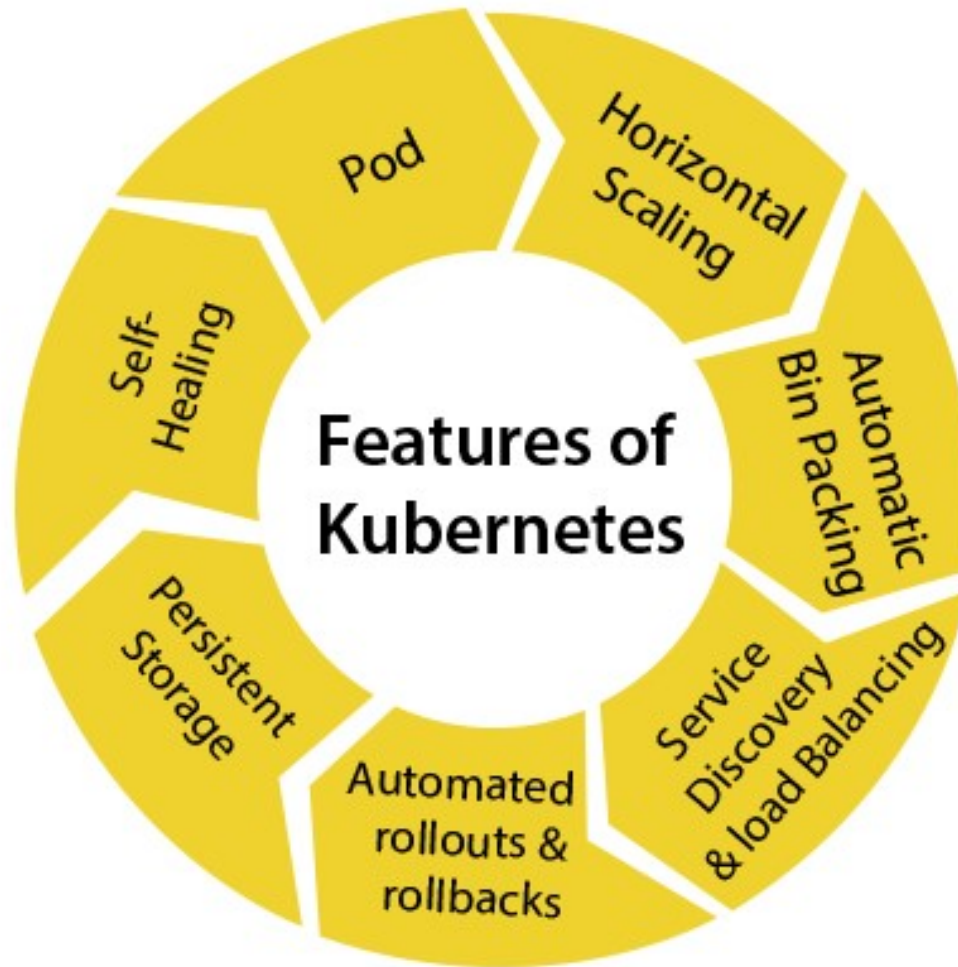
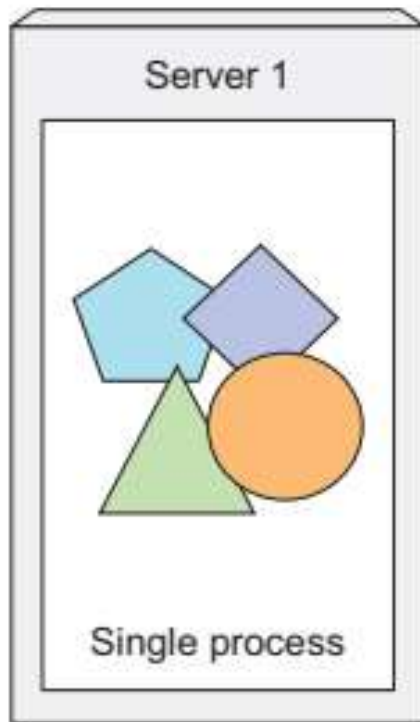


Introduction to Kubernetes

Features



Monolithic application



Microservices-based application

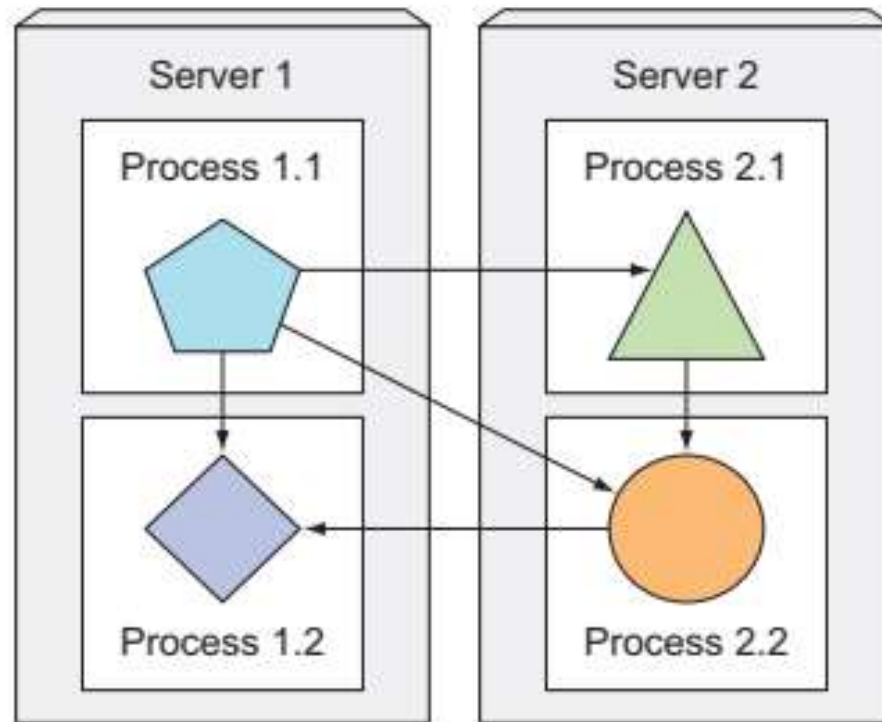
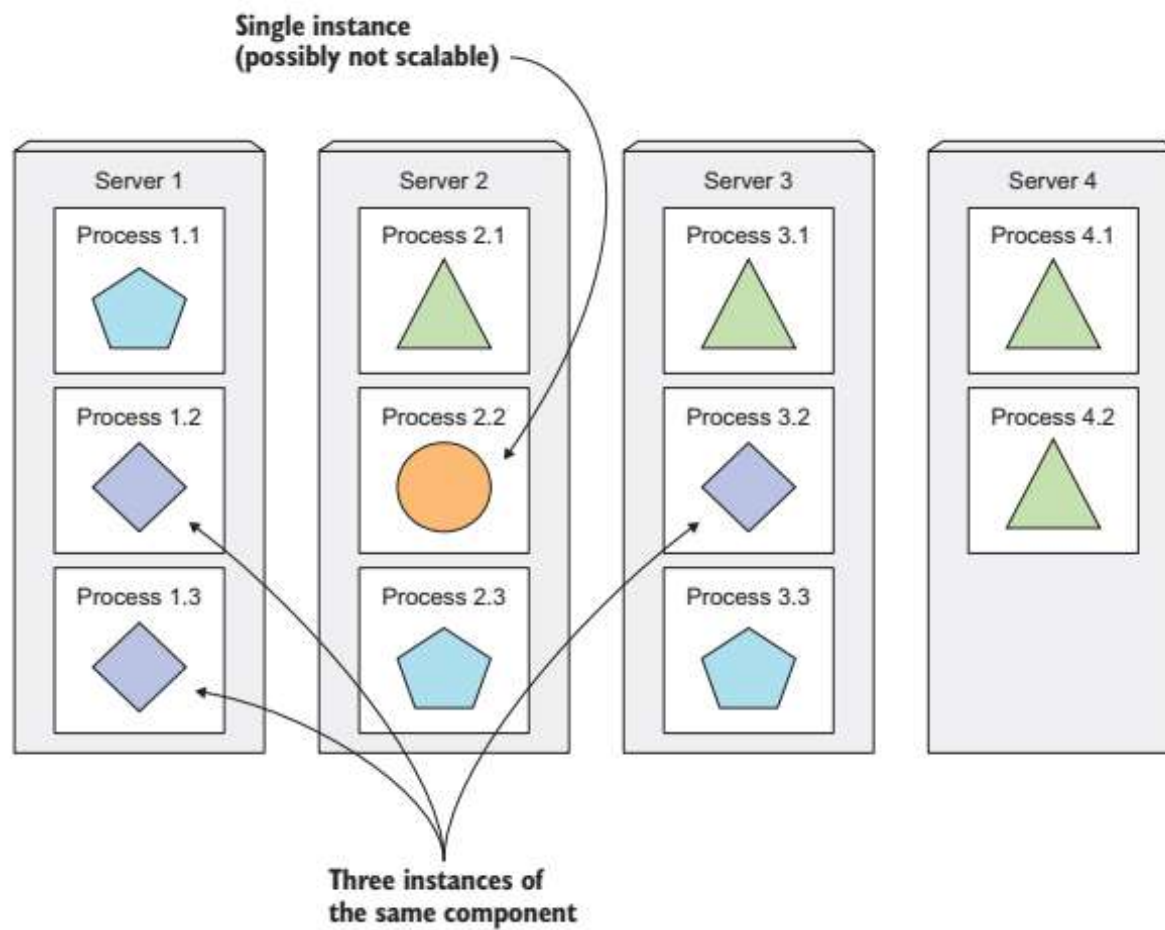
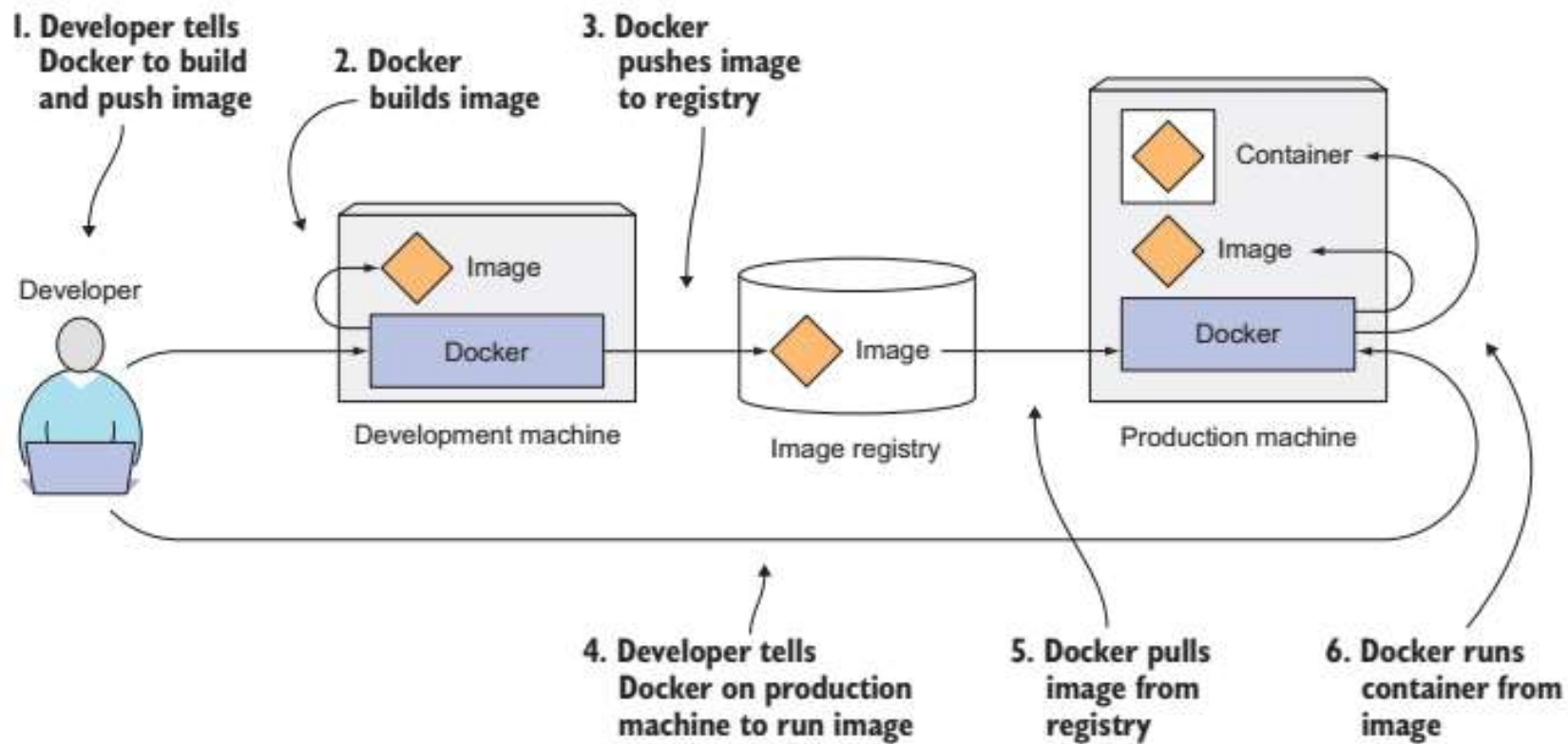
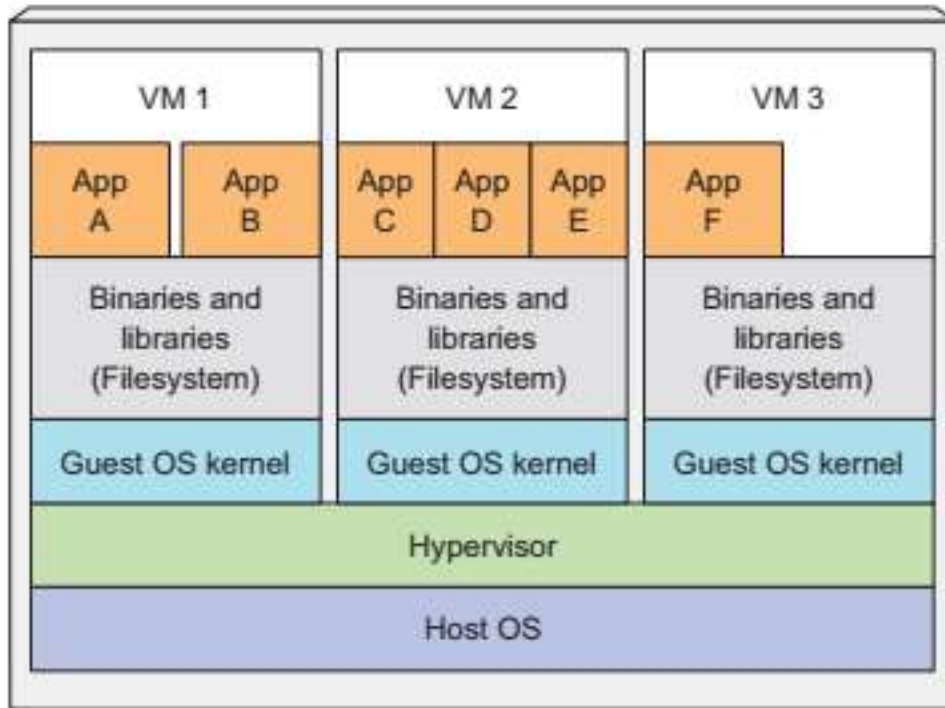


Figure 1.1 Components inside a monolithic application vs. standalone microservices

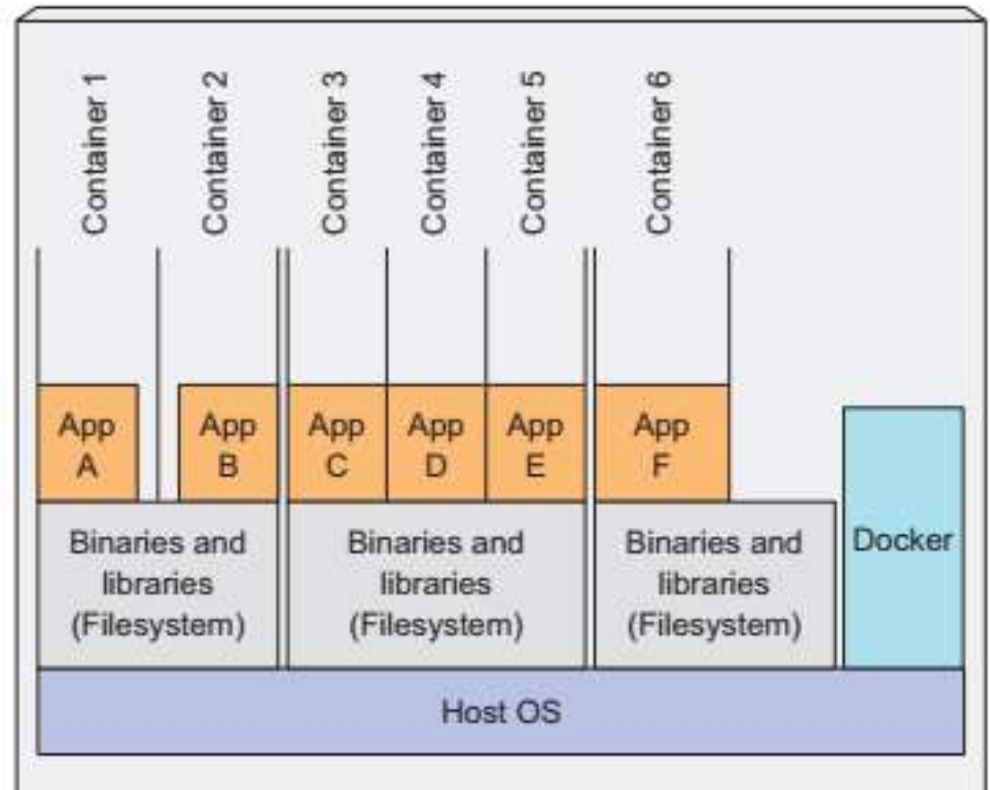




Host running multiple VMs



Host running multiple Docker containers



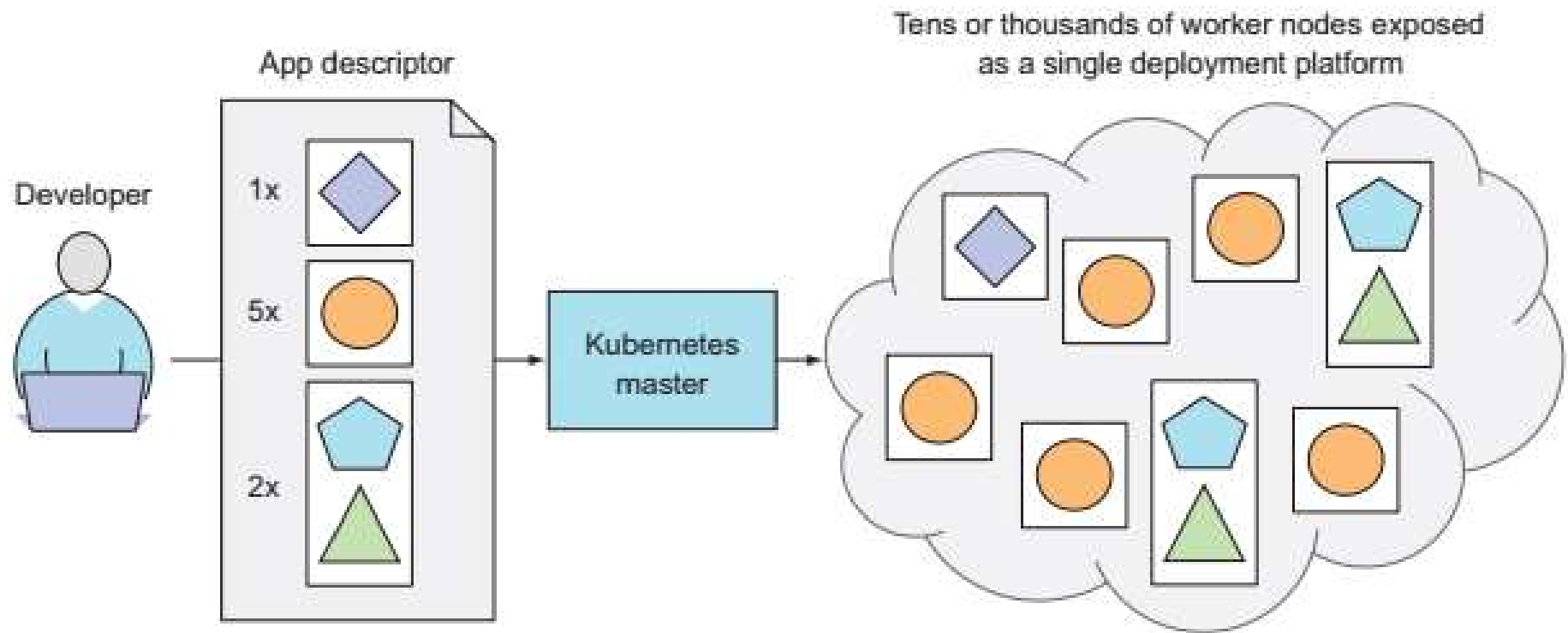


Figure 1.8 Kubernetes exposes the whole datacenter as a single deployment platform.

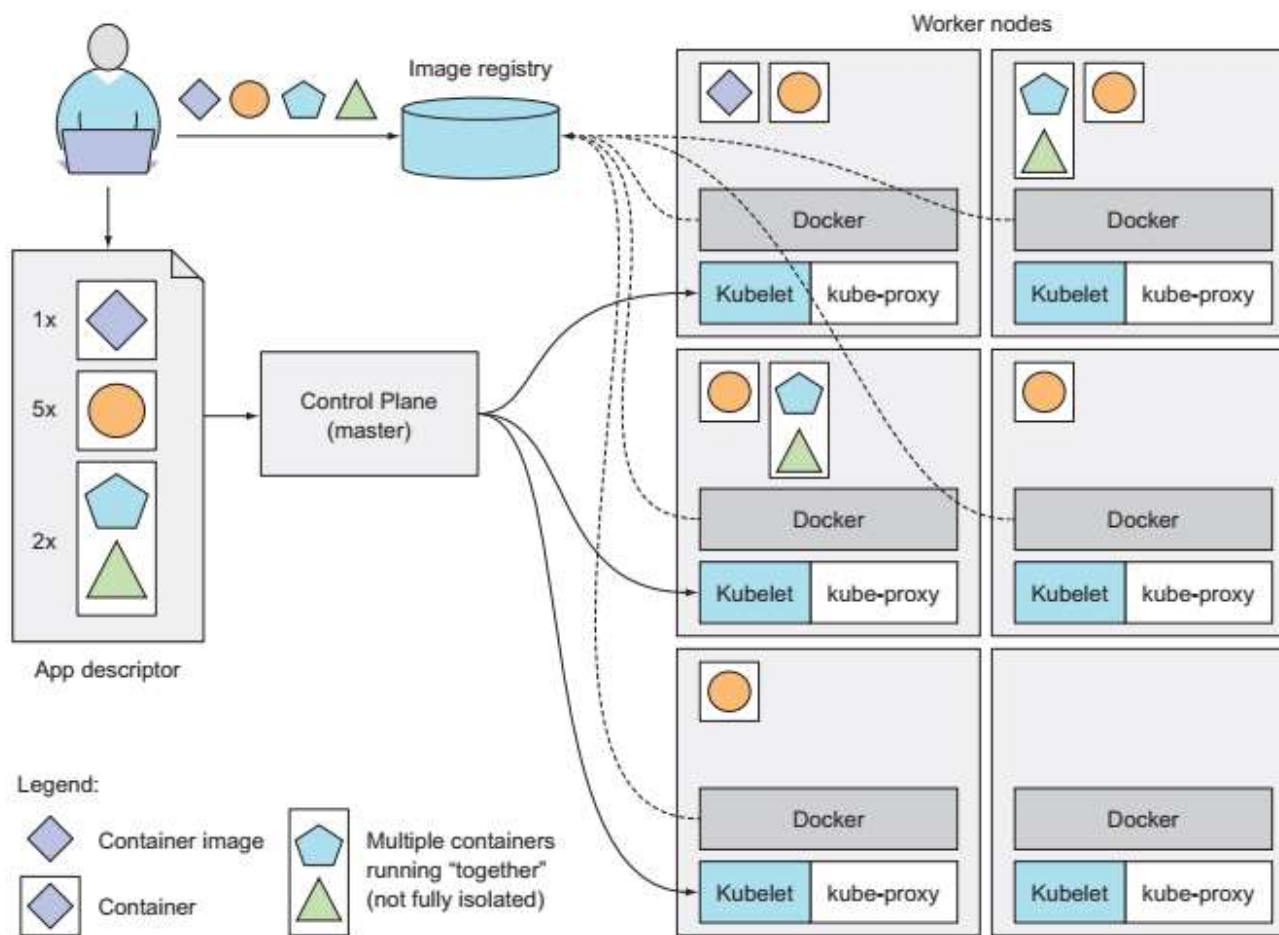


Figure 1.10 A basic overview of the Kubernetes architecture and an application running on top of it

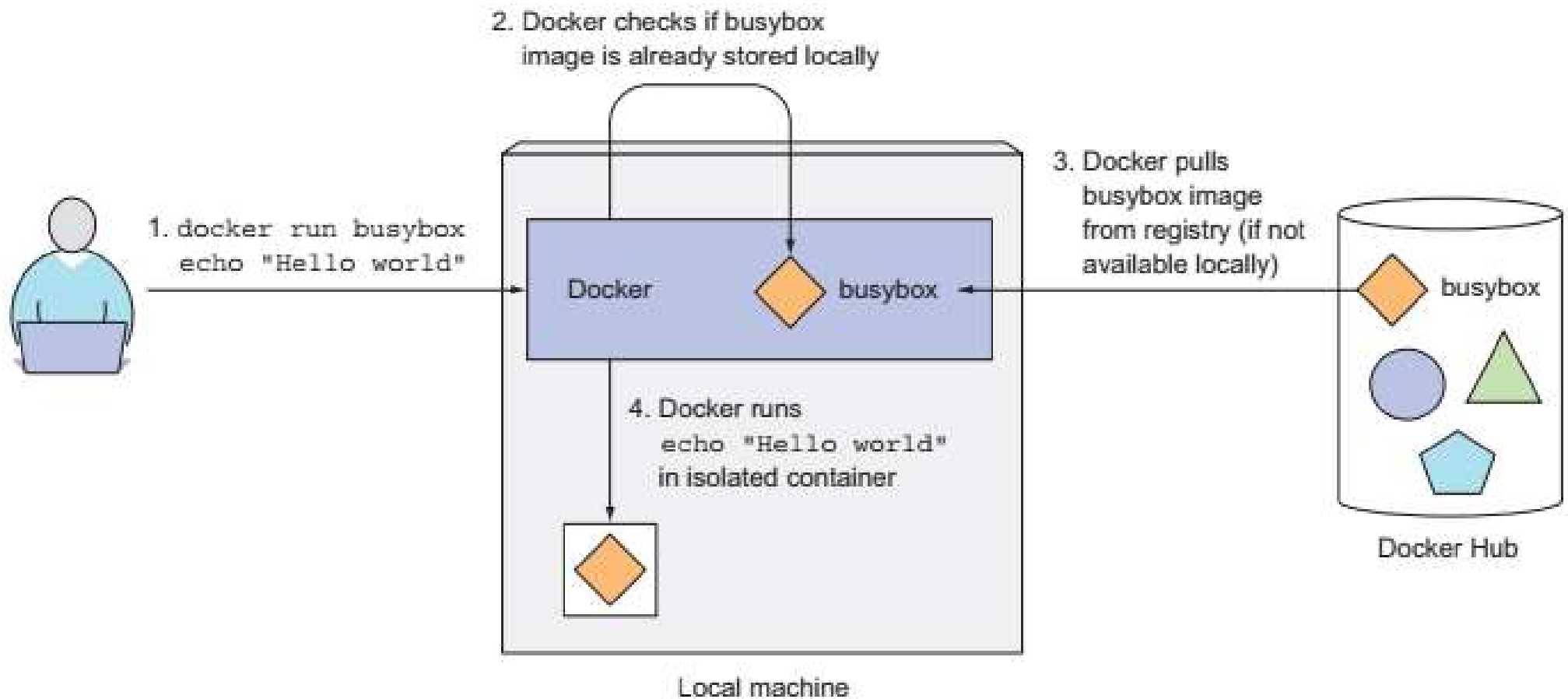


Figure 2.1 Running echo "Hello world" in a container based on the busybox container image

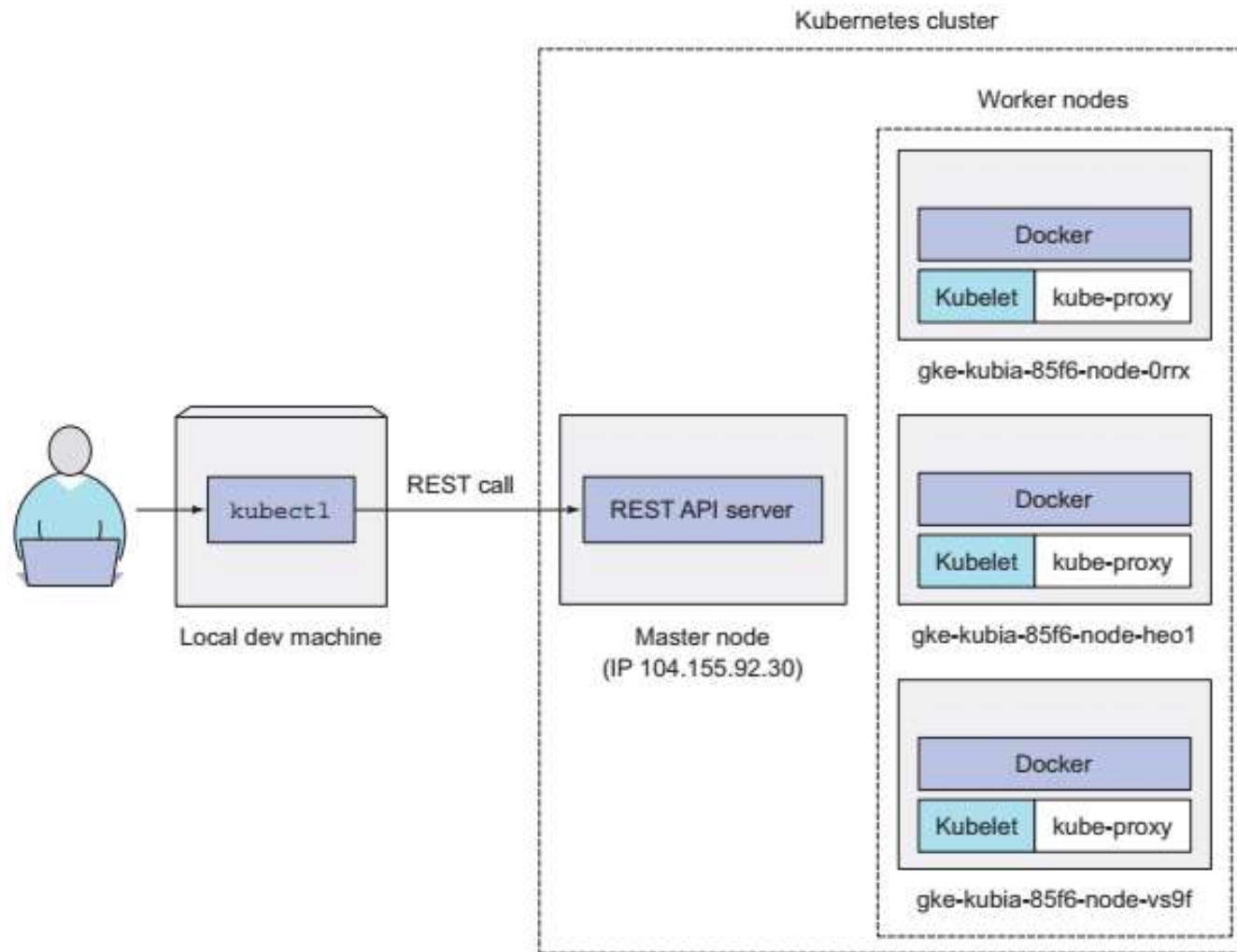


Figure 2.4 How you're interacting with your three-node Kubernetes cluster

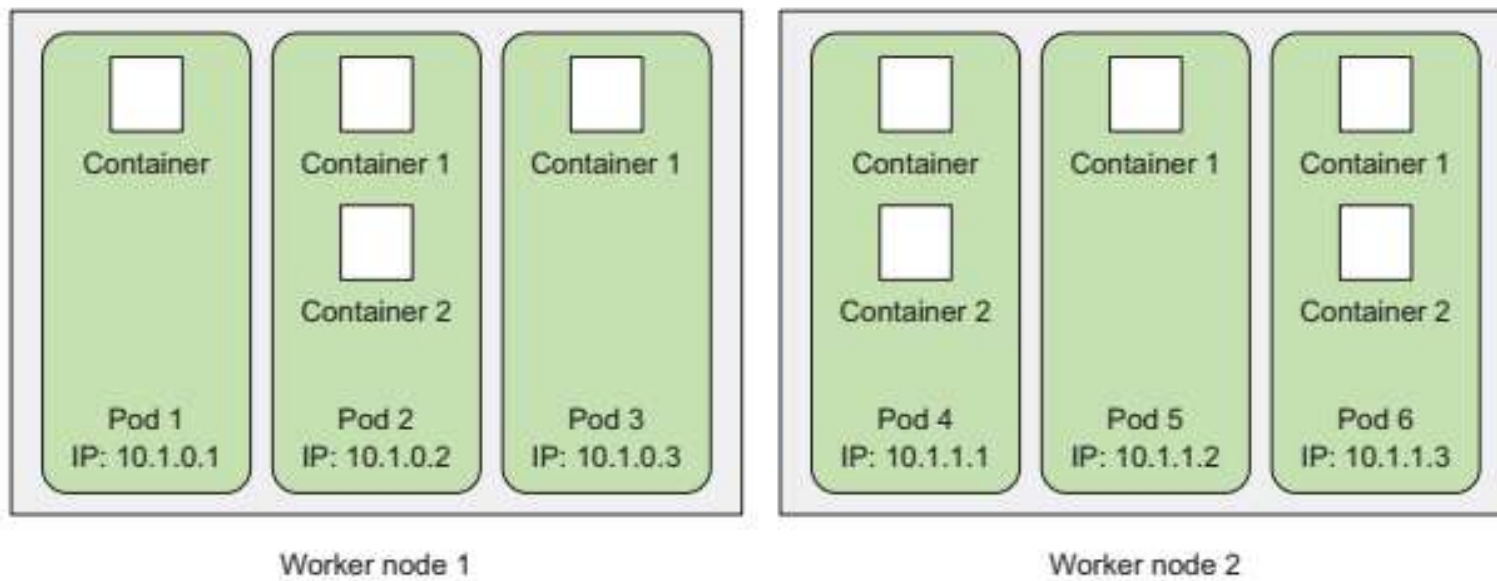


Figure 2.5 The relationship between containers, pods, and physical worker nodes

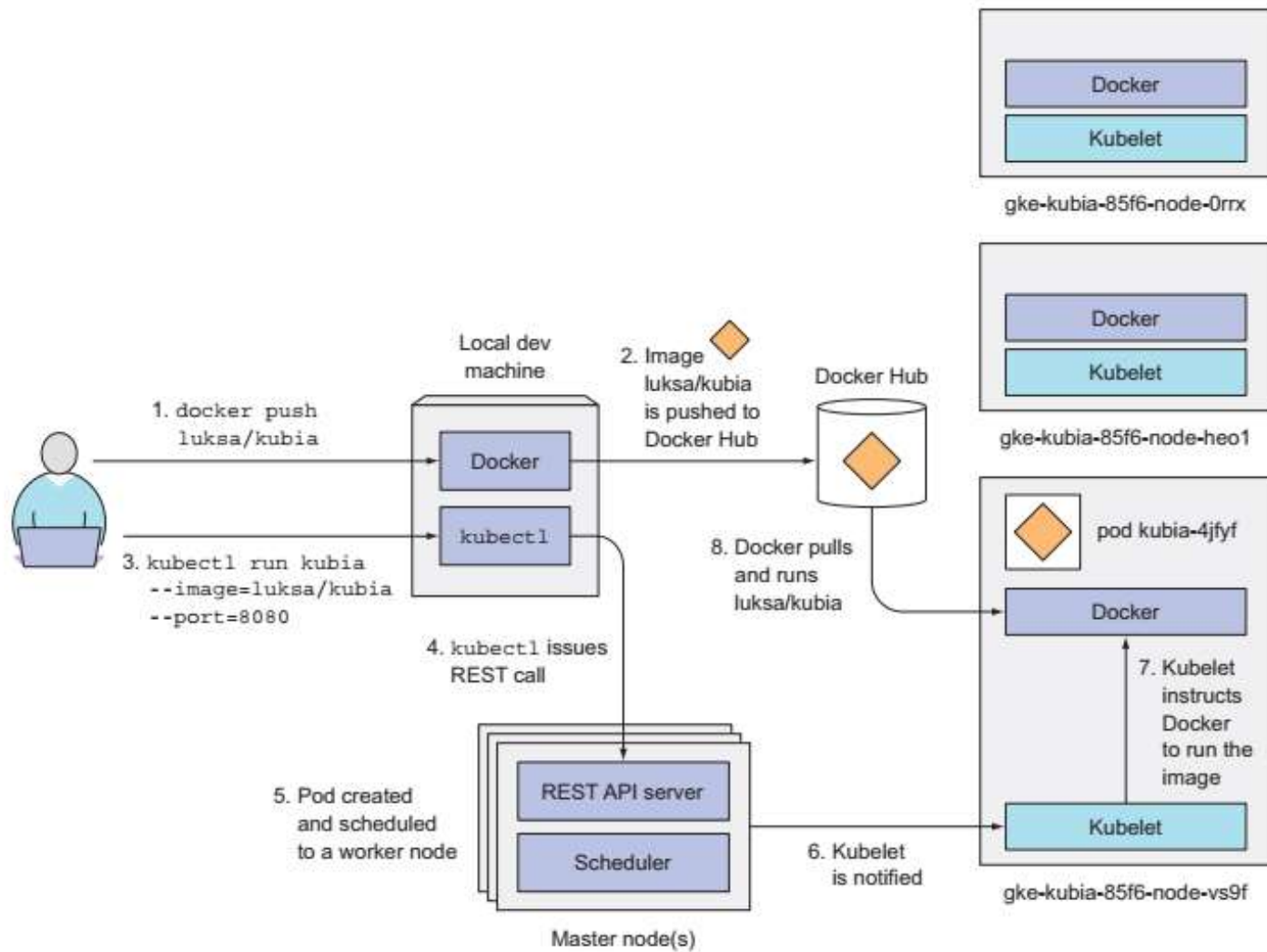


Figure 2.6 Running the luksa/kubia container image in Kubernetes

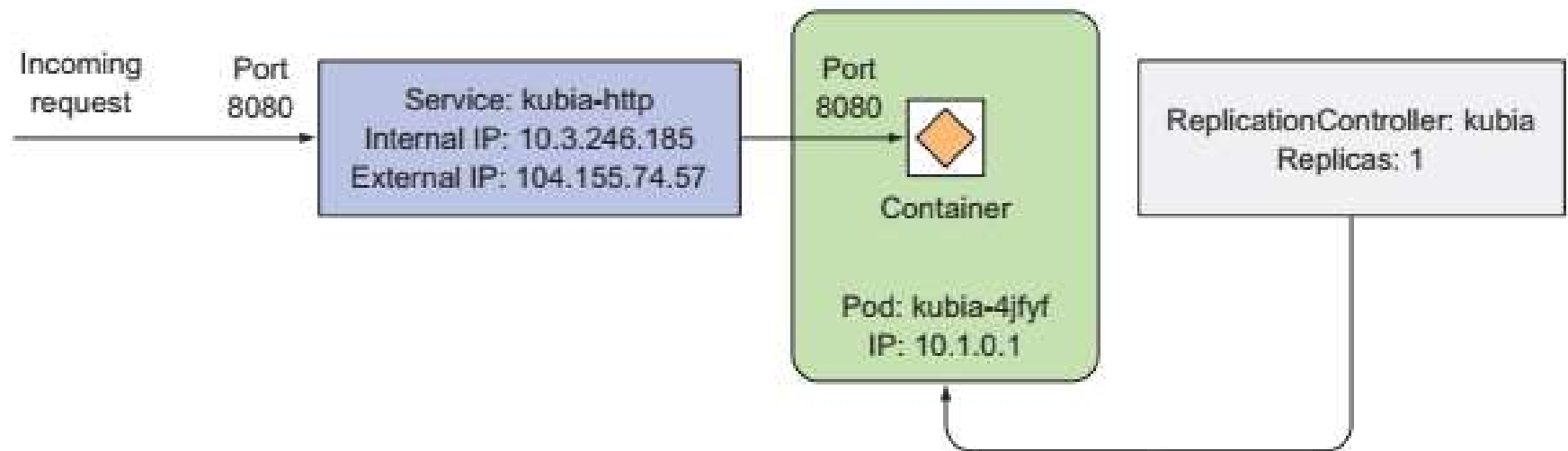


Figure 2.7 Your system consists of a ReplicationController, a Pod, and a Service.

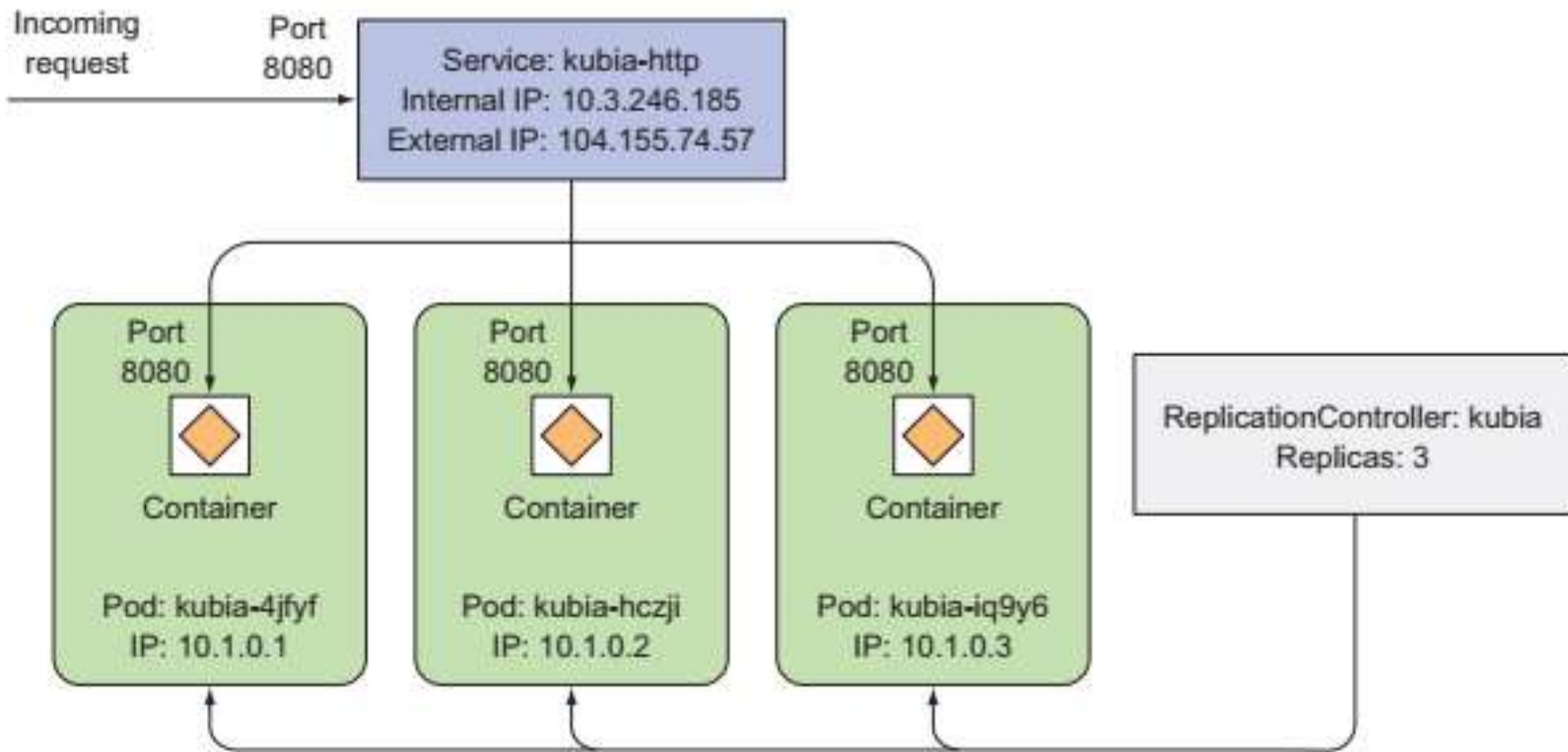


Figure 2.8 Three instances of a pod managed by the same ReplicationController and exposed through a single service IP and port.

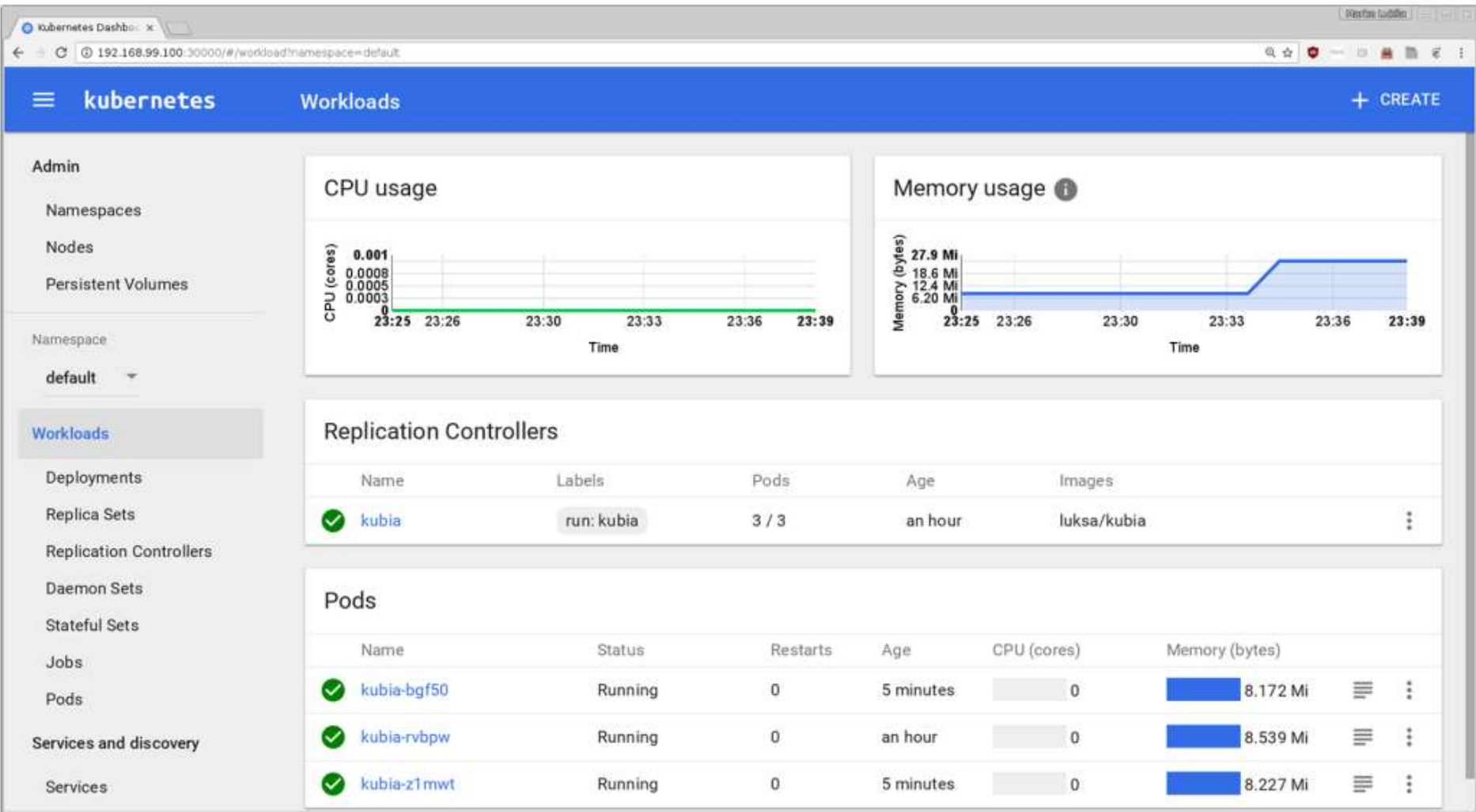


Figure 2.9 Screenshot of the Kubernetes web-based dashboard

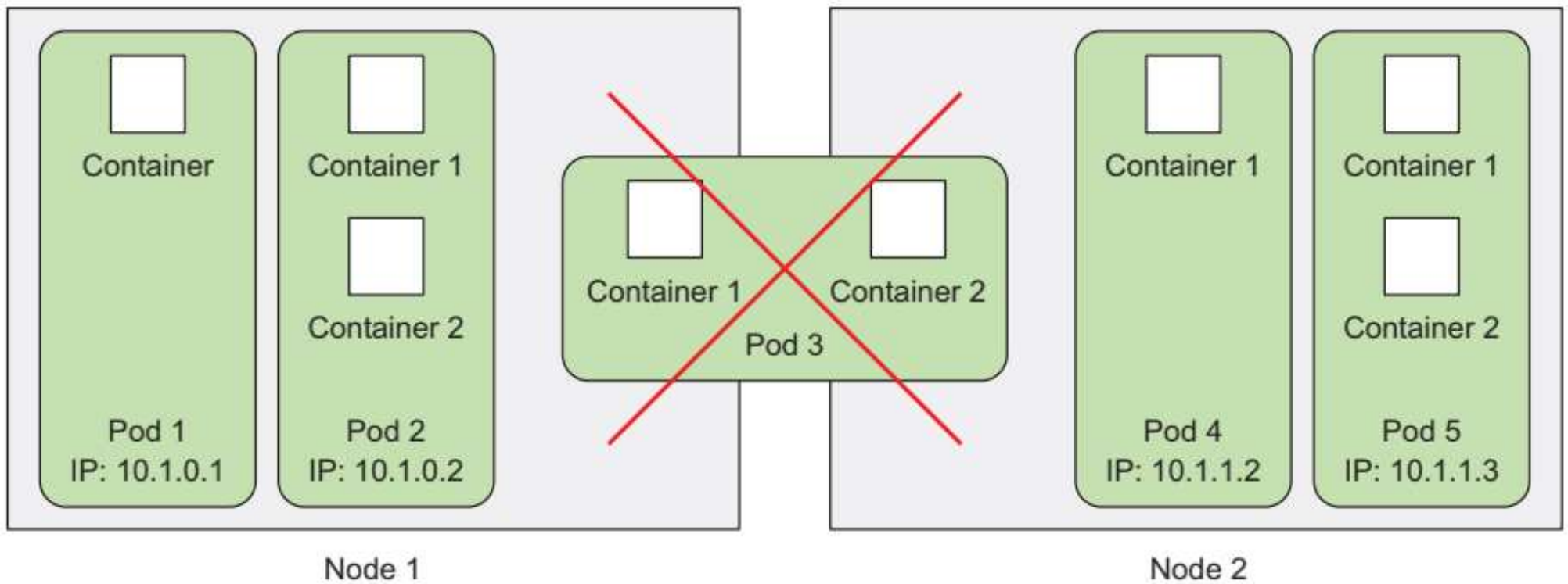


Figure 3.1 All containers of a pod run on the same node. A pod never spans two nodes.

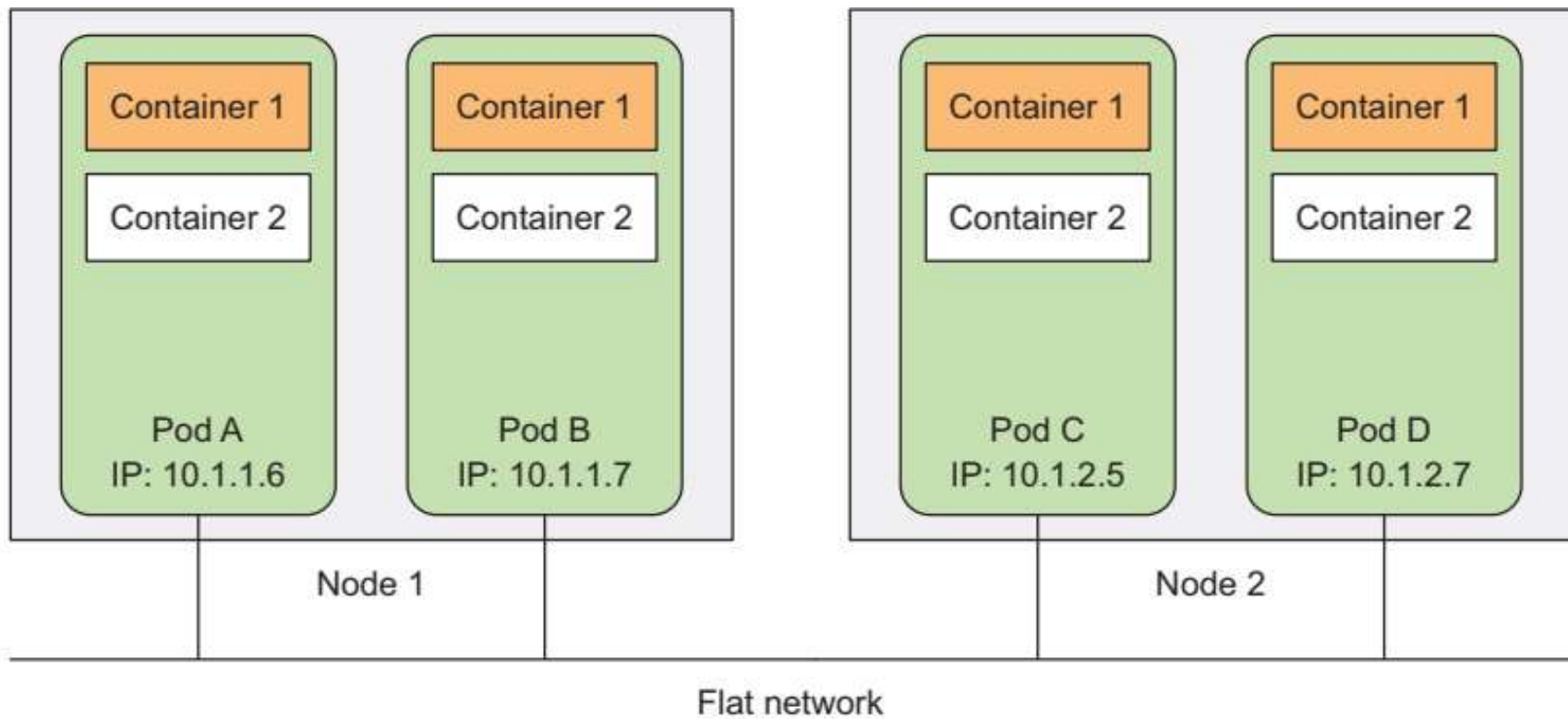


Figure 3.2 Each pod gets a routable IP address and all other pods see the pod under that IP address.

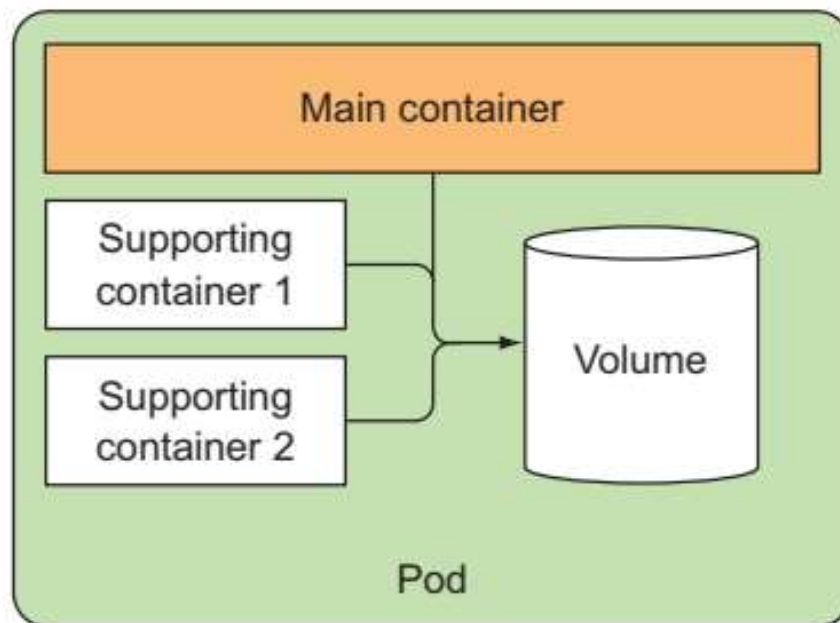
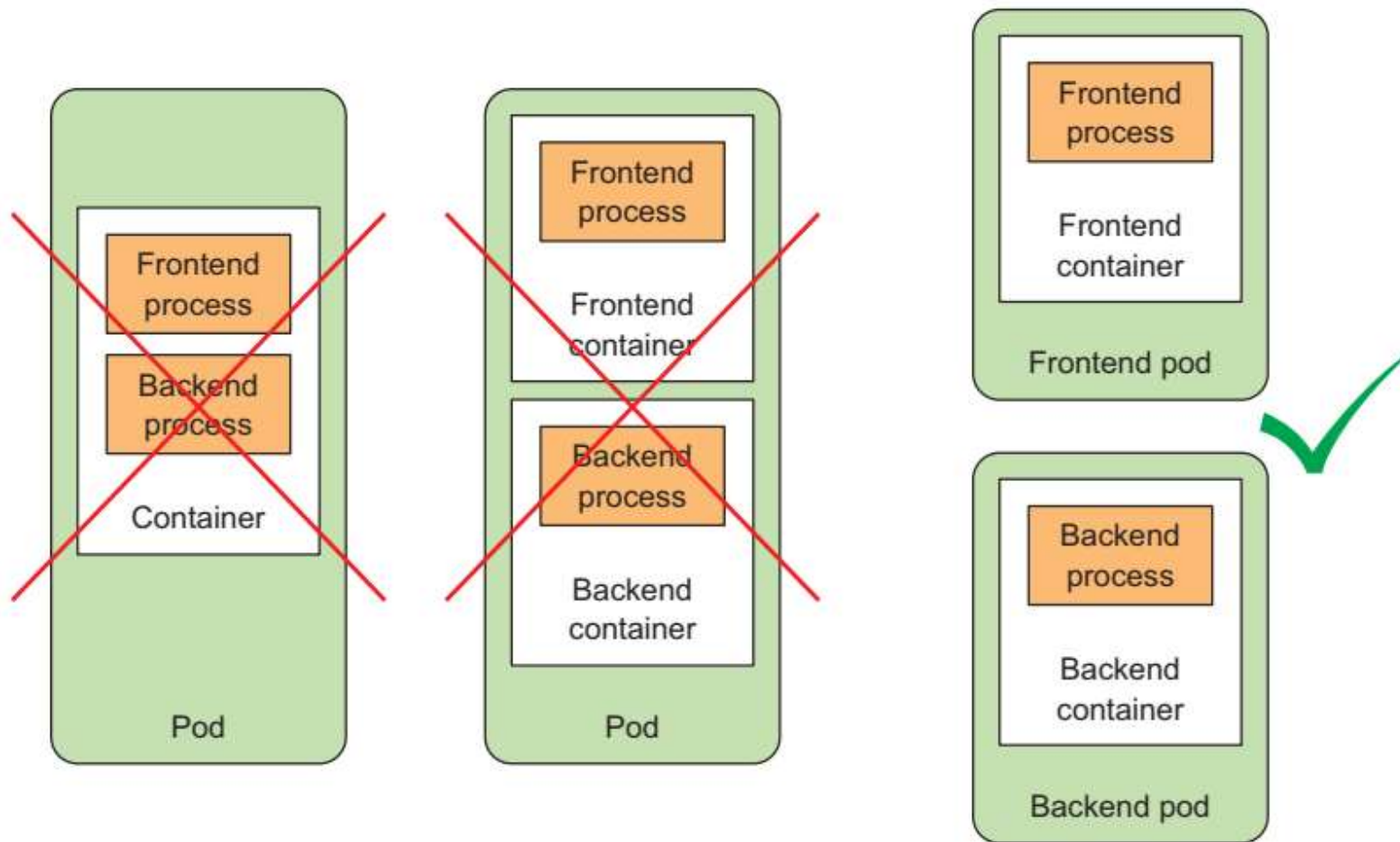
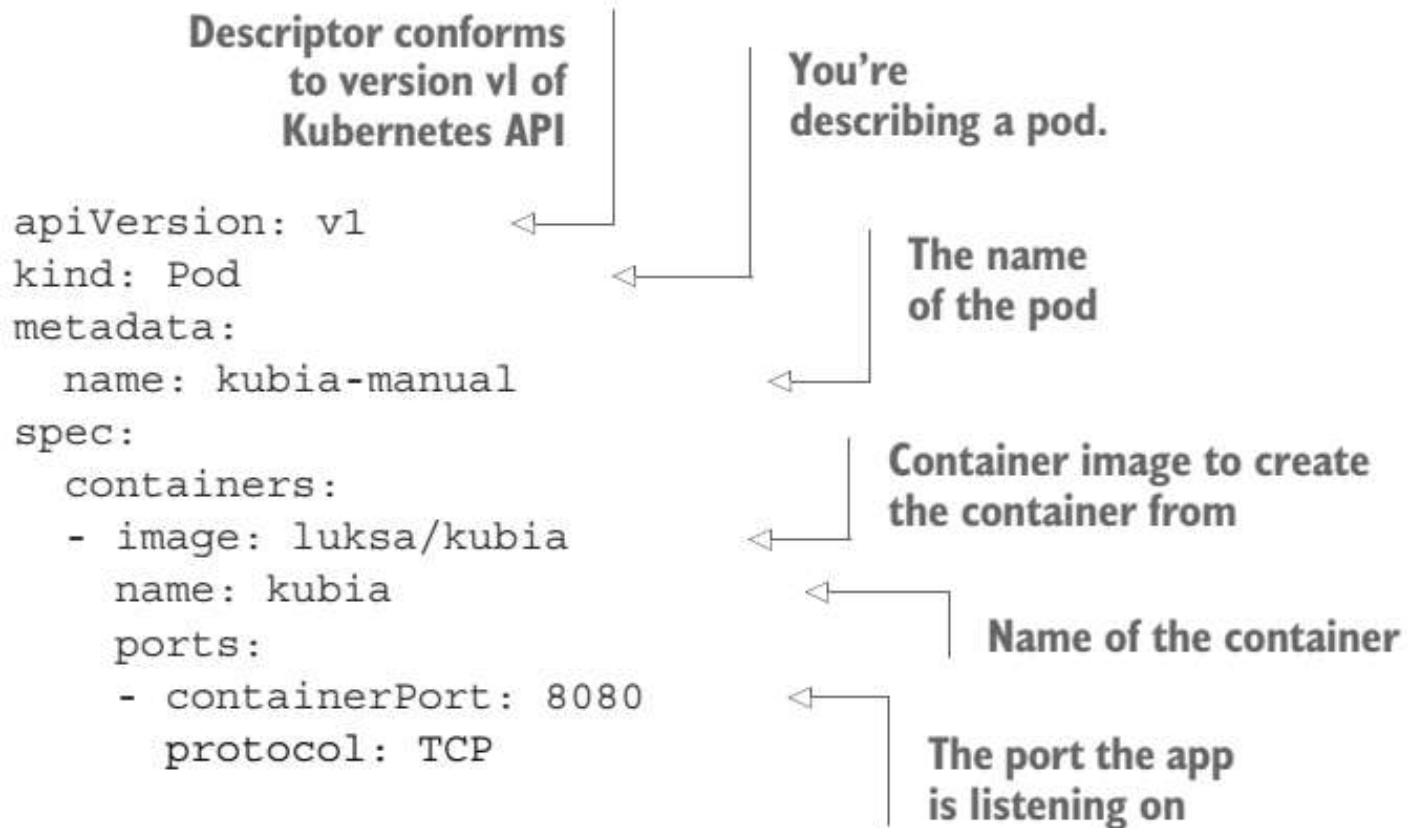


Figure 3.3 Pods should contain tightly coupled containers, usually a main container and containers that support the main one.



Listing 3.2 A basic pod manifest: kuba-manual.yaml



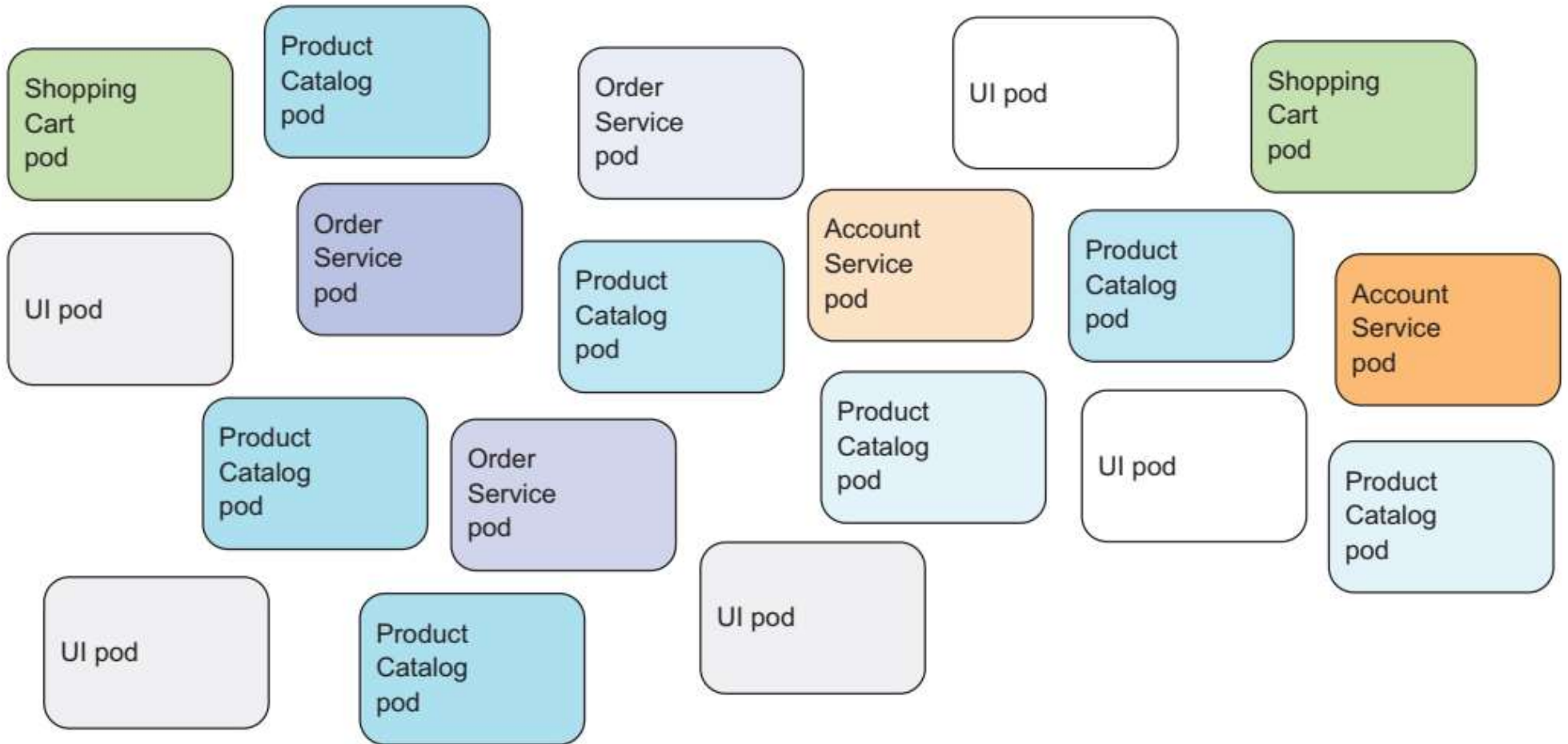


Figure 3.6 Uncategorized pods in a microservices architecture

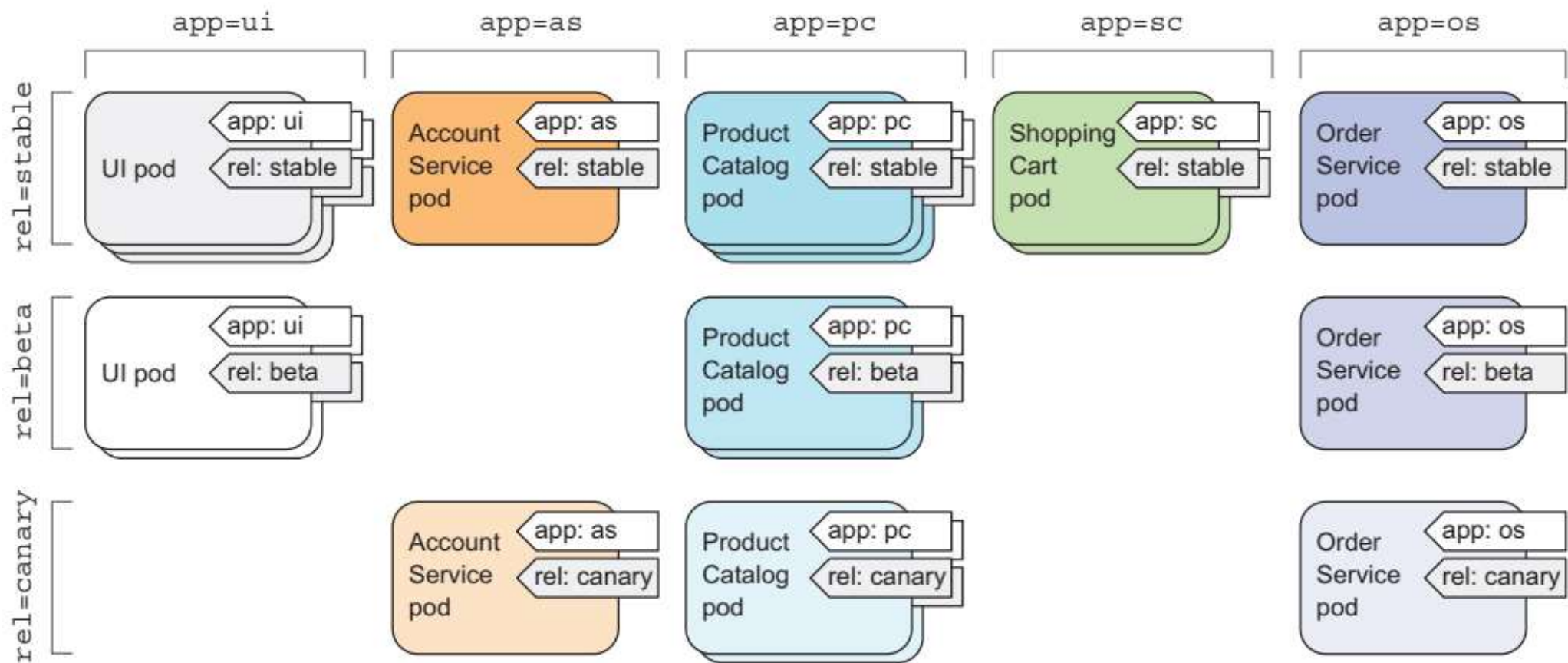


Figure 3.7 Organizing pods in a microservices architecture with pod labels

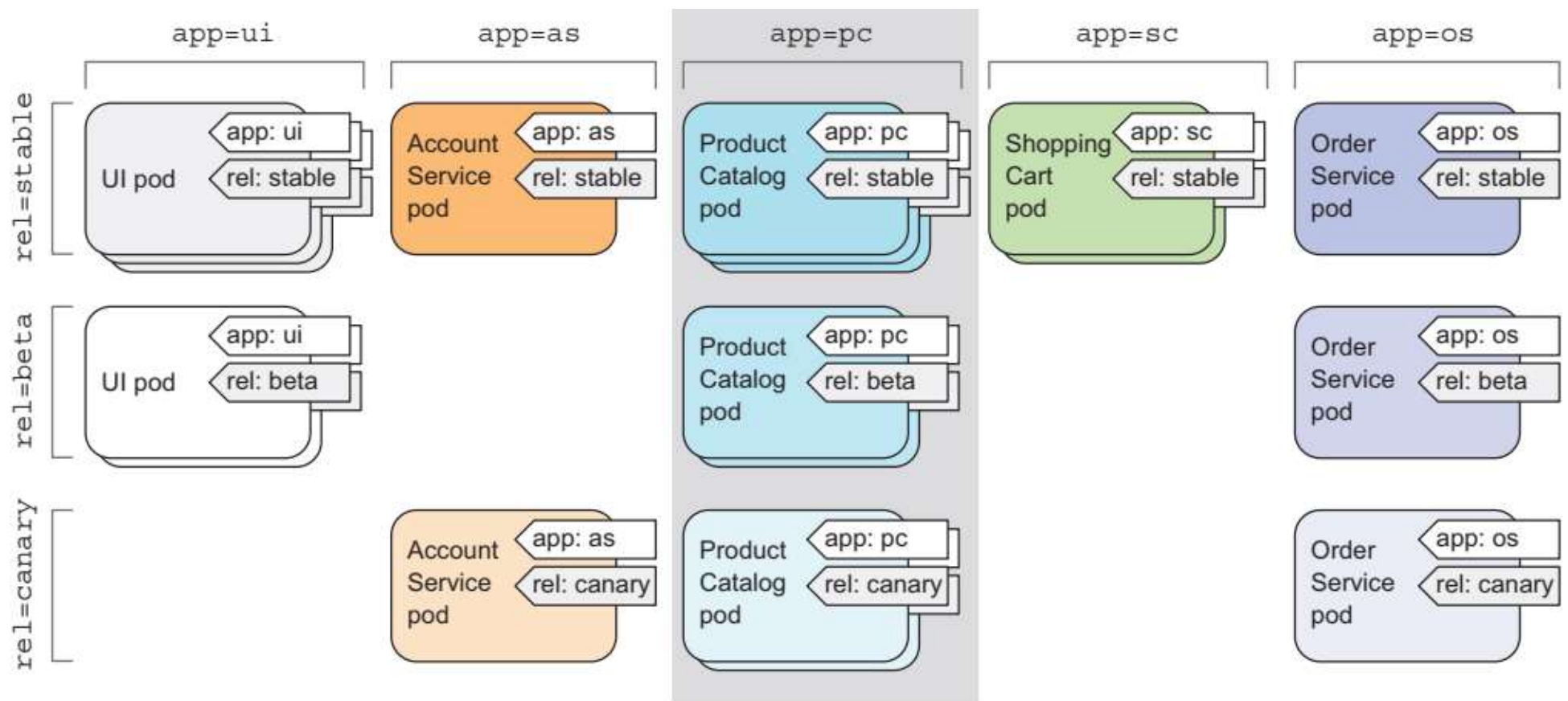


Figure 3.8 Selecting the product catalog microservice pods using the “app=pc” label selector

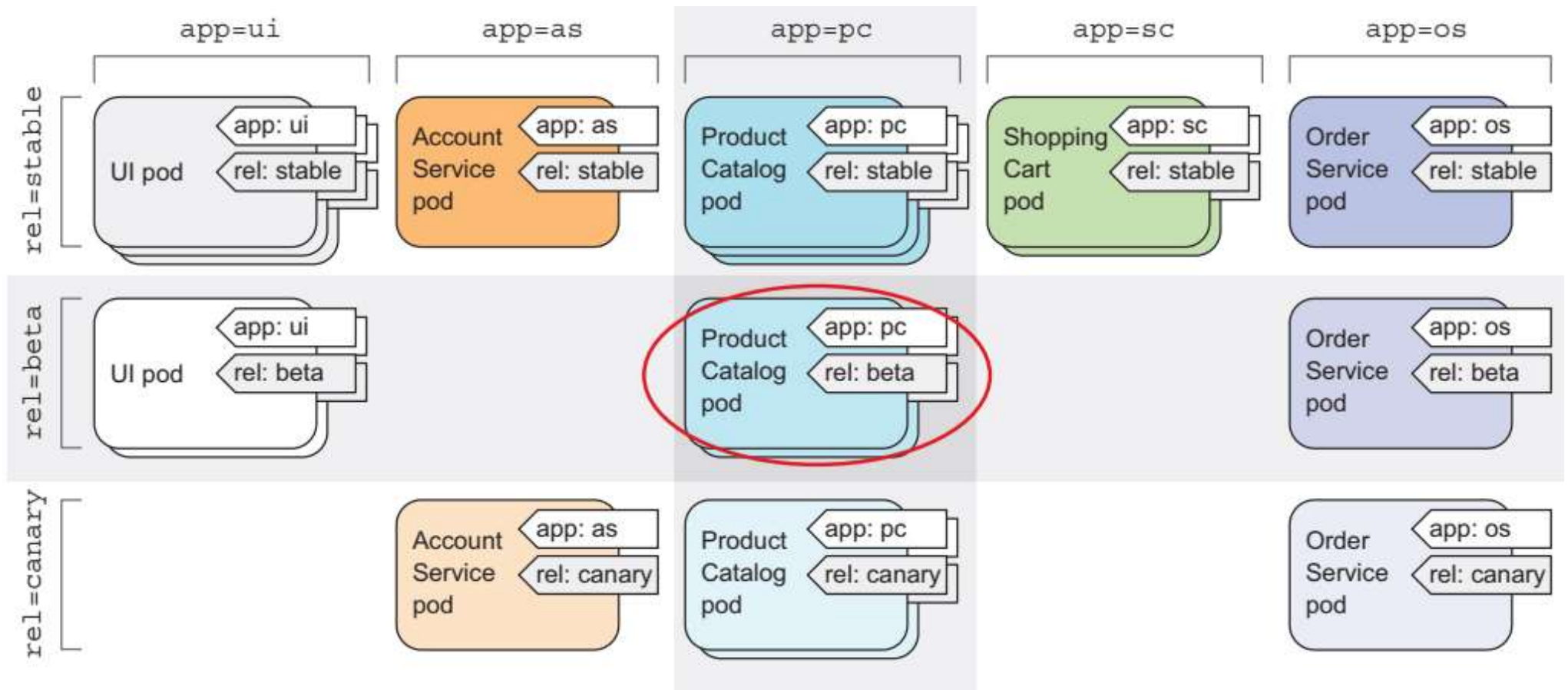


Figure 3.9 Selecting pods with multiple label selectors

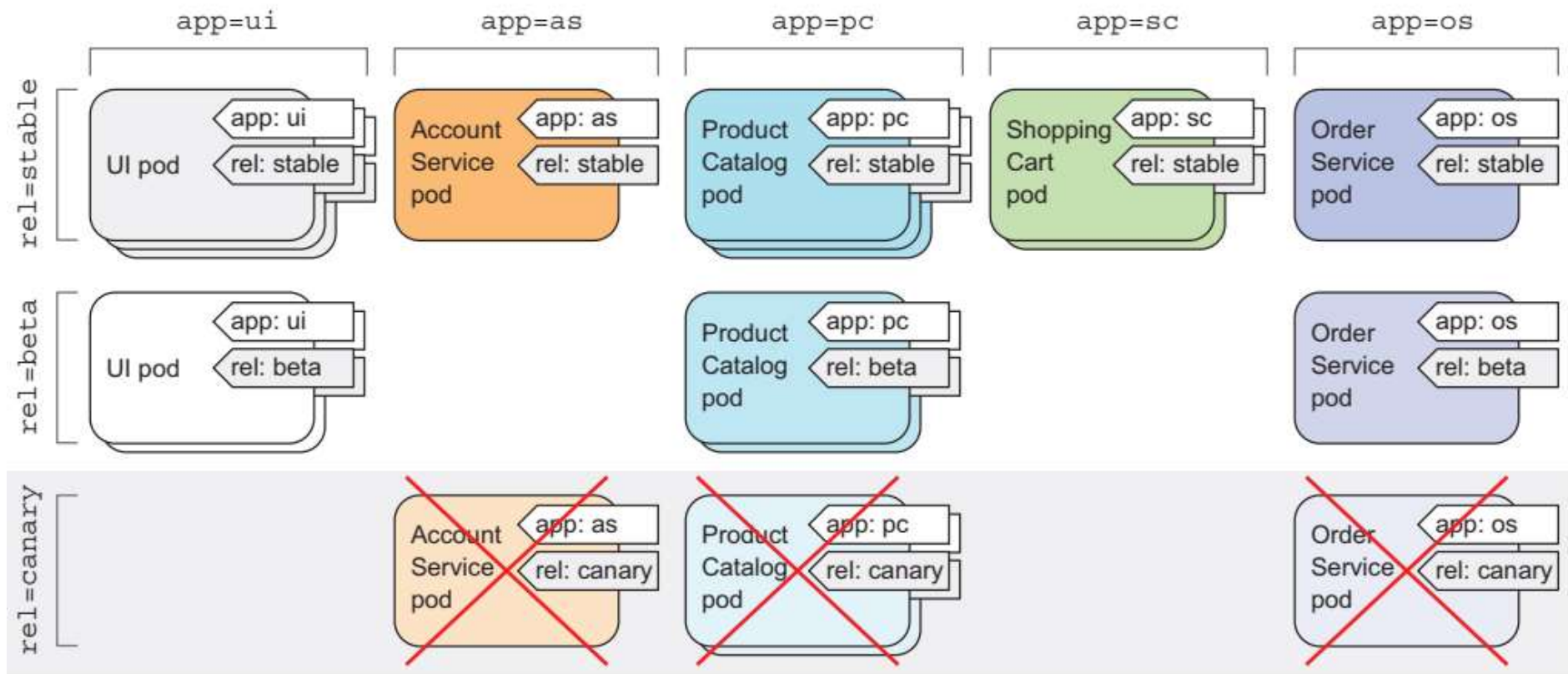


Figure 3.10 Selecting and deleting all canary pods through the `rel=canary` label selector

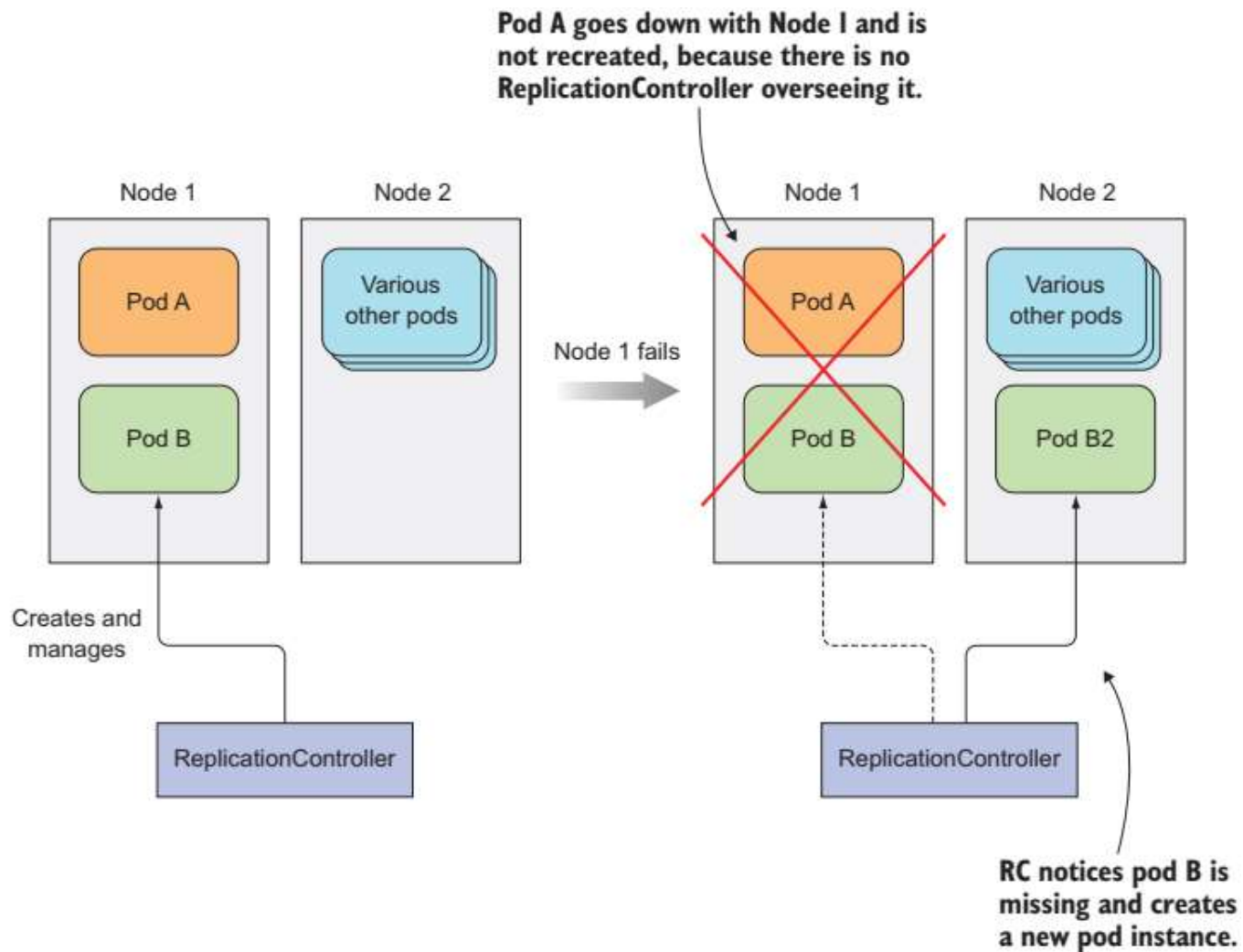


Figure 4.1 When a node fails, only pods backed by a ReplicationController are recreated.

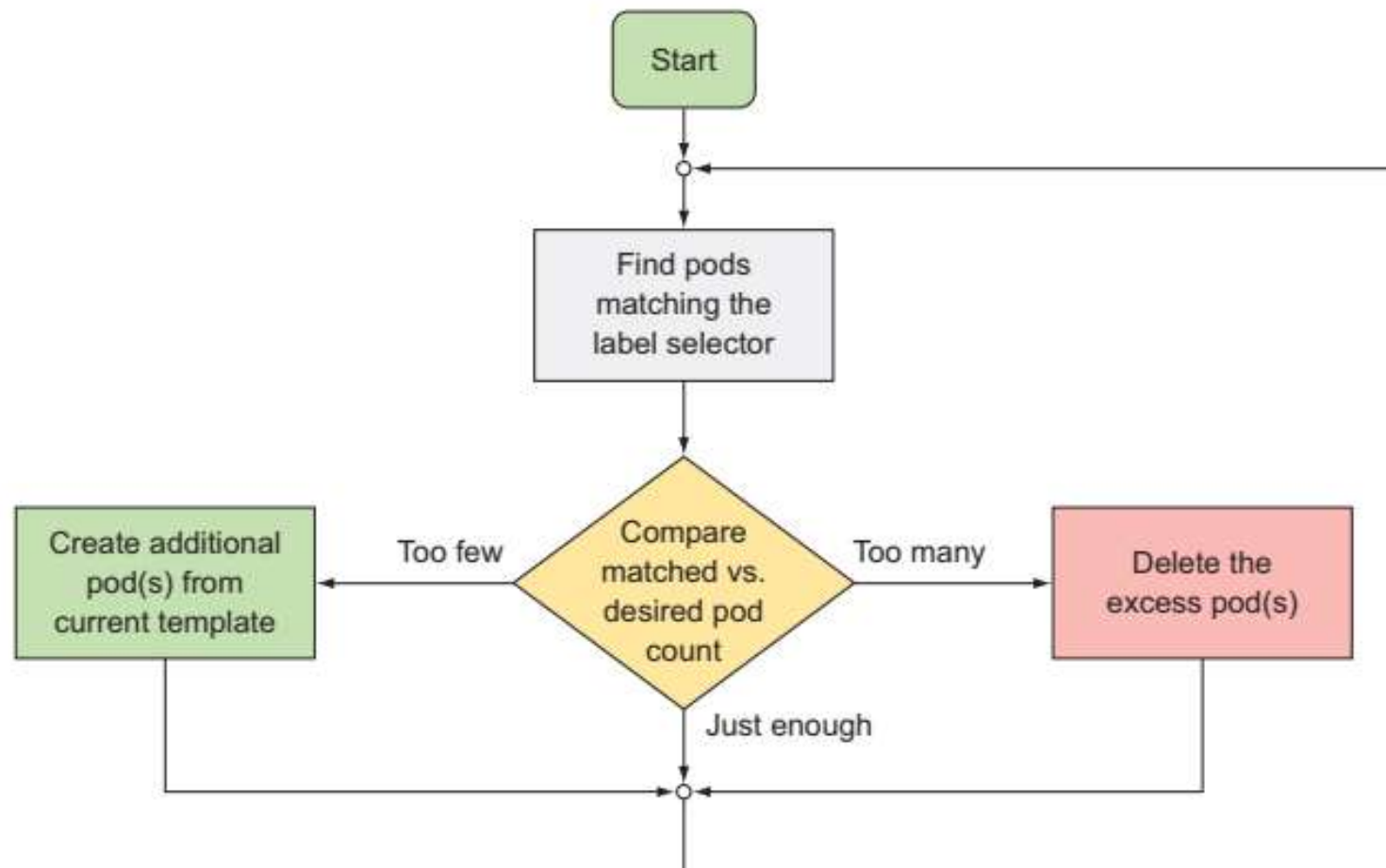


Figure 4.2 A ReplicationController's reconciliation loop

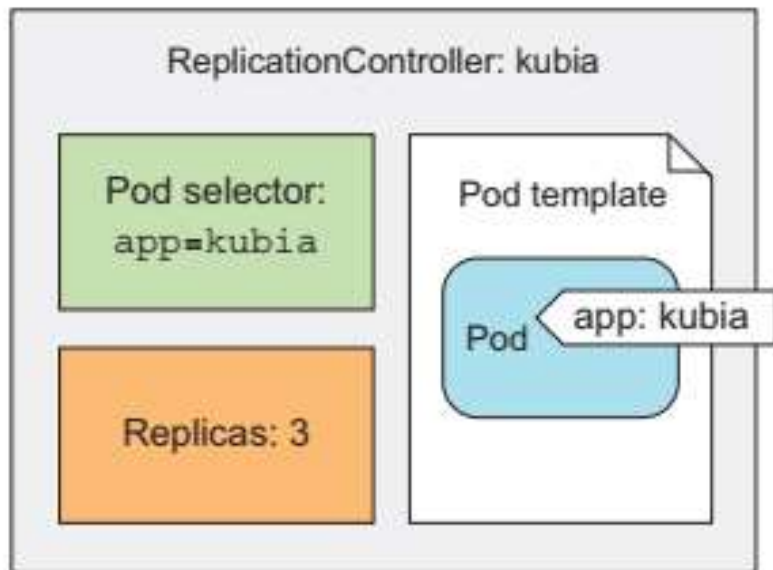


Figure 4.3 The three key parts of a `ReplicationController` (pod selector, replica count, and pod template)

Listing 4.4 A YAML definition of a ReplicationController: kubia-rc.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: kubia
spec:
  replicas: 3
  selector:
    app: kubia
```

This manifest defines a
ReplicationController (RC)

The name of this
ReplicationController

The desired number
of pod instances

The pod selector determining
what pods the RC is operating on

```
template:
  metadata:
    labels:
      app: kubia
  spec:
    containers:
      - name: kubia
        image: luksa/kubia
        ports:
          - containerPort: 8080
```

The pod template
for creating new
pods

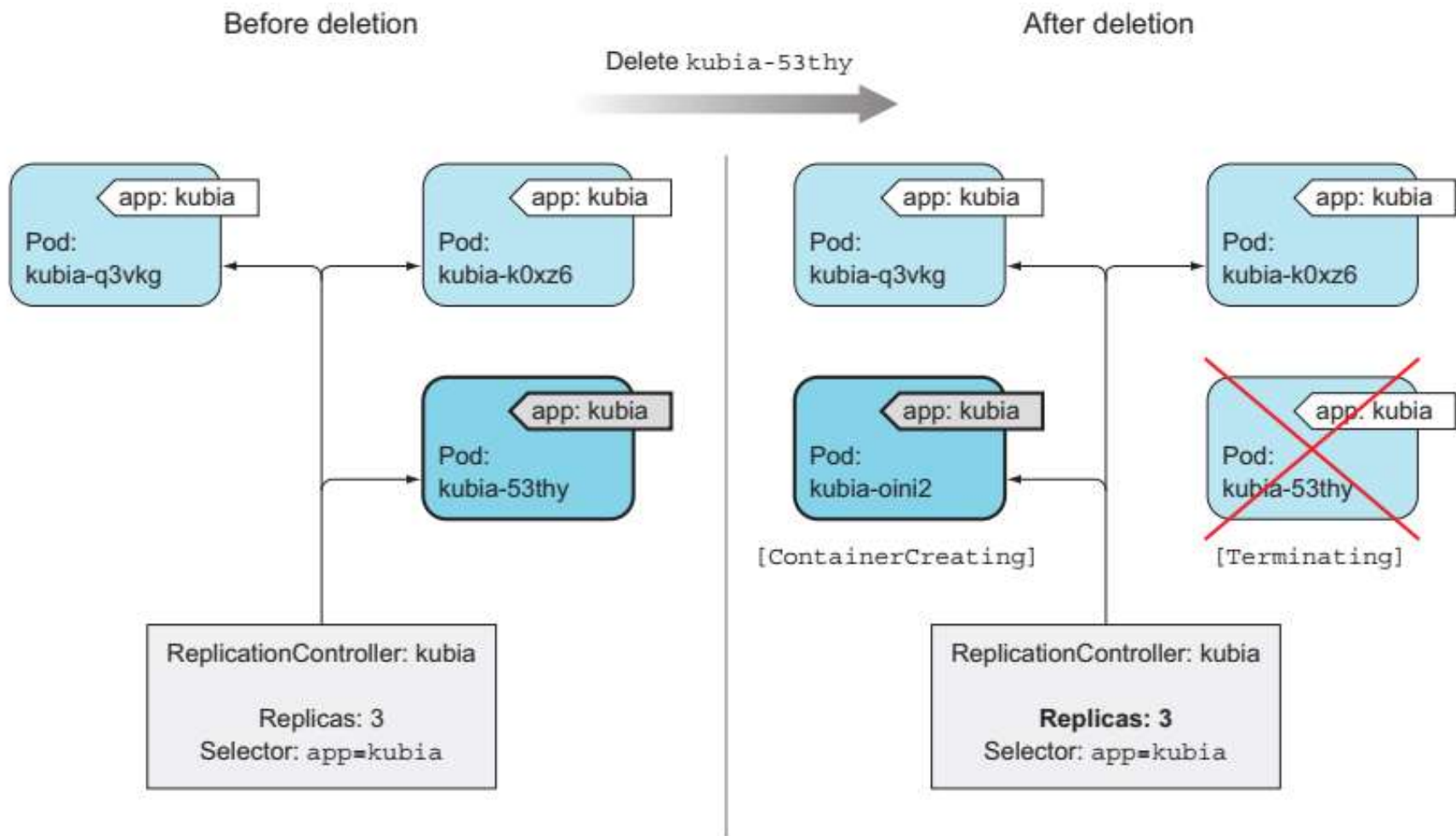


Figure 4.4 If a pod disappears, the ReplicationController sees too few pods and creates a new replacement pod.

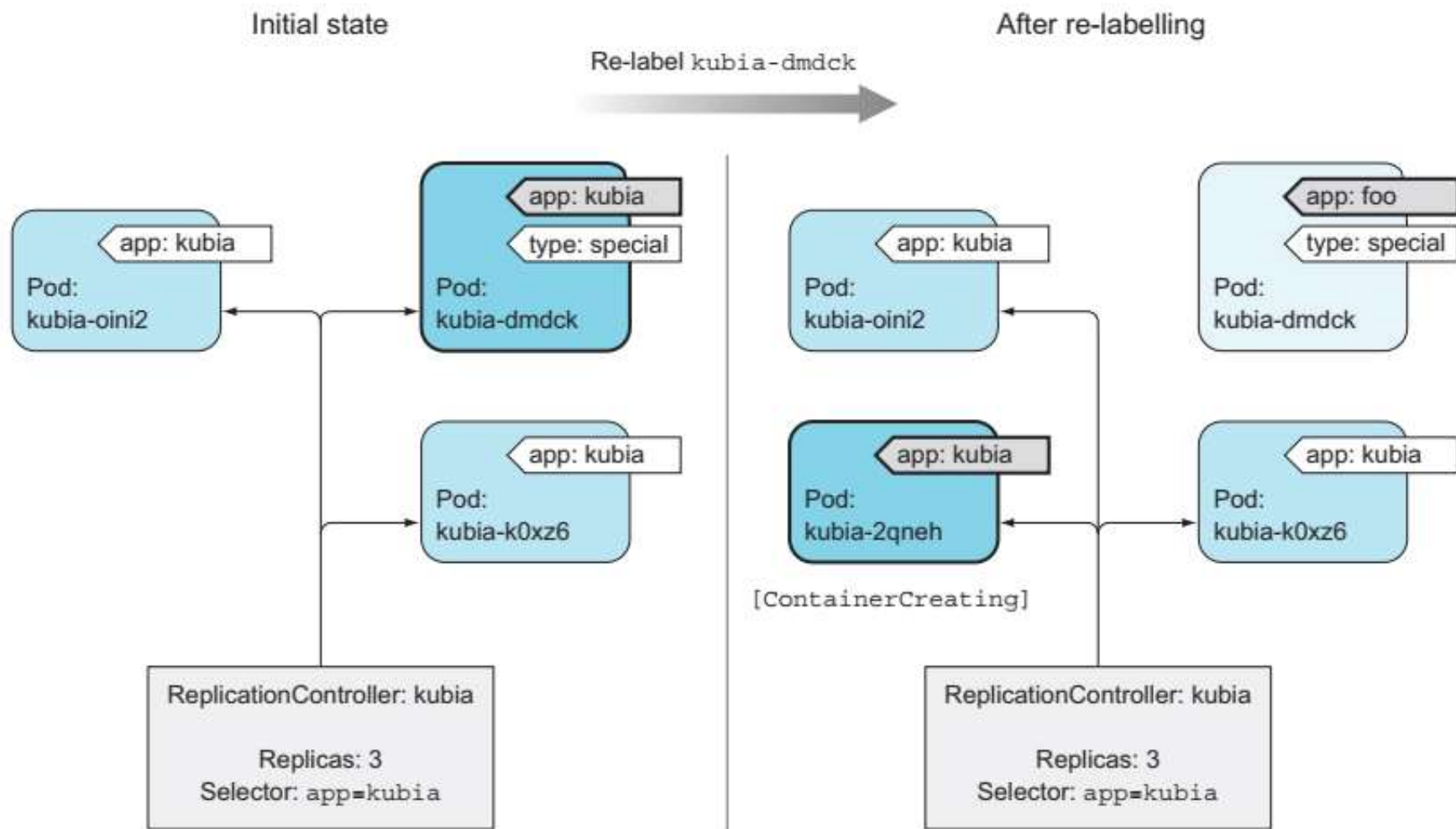


Figure 4.5 Removing a pod from the scope of a ReplicationController by changing its labels

Services

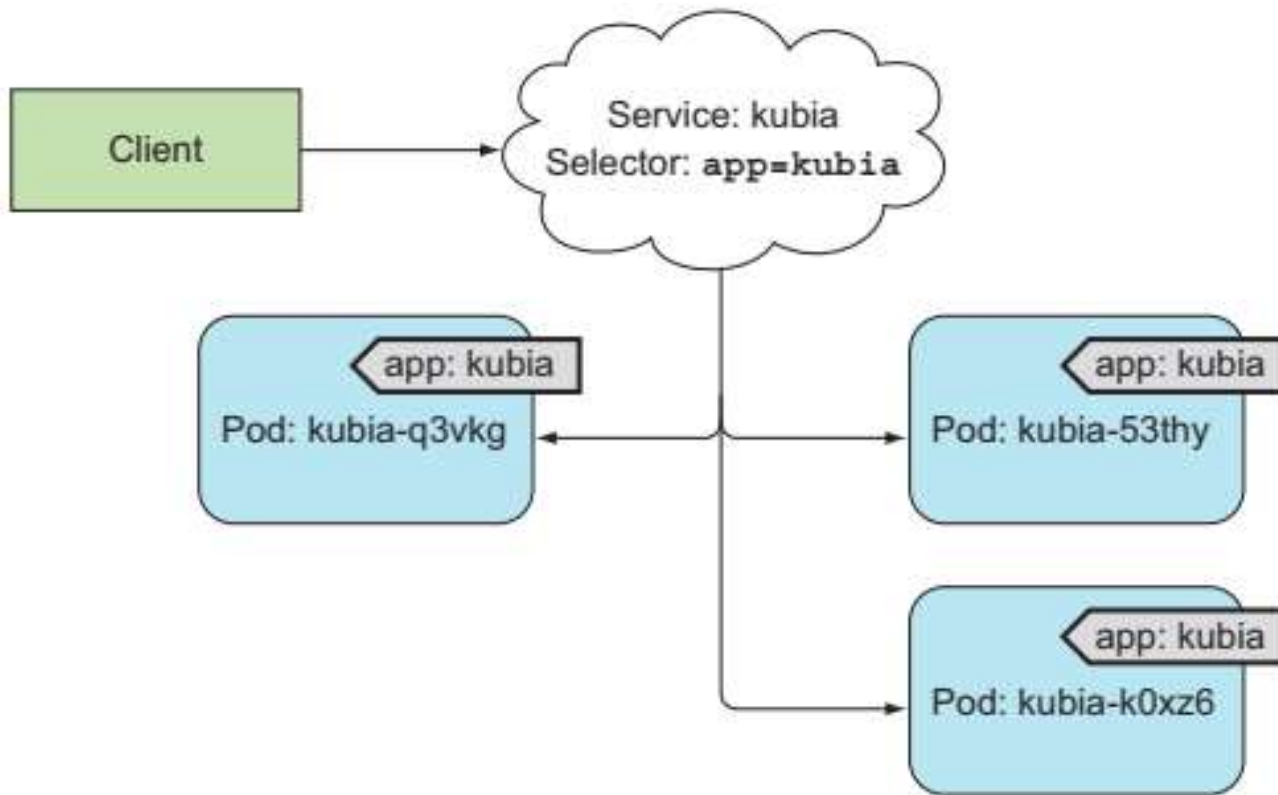


Figure 5.2 Label selectors determine which pods belong to the Service.

Listing 5.1 A definition of a service: kubia-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: kubia
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: kubia
```

**The port this service
will be available on**

**The container port the
service will forward to**

**All pods with the app=kubia
label will be part of this service.**

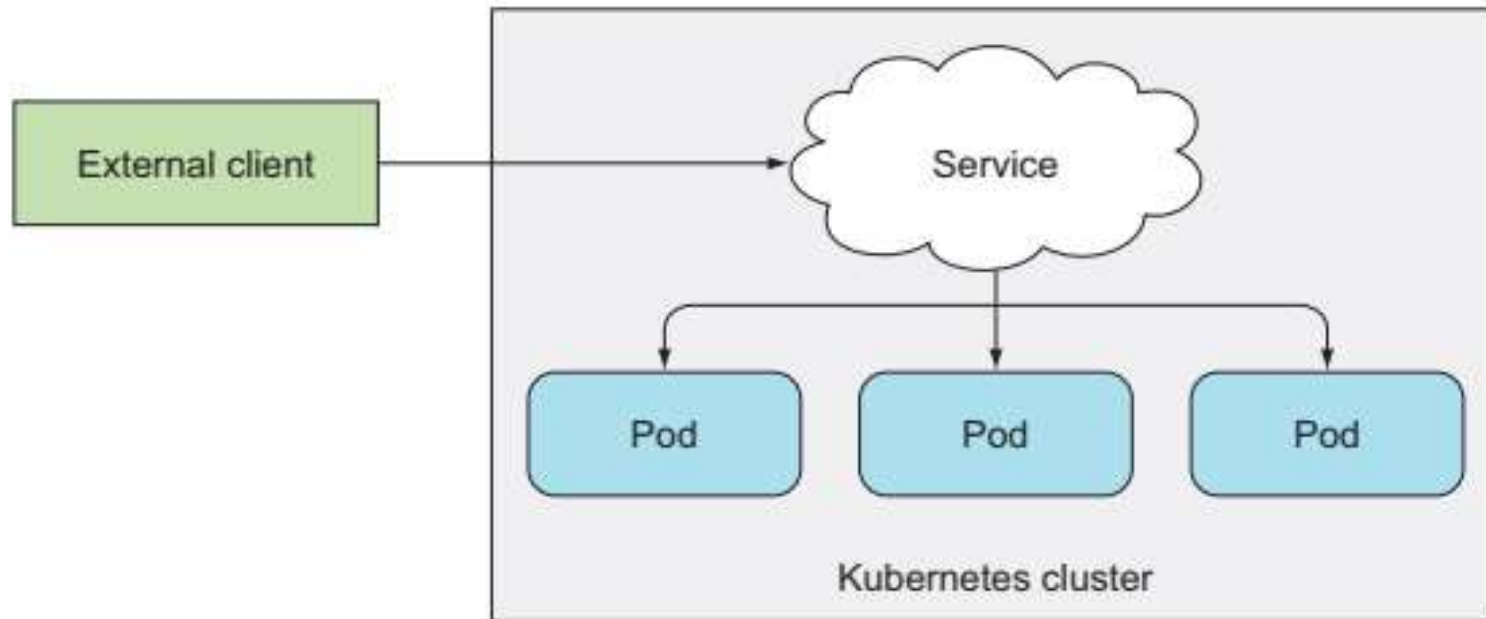
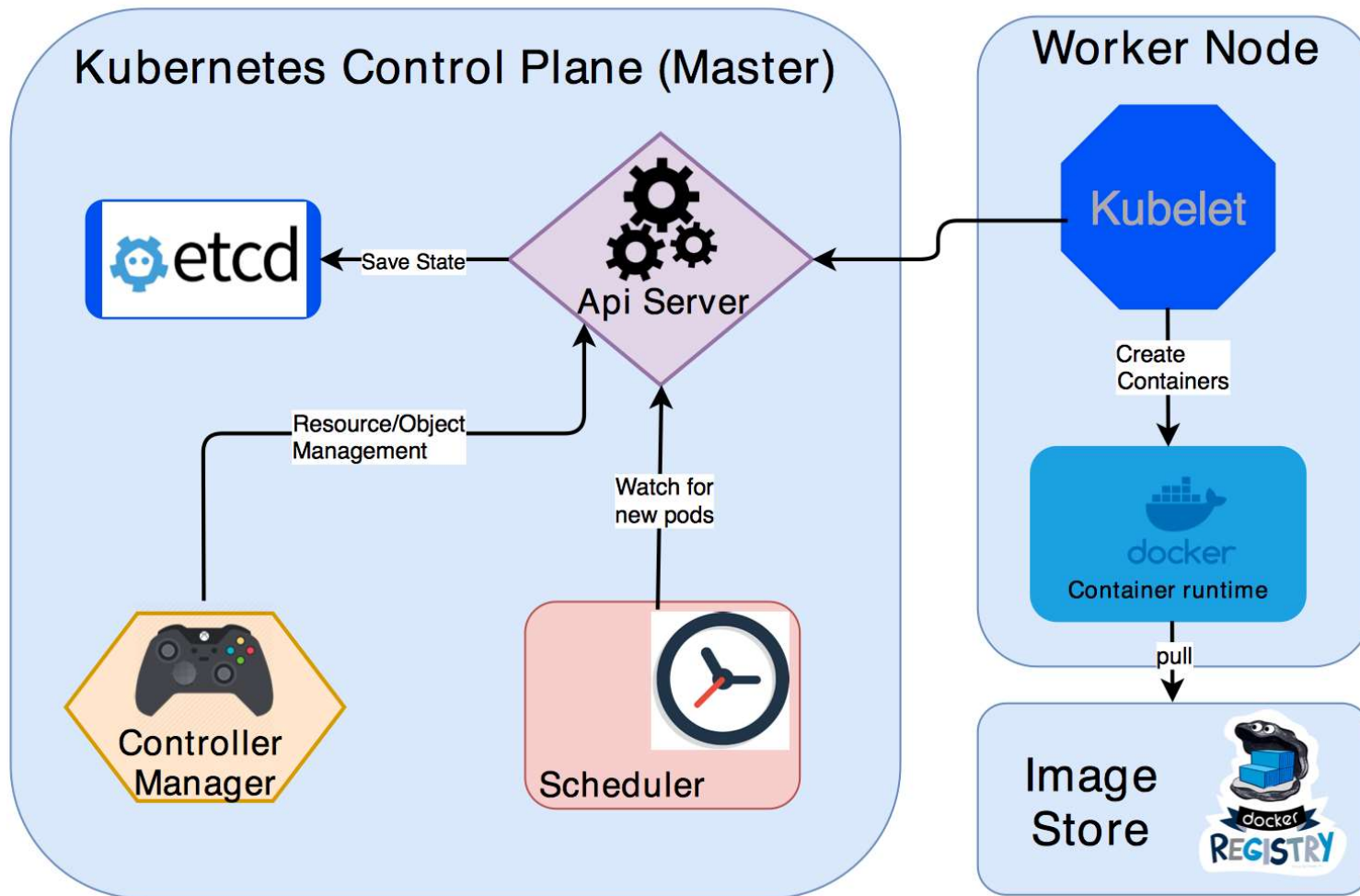
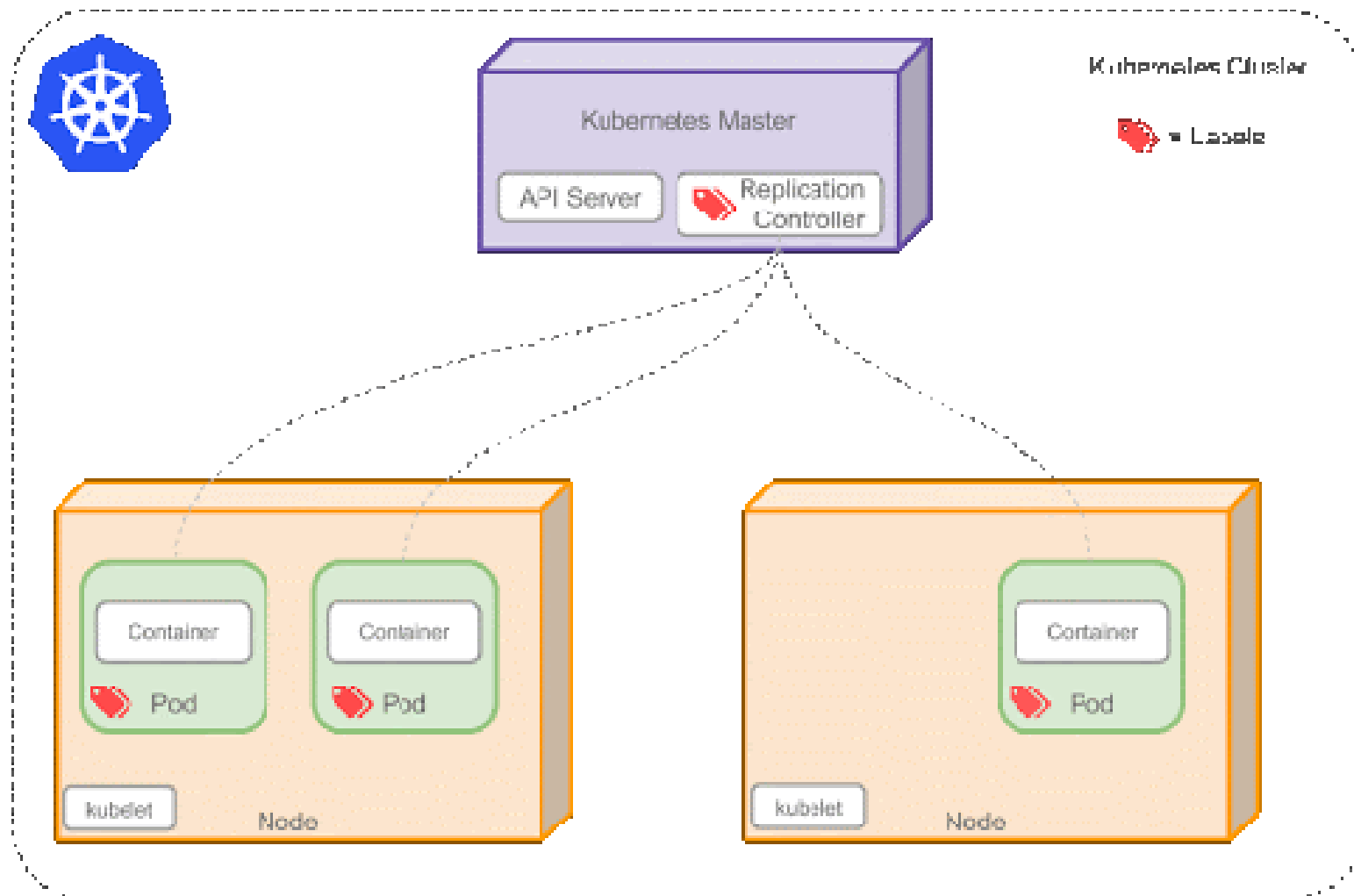


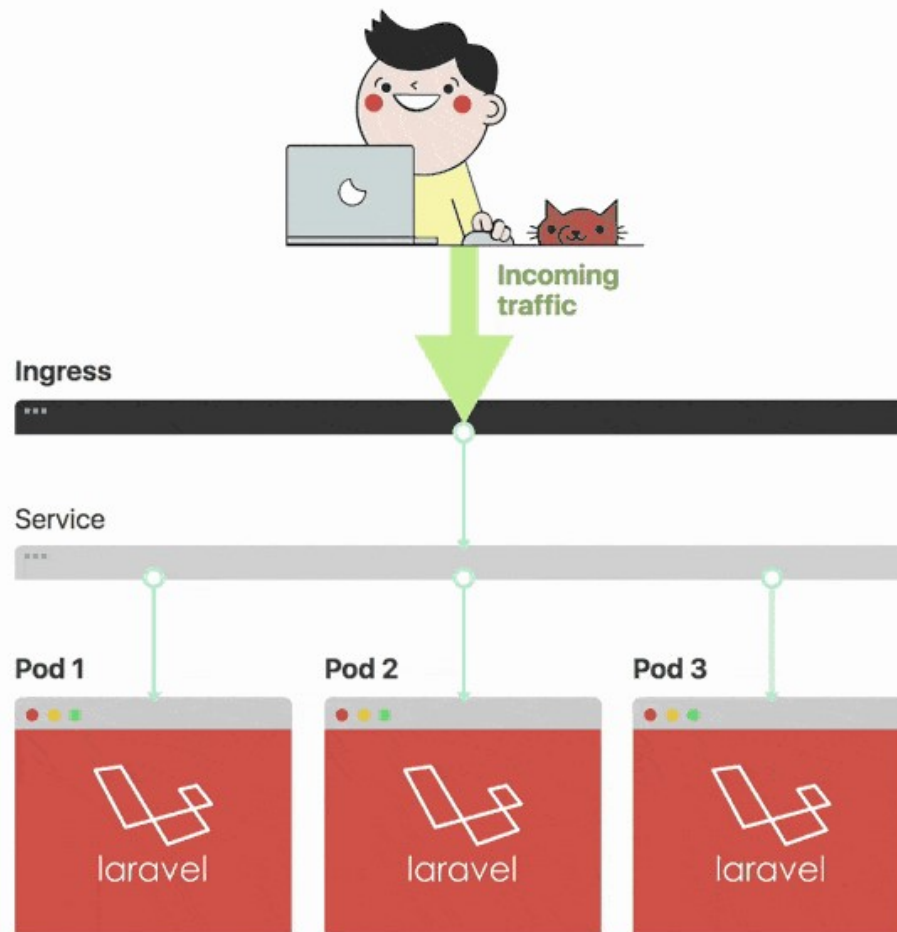
Figure 5.5 Exposing a service to external clients

Kubernetes Architecture



Kubernetes Cluster





Kubernetes Architecture

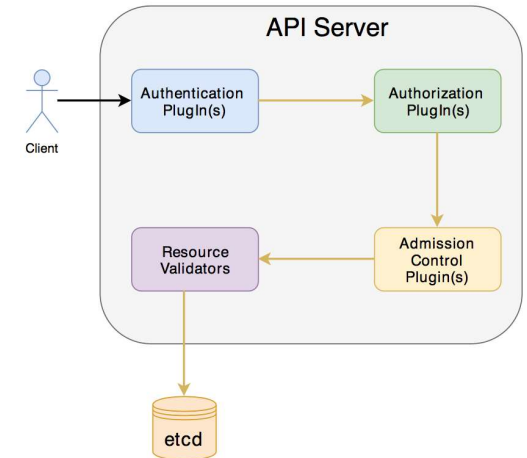
- A Kubernetes cluster consists of two main components:
 - Master (Control Plane)
 - Worker Nodes.
- Master has following components. These components are responsible for maintaining the state of the cluster:
 - etcd distributed key value store.
 - API Server.
 - Controller Manager
 - Scheduler
- Every worker node consists of the following components.
- These components are responsible for deploying and running the application containers.
 - Kubelet
 - Container Runtime (Docker)

Master Components

- etcd:
 - Stores the cluster status and metadata.
 - A distributed key value store
 - Provides reliable way of storing data across a cluster of machines.
 - API Server directly talk to etcd store.
 - K8s stores all its data under /registry directory in etcd.
- Api Server:
 - The central place for all other components.
 - Api Server will take care about validating the object before saving the information to etcd.
 - The client for the Api Server can be either kubectl (command line tool) or a Rest Api client.

Master Components

- Api Server:
 - Several plugin's are invoked by Api Server before creating/deleting/updating the object in etcd.
- Scheduler:
 - Allocate what node the pods needs to be created
- Controller Manager:
 - Make sure the actual state of the system converges towards the desired state.
 - Watch the API Server for changes to resources/objects and perform necessary actions like create/update/delete of the resource.



Worker Node components

- Kubelet
 - The agent that runs on each node in the cluster.
 - Monitors the Api Server for Pods
 - Start the pod's containers by instructing to docker runtime.
 - Monitors the status of running containers and reports to api server
 - Also do health checks for the container and restart if needed.
- Docker
 - Container runtime used by Kubelet for spinning up Containers

Other components

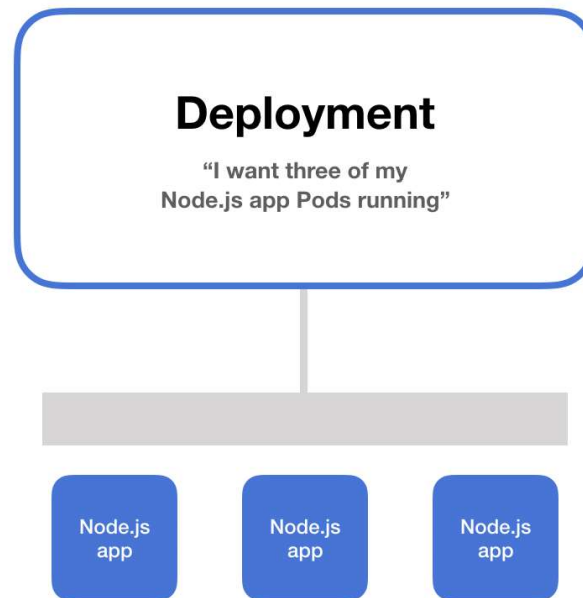
- Nodes:
 - Machine on which Kubernetes is installed.
 - This is where containers inside the pods will be launched by Kubernetes.
- Master Node:
 - Responsible for managing the cluster
 - A Kubernetes cluster also contains one or more master nodes that run the Kubernetes control plane
- Pod
 - Smallest deployable unit that can be managed by Kubernetes.
 - A logical group of one or more containers that share the same IP address

Kubernetes namespaces

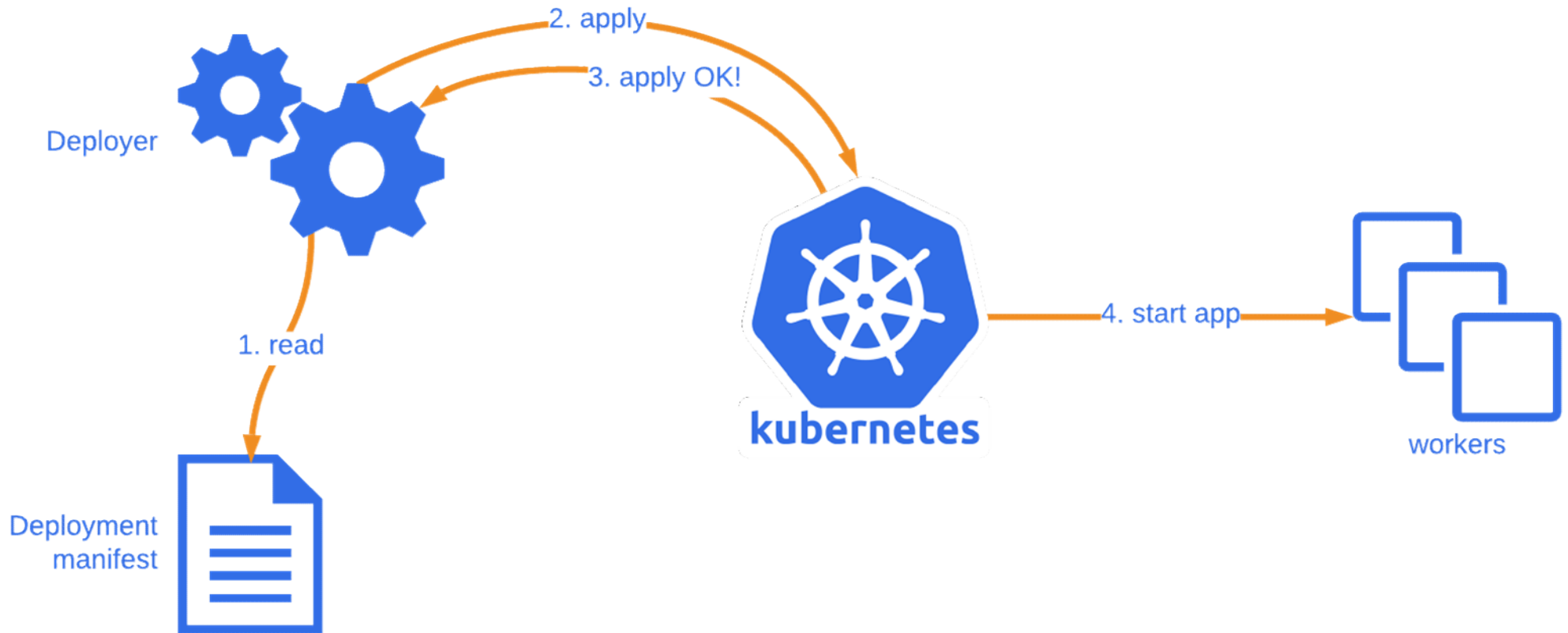
- Provide for a scope of Kubernetes resource, carving up your cluster in smaller units
 - `$ kubectl get ns`
 - `$ kubectl describe ns default`
 - `$ kubectl create namespace test`

Kubernetes deployment

- Everyone running applications on Kubernetes cluster uses a deployment.
- It's what you use to scale, roll out, and roll back versions of your applications.
- With a deployment, you tell Kubernetes how many copies of a Pod you want running. The deployment takes care of everything else.

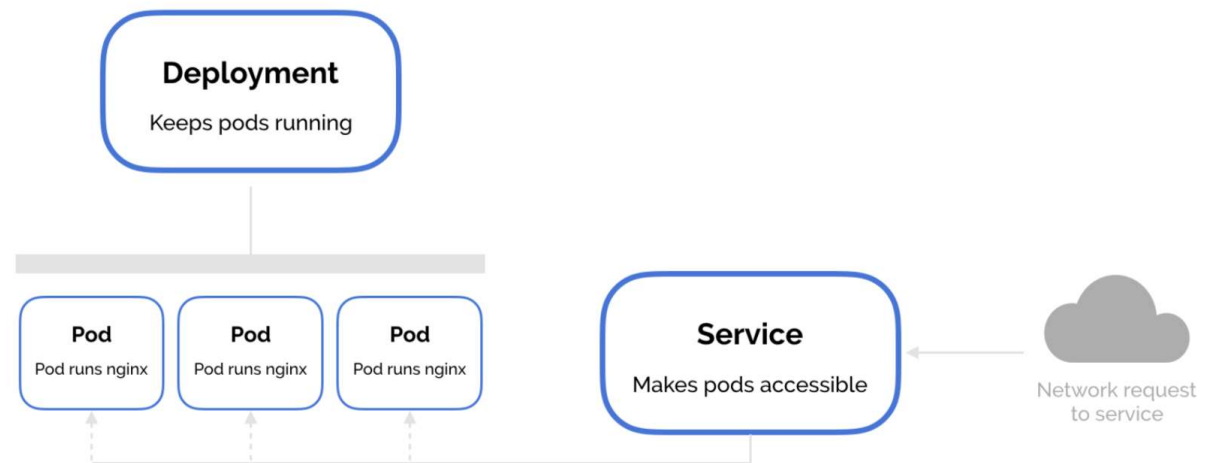


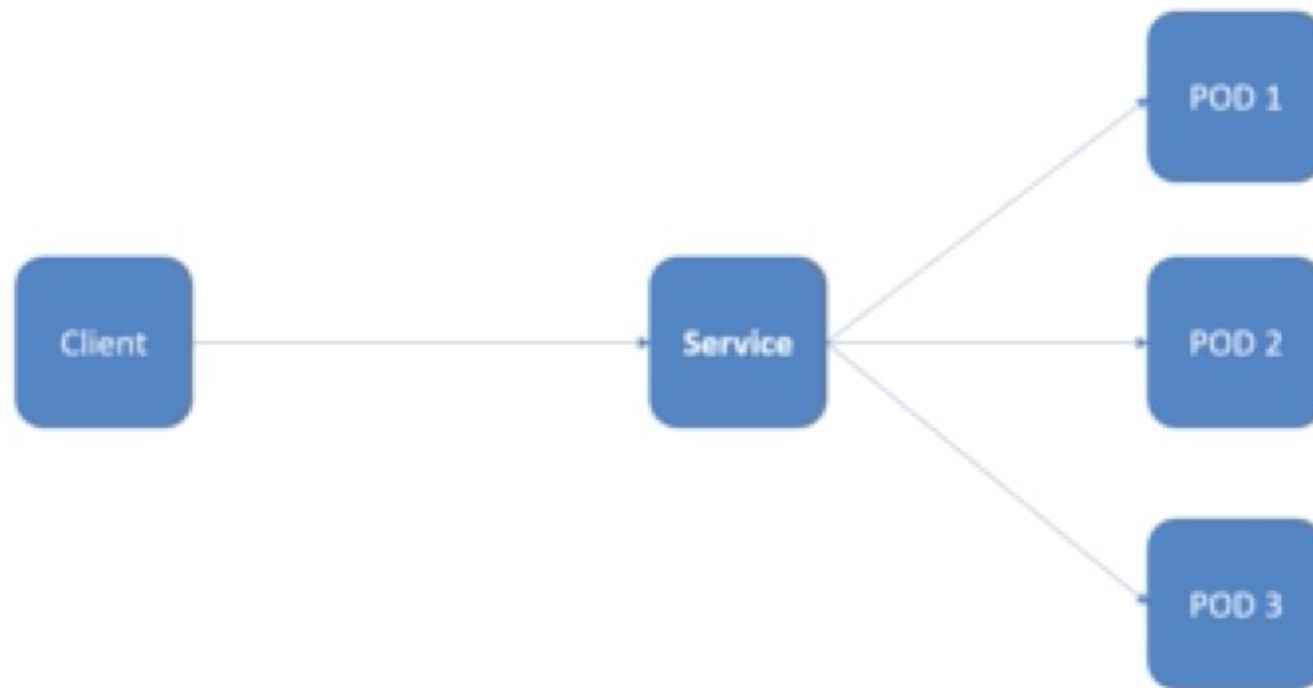
Kubernetes deployment



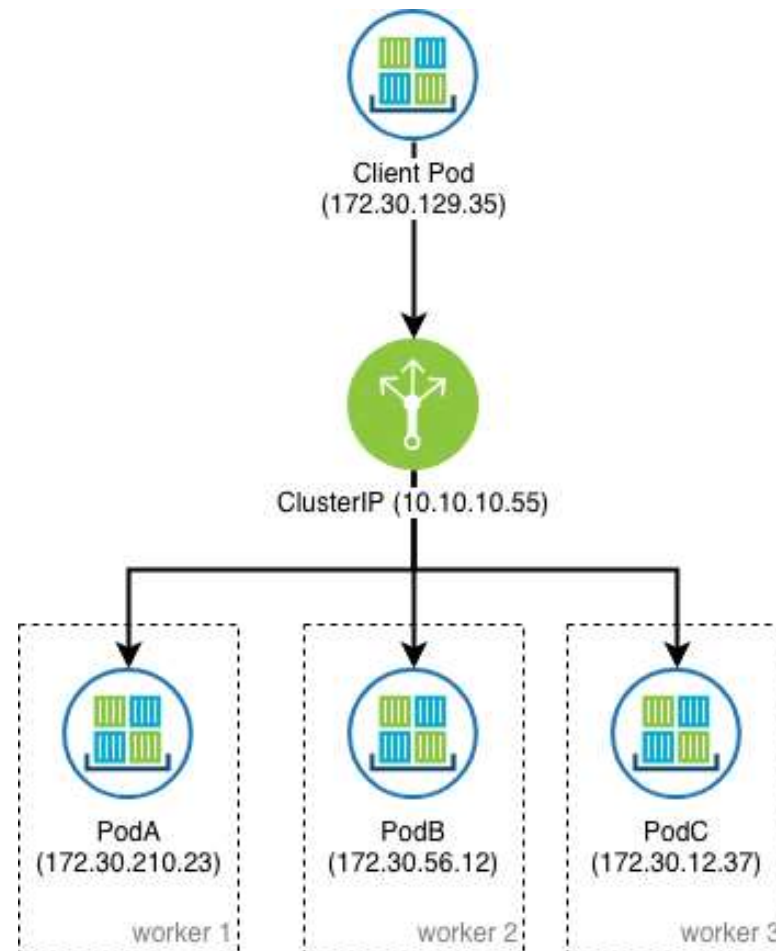
Deployment vs service

- A deployment is used to keep a set of pods running by creating pods from a template.
- A service is used to allow network access to a set of pods.
- To access a Deployment with one or many PODs, you need a Kubernetes Service endpoint mapped to the deployment using labels and selectors.

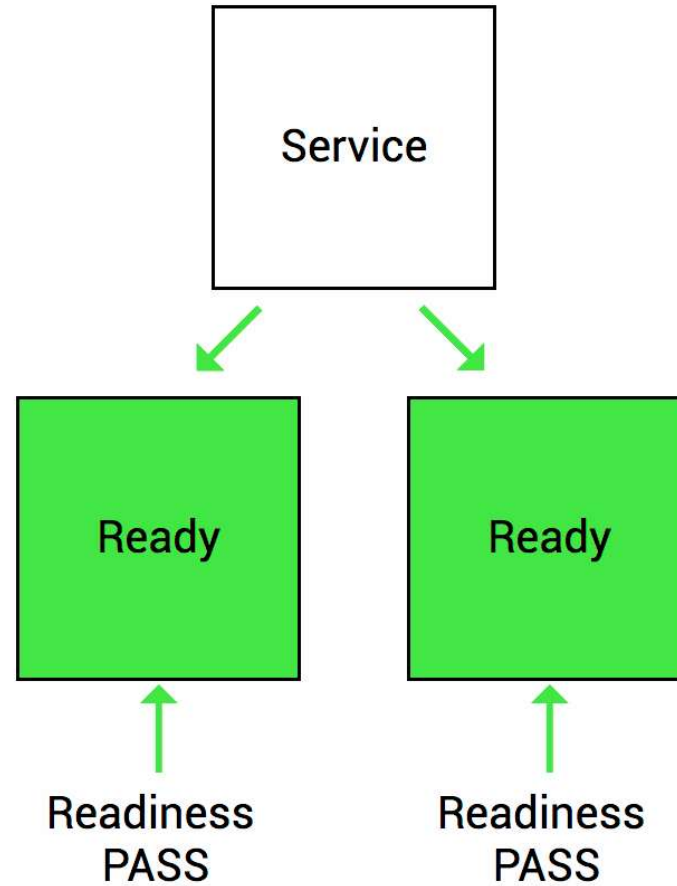




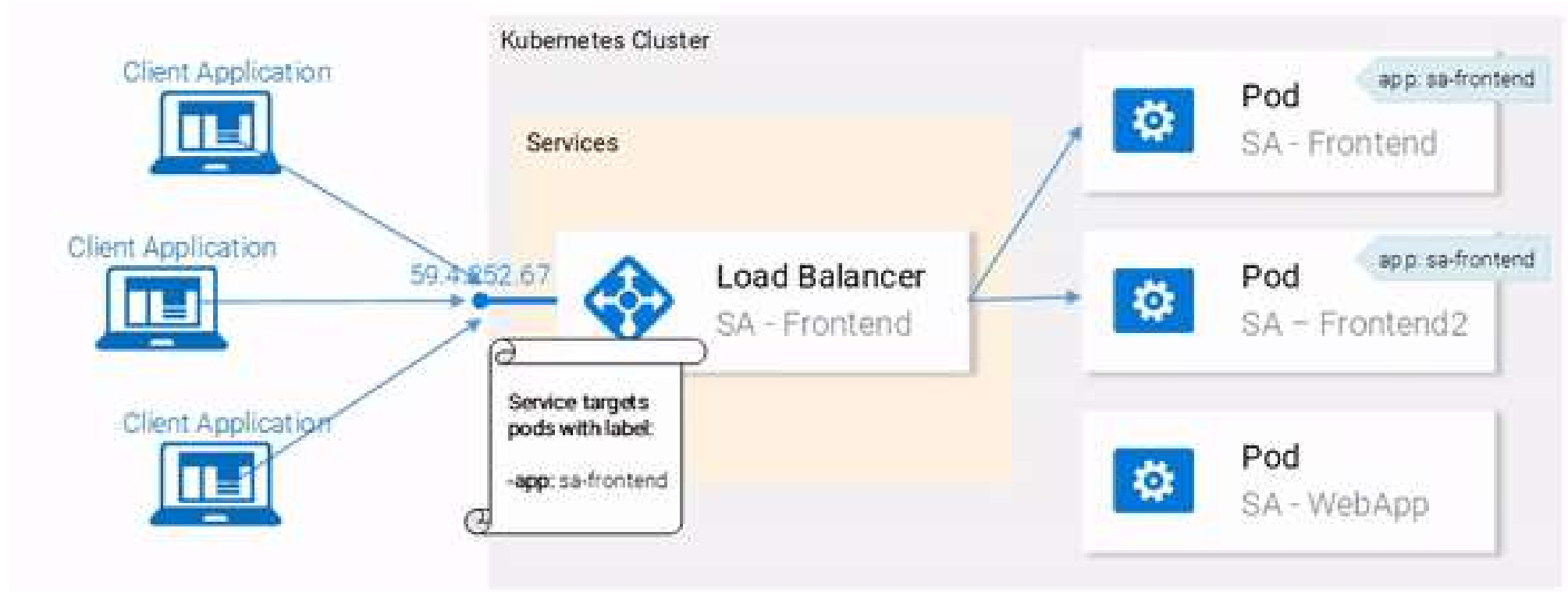
Service

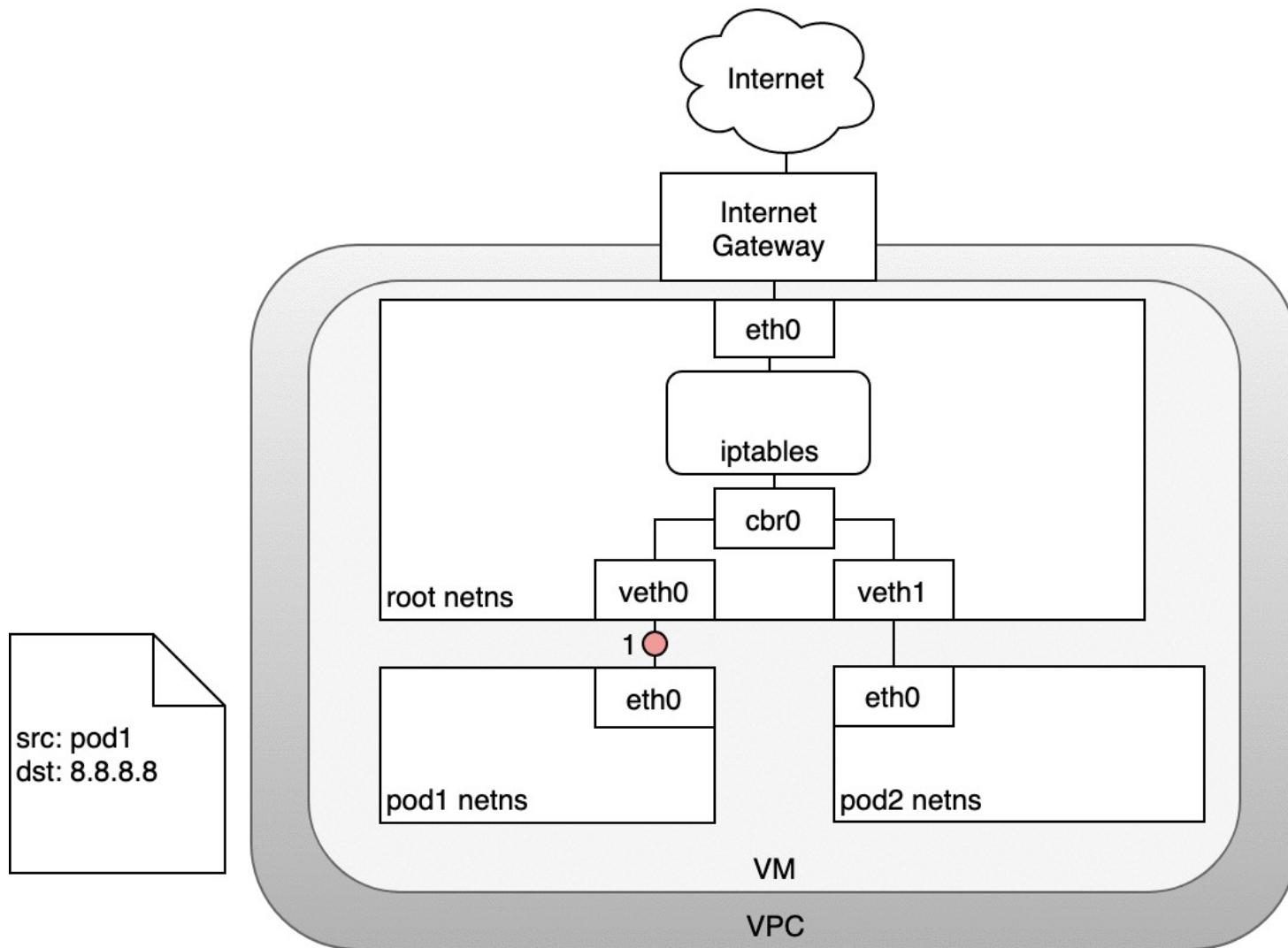


Service Readiness Probe



Load Balancer





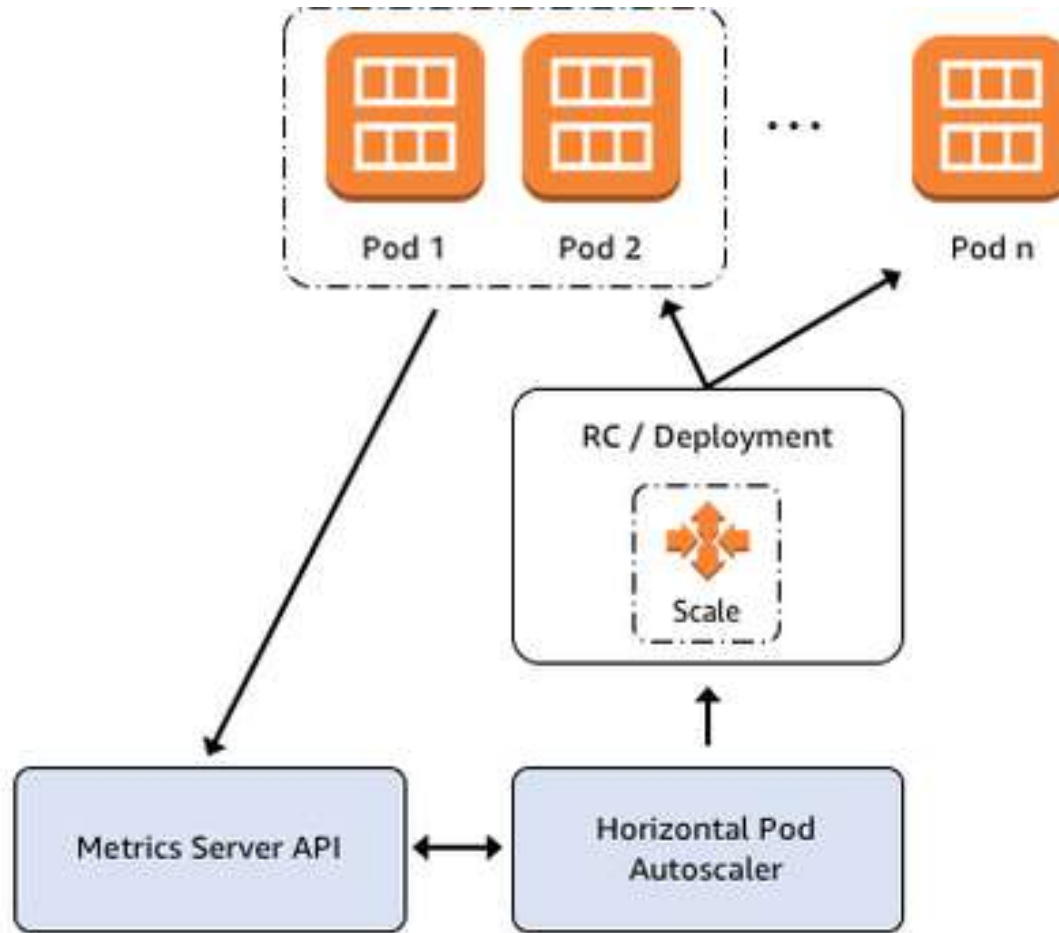
Installation - Online

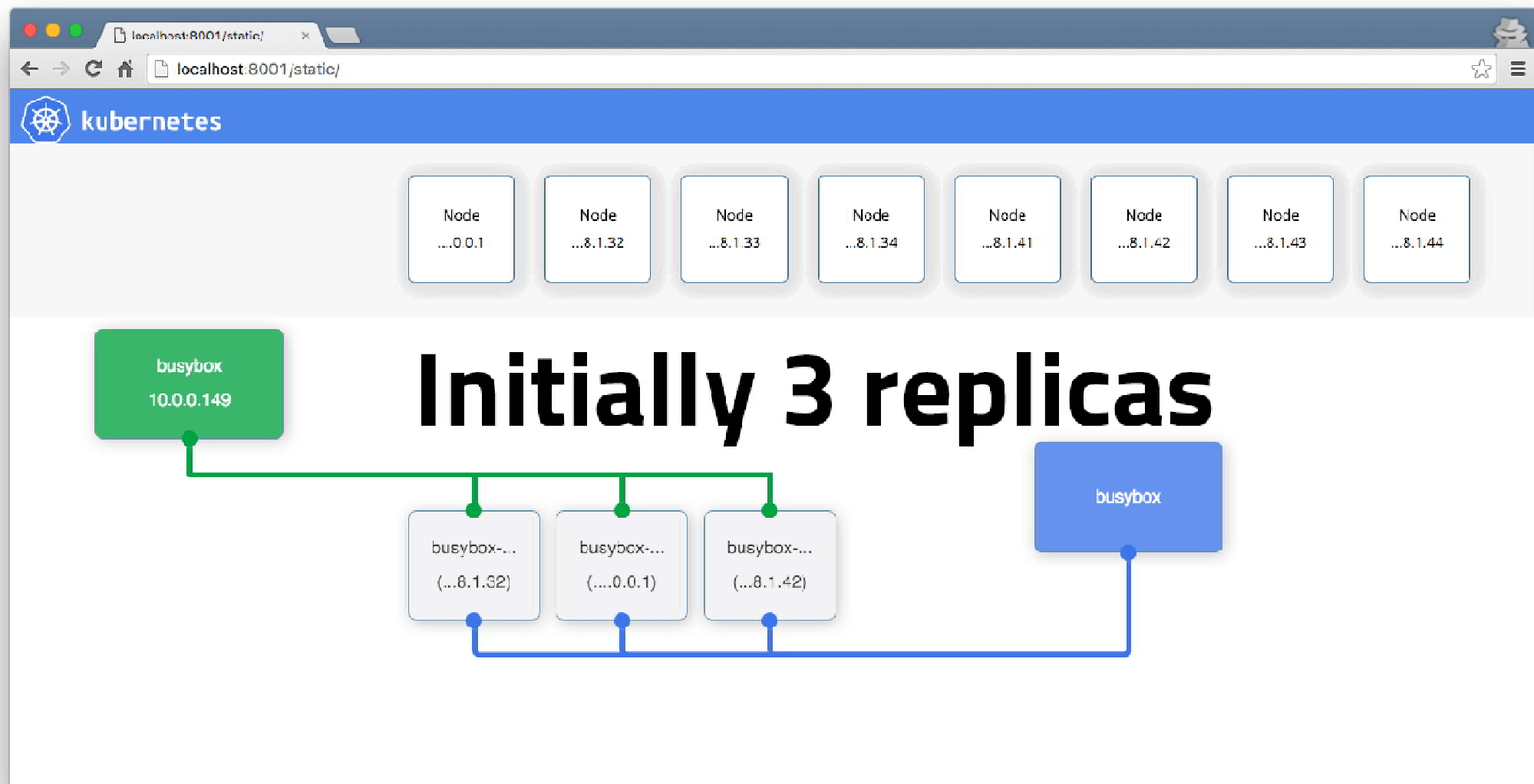
- <https://labs.play-with-k8s.com>

Scaling Kubernetes

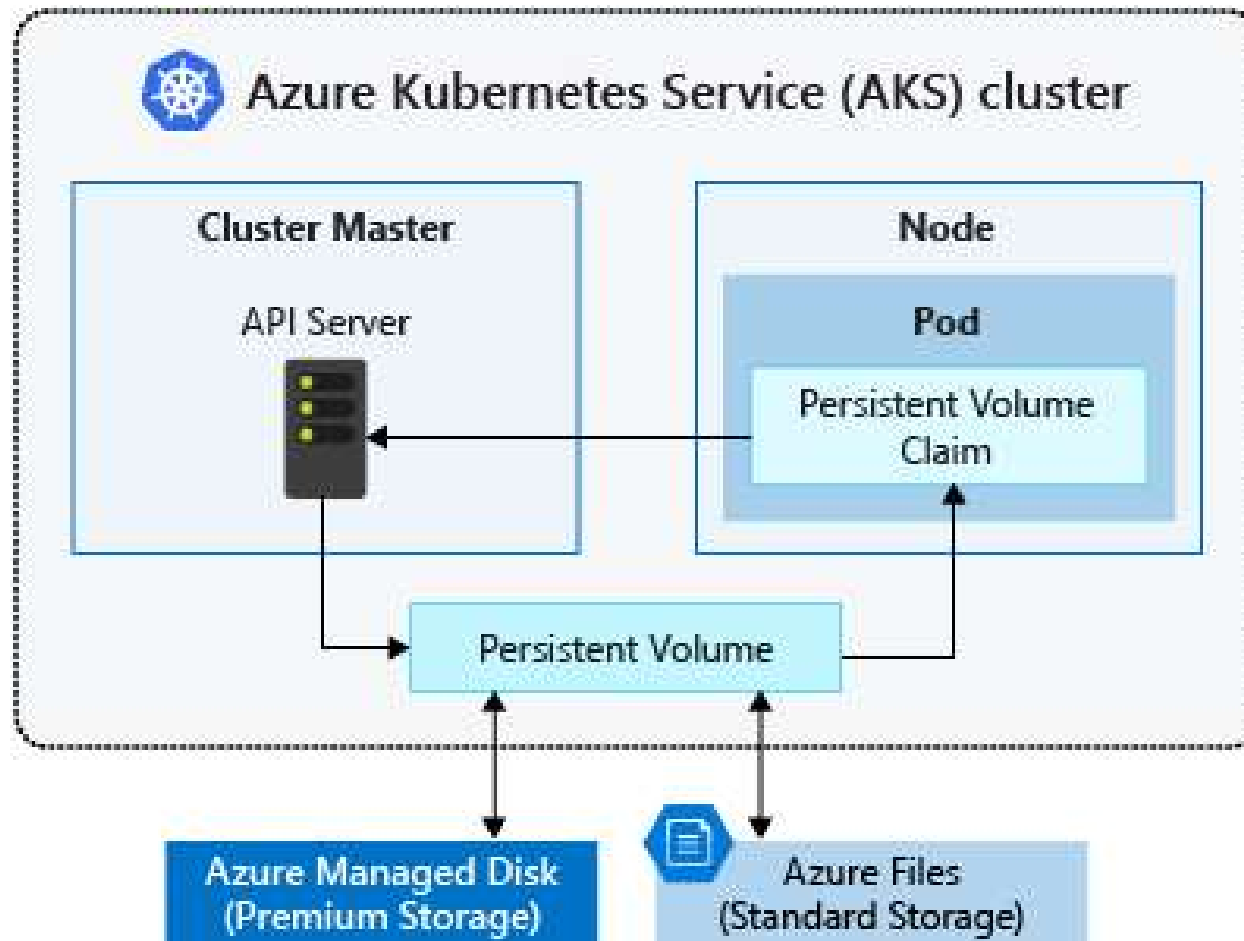
- Cluster scaling, sometimes called infrastructure-level scaling,
 - Refers to the (automated) process of adding or removing worker nodes based on cluster Utilization
- Application-level scaling, sometimes called pod scaling,
 - Refers to the (automated) process of manipulating pod characteristics based on a variety of metrics
 - CPU utilization
 - HTTP requests served per second etc
 - Two kinds of podlevel scalers exist
 - Horizontal Pod Autoscalers (HPAs), which increase or decrease the number of pod replicas depending on certain metrics.
 - Vertical Pod Autoscalers (VPAs), which increase or decrease the resource requirements of containers running in a pod.

Scaling Kubernetes

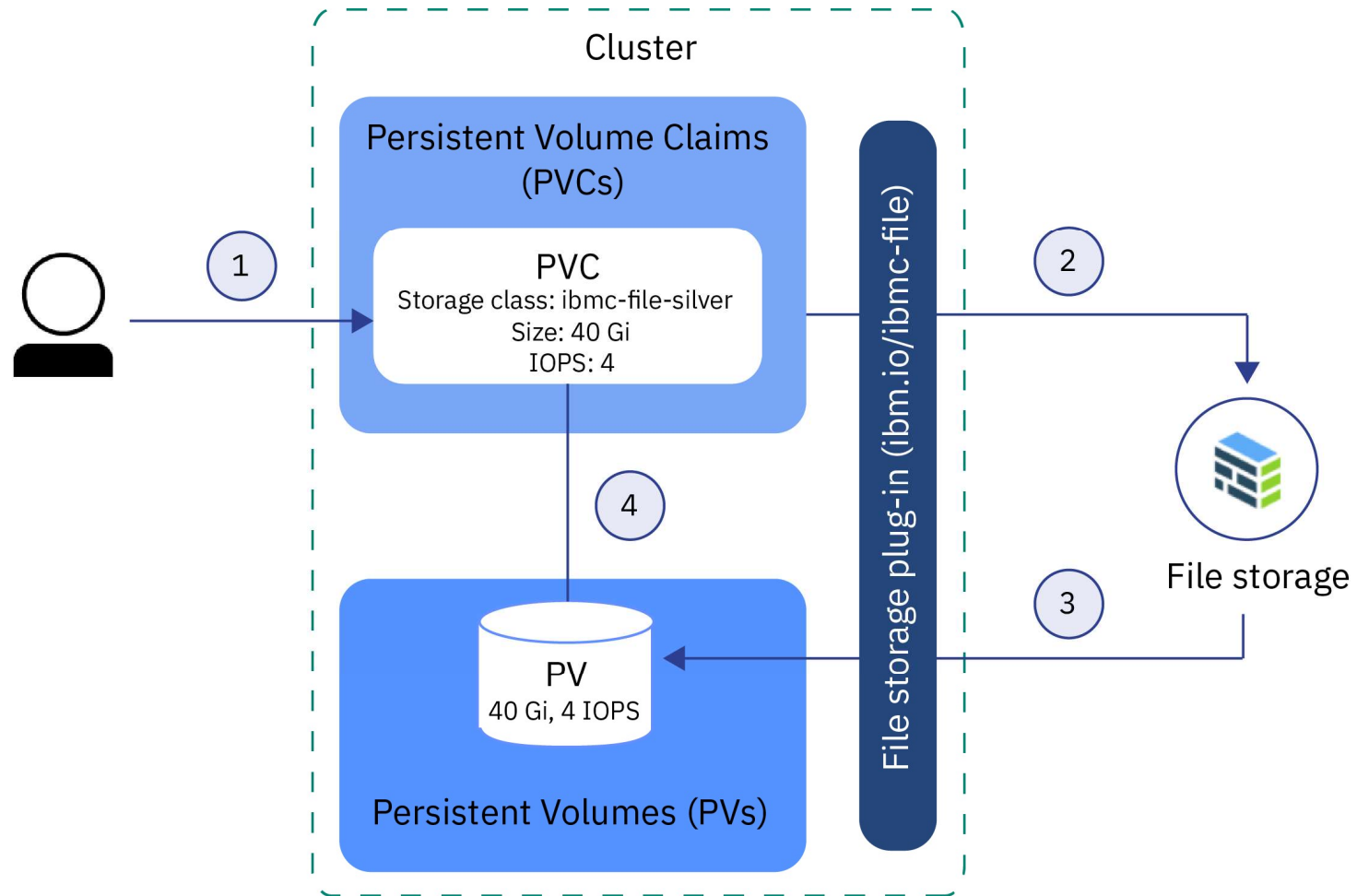




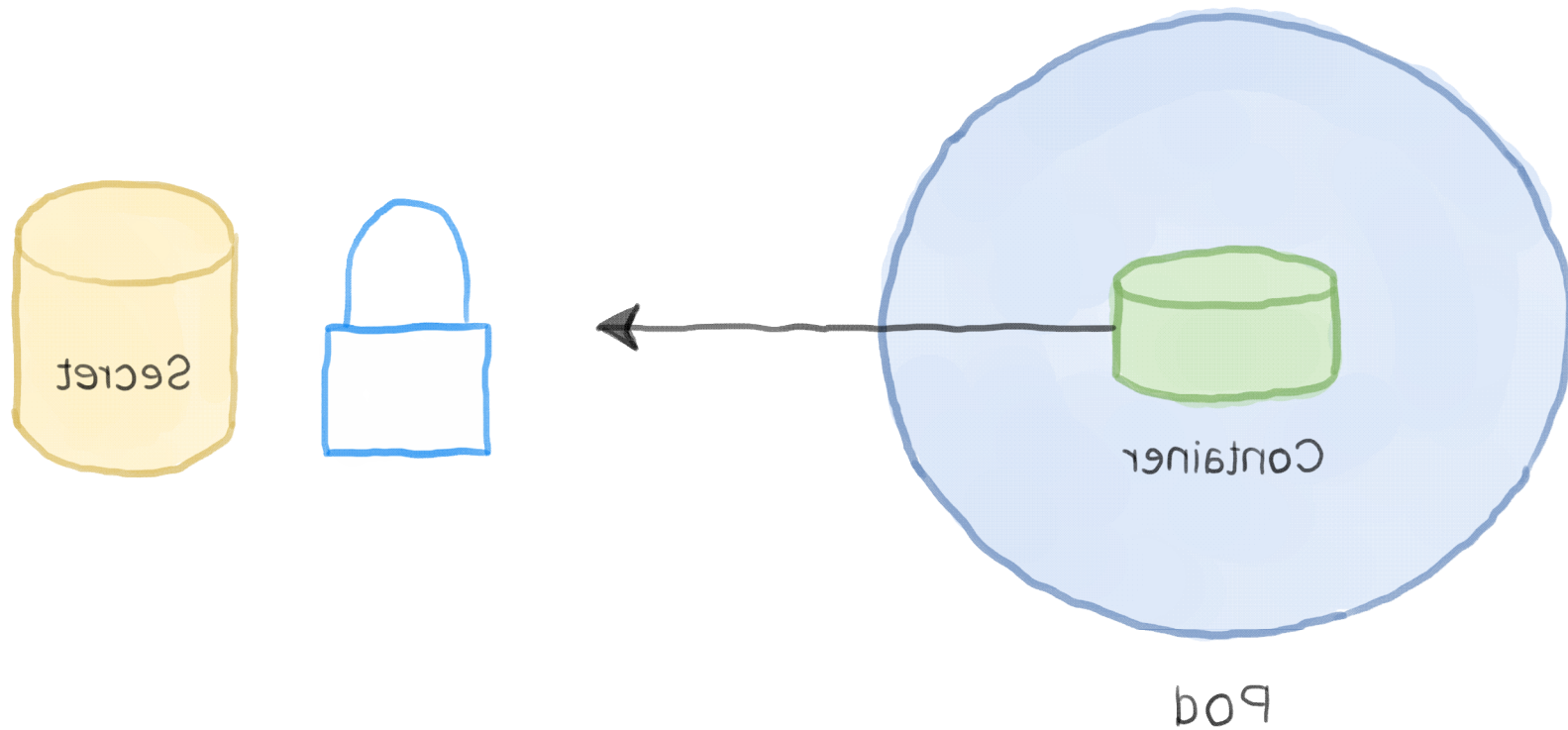
Persistent Volume



Persistent Volume



Secrets



Thanks