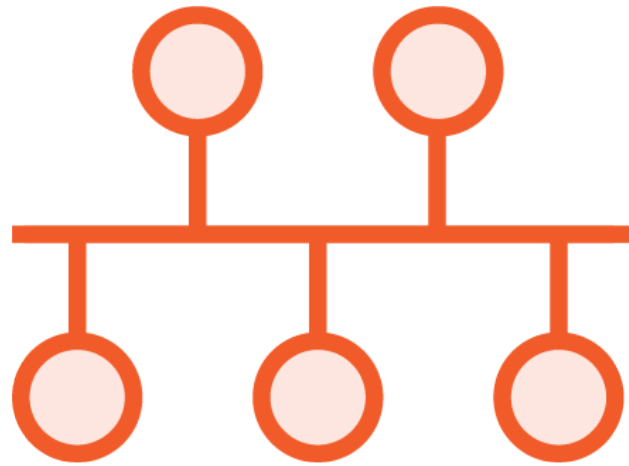
Common Anti-Pattern for Multi-container Pods



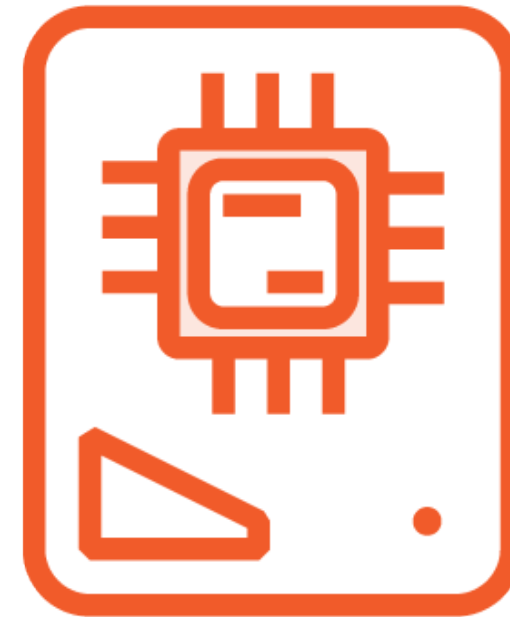
Recovery Options

Limits Scaling

Shares Resources Inside a Pod

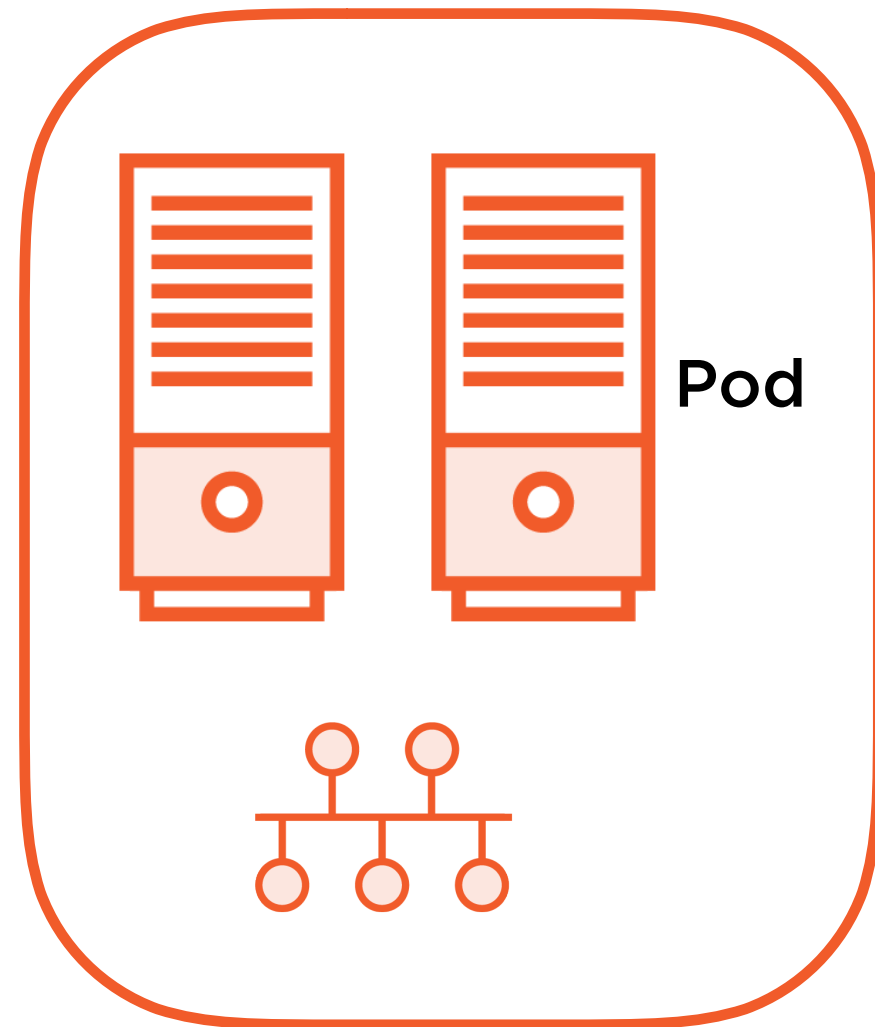


Networking



Storage

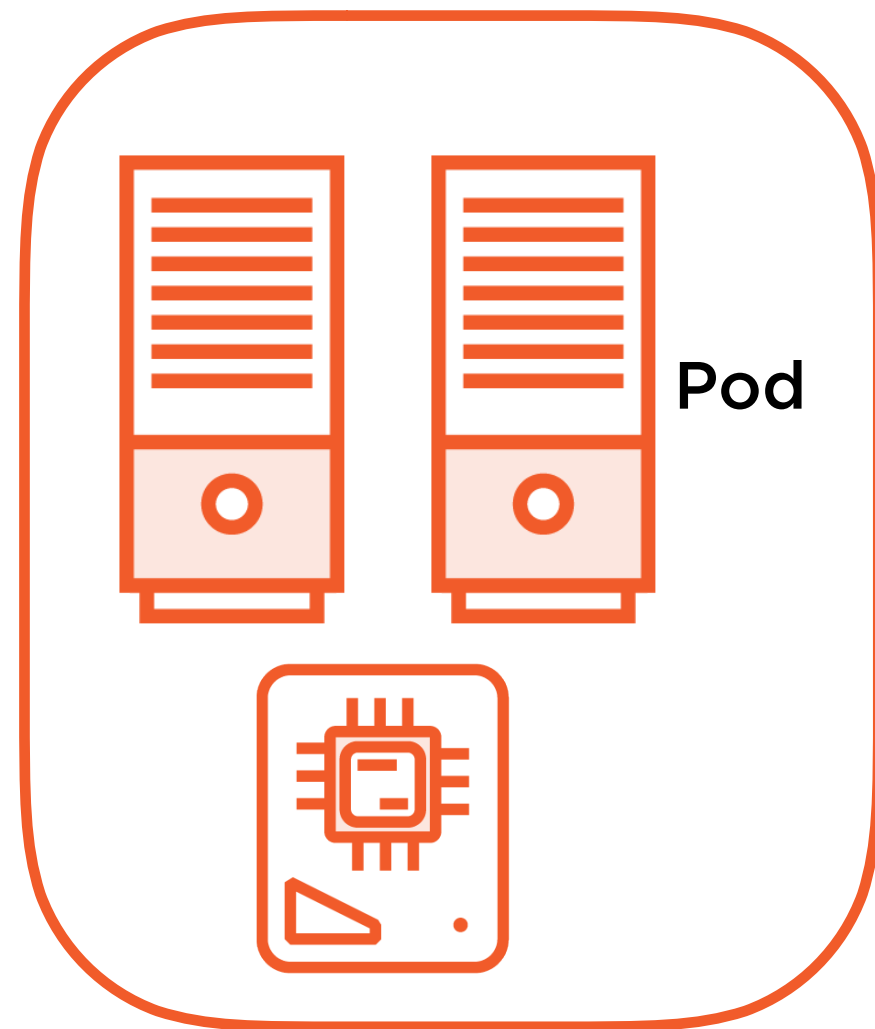
Shared Resources Inside a Pod - Networking



Shared loopback interface, used for communication over localhost

Be mindful of application port conflicts

Shared Resources Inside a Pod - Storage



Each container image has it's own file system

Volumes are defined at the Pod level

Shared amongst the containers in a Pod

Mounted into the containers' file system

Common way for containers to exchange data

Demo

Running Multi-container Pods

Sharing data between containers in a Pod

Pod Lifecycle

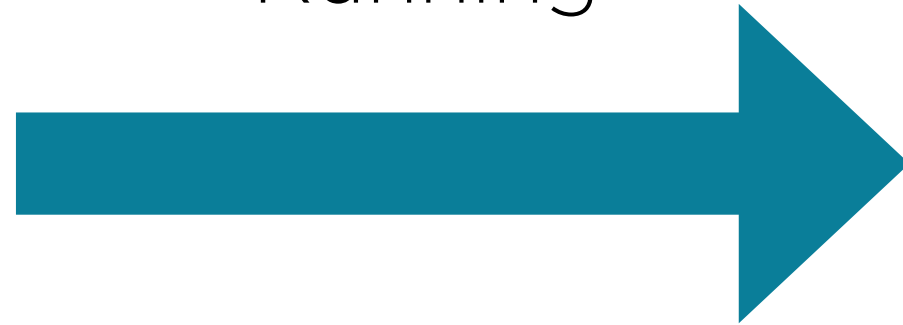
Creation



Administratively

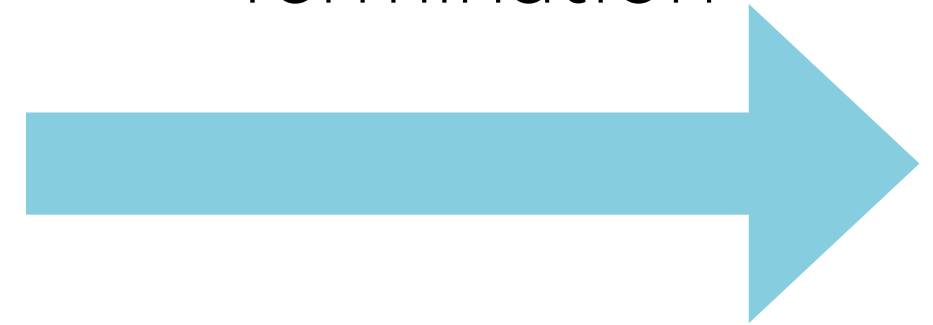
Controller

Running



Scheduled to a Node

Termination



Process is terminated/crashed

Pod is deleted

Evicted due to lack of resources

Node failure or maintenance

No Pod is redeployed

Stopping/Terminating Pods

Grace Period Timer
(30 sec default)

Pods changes to
Terminating

SIGTERM

Service Endpoints
and Controllers
updated

IF > Grace Period
SIGKILL

API and etcd are
updated

kubectl delete pod  --grace-period=<seconds>

Force Deletion - Immediately deletes records in API and etcd

kubectl delete pod --grace-period=0 --force

Persistency of Pods

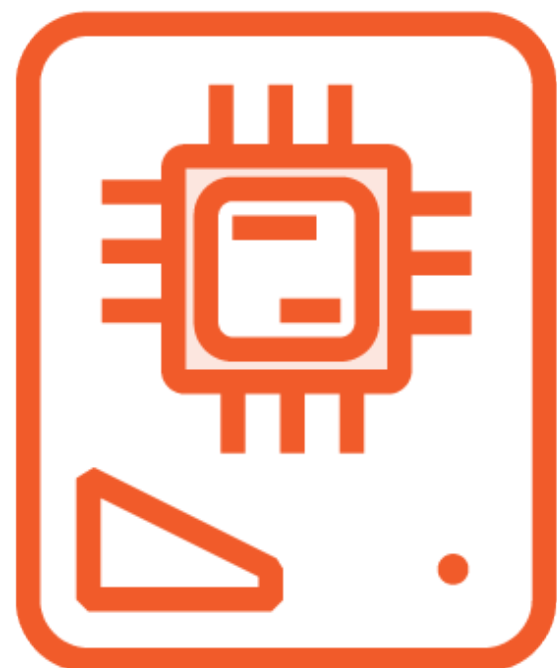


A Pod is never redeployed

If a Pod stops, a new one is created based on its Controller

Go back to the original container image(s) in the Pod definition

Persistency of Pods



Configuration is managed externally

Pod Manifests, secrets and ConfigMaps

Passing environment variables into containers

Data Persistency is managed externally

PersistentVolume

PersistentVolumeClaim

Container Restart Policy



A container in a Pod can restart independent of the Pod

Applies to containers inside a Pod and defined inside the Pod's Spec

The Pod is the environment the container runs in

Not rescheduled to another Node, but restarted by the Kubelet on that Node

Restarts with an exponential backoff, 10s, 20s, 40s capped at 5m and reset to 0 after 10m of successful runtime

Container Restart Policy



Always (default) - will restart all containers inside a Pod

OnFailure - Non-graceful termination

Never


```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx
  restartPolicy: OnFailure
```

Pod with Container Restart Policy

```
kubectl apply -f nginx.yaml
```

Demo

Pod lifecycle

Killing a container process

Container Restart Policy

Defining Pod Health



A Pod is considered ready when all containers are ready

But we'd like to be able to understand a little more about our applications

We can add additional intelligence to our Pod's state and health

Container Probes

`livenessProbe`

`readinessProbe`

livenessProbes



Runs a diagnostic check on a container

Per container setting

On failure, the Kubelet restarts the container

Container Restart Policy

Give Kubernetes a better understanding of our application

readinessProbes



Runs a diagnostic check on the container

Per container setting

On startup, your application won't receive traffic until ready

On failure, removes Pod from load balancing or replication controller

Applications have long startup times

Prevents users from seeing errors

Types of Diagnostic Checks for Probes

Exec

tcpSocket

httpGet

Process exit code

Successfully Open
a Port

Return Code
200 => and < 400

Success

Failure

Unknown

Configuring Container Probes



`initialDelaySeconds` - number of seconds after the container has started before running container probes

`periodSeconds` - probe interval, default 10 seconds

`timeoutSeconds` Probe timeout 1 seconds

`failureThreshold` - number of missed checks before reporting failure, default 3

`successThreshold` - number of probes to be considered successful and live, default 1

```
spec:
  containers:
    ...
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
```

livenessProbes and readinessProbes

Demo

Implementing container probes

- **livenessProbes**
- **readinessProbes**

Summary

Understanding Pods

Controllers and Pods

Multi-container Pods

Managing Pod Health with Probes

Thank You!

@nocentino

www.centinosystems.com