Kubernetes

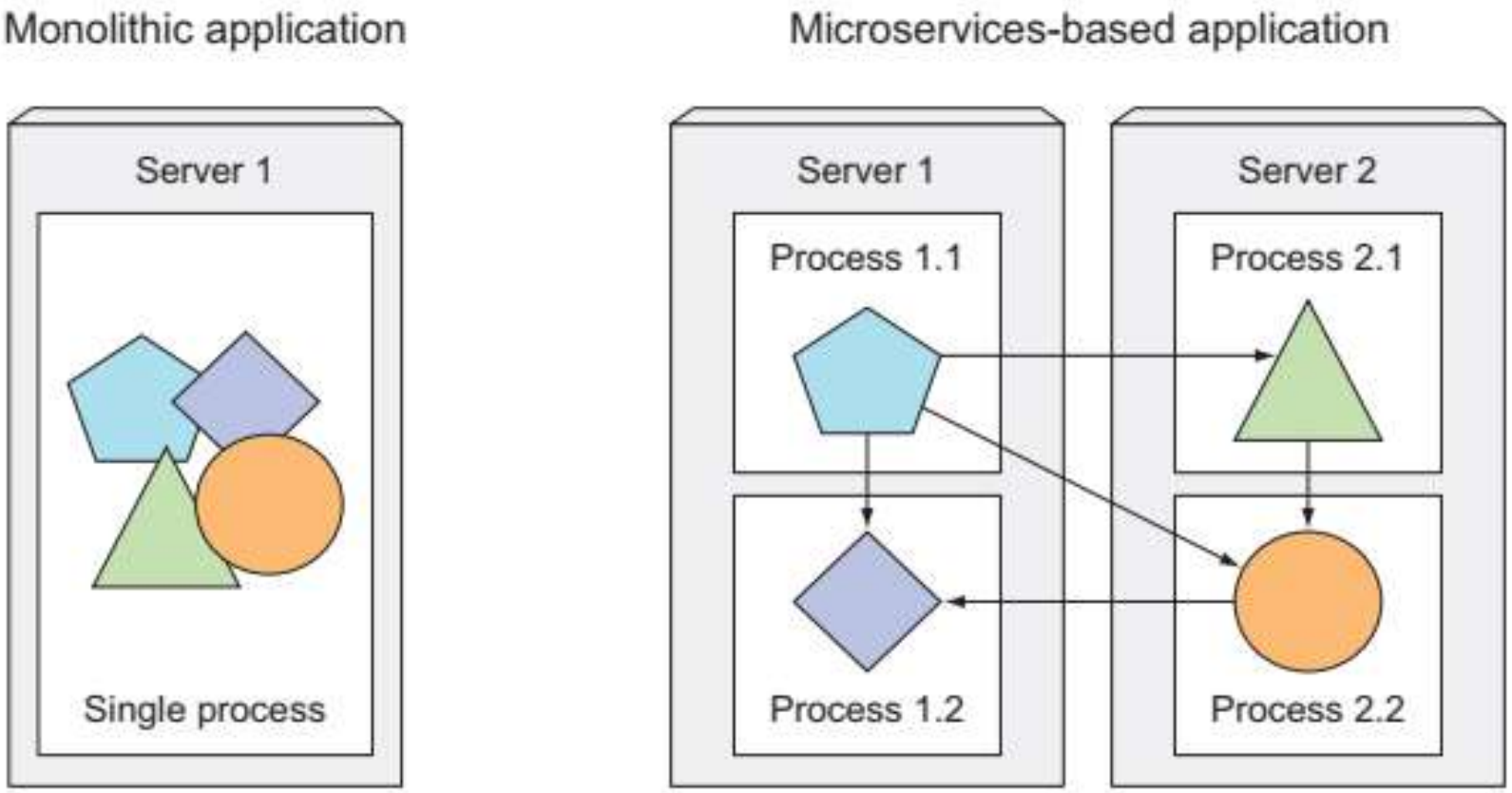
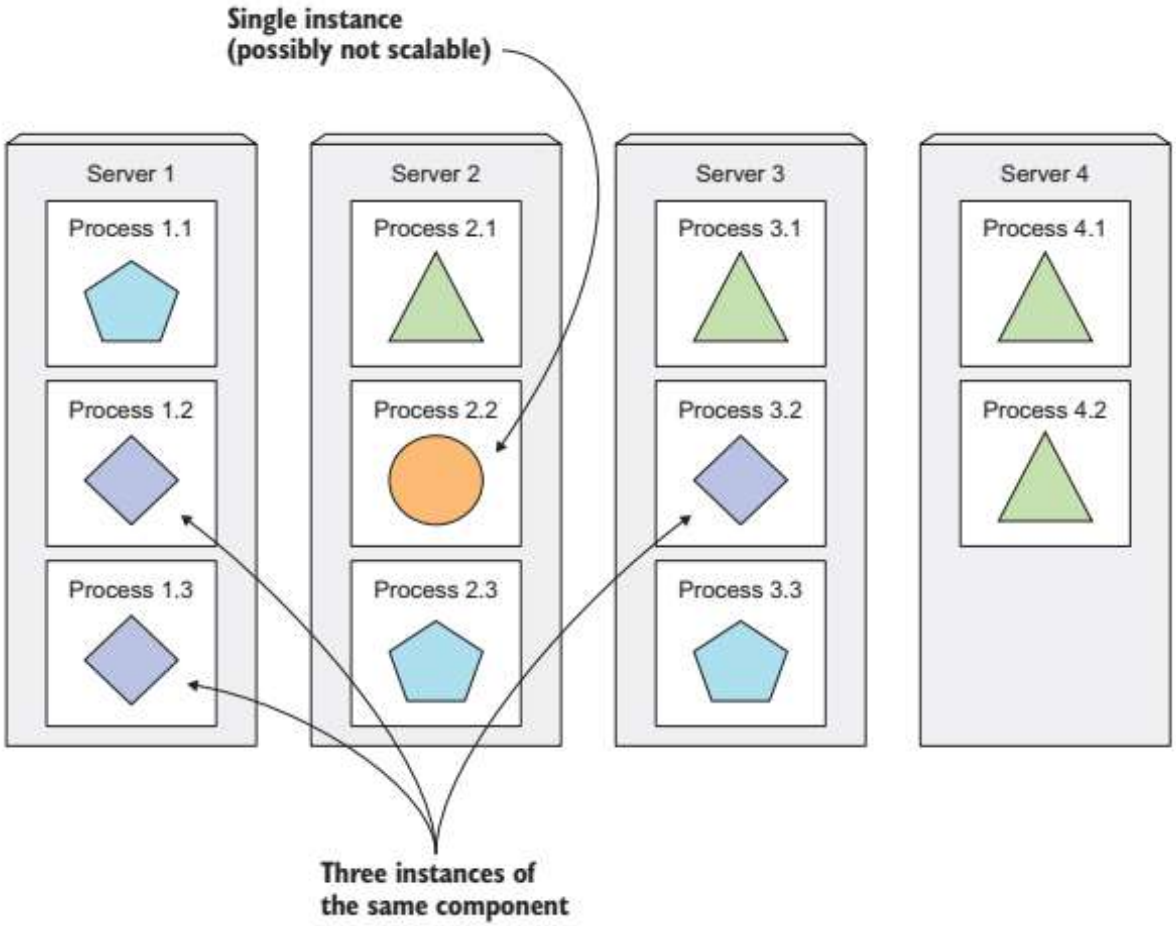
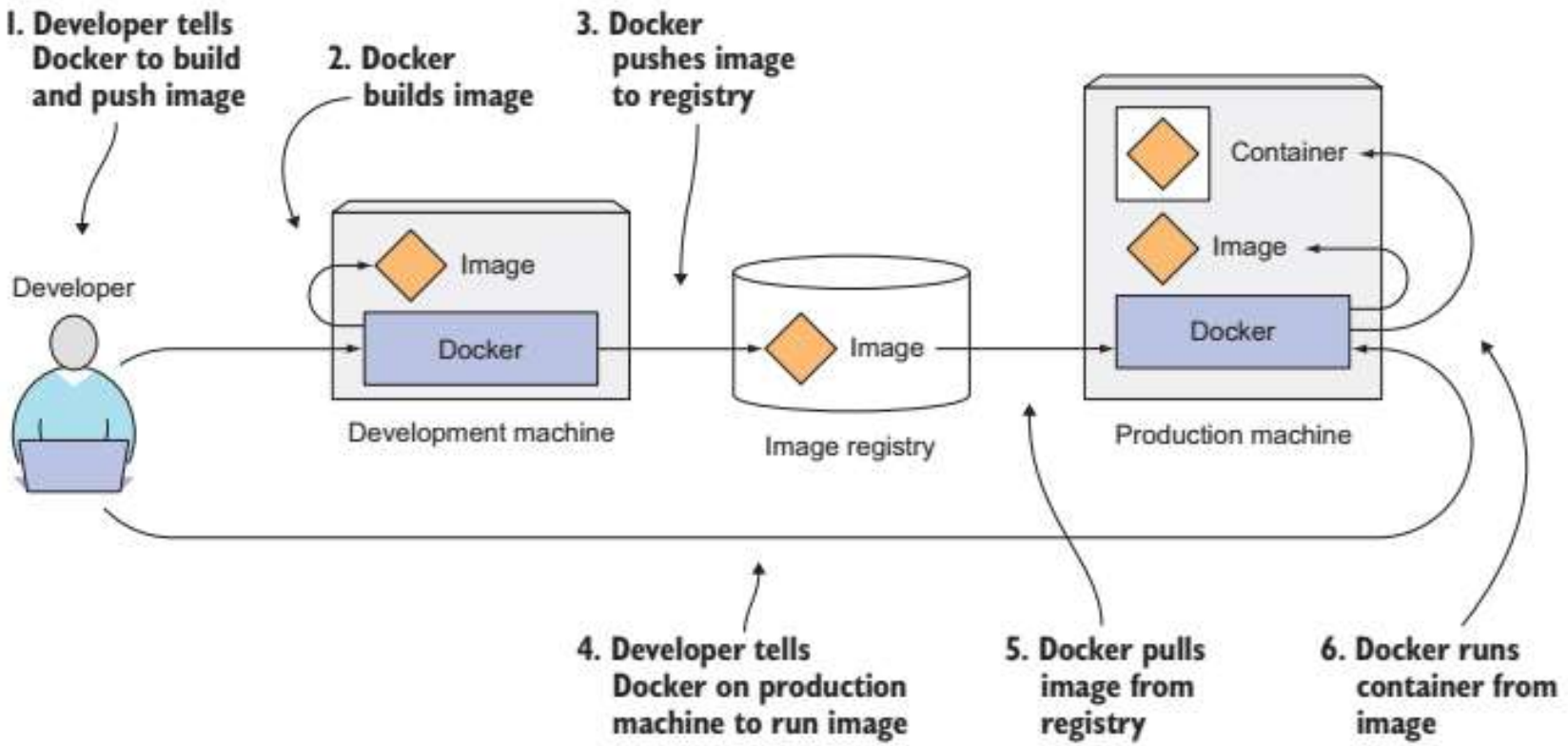
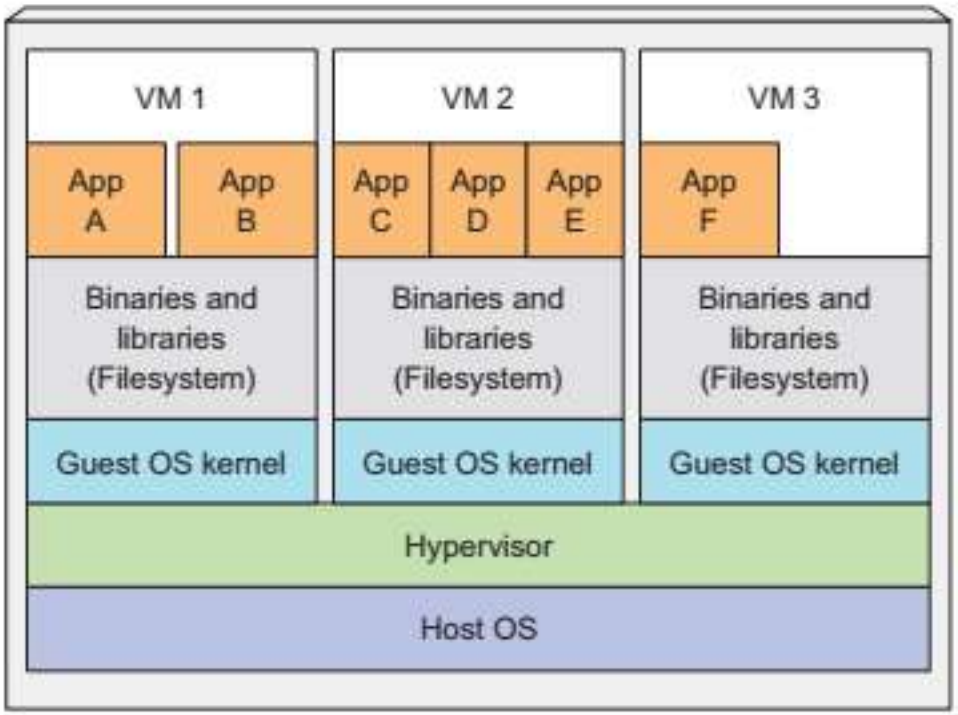


Figure 1.1 Components inside a monolithic application vs. standalone microservices

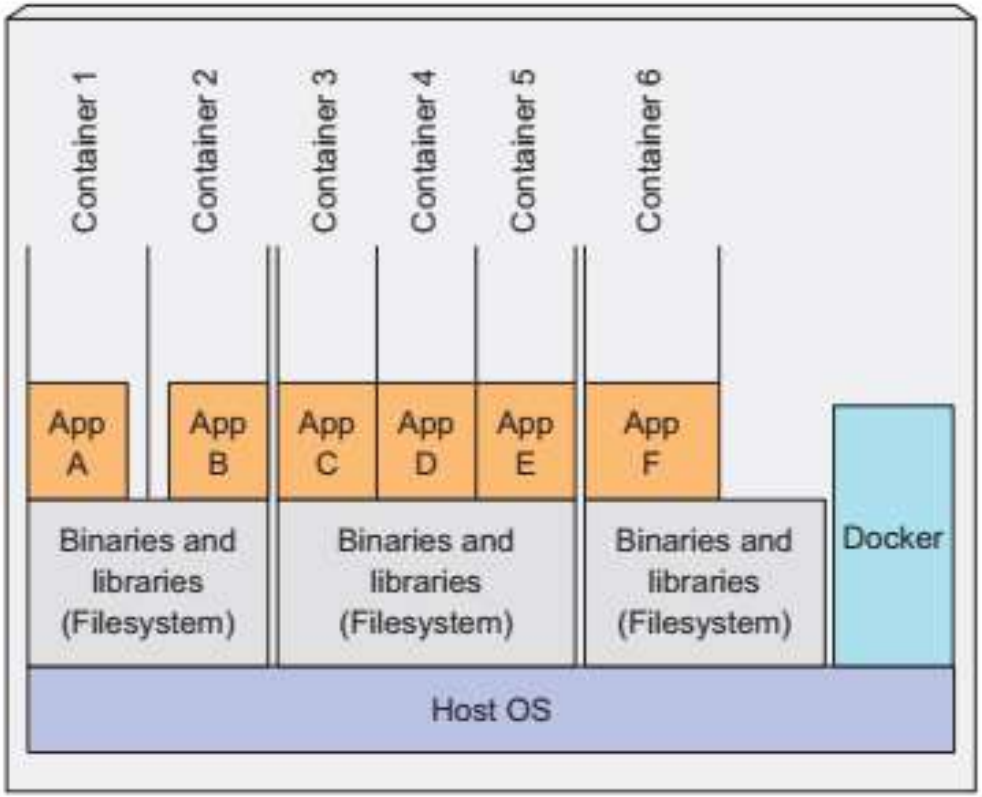

Single instance
(possibly not scalable)
Server 1
Process 1.1
Process 1.2
Process 1.3
Server 2
Process 2.1
Process 2.2
Process 2.3
Server 3
Process 3.1
Process 3.2
Process 3.3
Server 4
Process 4.1
Process 4.2
Three instances of
the same component


1. Developer tells Docker to build and push image
2. Docker builds image
3. Docker pushes image to registry
Developer
Image
Docker
Development machine
Image
Image registry
Container
Image
Docker
Production machine
4. Developer tells Docker on production machine to run image
5. Docker pulls image from registry
6. Docker runs container from image

Host running multiple VMs



Host running multiple Docker containers



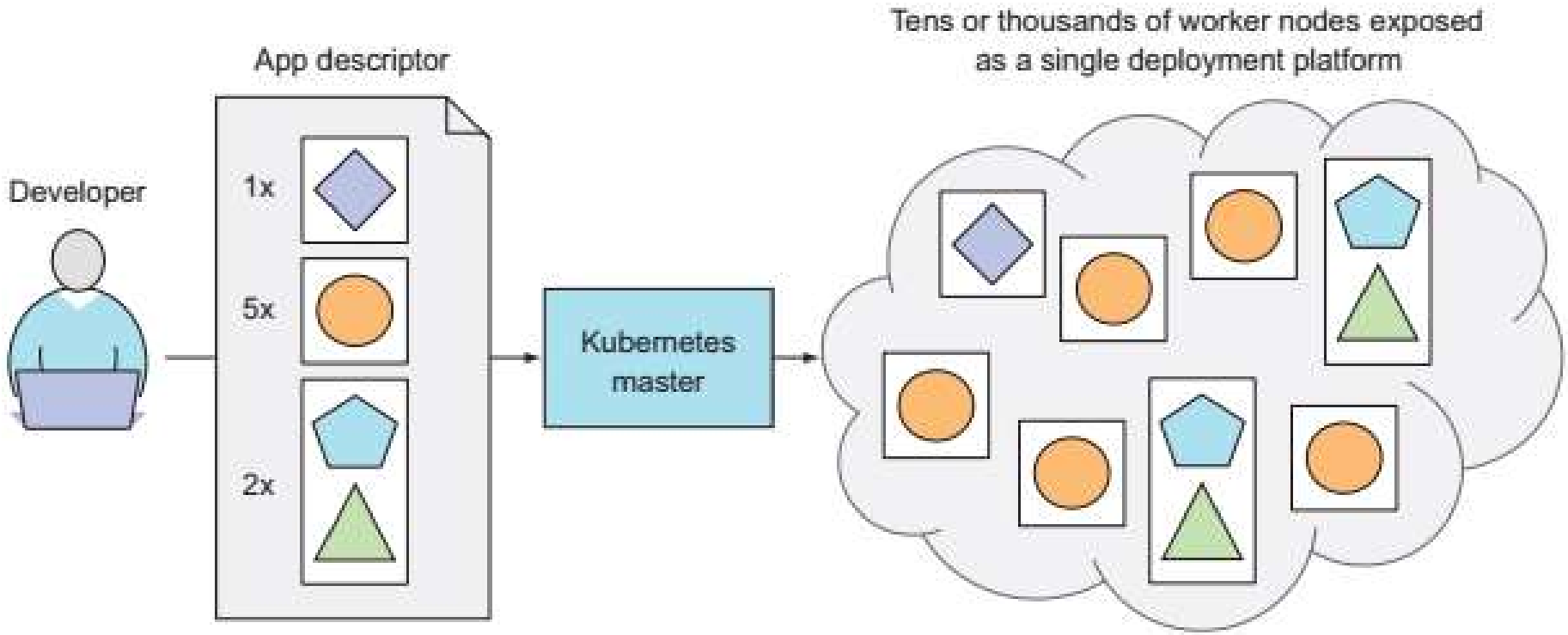


Figure 1.8 Kubernetes exposes the whole datacenter as a single deployment platform.

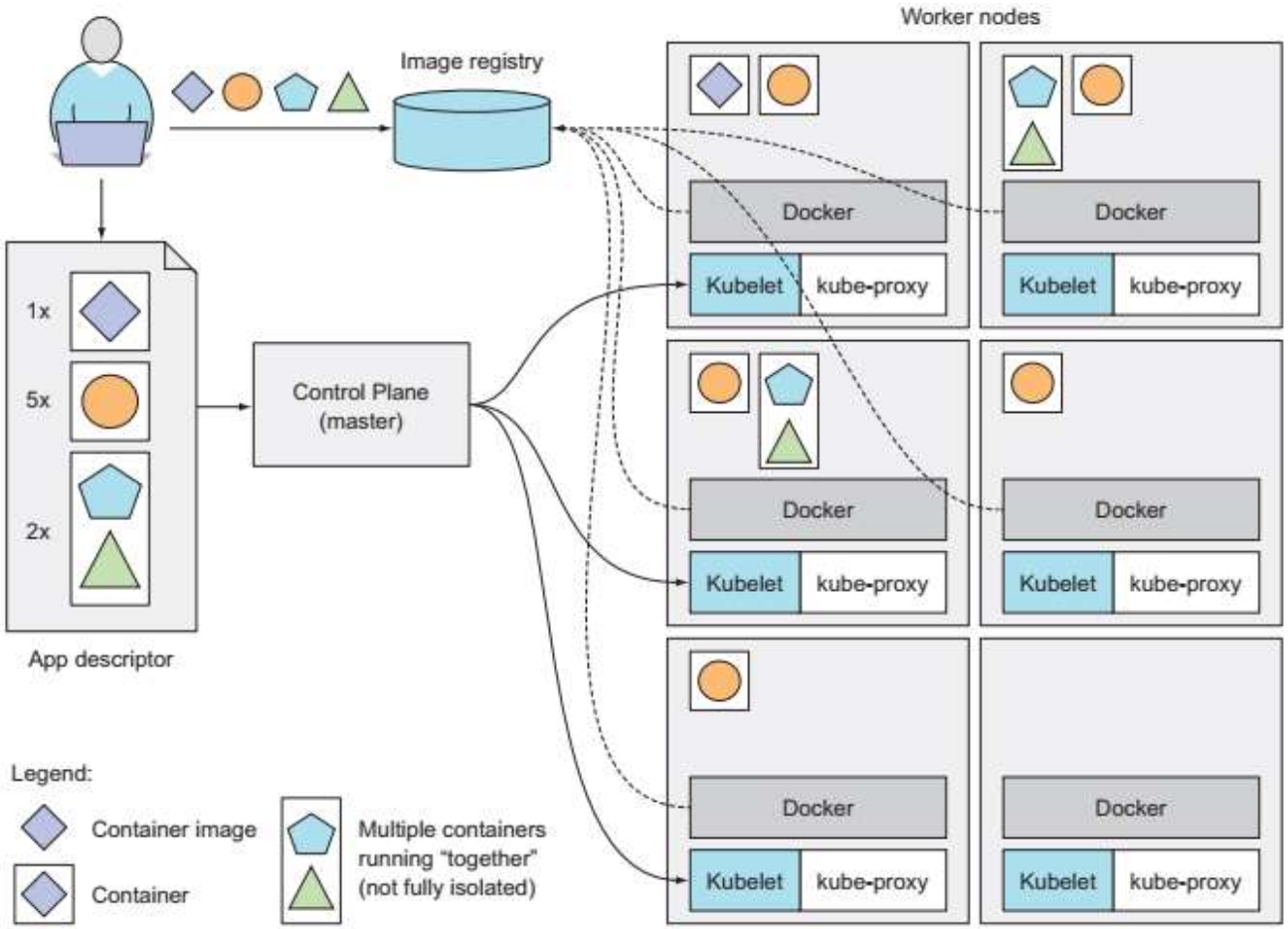


Figure 1.10 A basic overview of the Kubernetes architecture and an application running on top of it

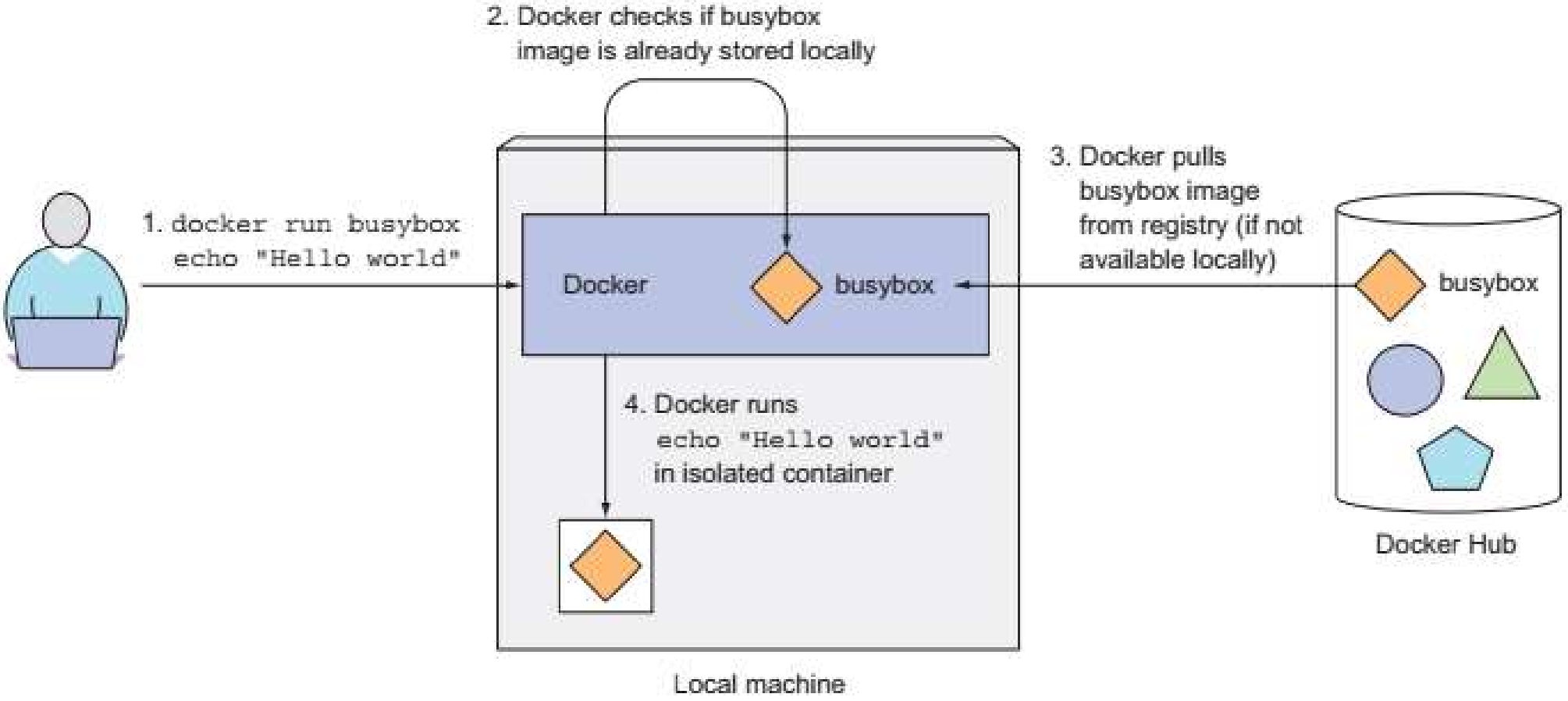


Figure 2.1 Running echo "Hello world" in a container based on the busybox container image

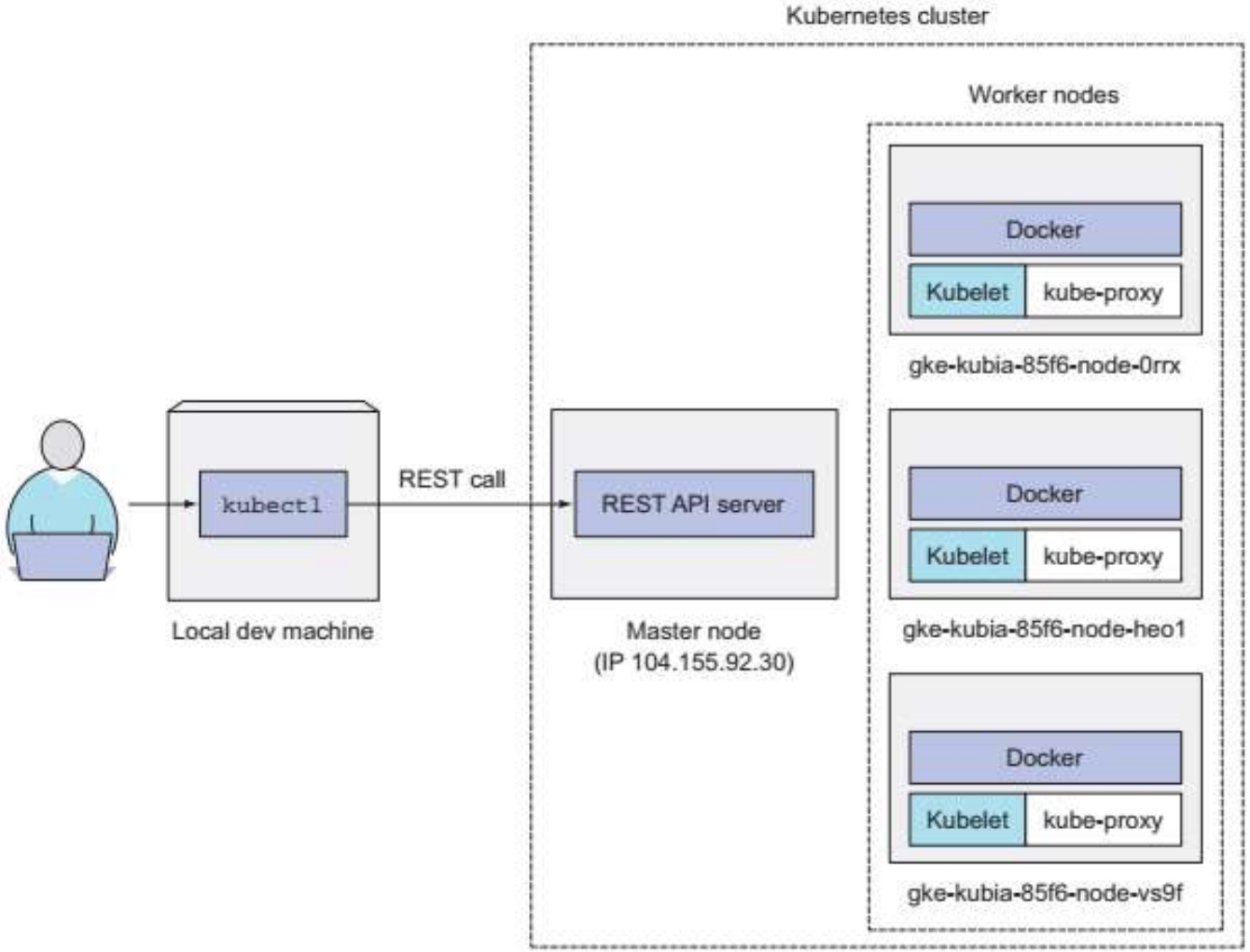


Figure 2.4 How you're interacting with your three-node Kubernetes cluster

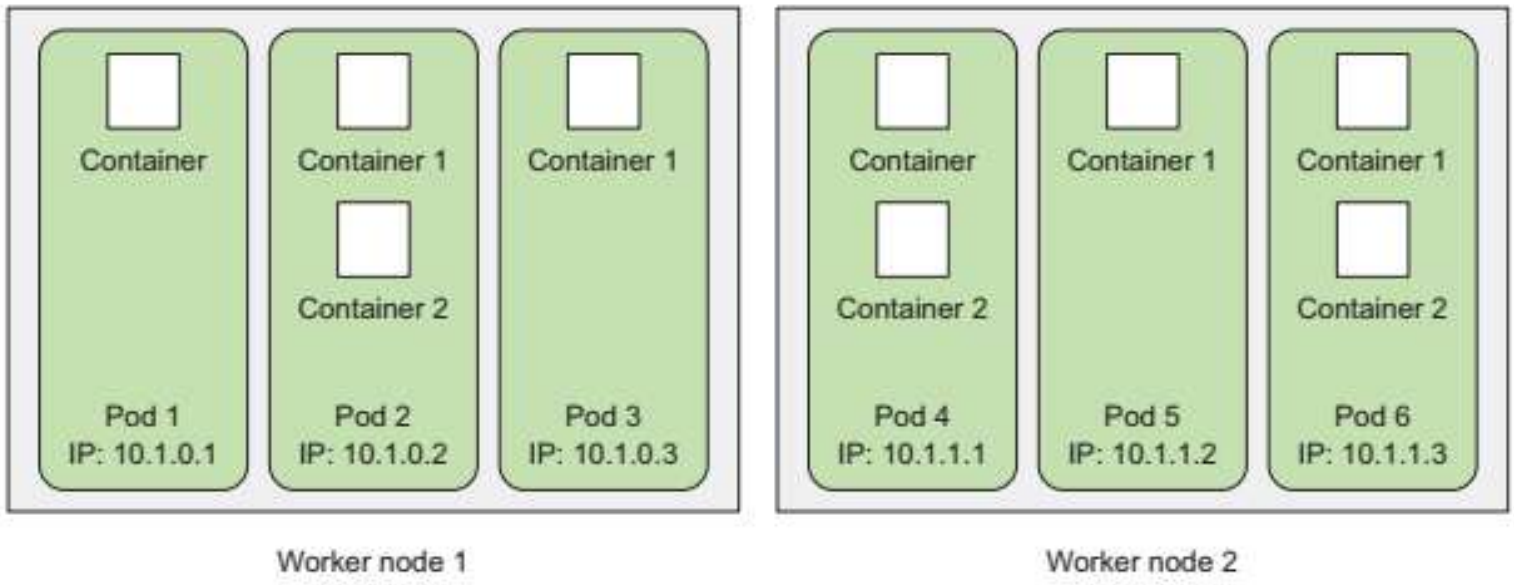


Figure 2.5 The relationship between containers, pods, and physical worker nodes

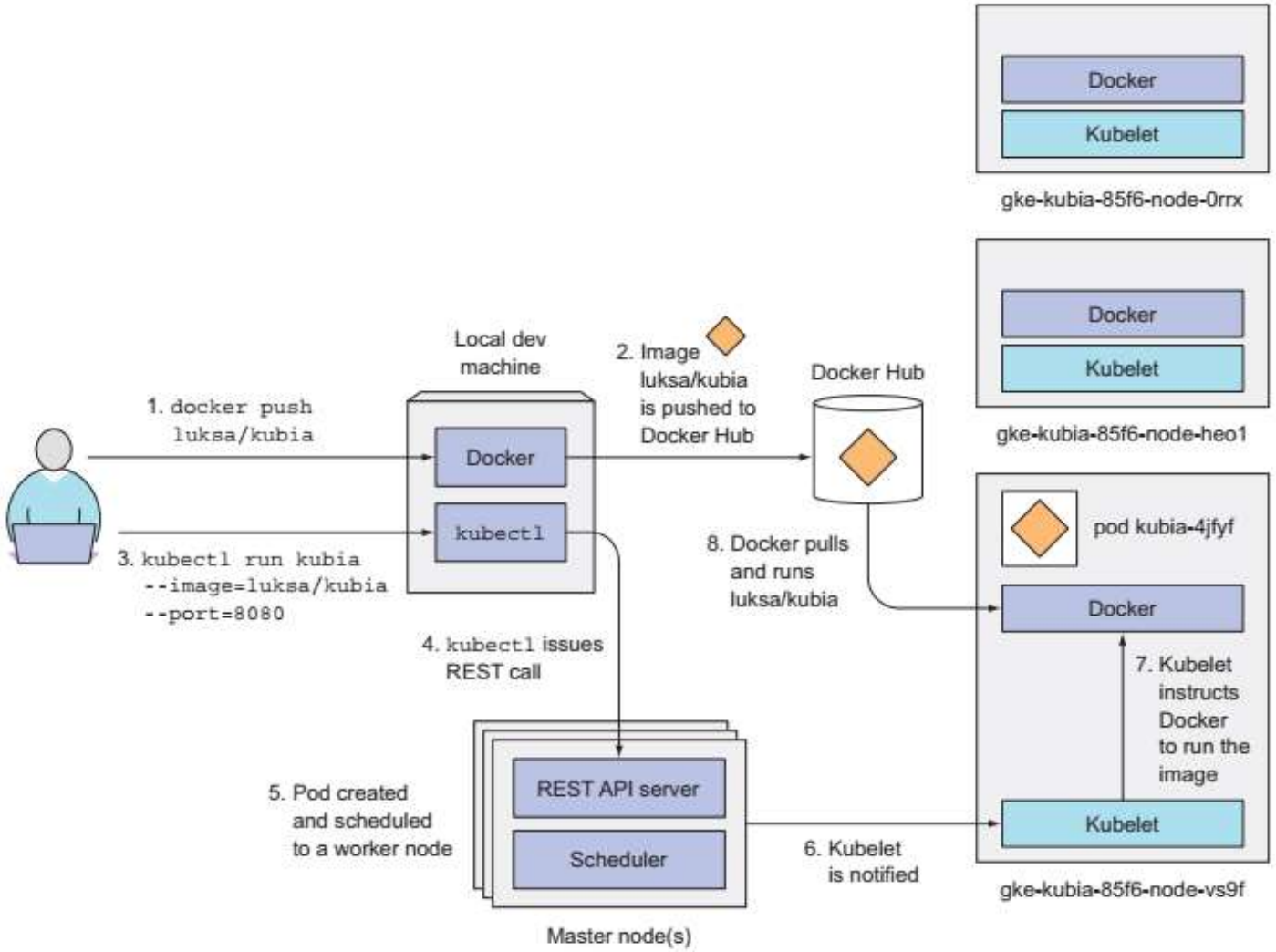


Figure 2.6 Running the `luksa/kubia` container image in Kubernetes

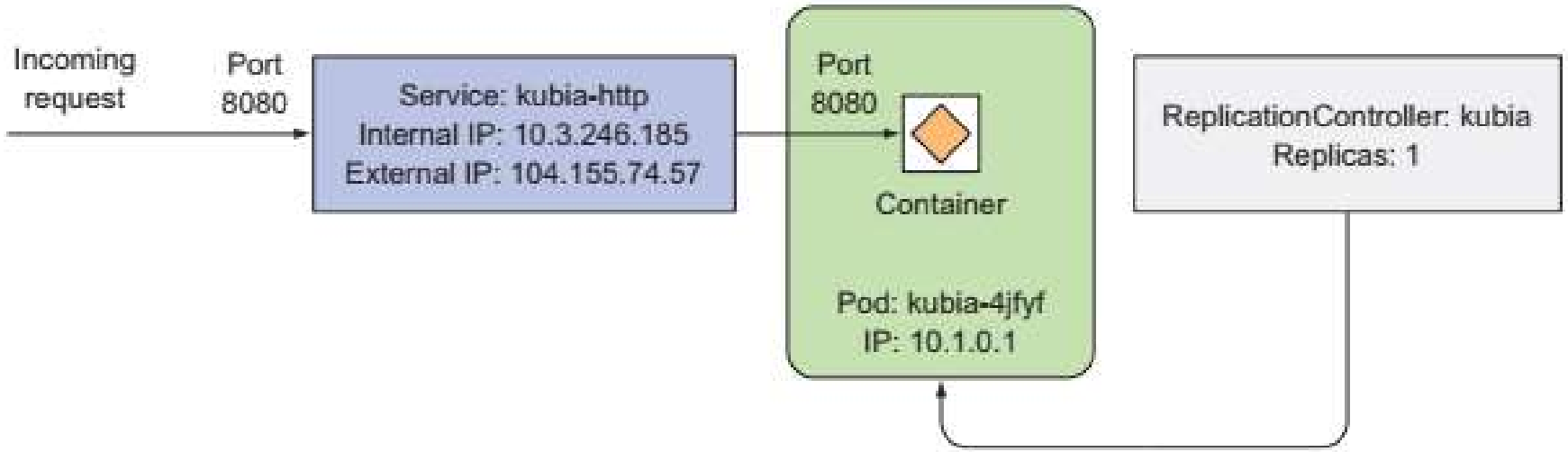


Figure 2.7 Your system consists of a ReplicationController, a Pod, and a Service.

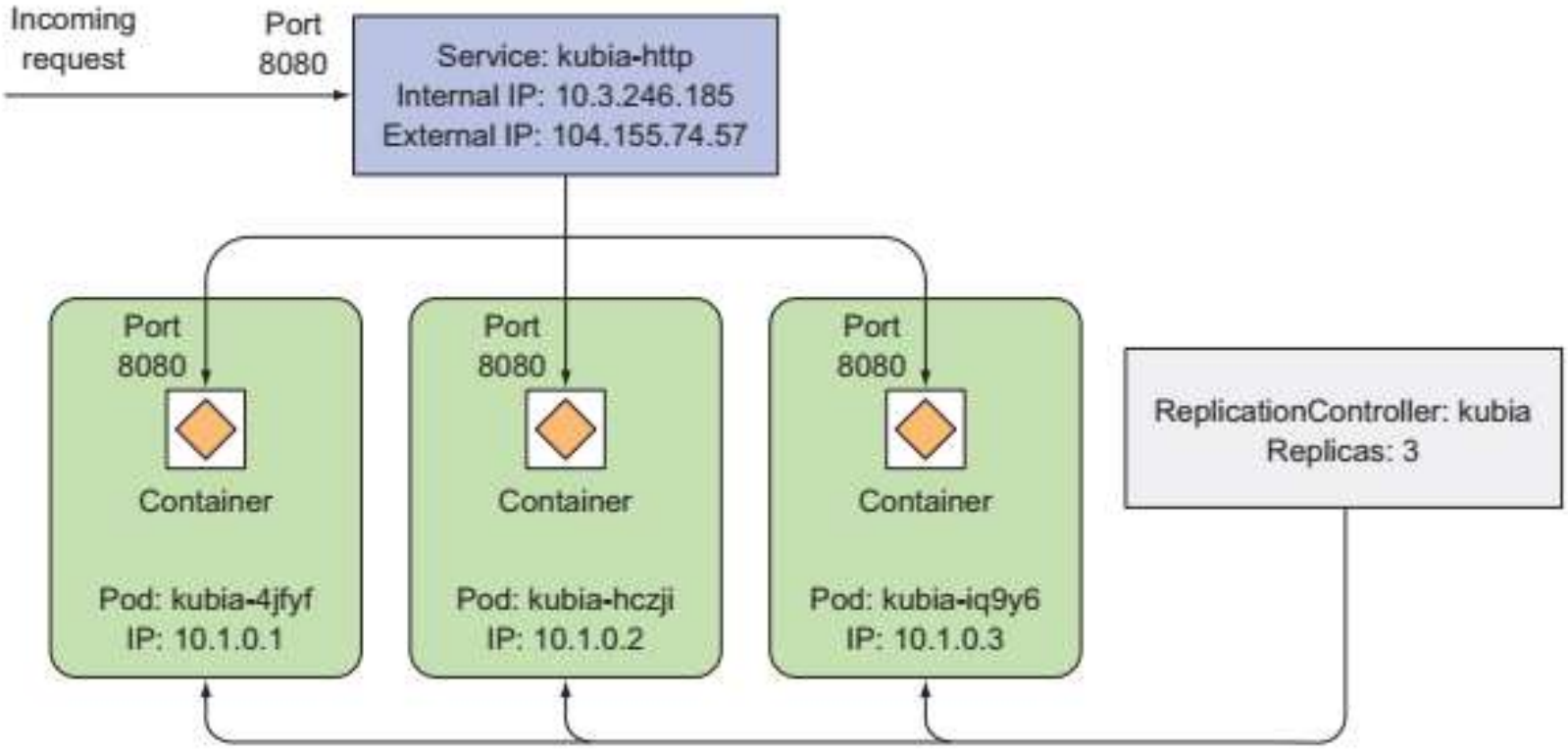


Figure 2.8 Three instances of a pod managed by the same ReplicationController and exposed through a single service IP and port.

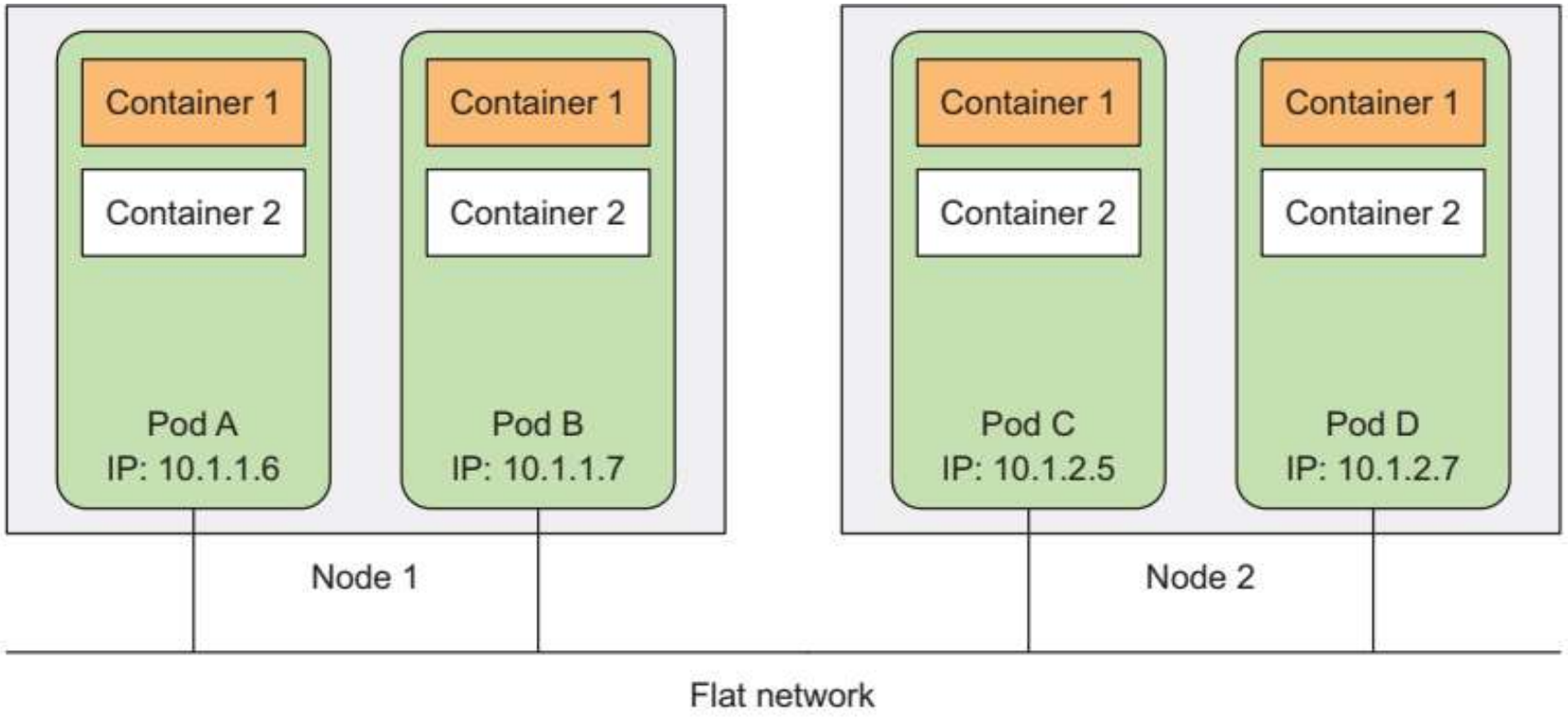
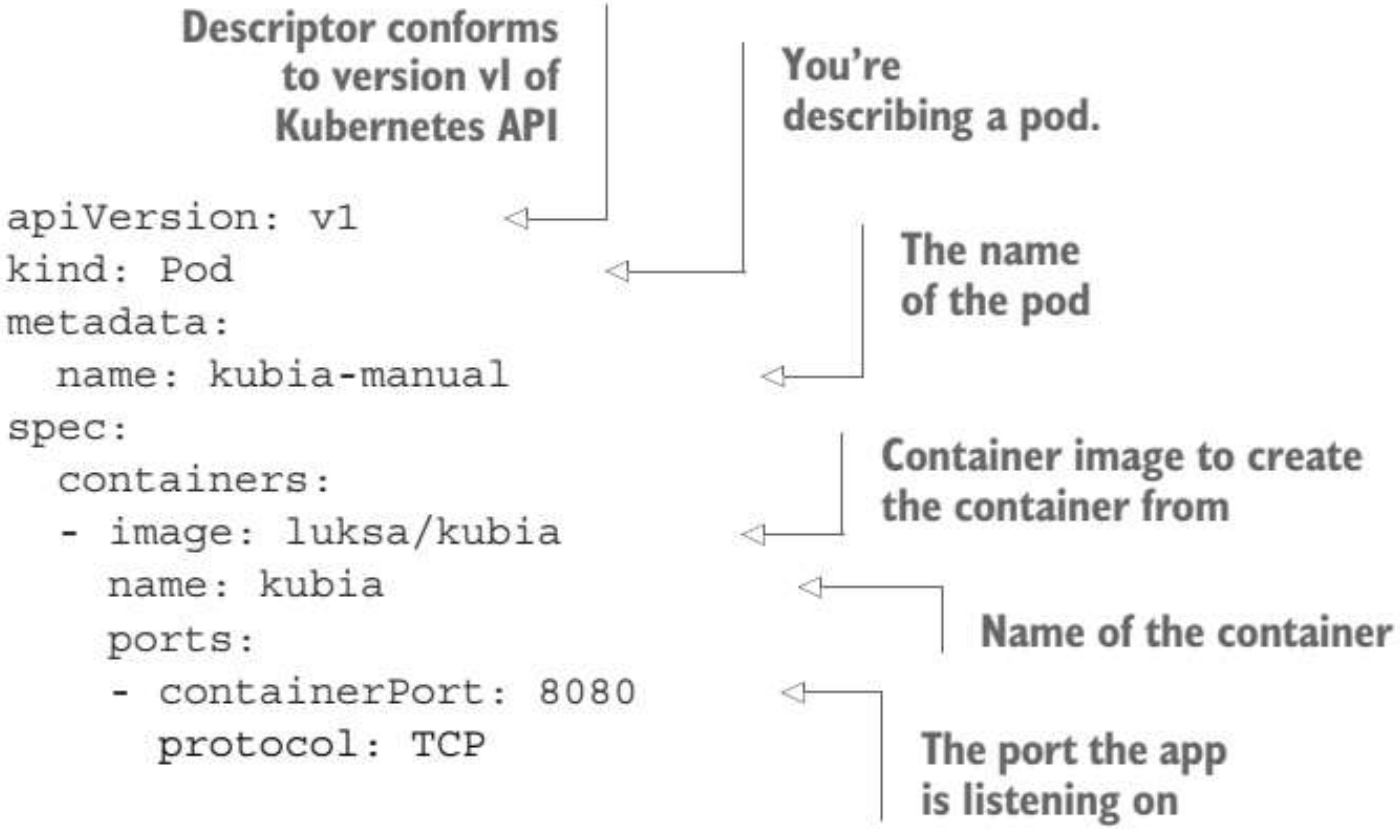


Figure 3.2 Each pod gets a routable IP address and all other pods see the pod under that IP address.

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




```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-manual
spec:
  containers:
  - image: luksa/kubia
    name: kubia
    ports:
    - containerPort: 8080
      protocol: TCP
```

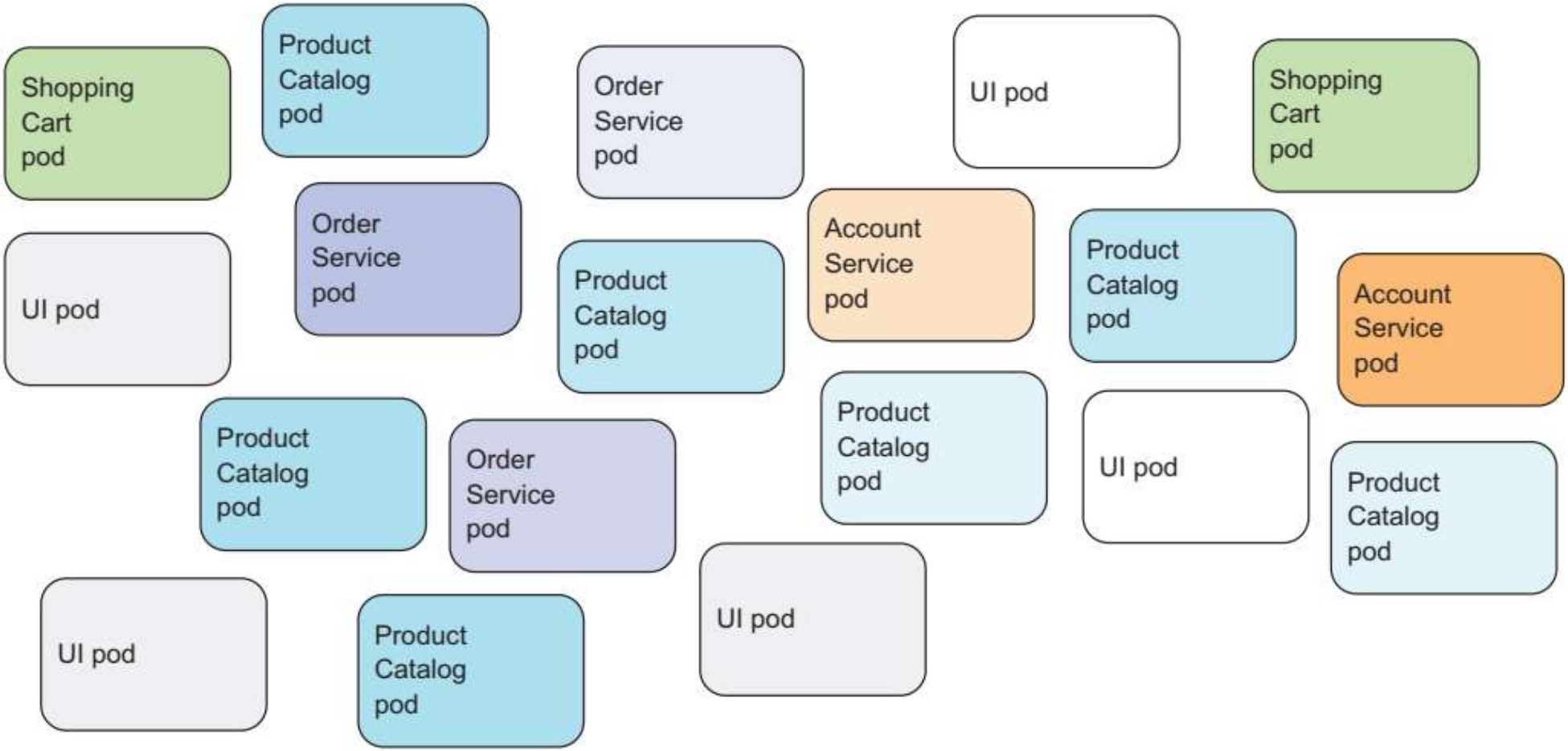


Figure 3.6 Uncategorized pods in a microservices architecture

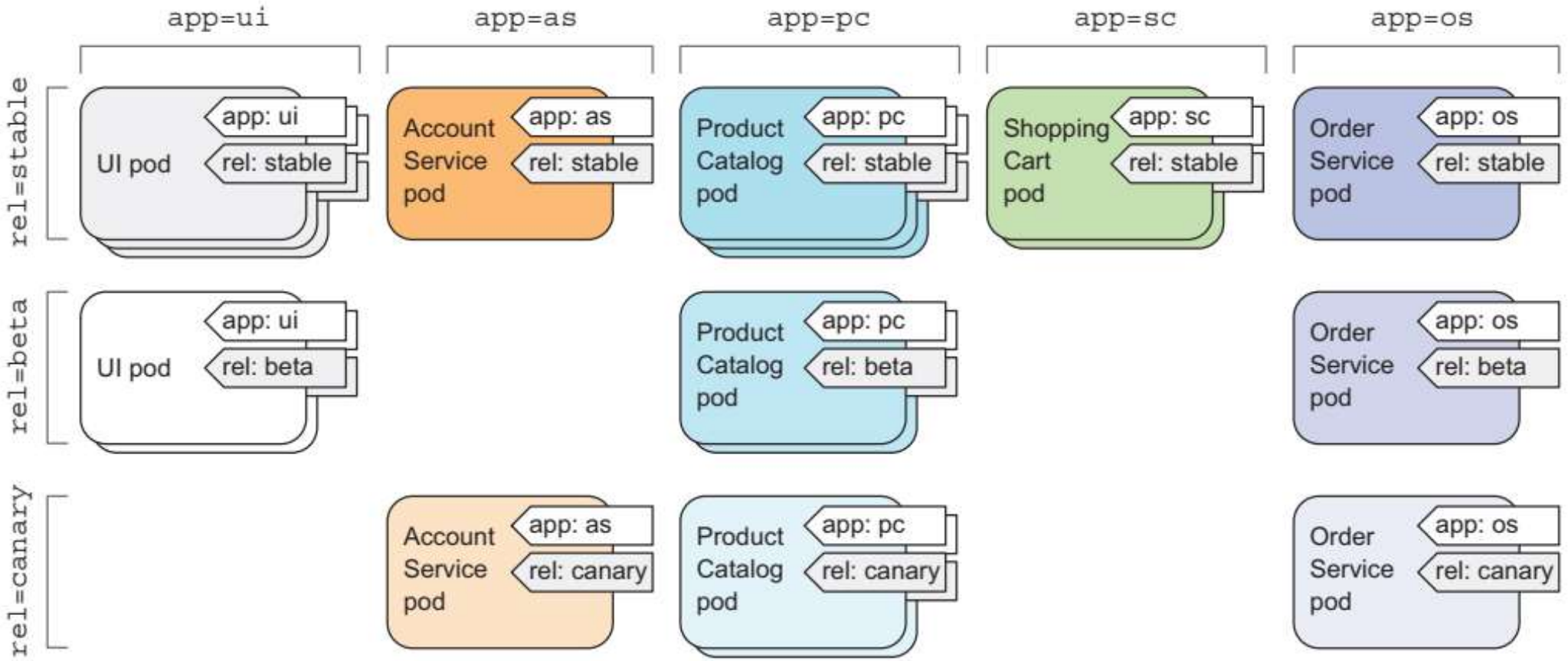


Figure 3.7 Organizing pods in a microservices architecture with pod labels

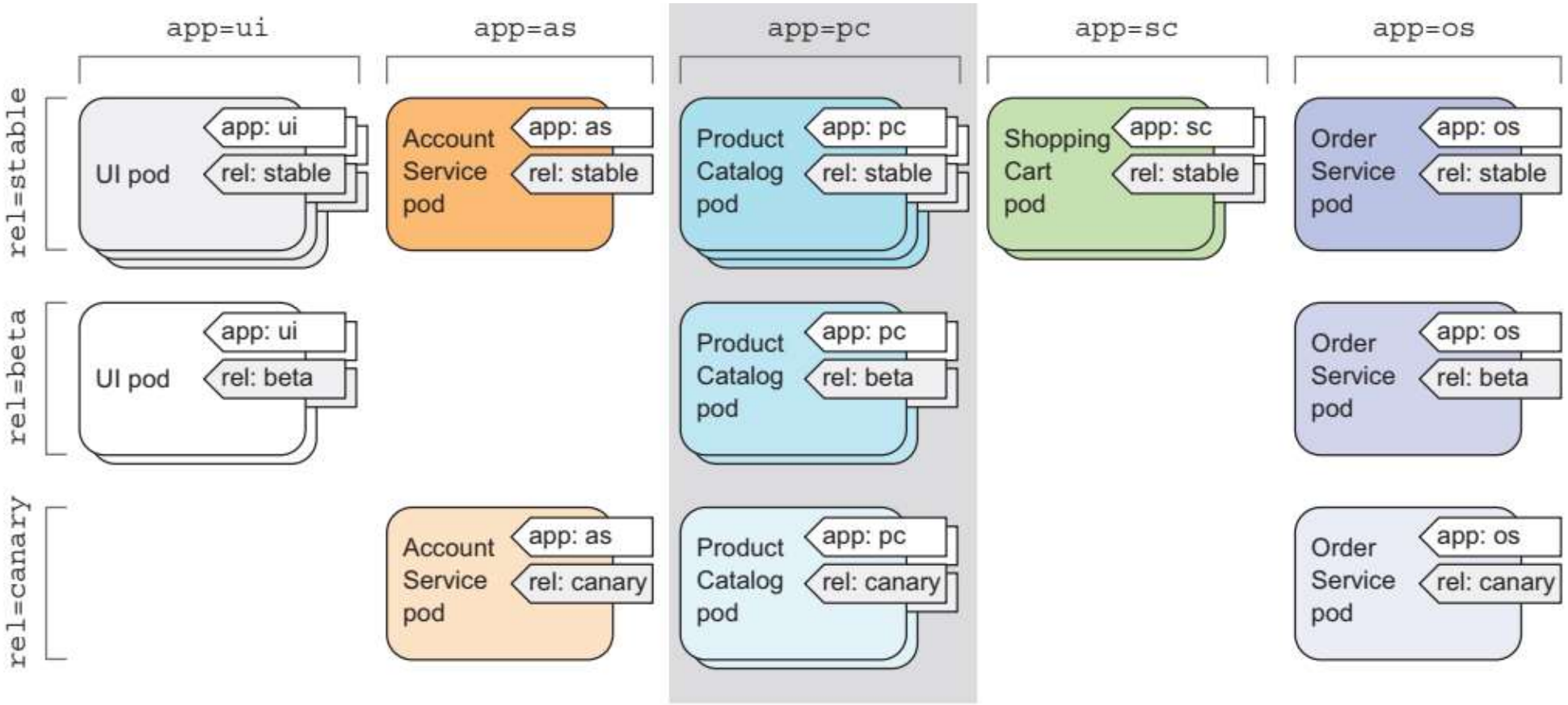


Figure 3.8 Selecting the product catalog microservice pods using the "app=pc" label selector

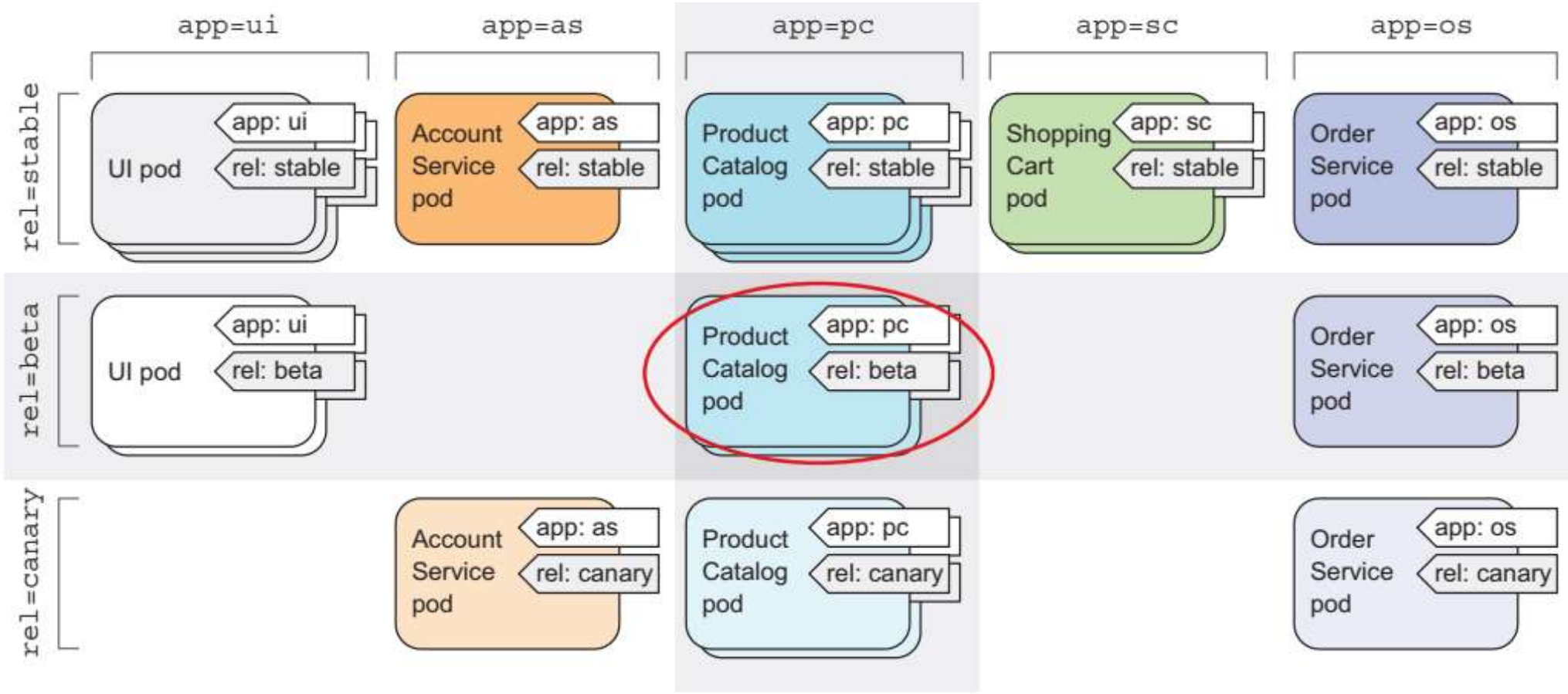


Figure 3.9 Selecting pods with multiple label selectors

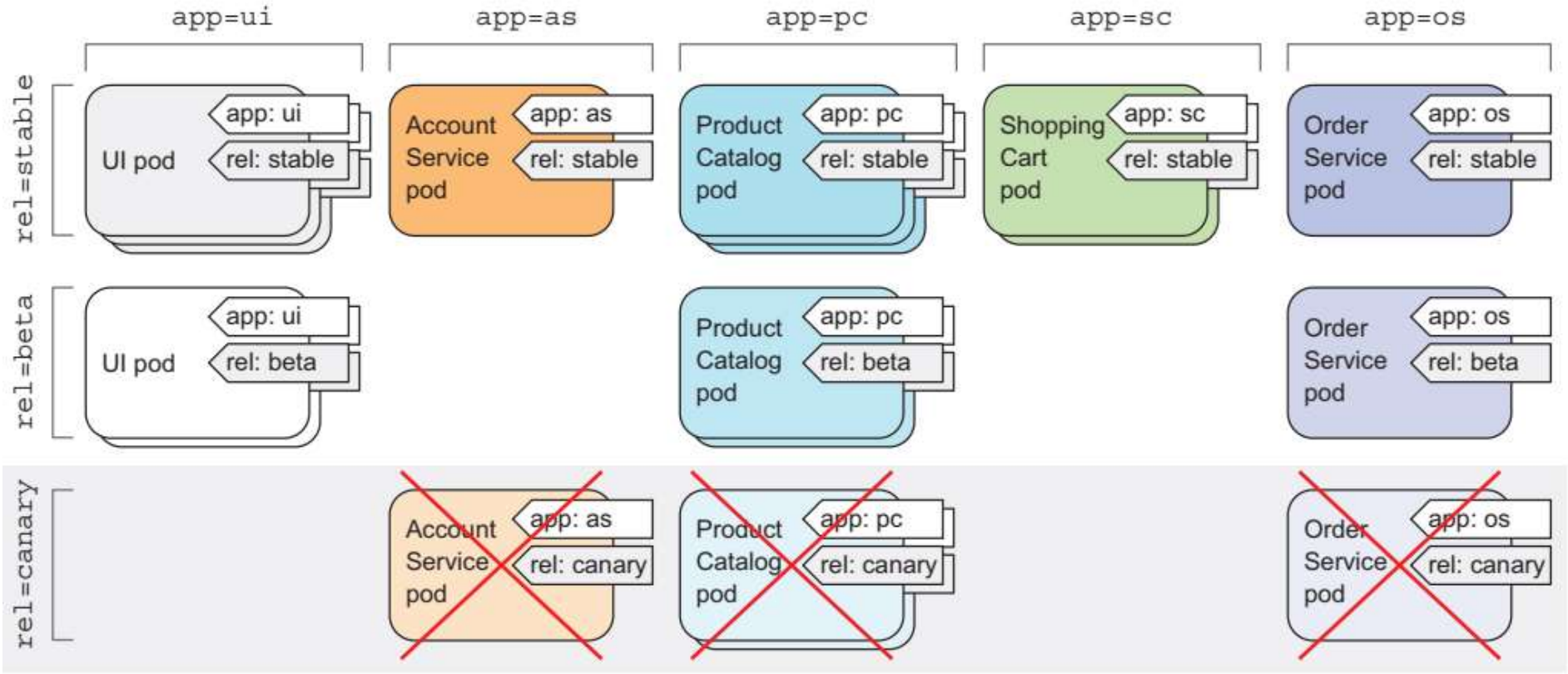


Figure 3.10 Selecting and deleting all canary pods through the `rel=canary` label selector

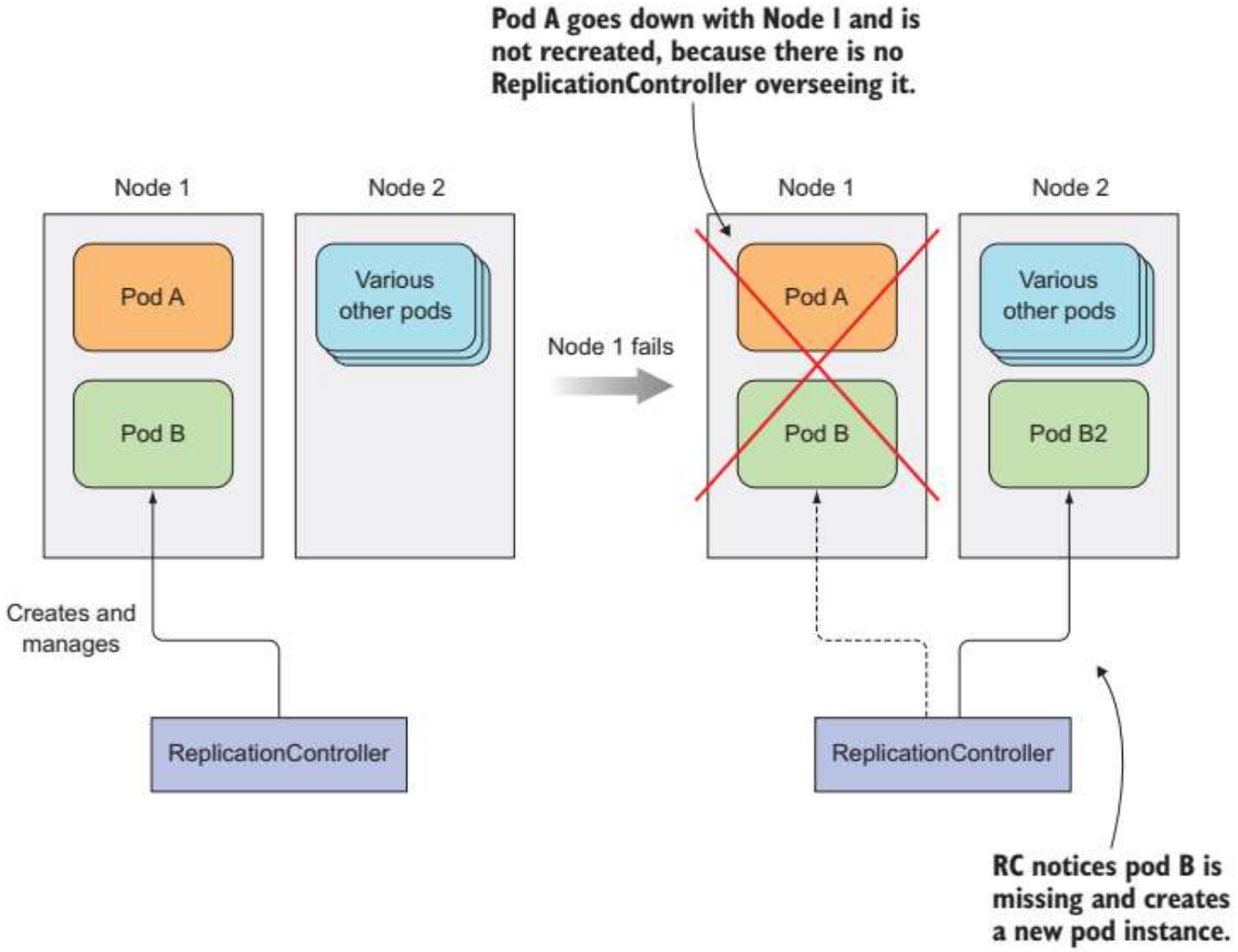


Figure 4.1 When a node fails, only pods backed by a ReplicationController are recreated.

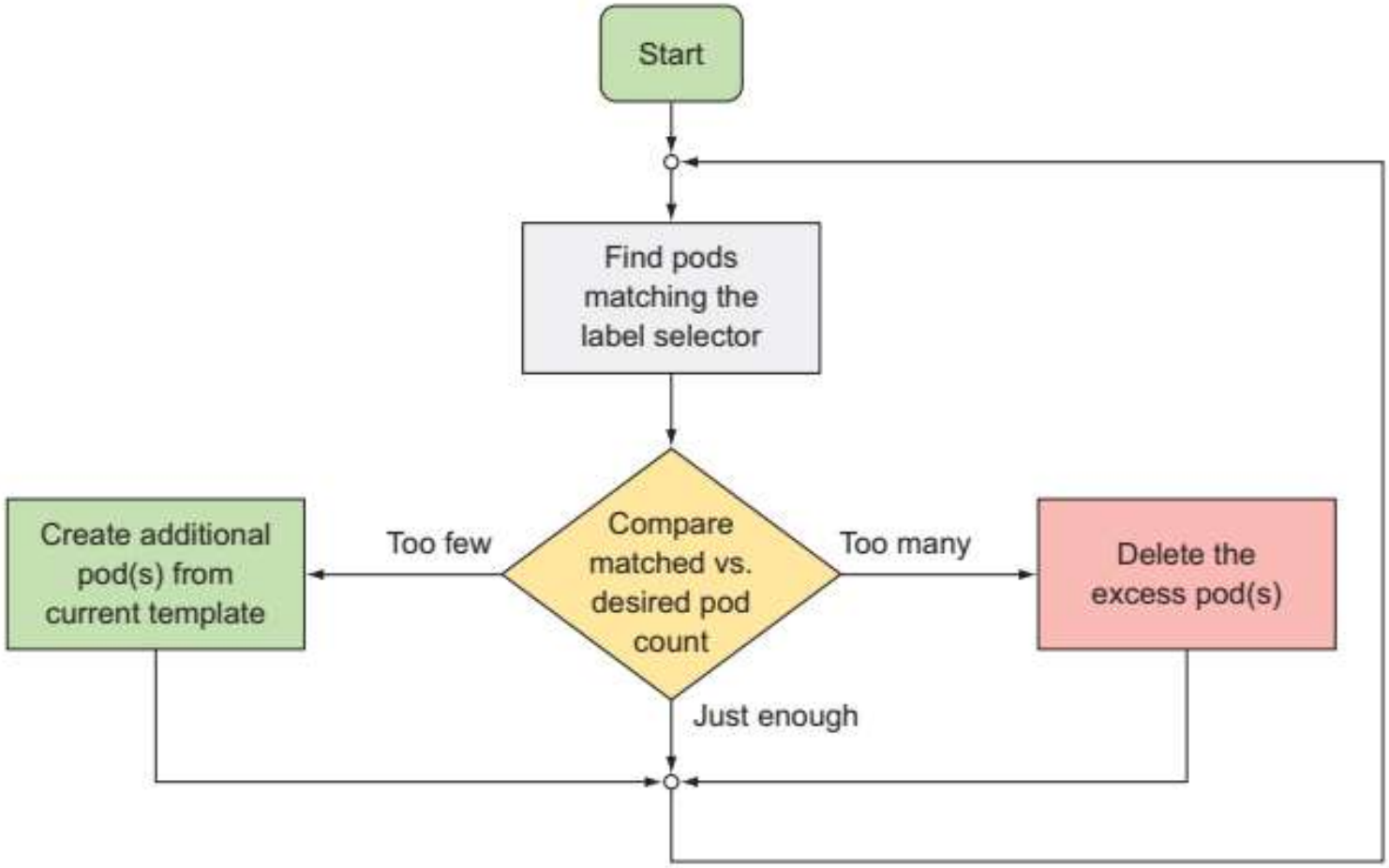


Figure 4.2 A ReplicationController's reconciliation loop

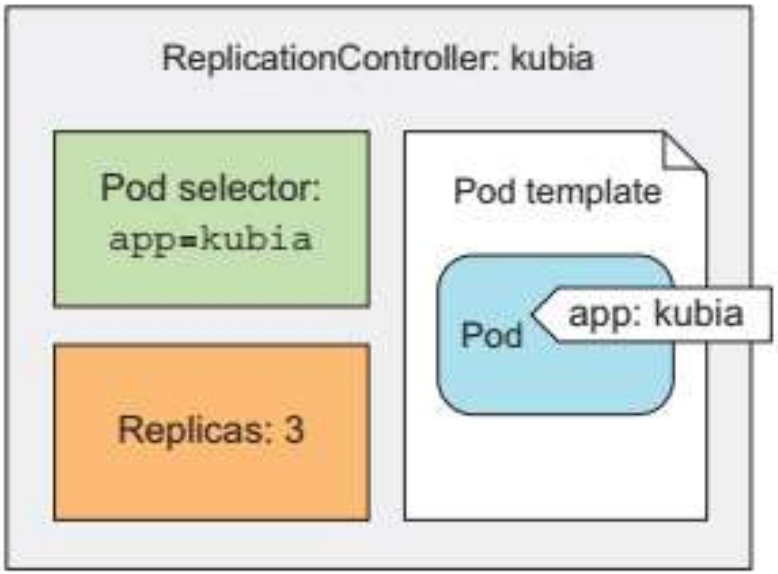


Figure 4.3 The three key parts of a ReplicationController (pod selector, replica count, and pod template)

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

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

The pod template for creating new pods

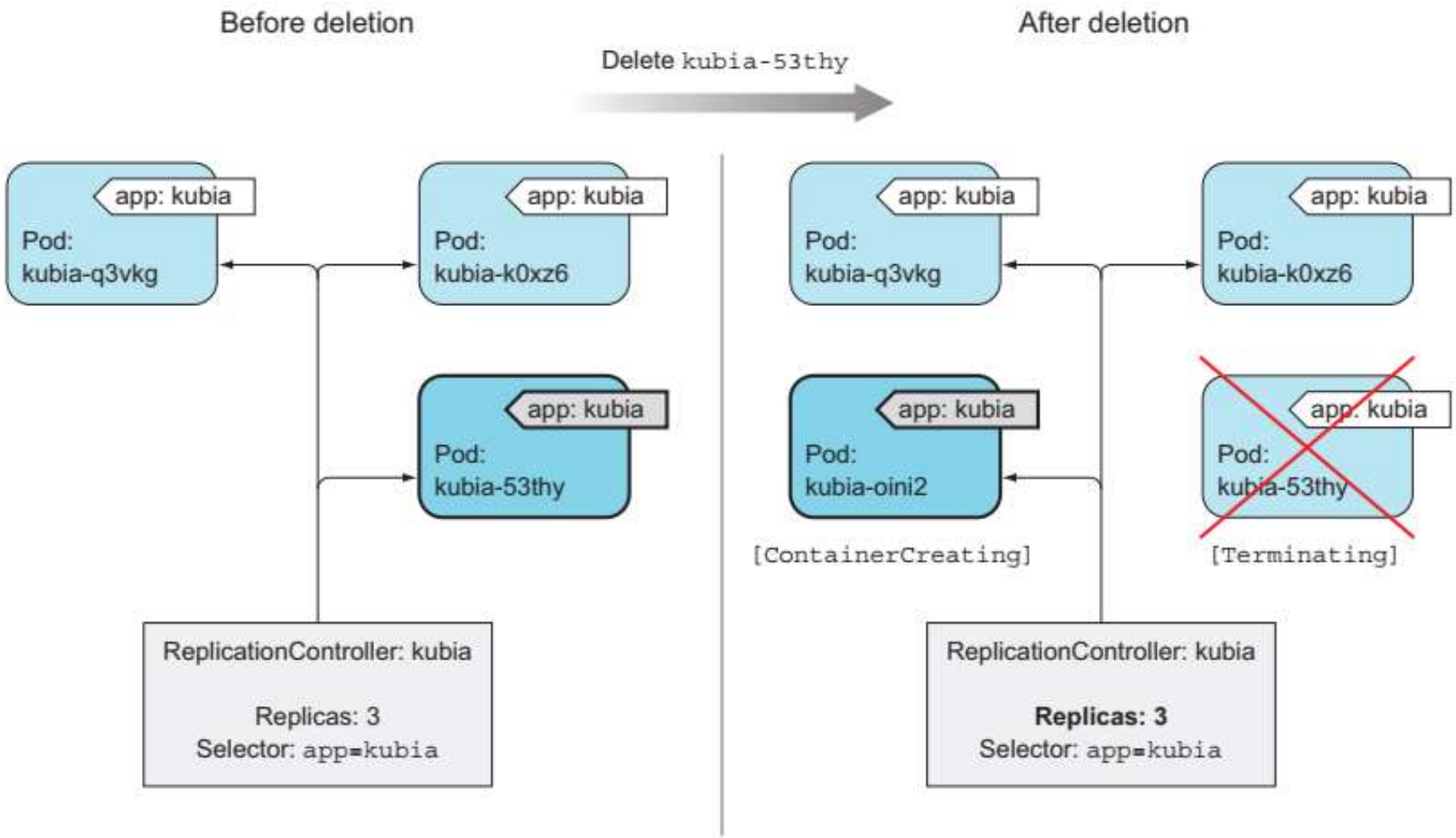


Figure 4.4 If a pod disappears, the ReplicationController sees too few pods and creates a new replacement pod.

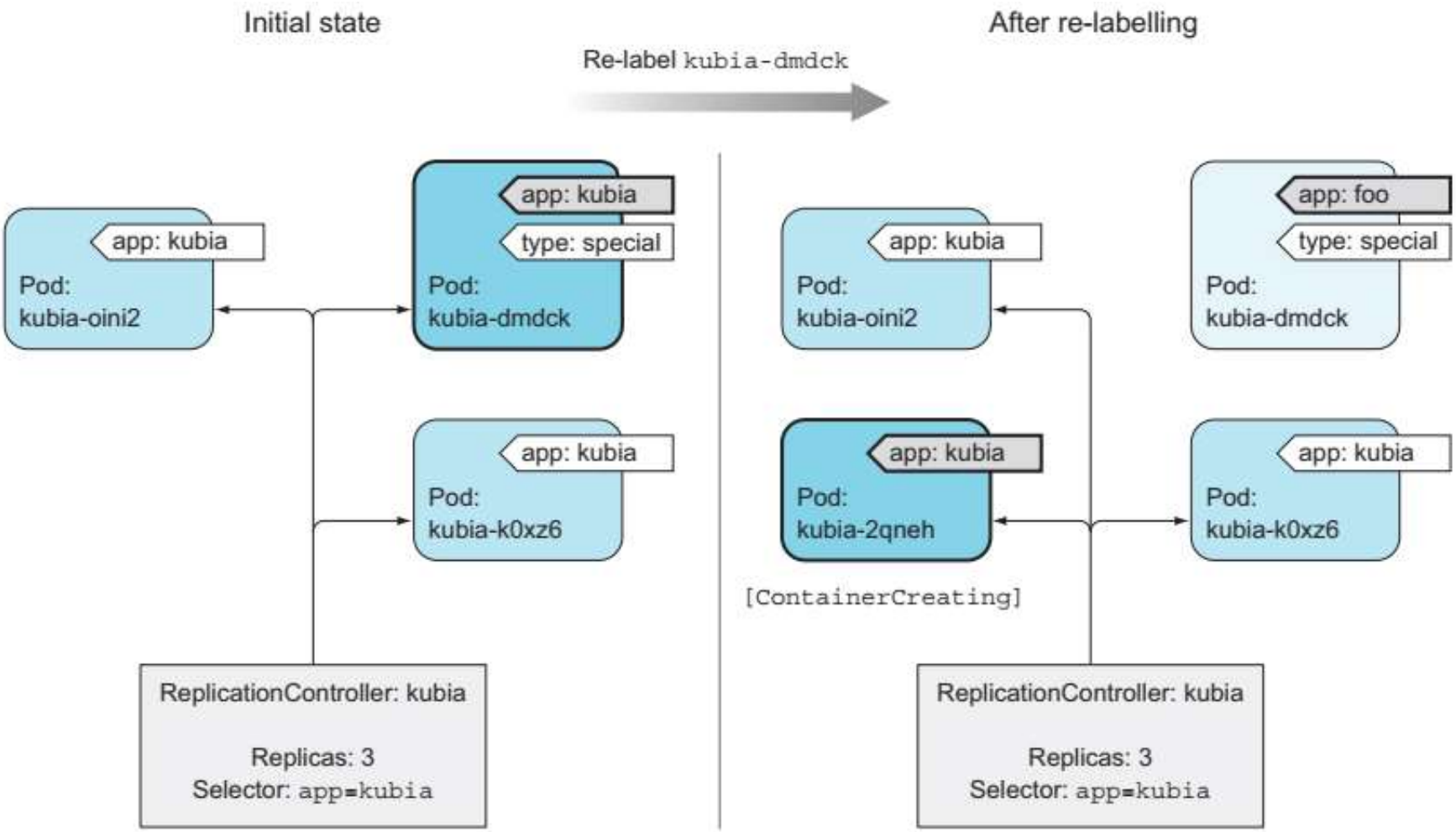


Figure 4.5 Removing a pod from the scope of a ReplicationController by changing its labels

Services

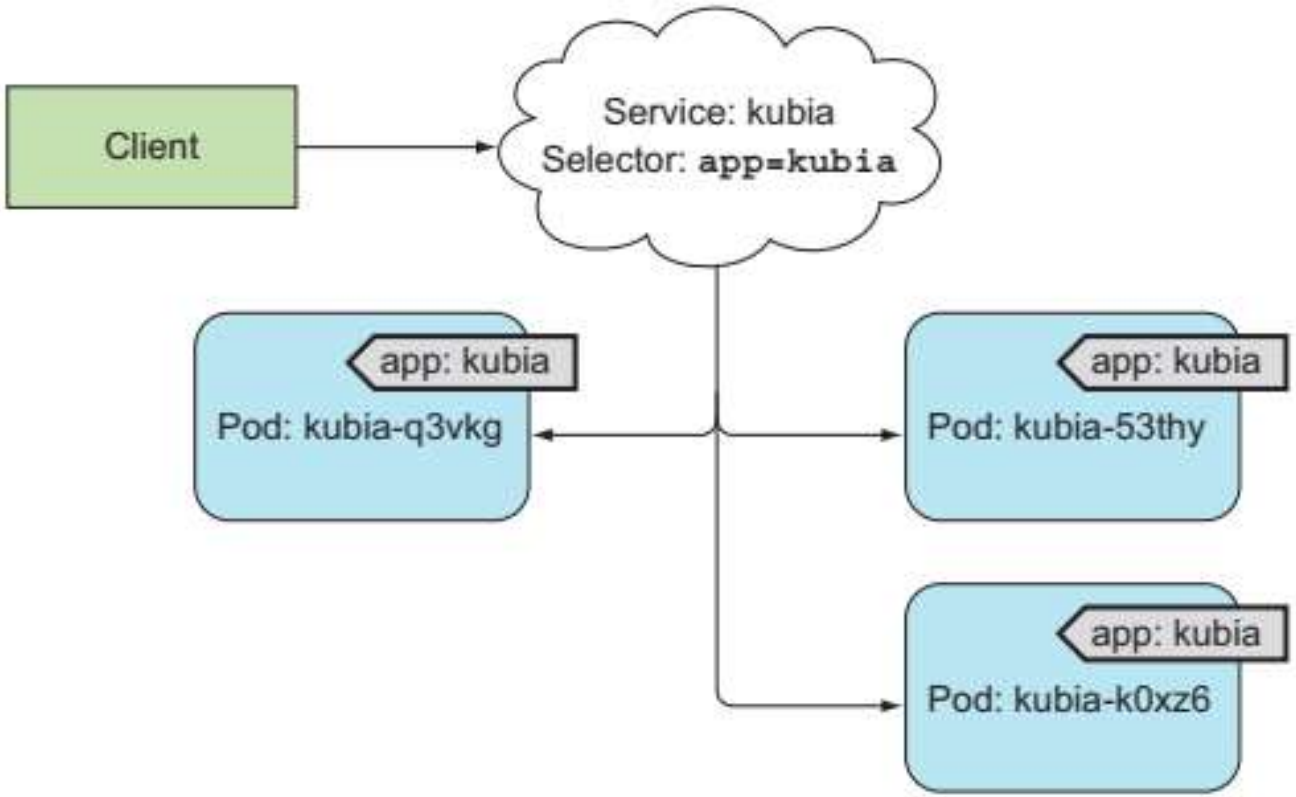


Figure 5.2 Label selectors determine which pods belong to the Service.

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.

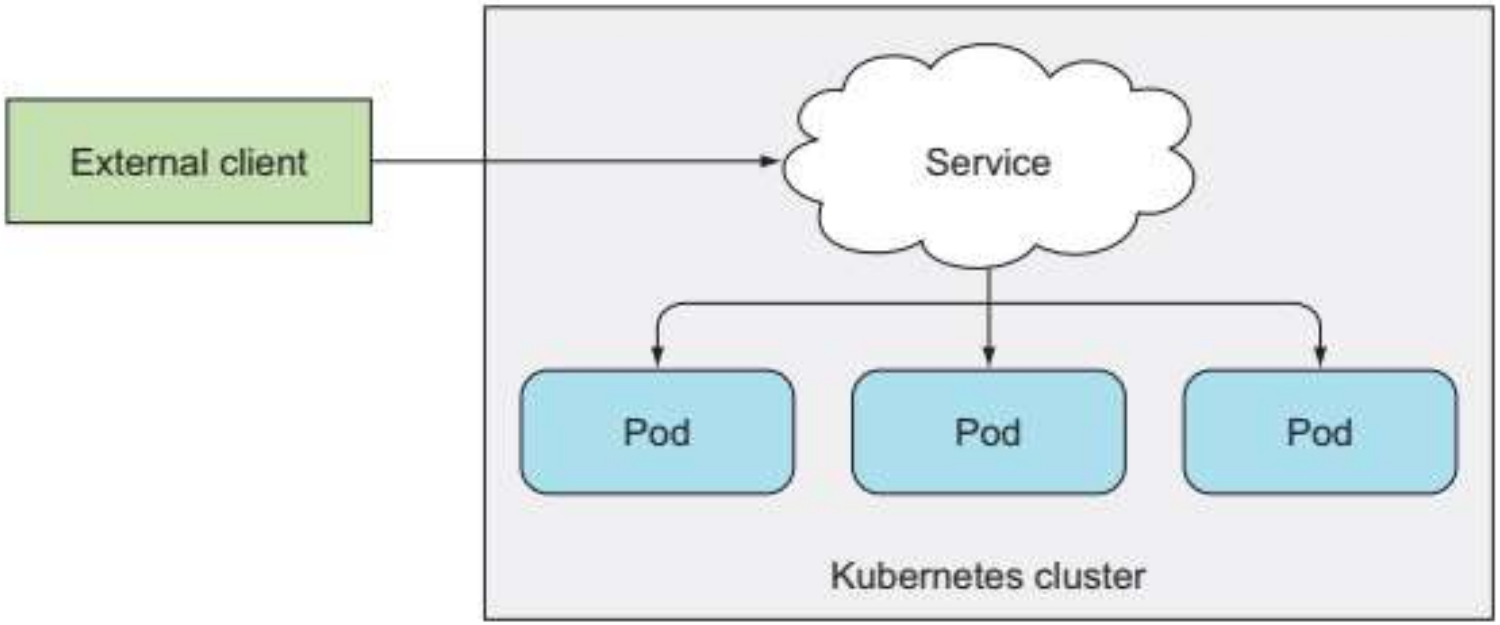
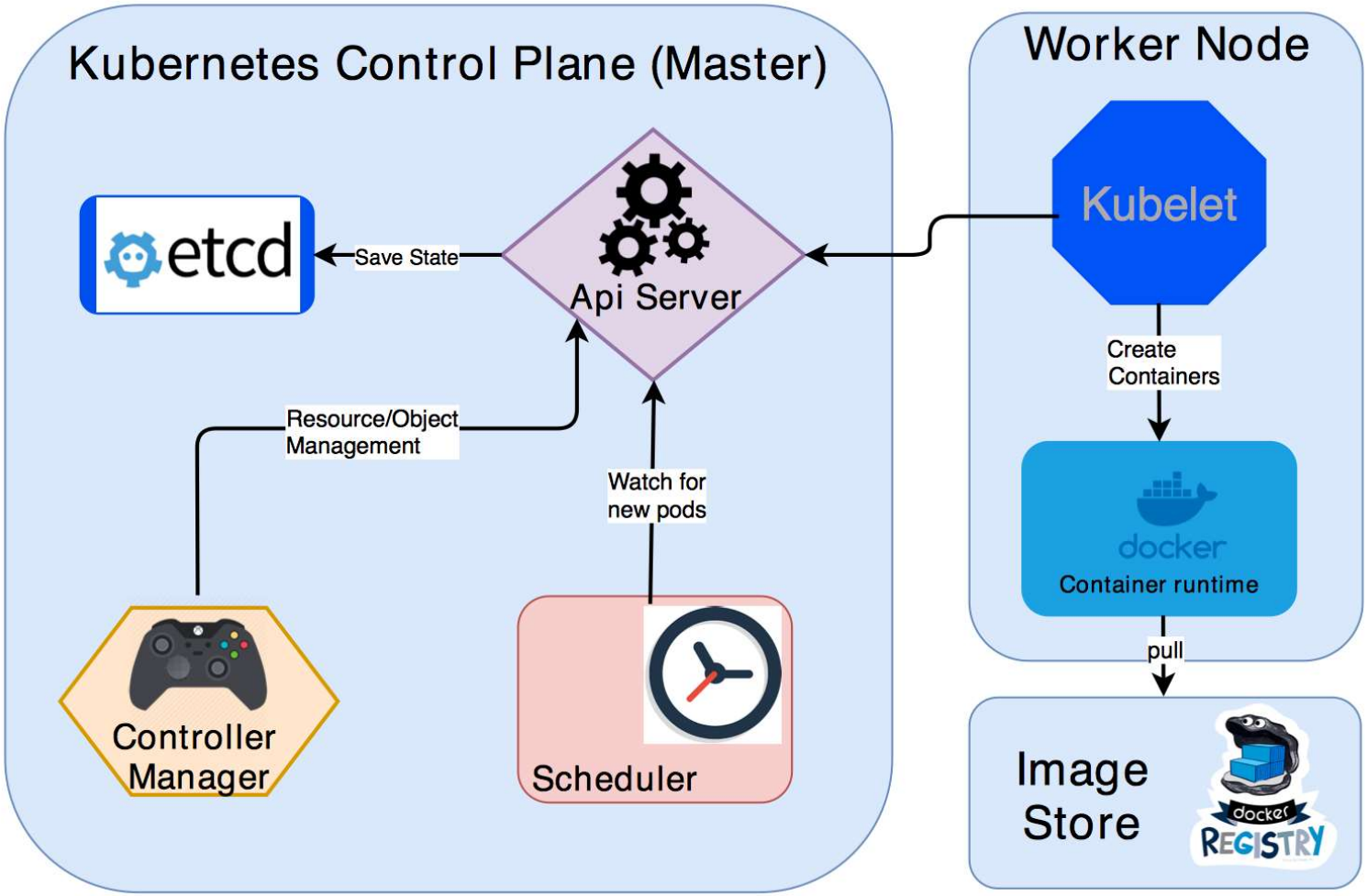


Figure 5.5 Exposing a service to external clients

Kubernetes Architecture

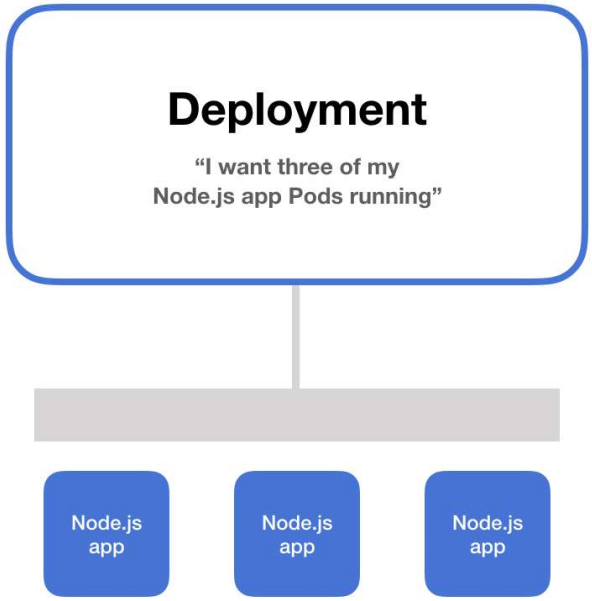


Kubernetes namespaces

- 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.

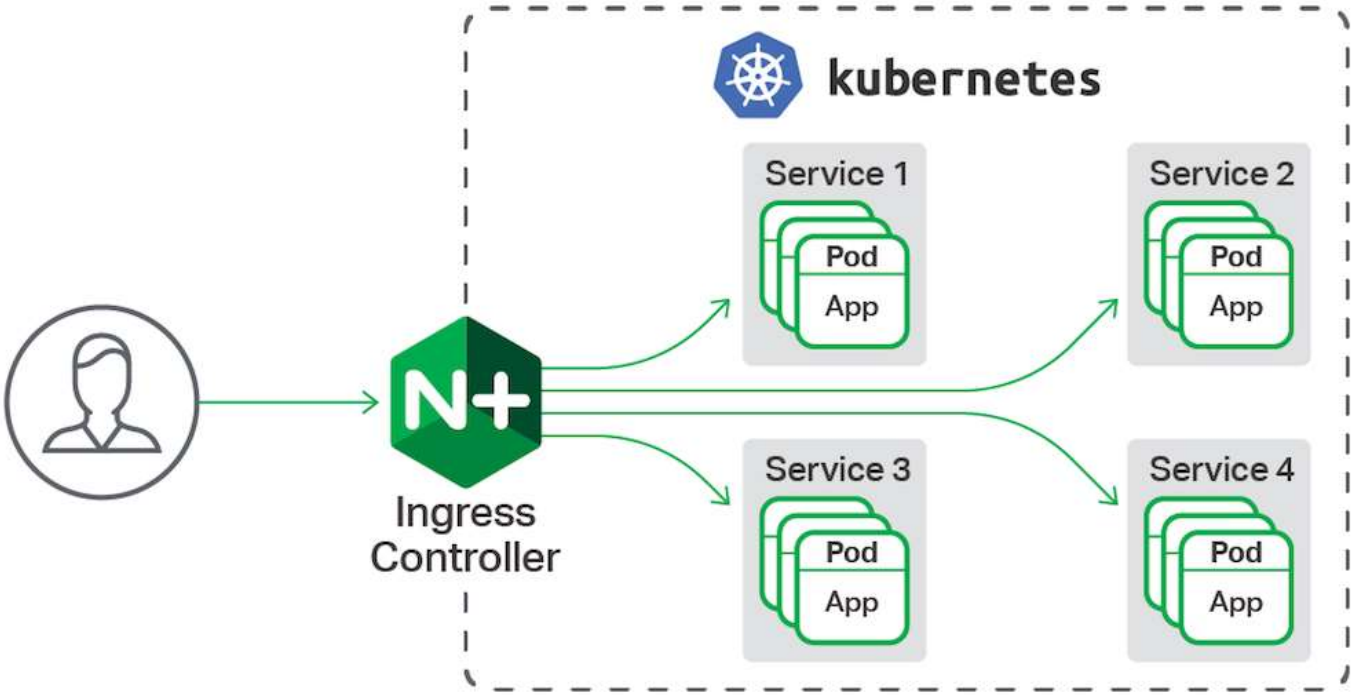


Service and Ingress

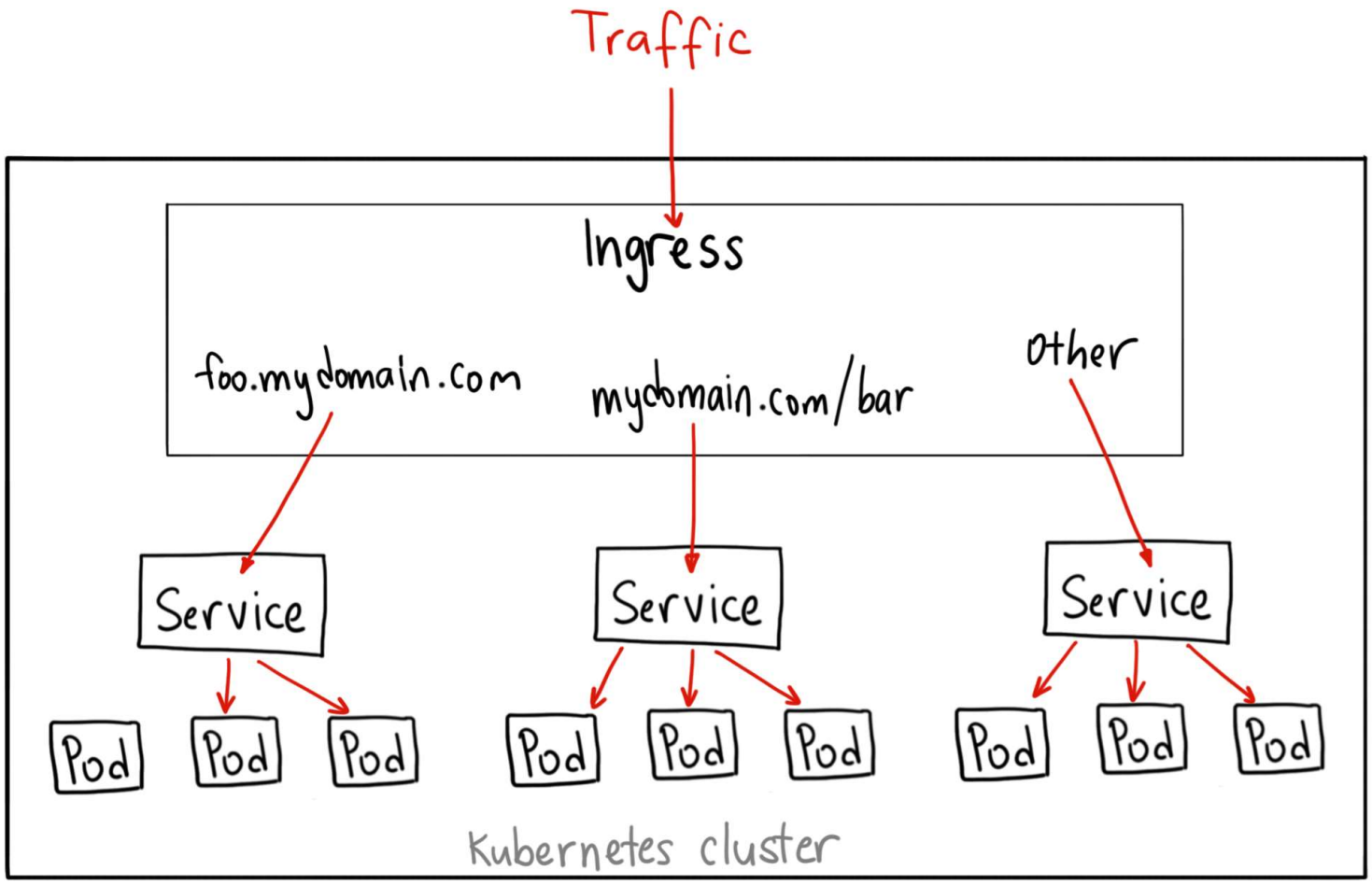
- To consume your deployment, you will need to create ingress rules that expose your deployment to the external world.
- Kubernetes Ingress is a resource to add rules for routing traffic from external sources to the services in the kubernetes cluster

Service and Ingress

- To confirm:
  - kubectl get pods -n ingress-nginx



Service and Ingress



Thanks