TEL4VN

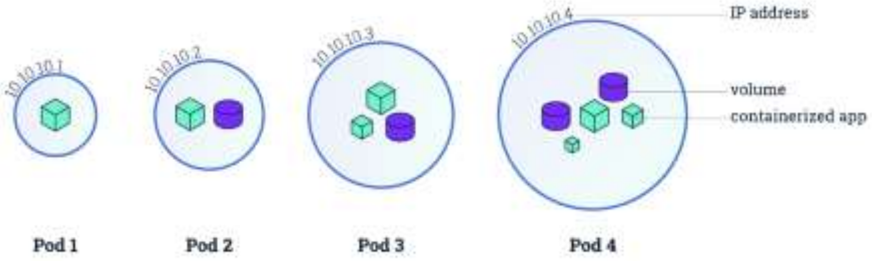
Understanding k8s workloads

THAO LUONG
03/2022

Content

- ❑ Understanding Pods, Nodes
- ❑ Controllers and Pods
- ❑ Deployment , ReplicaSet
- ❑ Multi-container Pods
- ❑ Managing Pod Heath with Probes

What is a Pod?



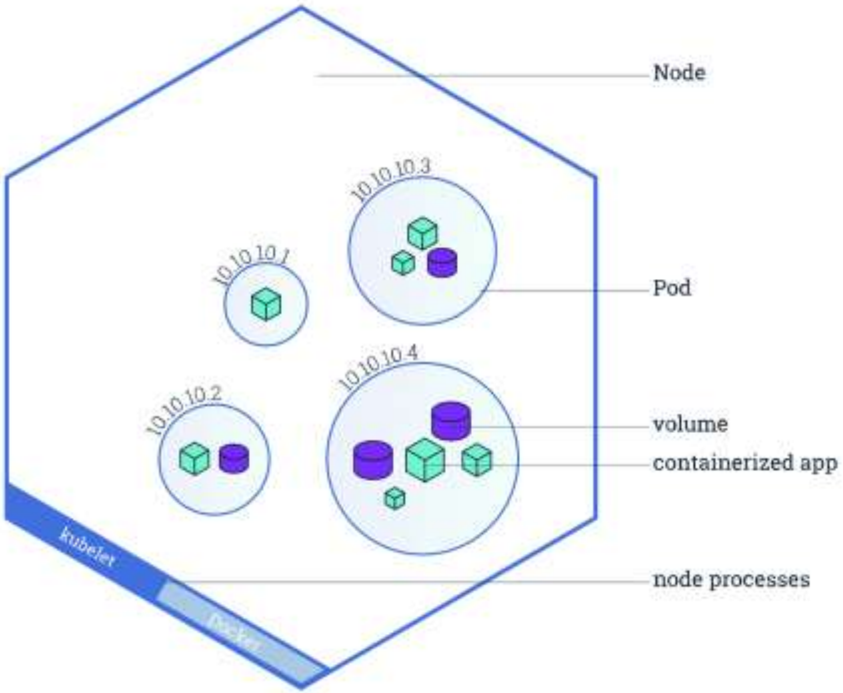
- A group of whales called “pod” = a group of one or more containers
- Shared resources like networking, storages and specification (image version, container ports…)
- “Logical host”
- Atomic unit on the k8s platform

Pods & Nodes

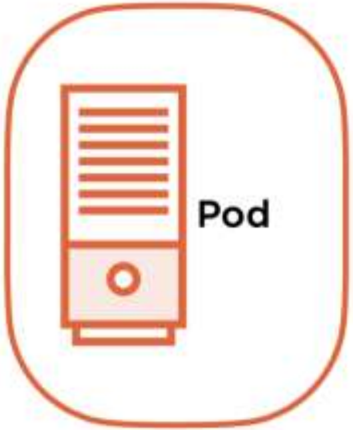
- Node can be a VM, bare-metal or even Docker container (kubernetes in Docker)

- Node runs at a least: kubelet + container runtime

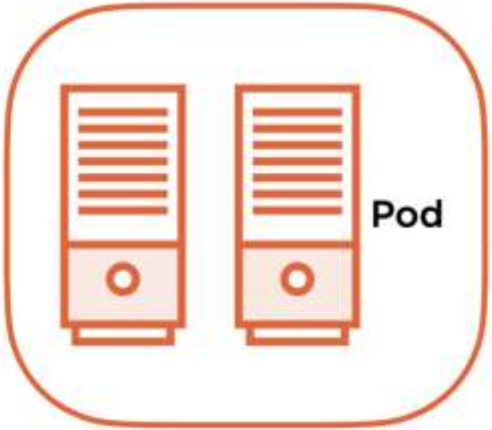
- Node failure -> identical Pods are scheduled on other nodes.



How pods manage containers



Single Container Pods

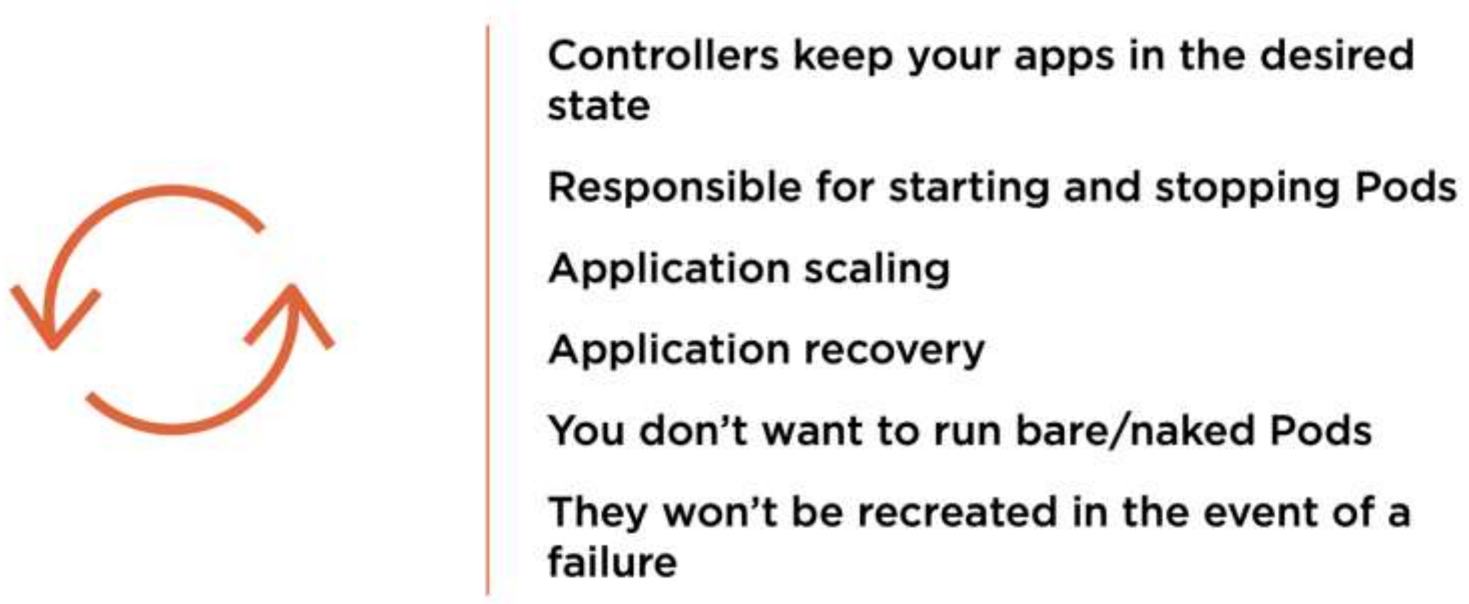


Multi-Container Pods

Single Pod manifest

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world-pod
spec:
  containers:
  - name: hello-world
    image: gcr.io/google-samples/hello-app:1.0
    ports:
    - containerPort: 80
```

Controllers & Pods



Controllers keep your apps in the desired state

Responsible for starting and stopping Pods

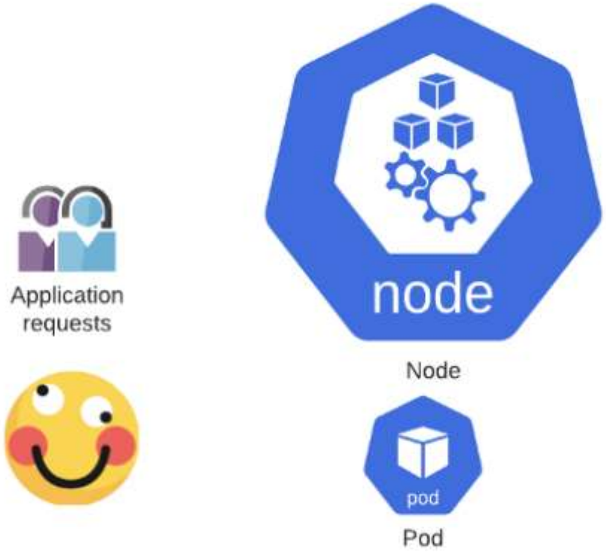
Application scaling

Application recovery

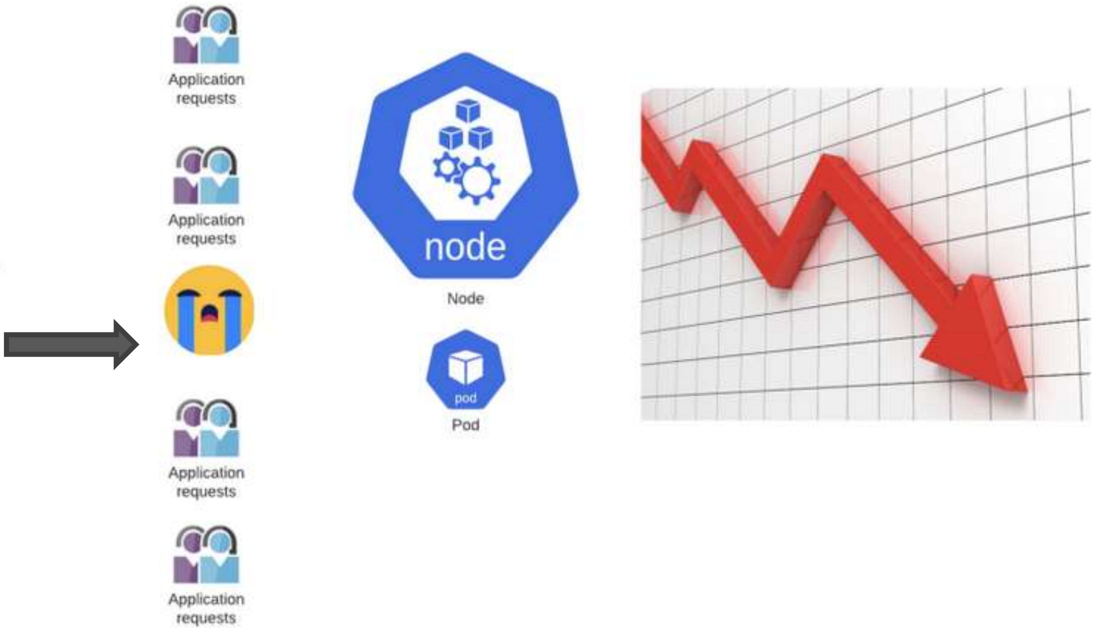
You don’t want to run bare/naked Pods

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

Understanding ReplicaSets

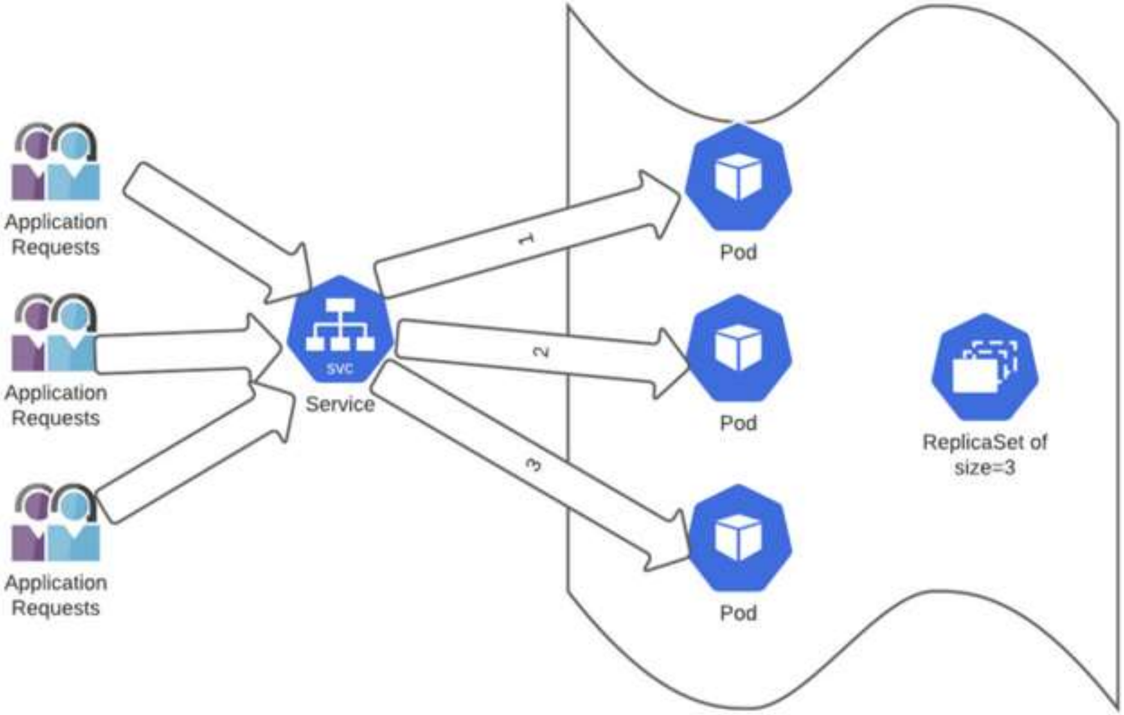


Single POD application



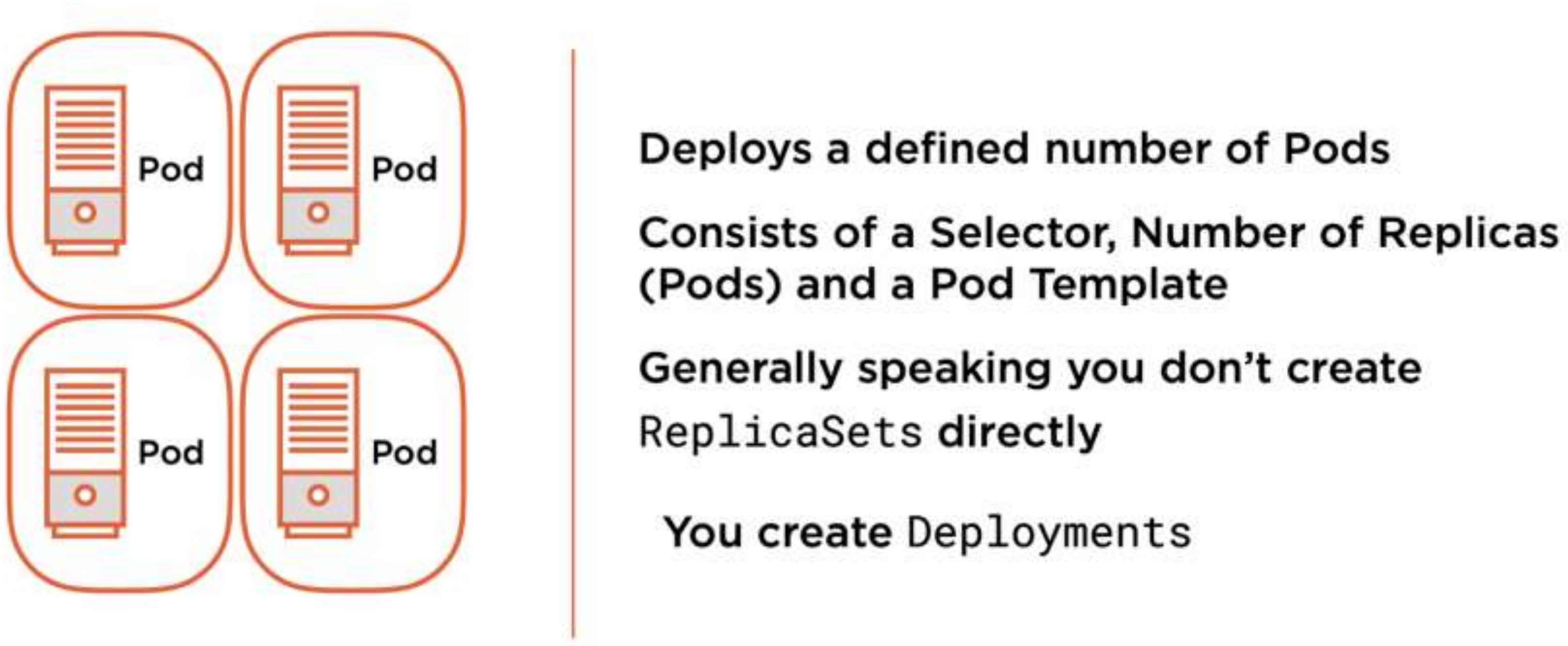
Single pod with increased application requests seem to bring a lot of sadness.

Understanding ReplicaSets



User requests getting routed to application PODs through the K8s service object.

Understanding ReplicaSets



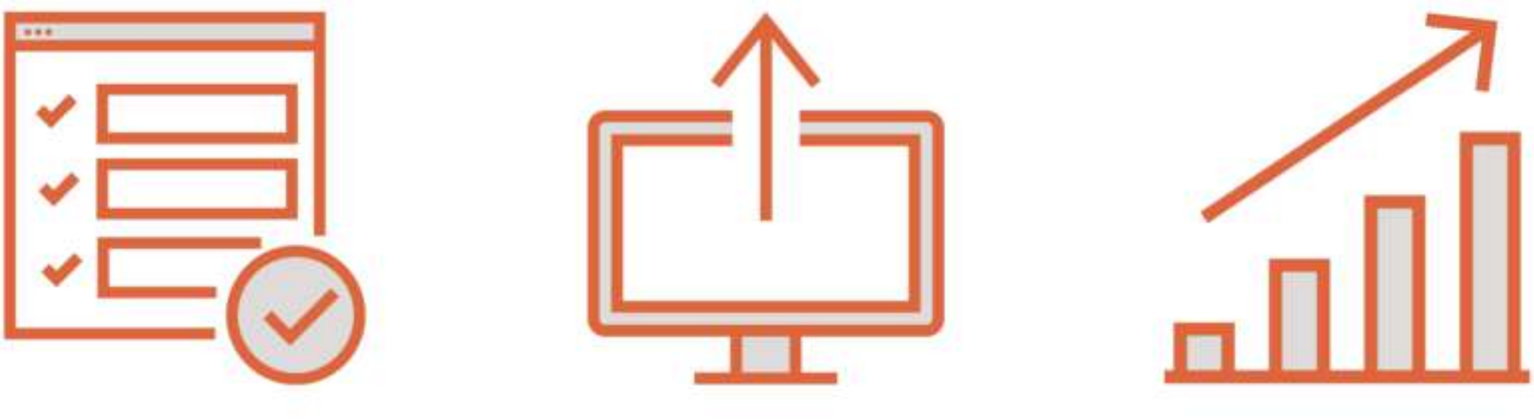
Deploys a defined number of Pods

Consists of a Selector, Number of Replicas (Pods) and a Pod Template

Generally speaking you don’t create `ReplicaSets` directly

You create `Deployments`

Managing app state with deployment

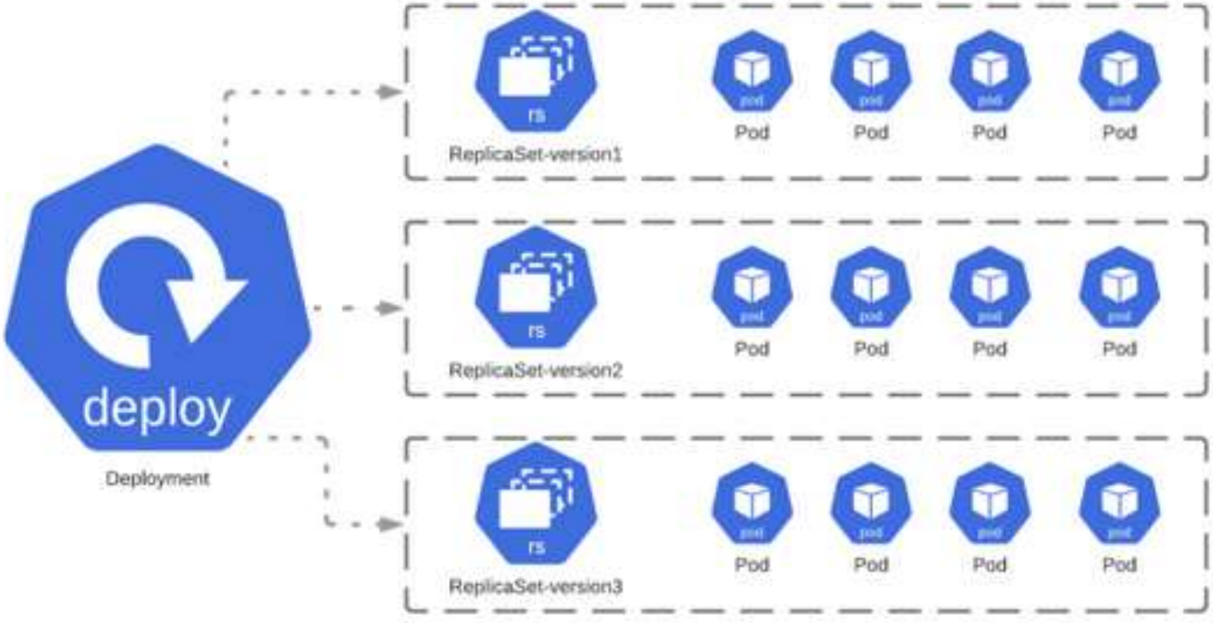


Creating

Updating

Scaling

Managing app state with deployment



The relation between Deployment, Replicaset & POD

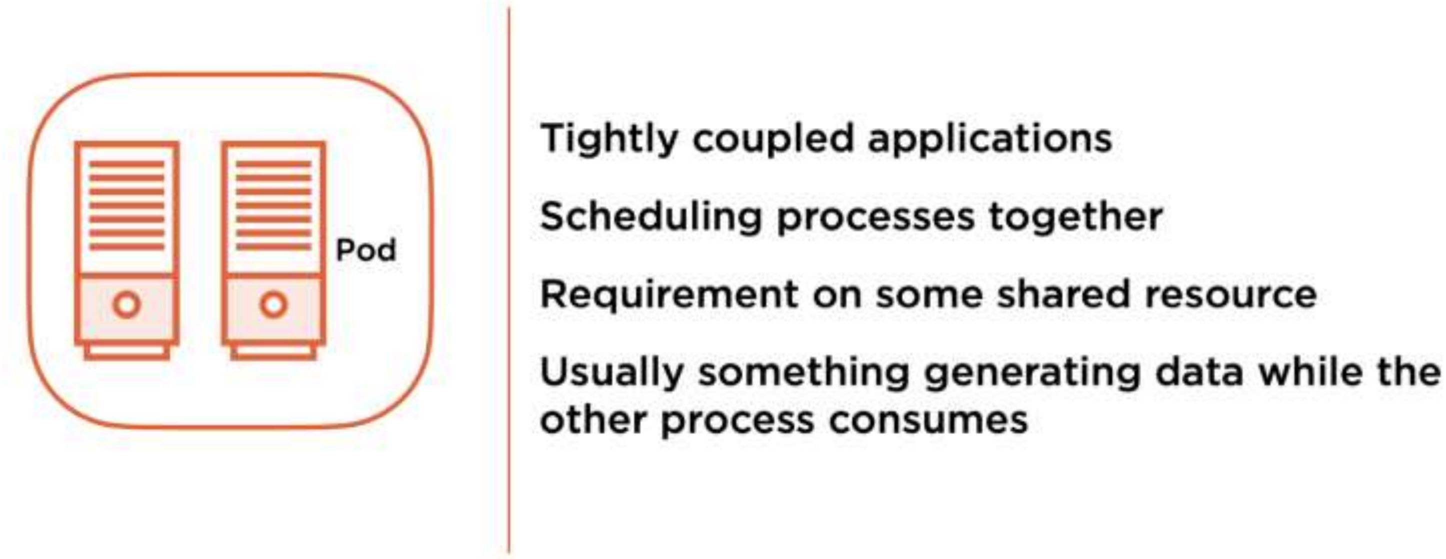
Pod/ReplicaSet/Deployment

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
    name: hello-world
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-world-pod
  template:
    metadata:
      labels:
        app: hello-world-pod
    spec:
      containers:
      - name: hello-world
        image: gcr.io/google-samples/hello-app:1.0
```

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-world-pod
spec:
  containers:
  - name: hello-world
    image: gcr.io/google-samples/hello-app:1.0
    ports:
    - containerPort: 8080
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-1
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-world-1
  template:
    metadata:
      labels:
        app: hello-world-1
    spec:
      containers:
      - name: hello-world
        image: gcr.io/google-samples/hello-app:1.0
        ports:
        - containerPort: 8080
```

Multi-container Pods



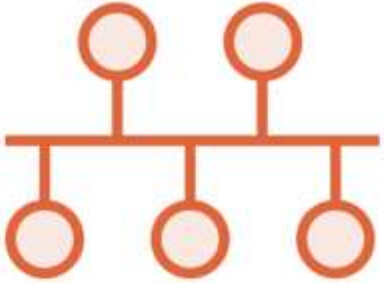
Tightly coupled applications

Scheduling processes together

Requirement on some shared resource

Usually something generating data while the other process consumes

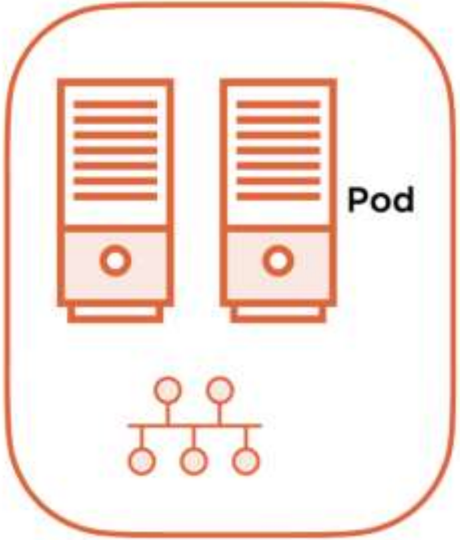
Share resources inside a pod



Networking

Storage

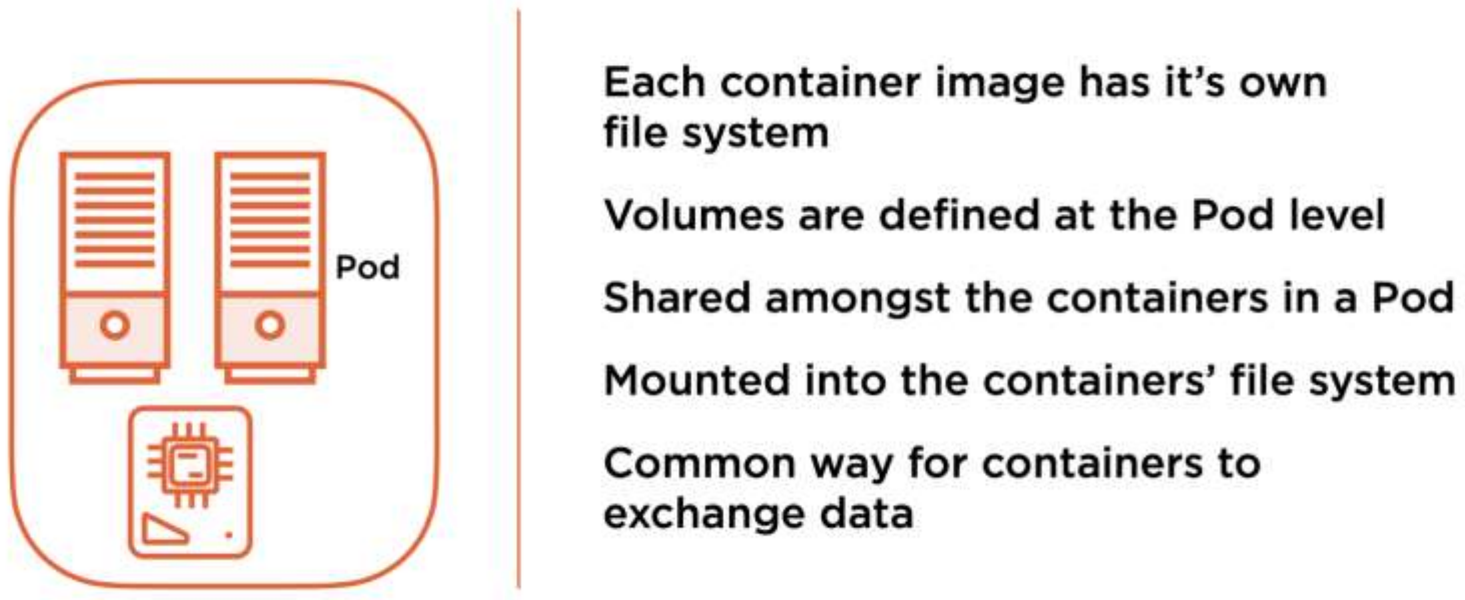
Shared networking



Shared loopback interface, used for communication over localhost

Be mindful of application port conflicts

Shared storage



Each container image has it's own file system

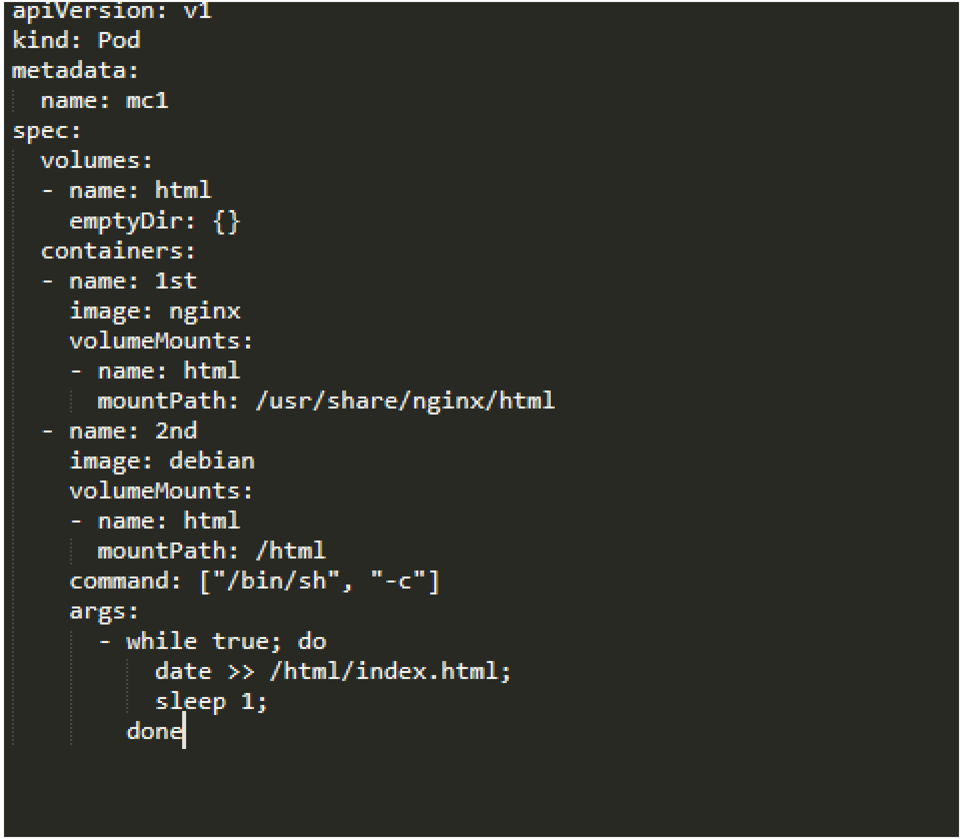
Volumes are defined at the Pod level

Shared amongst the containers in a Pod

Mounted into the containers' file system

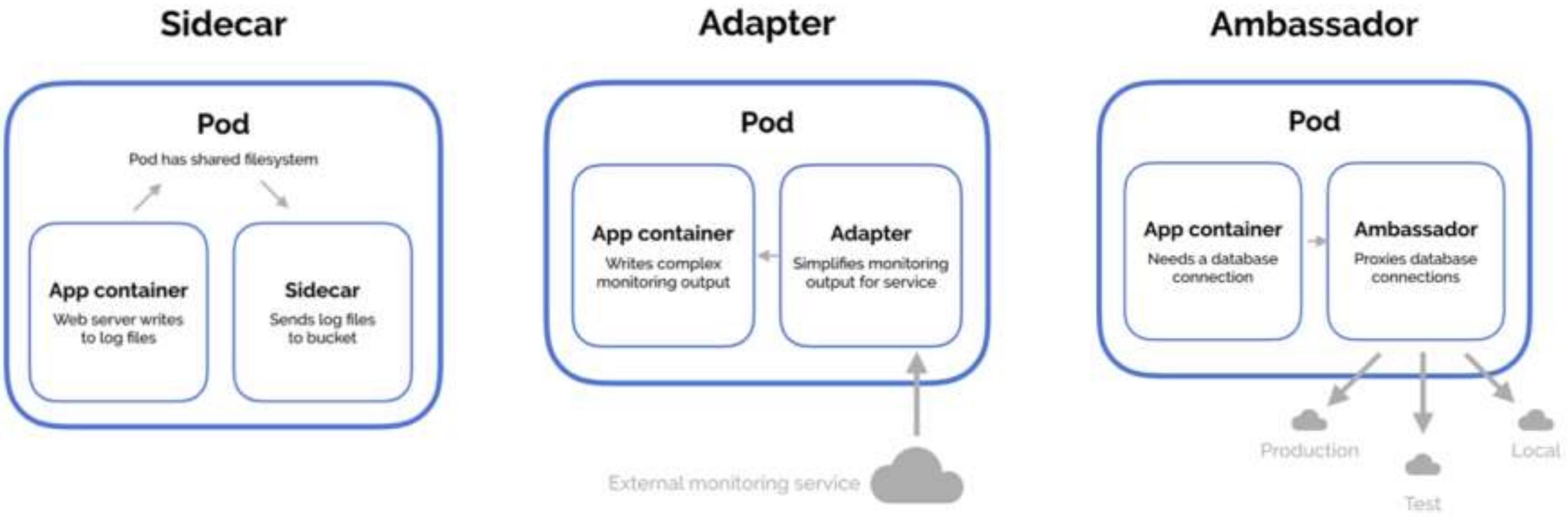
Common way for containers to exchange data

Shared storage

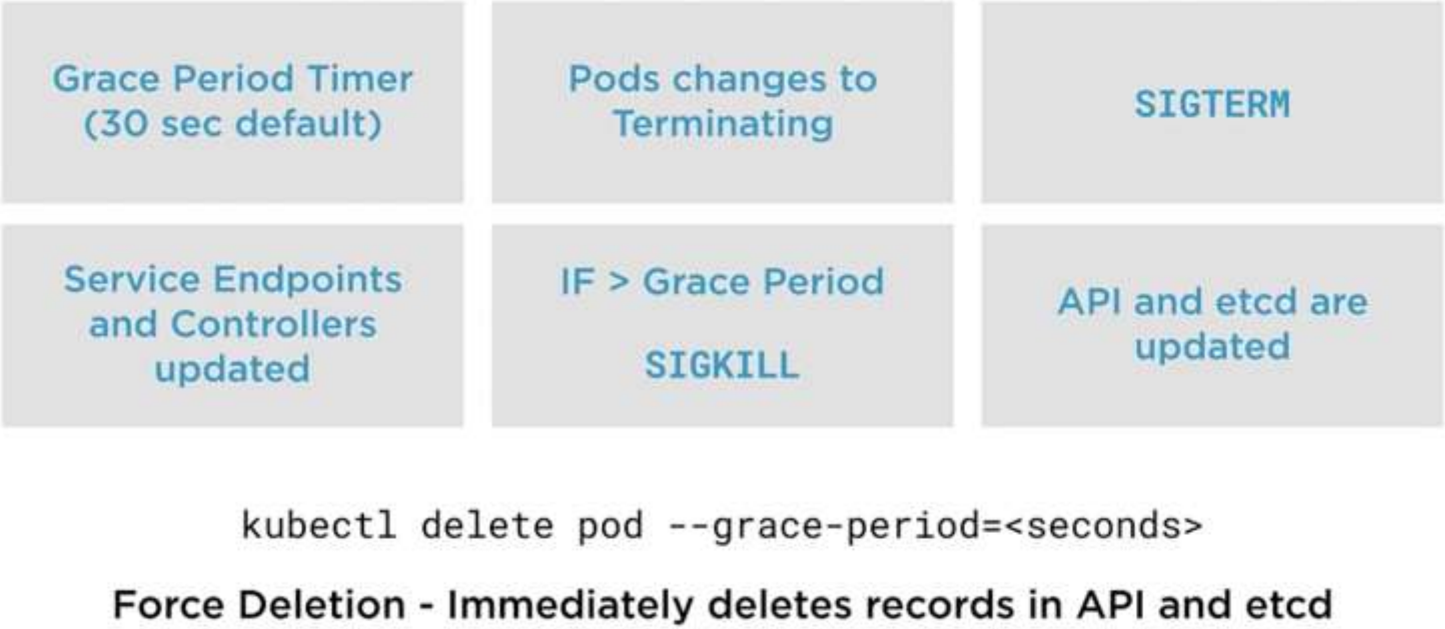


```
apiVersion: v1
kind: Pod
metadata:
  name: mc1
spec:
  volumes:
  - name: html
    emptyDir: {}
  containers:
  - name: 1st
    image: nginx
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  - name: 2nd
    image: debian
    volumeMounts:
    - name: html
      mountPath: /html
    command: ["/bin/sh", "-c"]
    args:
      - while true; do
          date >> /html/index.html;
          sleep 1;
        done
```

Common multi-container pods design


Sidecar
Pod
Pod has shared filesystem
App container
Web server writes to log files
Sidecar
Sends log files to bucket
Adapter
Pod
App container
Writes complex monitoring output
Adapter
Simplifies monitoring output for service
External monitoring service
Ambassador
Pod
App container
Needs a database connection
Ambassador
Proxies database connections
Production
Test
Local

Terminating a pod



Grace Period Timer (30 sec default)

Pods changes to Terminating

SIGTERM

Service Endpoints and Controllers updated

IF > Grace Period

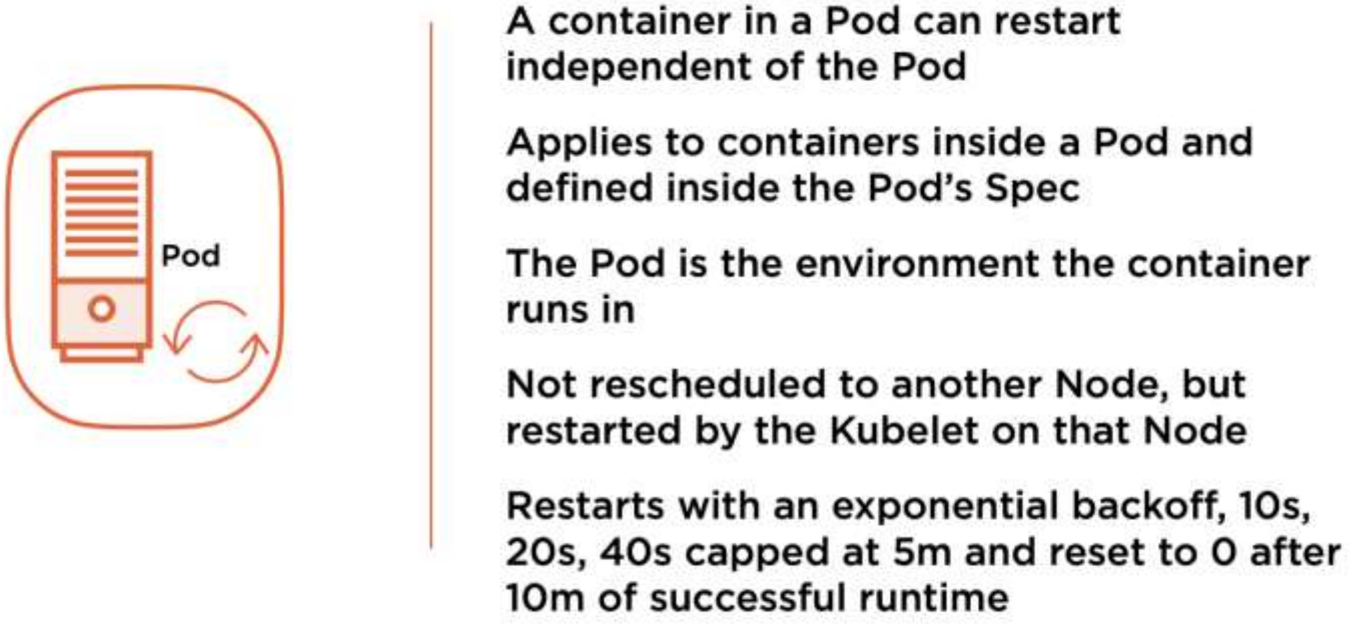
SIGKILL

API and etcd are updated

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

Force Deletion - Immediately deletes records in API and etcd

Container Restart Policy



A container in a Pod can restart independent of the Pod

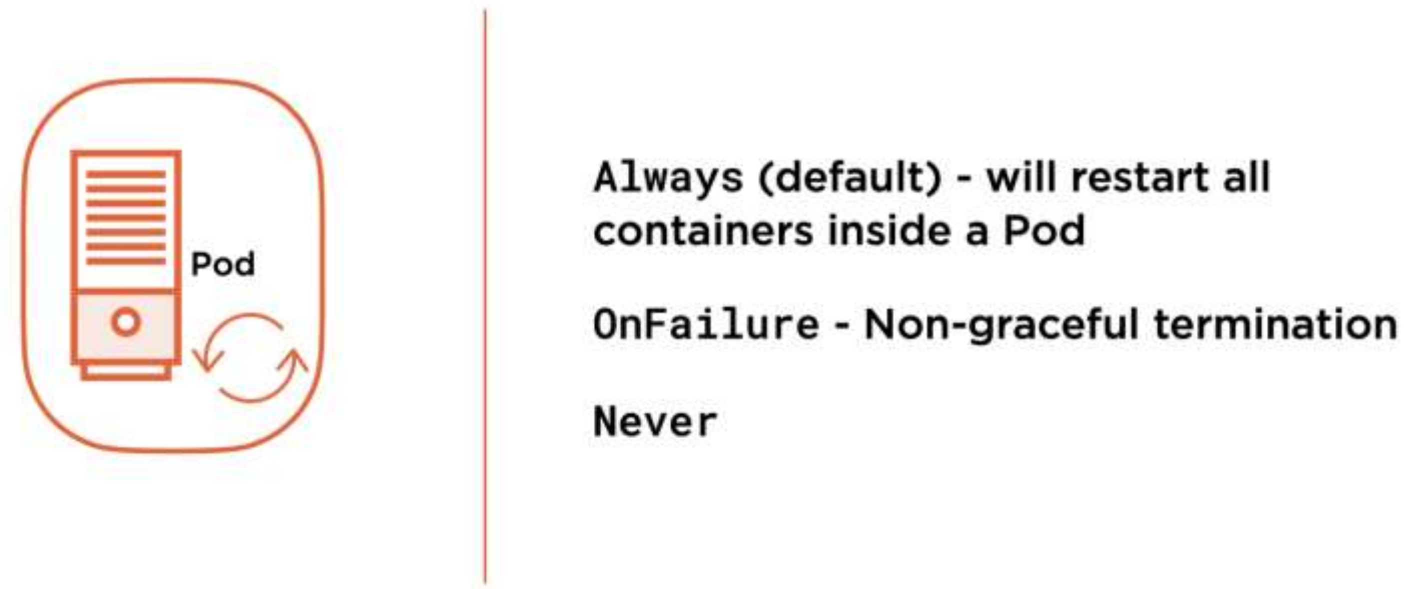
Applies to containers inside a Pod and defined inside the Pod’s Spec

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

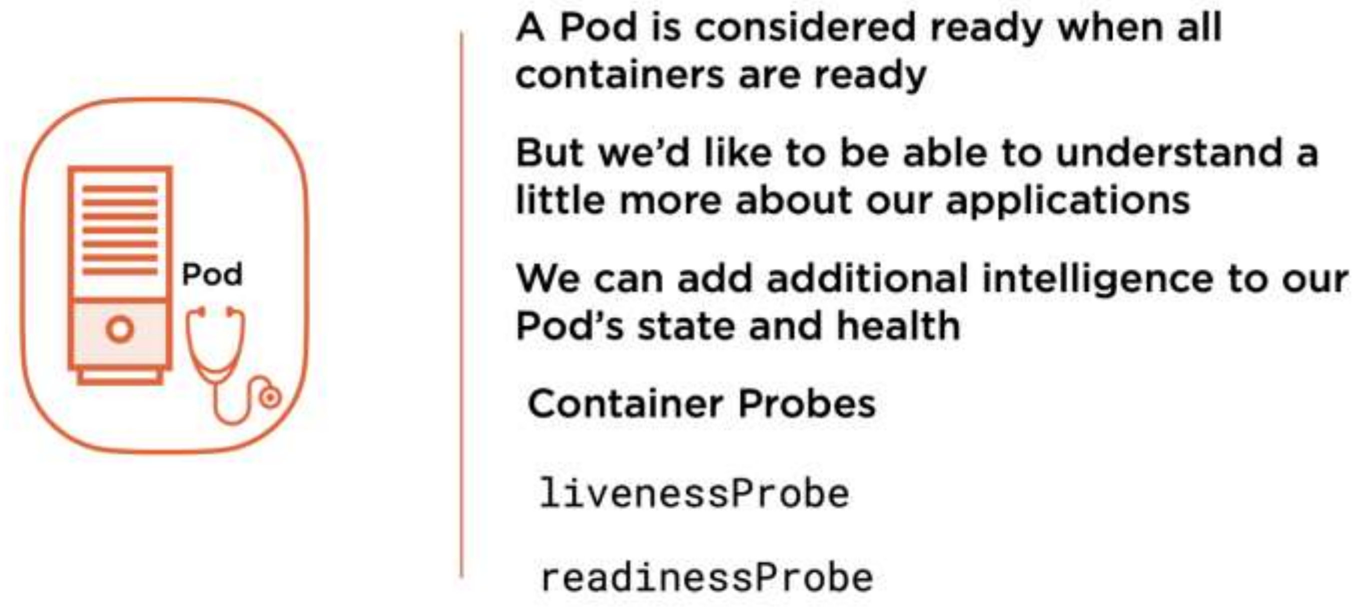


Always (default) - will restart all containers inside a Pod

OnFailure - Non-graceful termination

Never

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

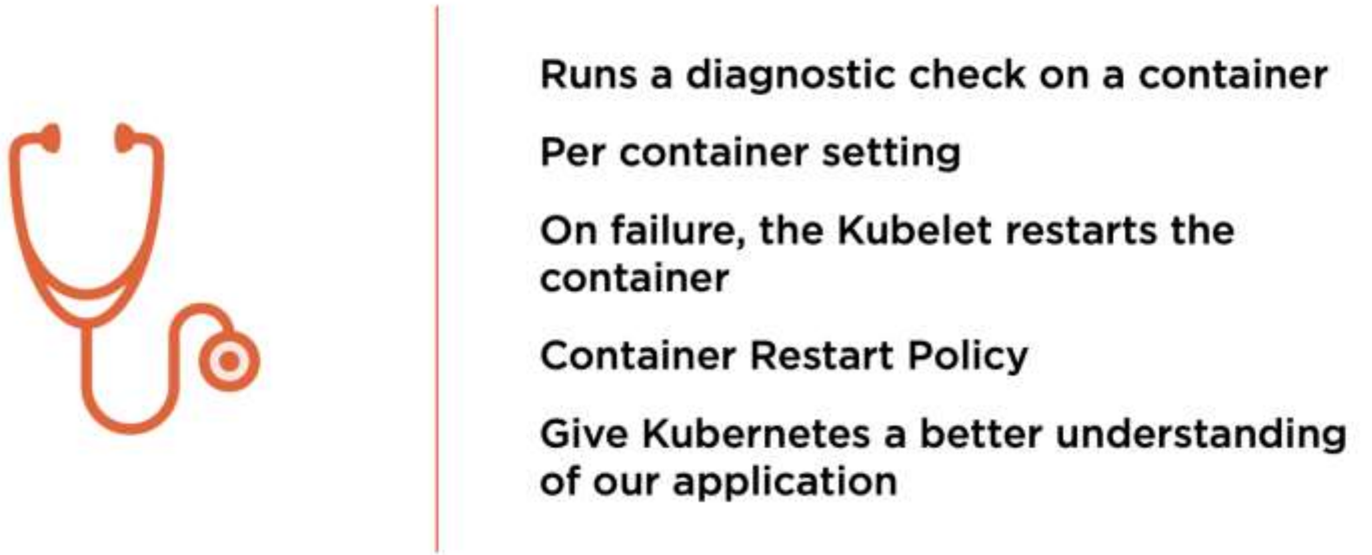
We can add additional intelligence to our Pod's state and health

Container Probes

```
livenessProbe

readinessProbe
```

livenessProbes



Runs a diagnostic check on a container

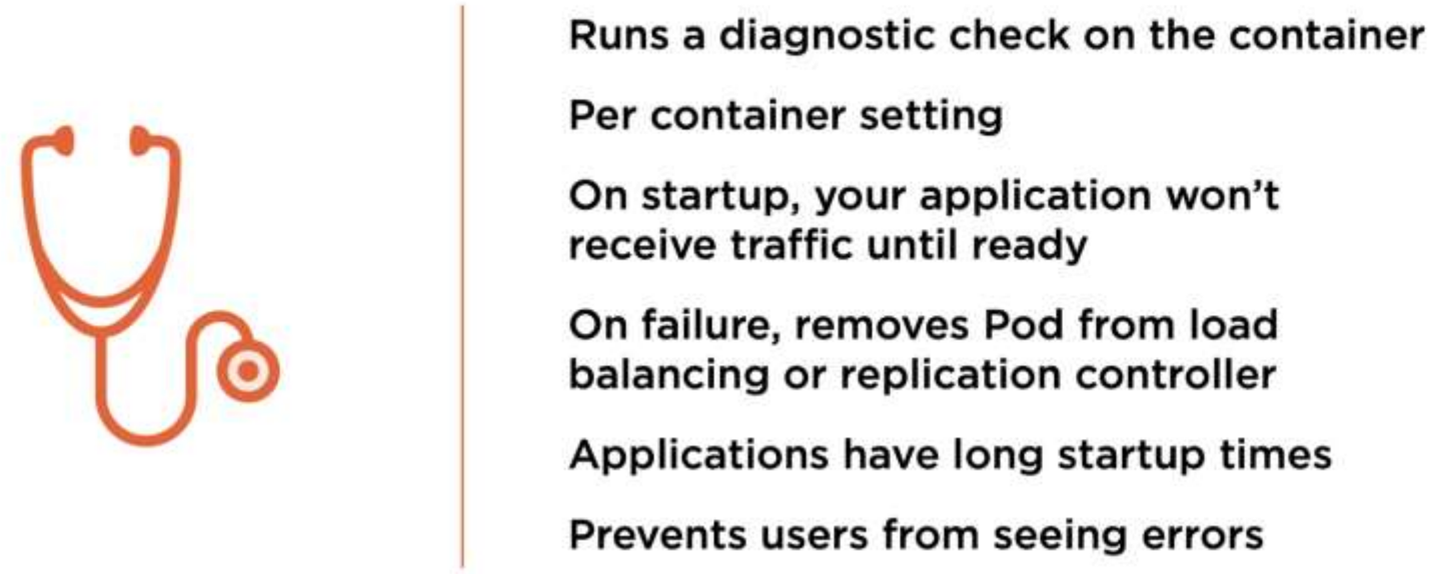
Per container setting

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

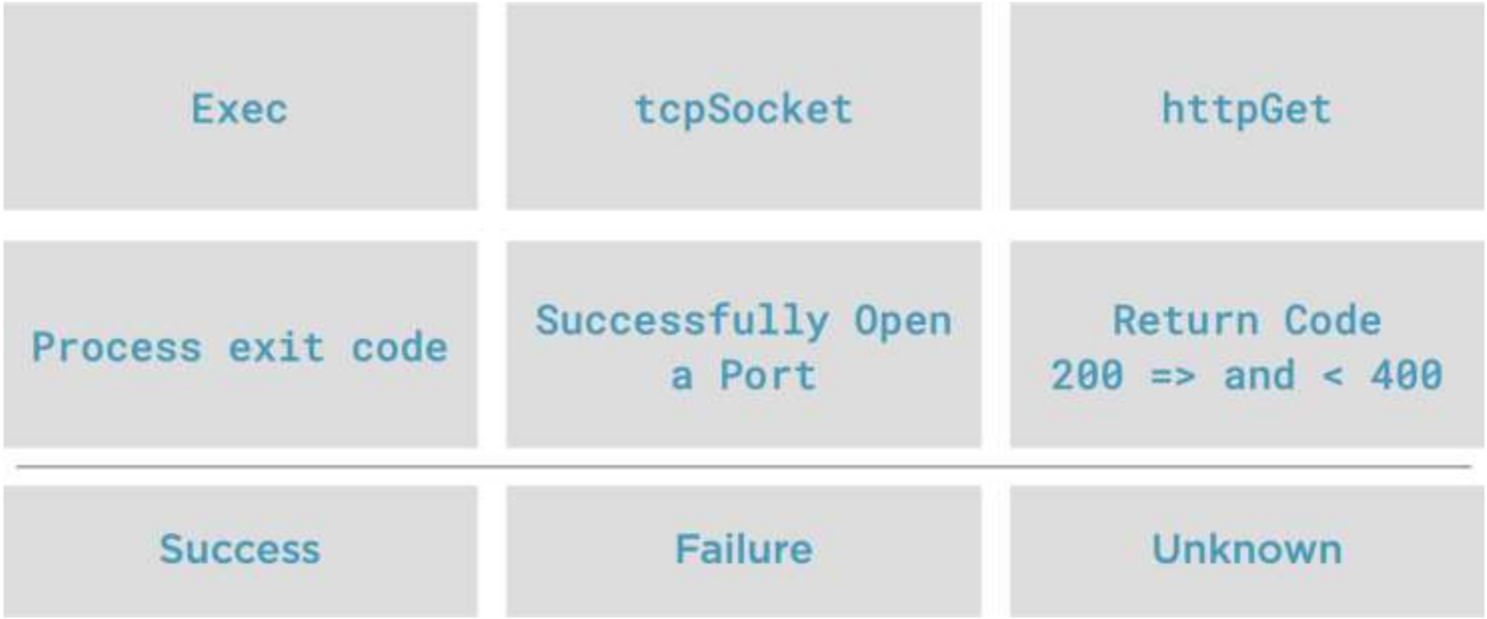
On startup, your application won’t receive traffic until ready

On failure, removes Pod from load balancing or replication controller

Applications have long startup times

Prevents users from seeing errors

Type of checks



Exec	tcpSocket	httpGet
Process exit code	Successfully Open a Port	Return Code 200 => and < 400
Success	Failure	Unknown

Configuring 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

Q
A