Kubernetes 101

A Cluster Operating System

Mikaël Barbero

EclipseCon France — June 13, 2018

Who is familiar with Linux containers technology (like Docker)?

What is Kubernetes?

- Kubernetes is a container management/orchestration system
- It runs and manages containerized applications on a cluster
- What does that really mean?

Kubernetes, basic features

- Start 5 containers using image atseashop/api:v1.3
  - Place an internal load balancer in front of these containers
- Start 10 containers using image atseashop/webfront:v1.3
  - Place a public load balancer in front of these containers
- It's Black Friday (or Christmas), traffic spikes, grow our cluster and add containers
- New release! Replace my containers with the new image atseashop/webfront:v1.4
- Keep processing requests during the upgrade; update my containers one at a time

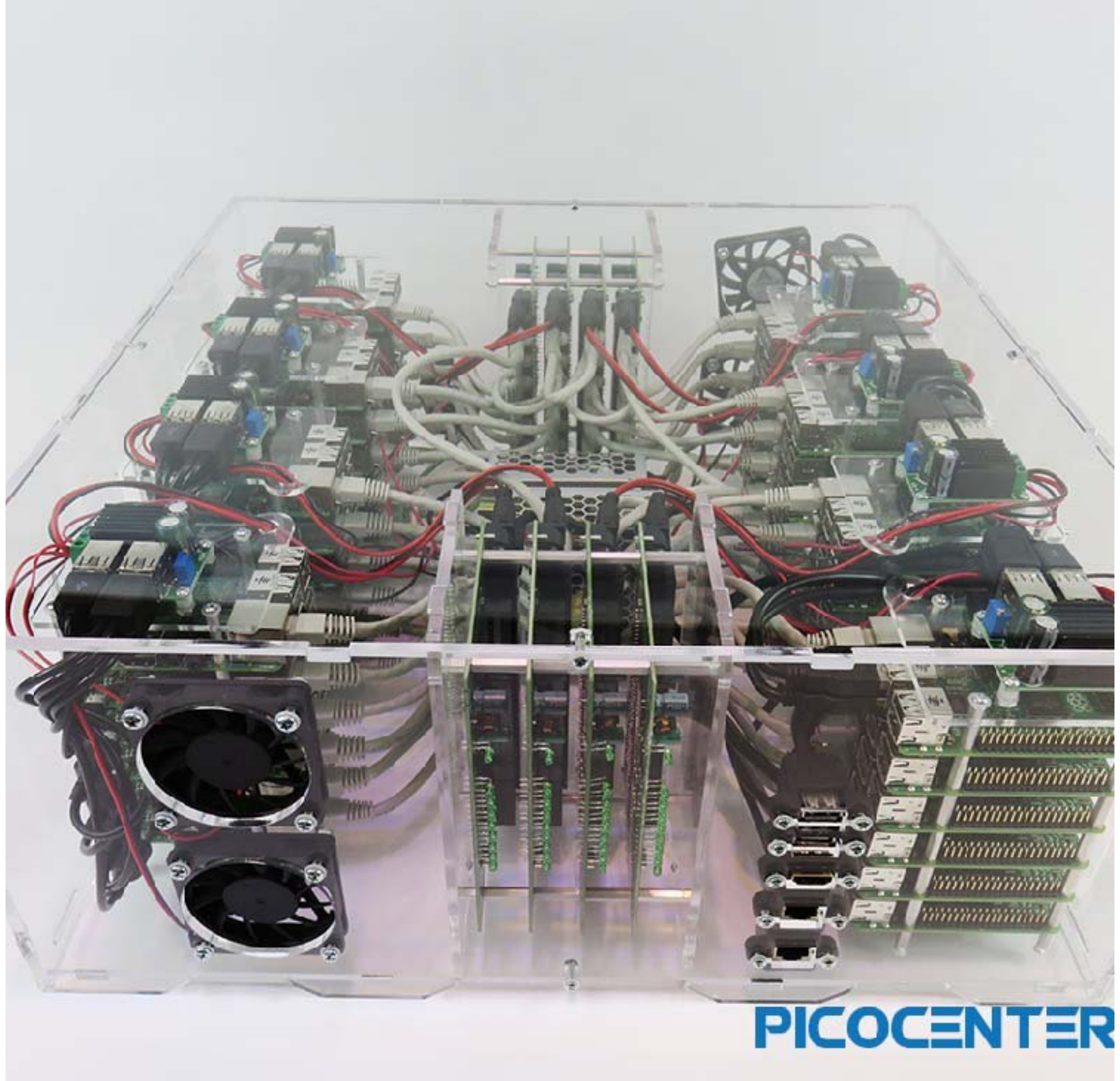
Kubernetes, advanced features

- Autoscaling
- Blue/green deployment, canary deployment
- Long running services, but also batch (one-off) jobs
- Overcommit the cluster and evict low-priority jobs
- Run services with stateful data (databases etc.)
- Fine-grained access control defining what can be done by whom on which resources
- Integrating third party services (service catalog)
- Automating complex tasks (operators)

What is a cluster?

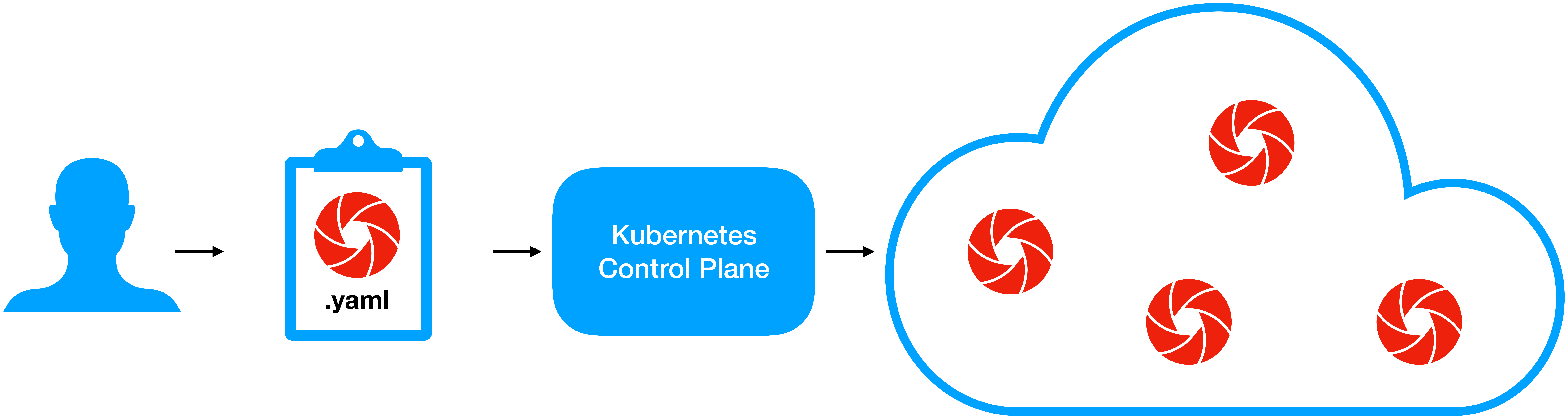
A Kubernetes cluster is a set of nodes. A node is either

- Bare iron machine
- Virtual machine



https://www.picocluster.com/

30,000 foot view

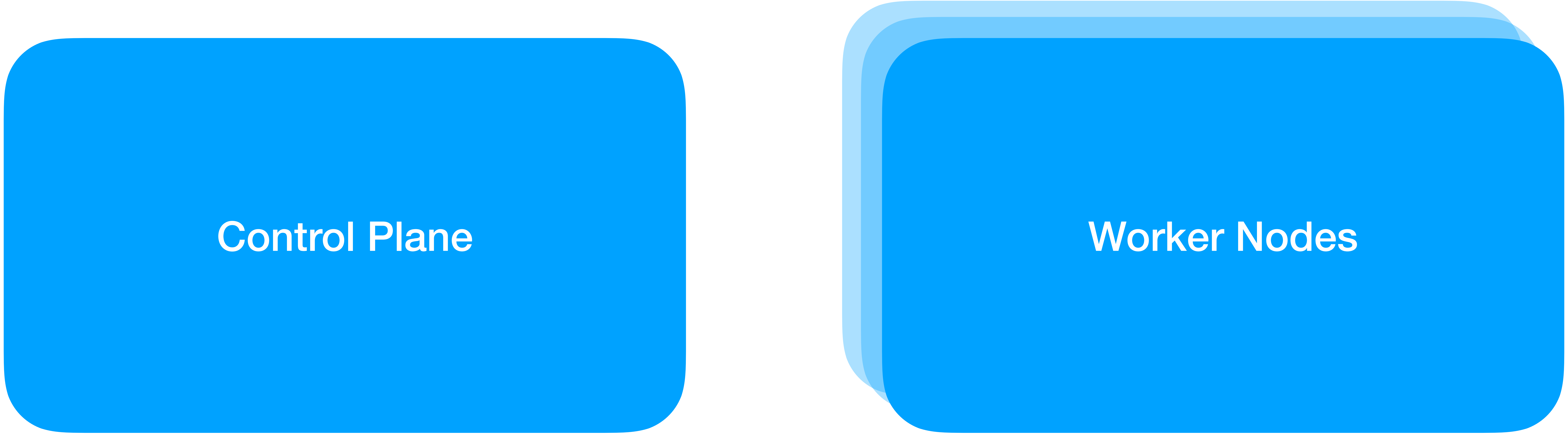


Developer

Application description

10s or 100s of nodes

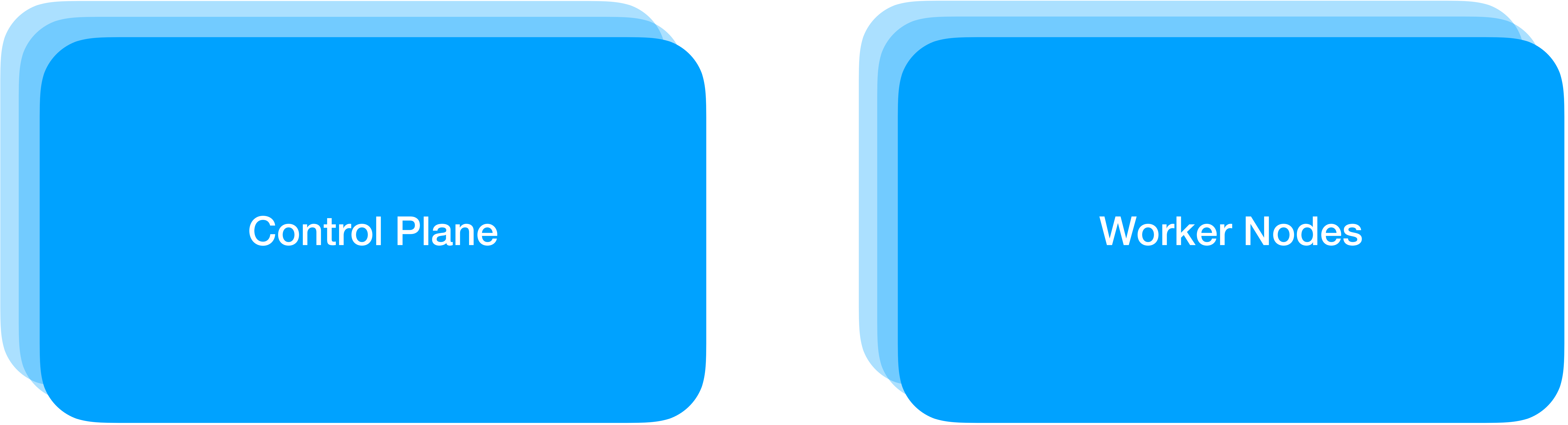
Kubernetes Architecture



Master node(s) host the Kubernetes control plane that controls and manages the cluster

Worker nodes run the actual applications

Kubernetes Architecture



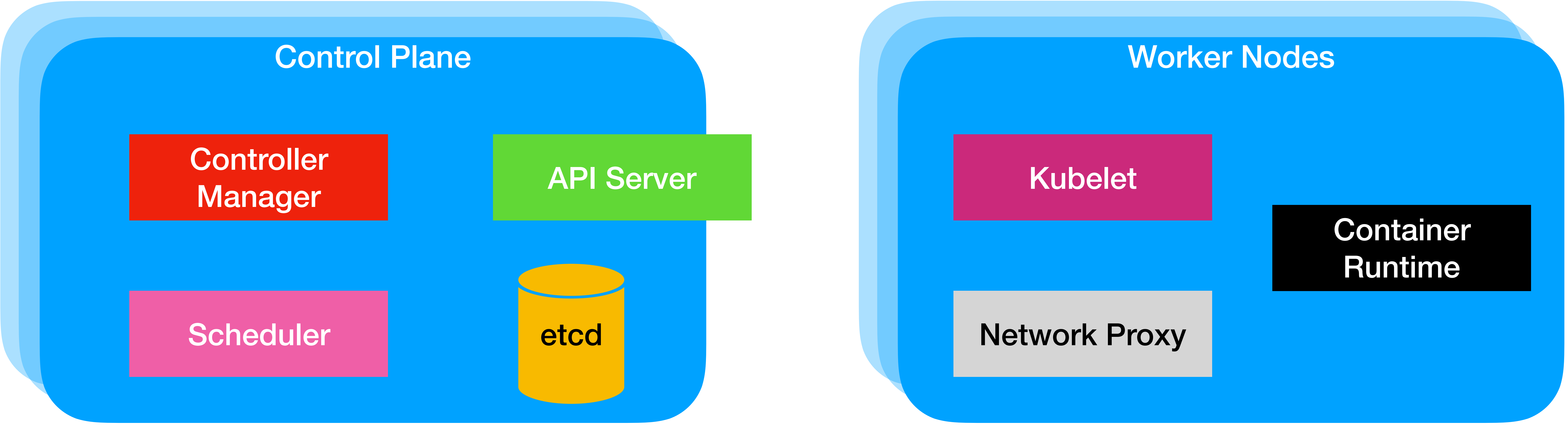
Control Plane

Worker Nodes

Master node(s) host the Kubernetes control plane that controls and manages the cluster

Worker nodes run the actual applications

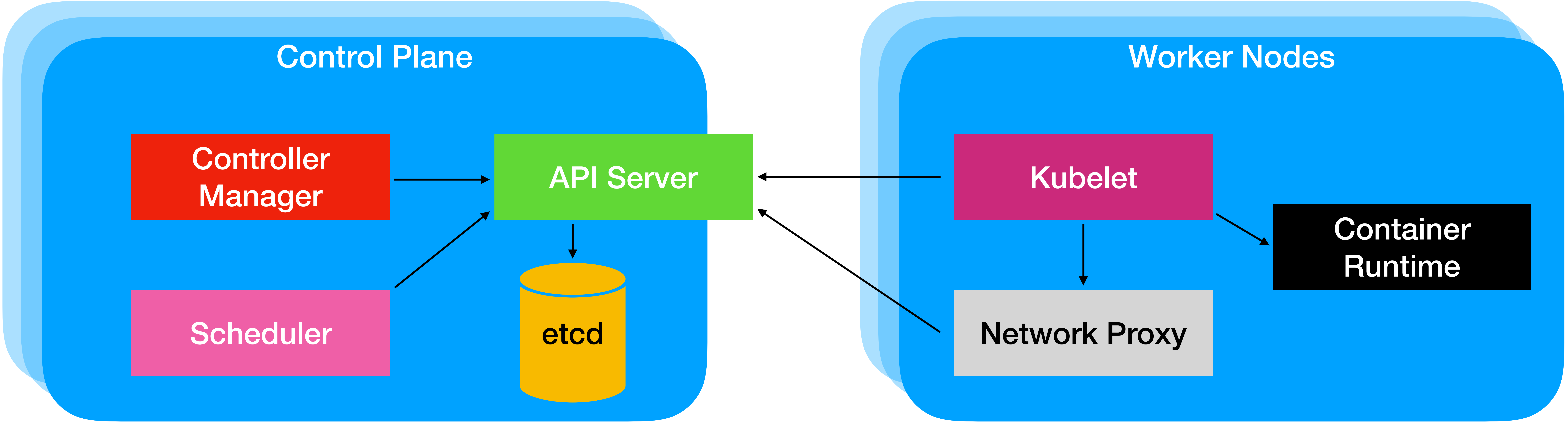
Kubernetes Architecture



Master node(s) host the Kubernetes control plane that control and manage the cluster

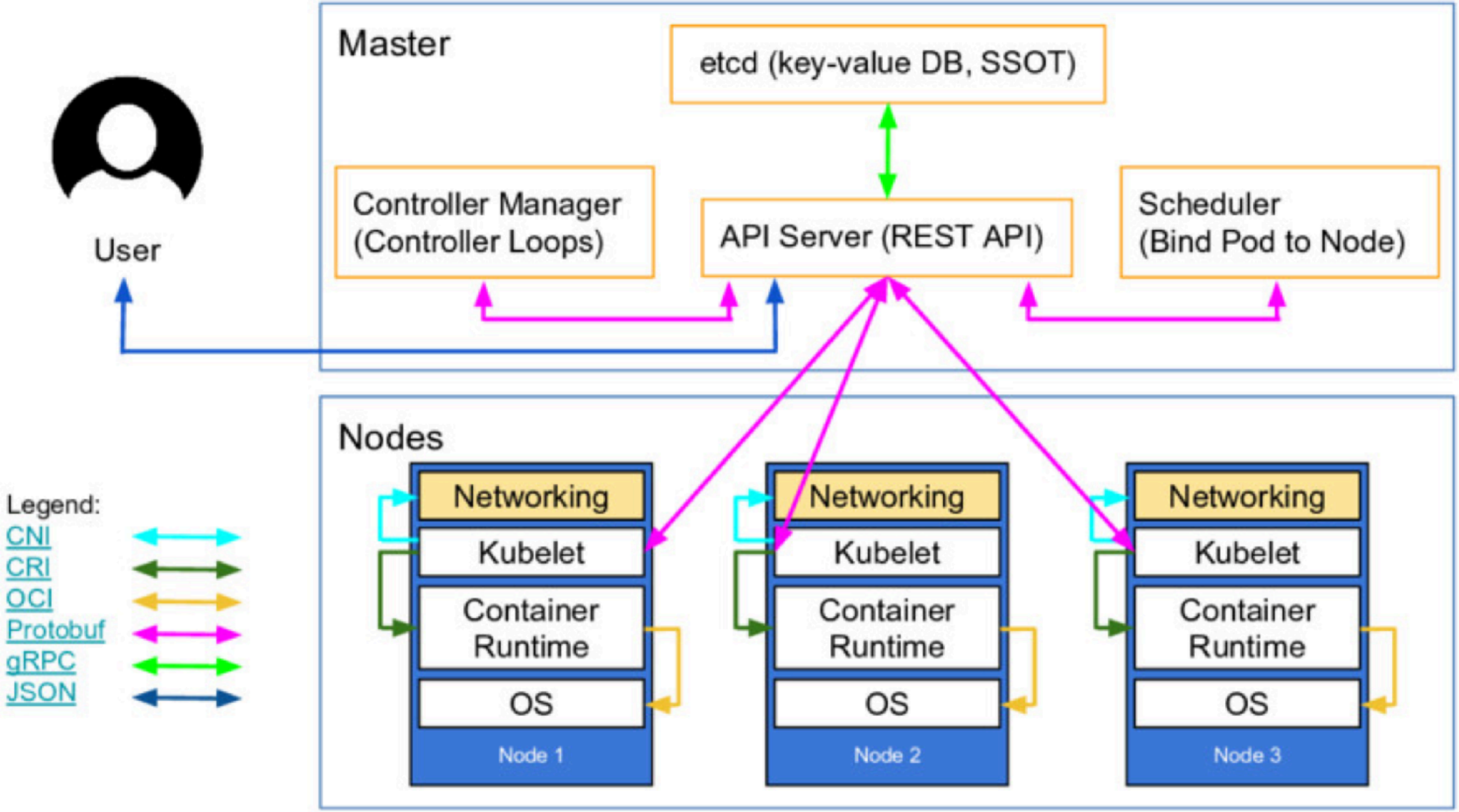
Worker nodes run the actual applications

Kubernetes Architecture



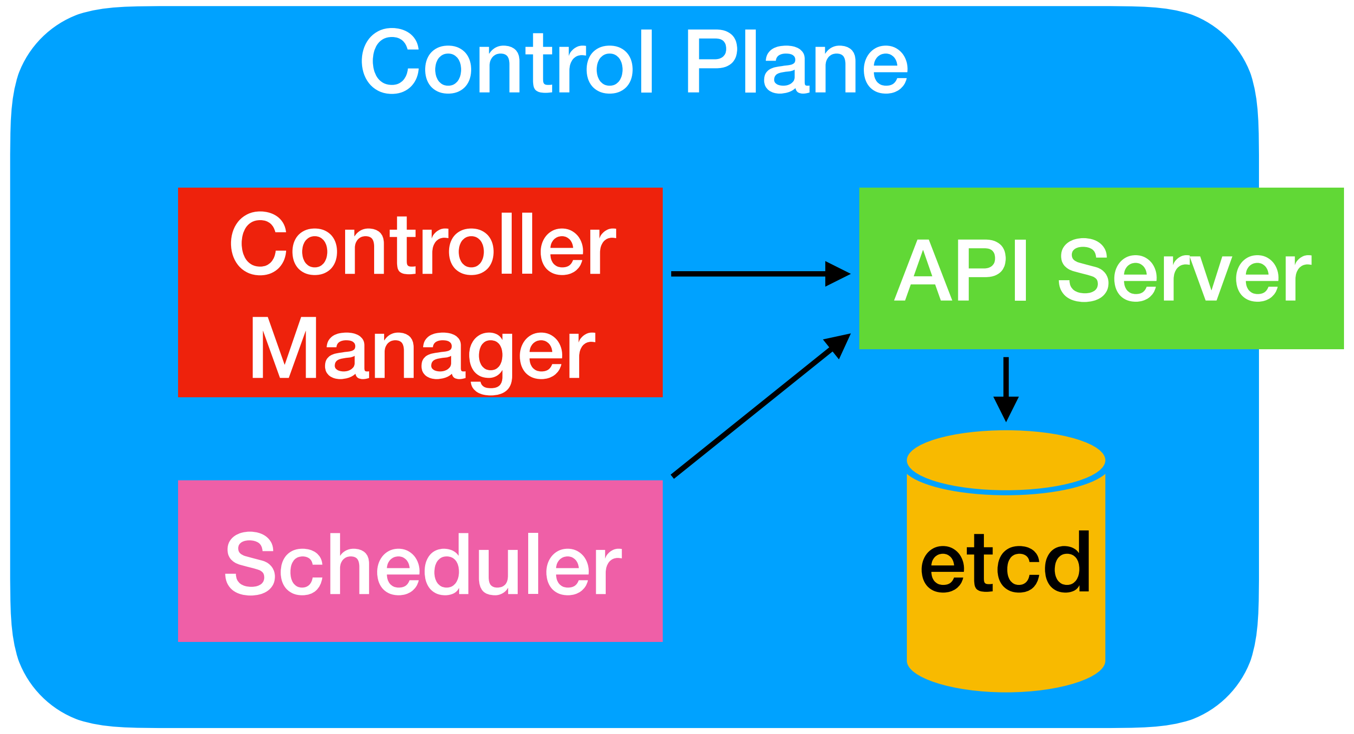
Master node(s) host the Kubernetes control plane that control and manage the cluster

Worker nodes run the actual applications



Lucas Käldström
https://speakerdeck.com/luxas/kubeadm-cluster-creation-internals-from-self-hosting-to-upgradability-and-ha?slide=7

Control Plane



Multiple components that can run on a single master node or be split across multiple (master) nodes and replicated to ensure high availability:

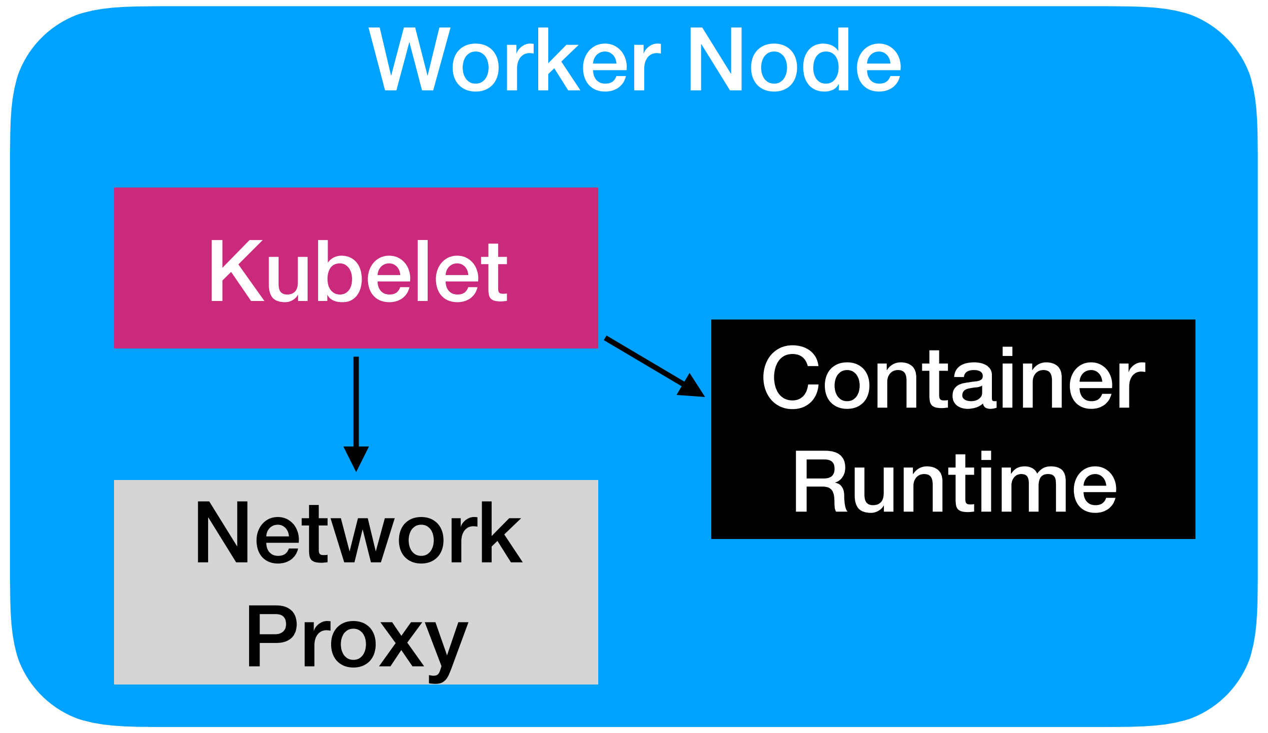
- **API Server**: communication center for developers, sysadmin and other Kubernetes components

- **Scheduler**: assigns a worker node to each deployable component

- **Controller Manager**: performs cluster-level functions (replication, keeping track of worker nodes, handling nodes failures…)

- **etcd**: reliable distributed data store where the cluster configuration is persisted

Worker Node



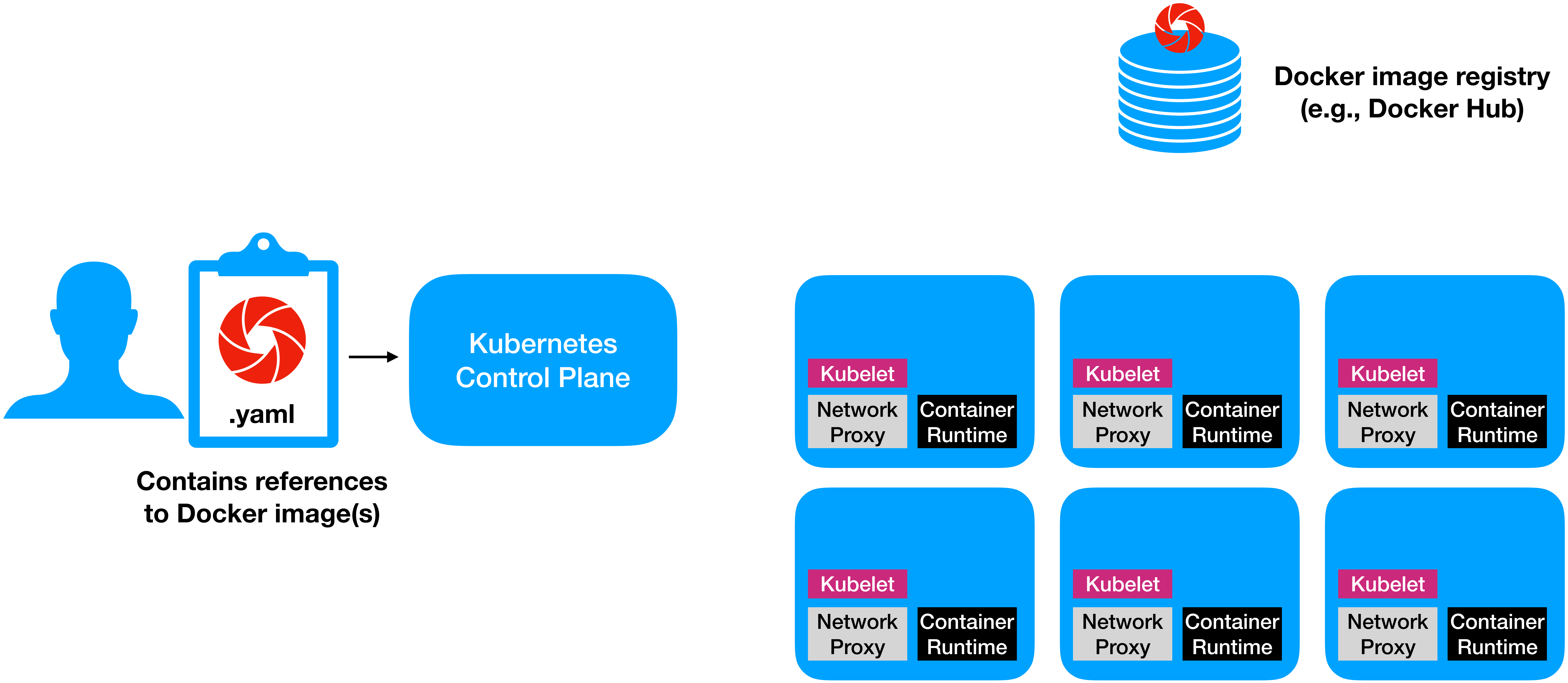
Machines that run containerized applications. It runs, monitors and provides services to applications via components:

Docker, rkt, or another container runtime: runs the containers

Kubelet: talks to API server and manages containers on its node

Network Proxy: load balance network traffic between application components

Running a Kubernetes application



Docker image registry
(e.g., Docker Hub)

.yaml

Contains references
to Docker image(s)

Kubernetes
Control Plane

Kubelet

Network Proxy

Container Runtime

Kubelet

Network Proxy

Container Runtime

Kubelet

Network Proxy

Container Runtime

Kubelet

Network Proxy

Container Runtime

Kubelet

Network Proxy

Container Runtime

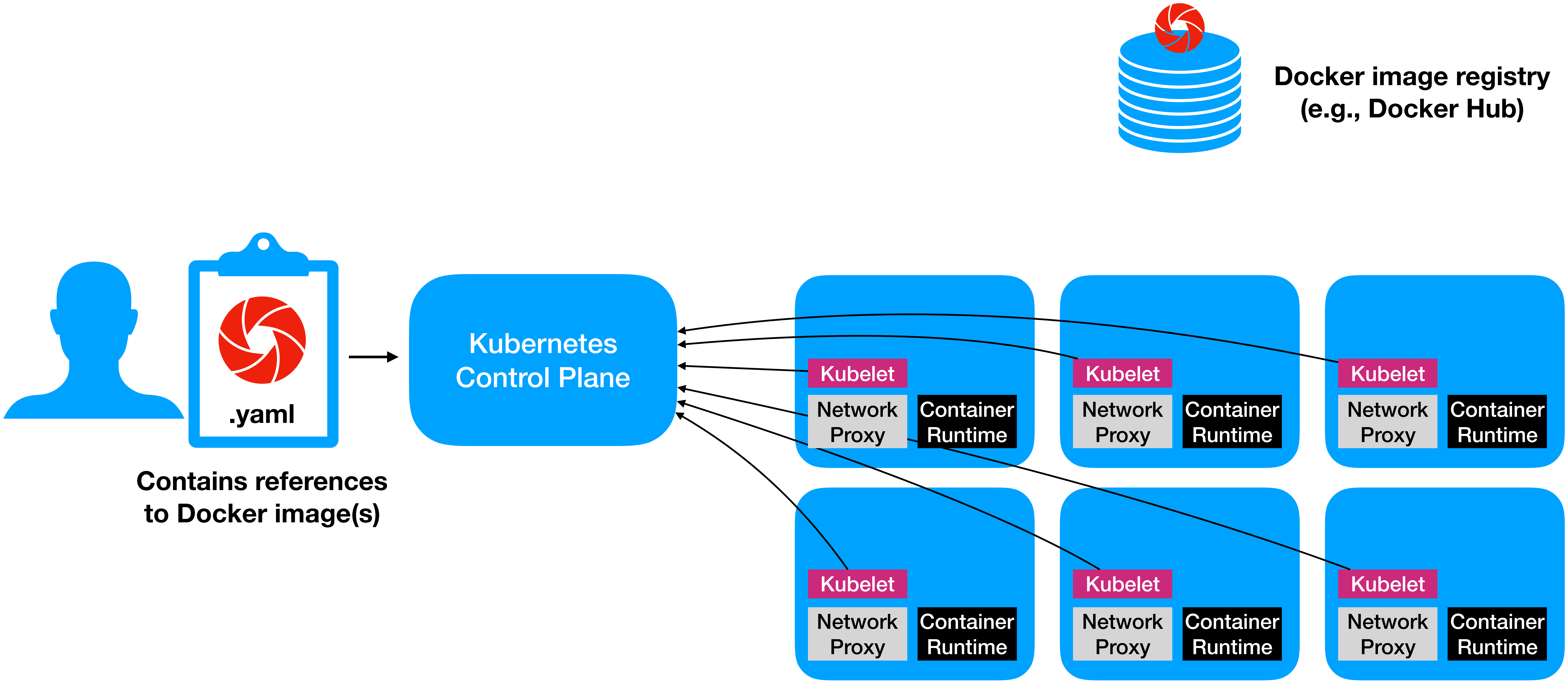
Kubelet

Network Proxy

Container Runtime

Worker Nodes

Running a Kubernetes application



Docker image registry
(e.g., Docker Hub)

.yaml

Contains references
to Docker image(s)

Kubernetes
Control Plane

Kubelet

Network
Proxy

Container
Runtime

Kubelet

Network
Proxy

Container
Runtime

Kubelet

Network
Proxy

Container
Runtime

Kubelet

Network
Proxy

Container
Runtime

Kubelet

Network
Proxy

Container
Runtime

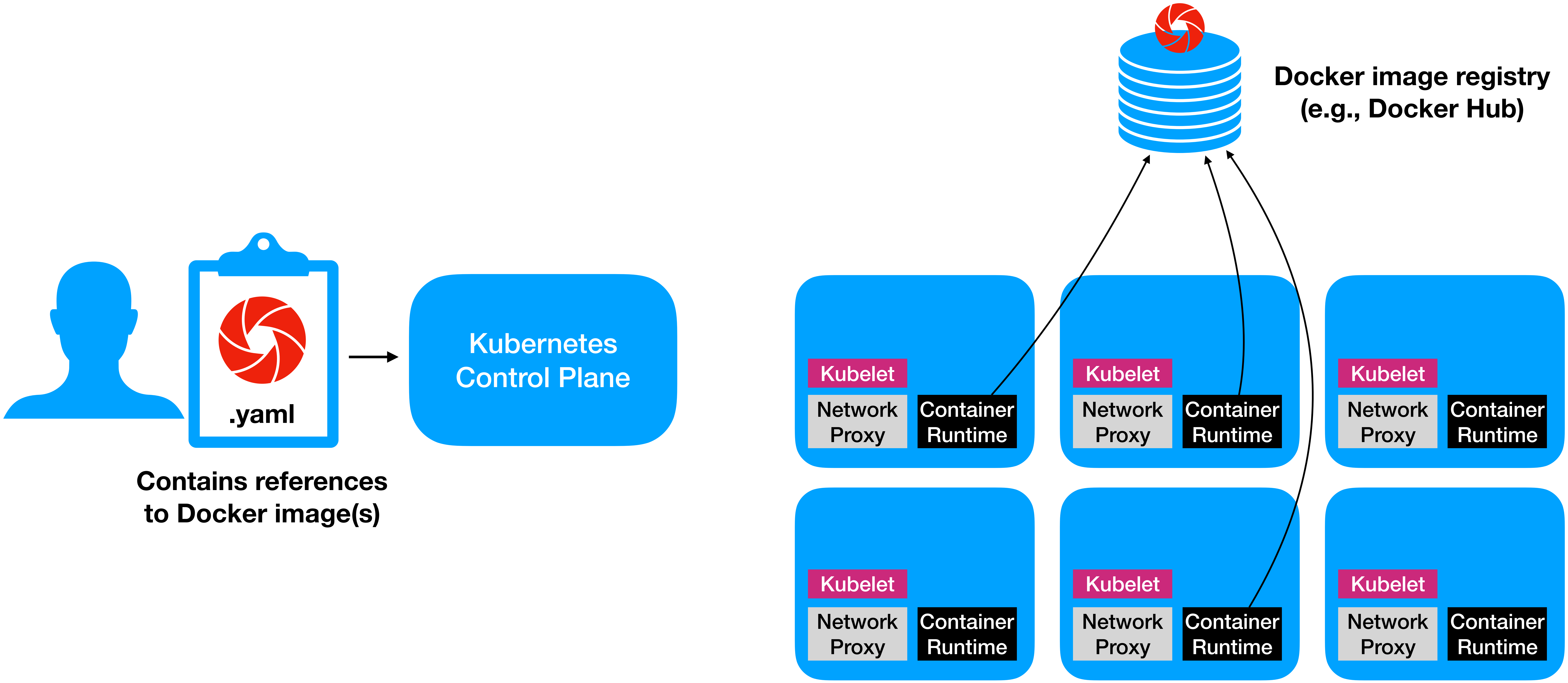
Kubelet

Network
Proxy

Container
Runtime

Worker Nodes

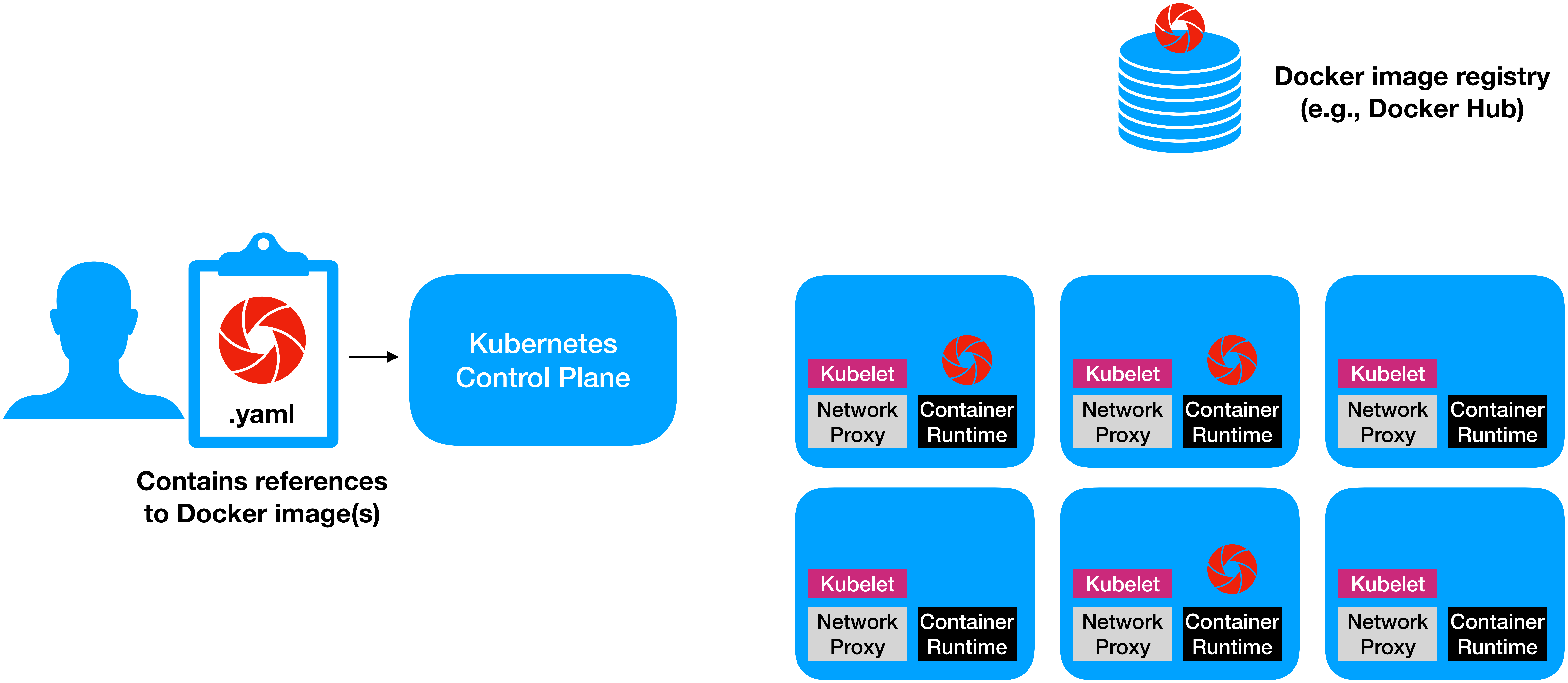
Running a Kubernetes application



Contains references
to Docker image(s)

Worker Nodes

Running a Kubernetes application



Docker image registry
(e.g., Docker Hub)

.yaml

Contains references
to Docker image(s)

Kubernetes
Control Plane

Kubelet

Network
Proxy

Container
Runtime

Kubelet

Network
Proxy

Container
Runtime

Kubelet

Network
Proxy

Container
Runtime

Kubelet

Network
Proxy

Container
Runtime

Kubelet

Network
Proxy

Container
Runtime

Kubelet

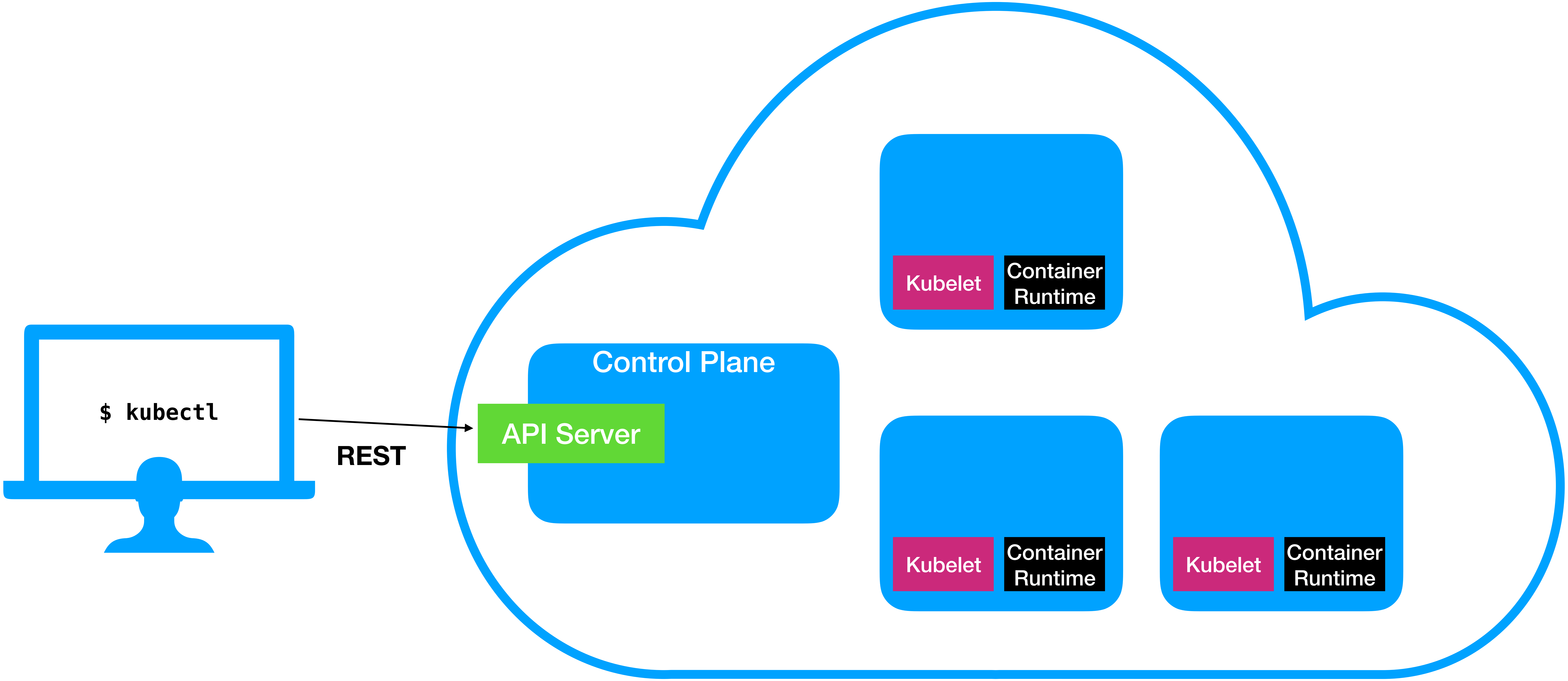
Network
Proxy

Container
Runtime

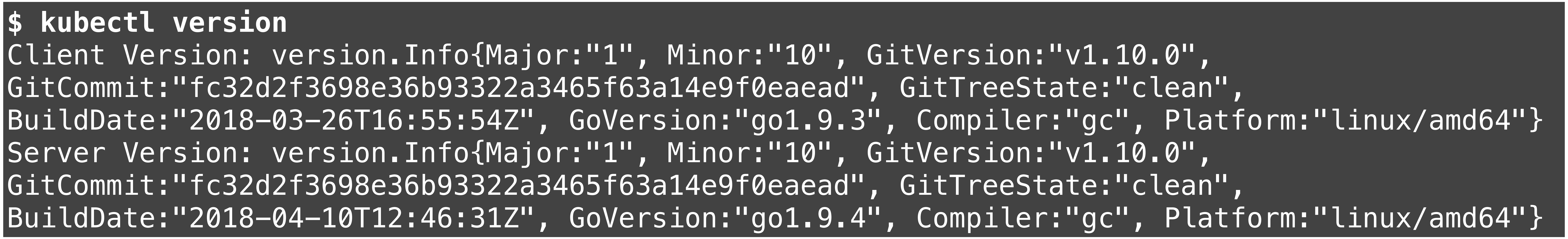
Worker Nodes

ECLIPSE
FOUNDATION

Working with a cluster



Working with a cluster

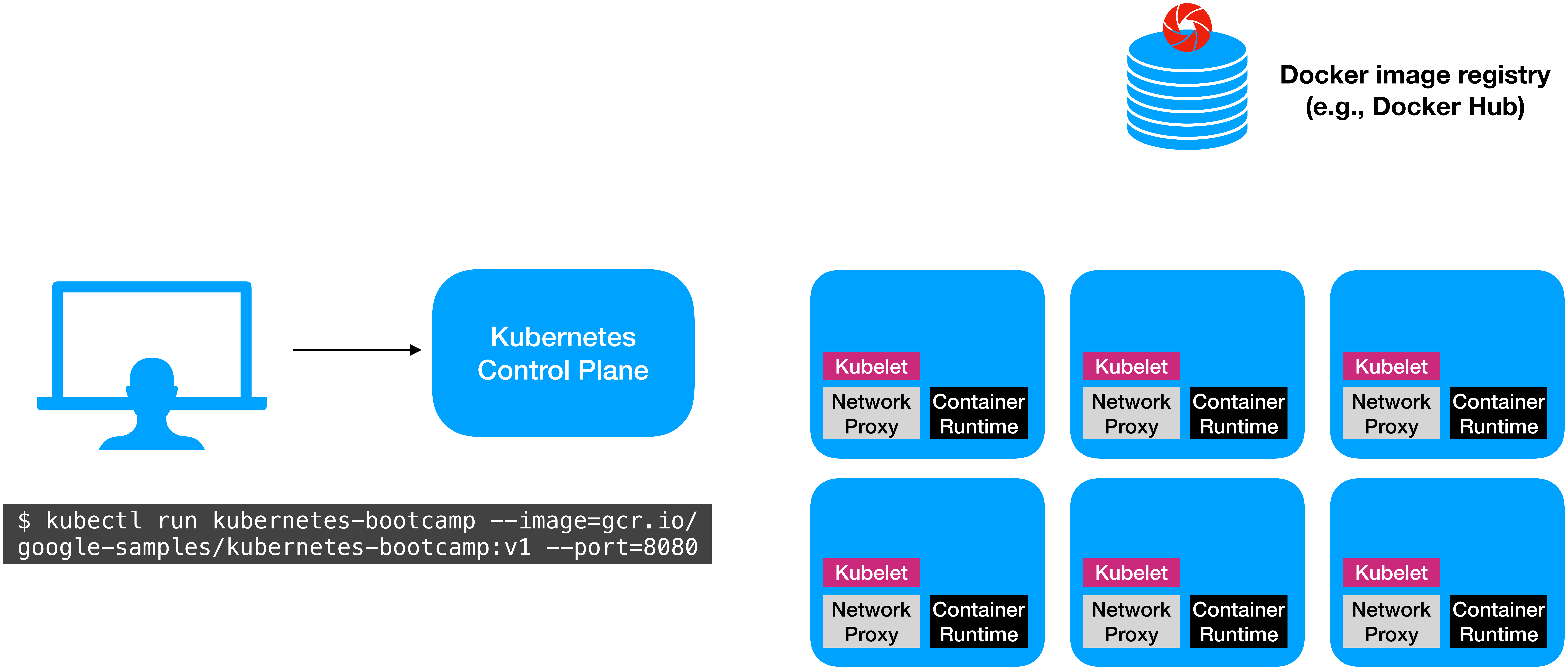


```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"10", GitVersion:"v1.10.0",
GitCommit:"fc32d2f3698e36b93322a3465f63a14e9f0eaead", GitTreeState:"clean",
BuildDate:"2018-03-26T16:55:54Z", GoVersion:"go1.9.3", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"10", GitVersion:"v1.10.0",
GitCommit:"fc32d2f3698e36b93322a3465f63a14e9f0eaead", GitTreeState:"clean",
BuildDate:"2018-04-10T12:46:31Z", GoVersion:"go1.9.4", Compiler:"gc", Platform:"linux/amd64"}
```

Running a container in a Pod

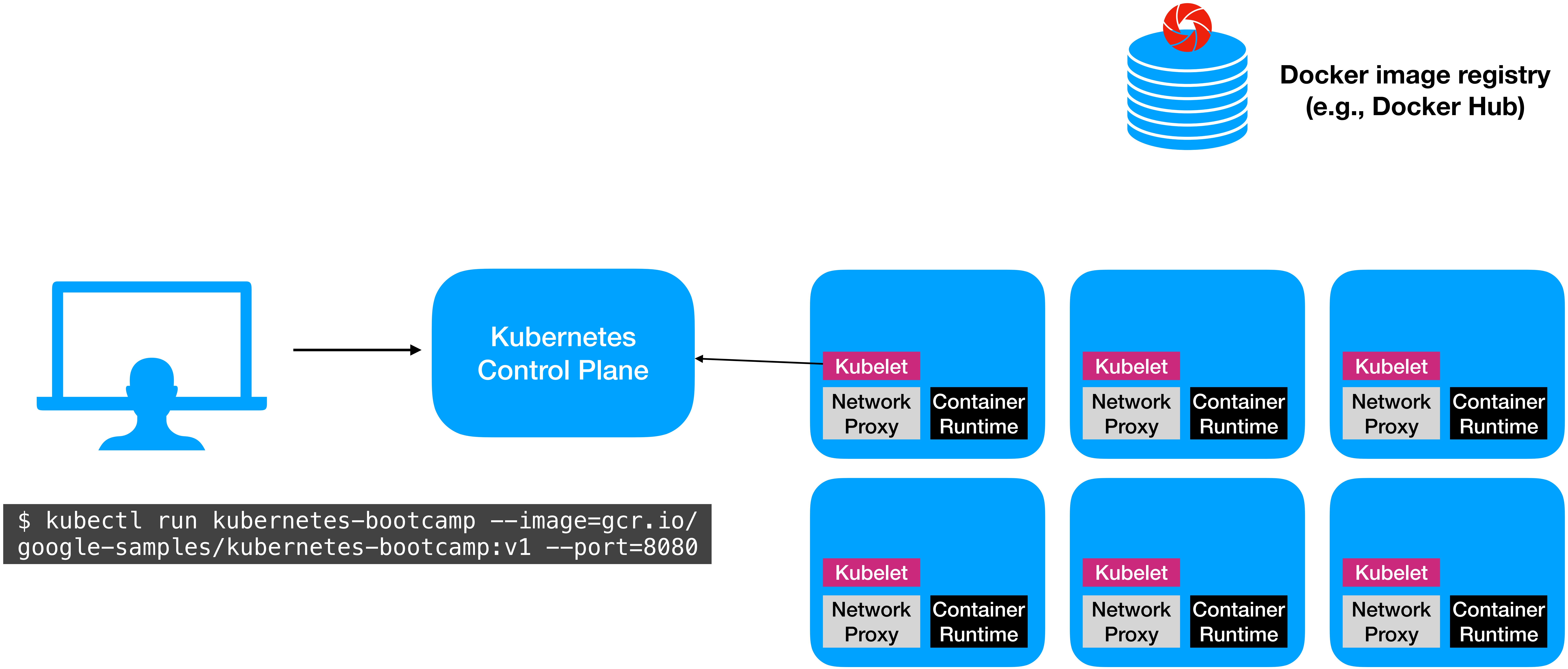
```
$ kubectl run kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1
--port=8080
deployment.apps "kubernetes-bootcamp" created

$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1         1         1            1           35s
```

Behind the scene



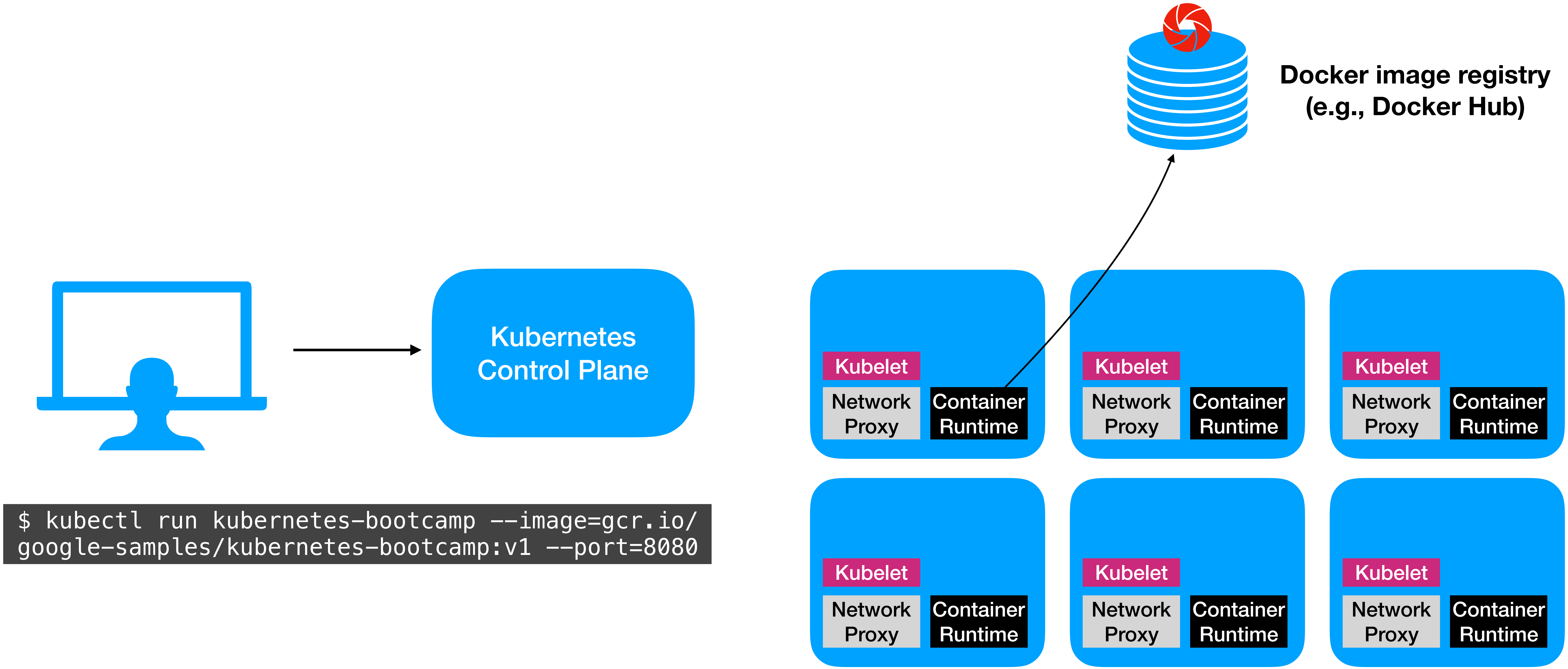
```
$ kubectl run kubernetes-bootcamp --image=gcr.io/
google-samples/kubernetes-bootcamp:v1 --port=8080
```

Behind the scene



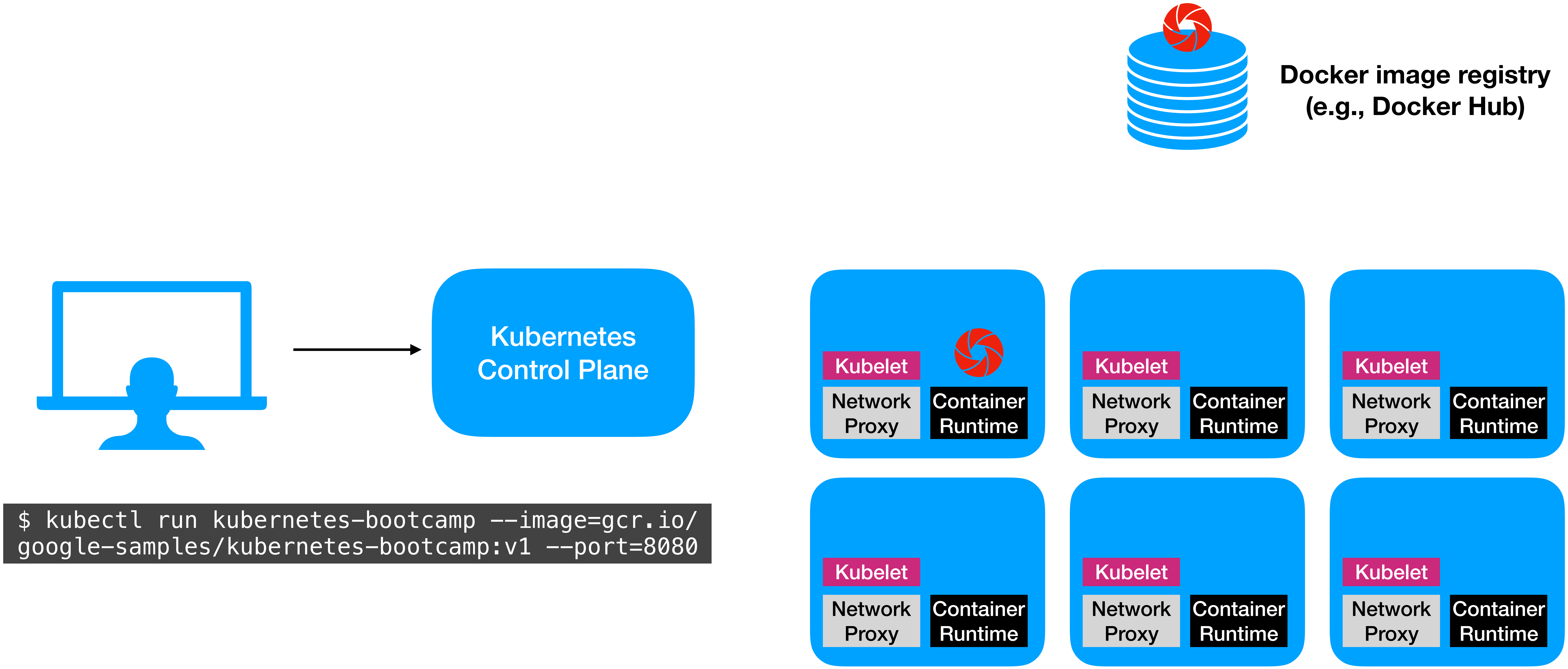
```
$ kubectl run kubernetes-bootcamp --image=gcr.io/
google-samples/kubernetes-bootcamp:v1 --port=8080
```

Behind the scene



```
$ kubectl run kubernetes-bootcamp --image=gcr.io/
google-samples/kubernetes-bootcamp:v1 --port=8080
```

Behind the scene

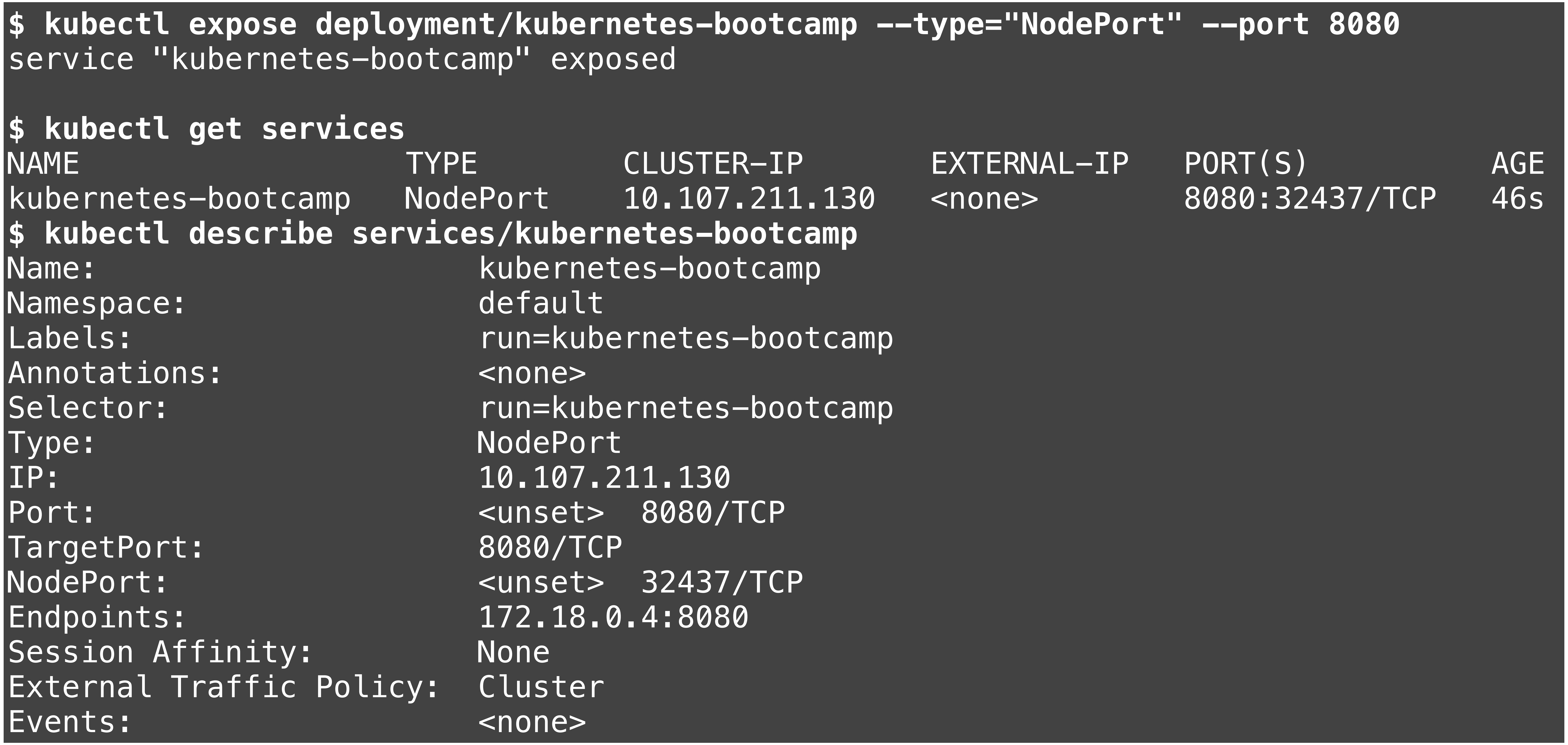


```
$ kubectl run kubernetes-bootcamp --image=gcr.io/
google-samples/kubernetes-bootcamp:v1 --port=8080
```

Behind the scene

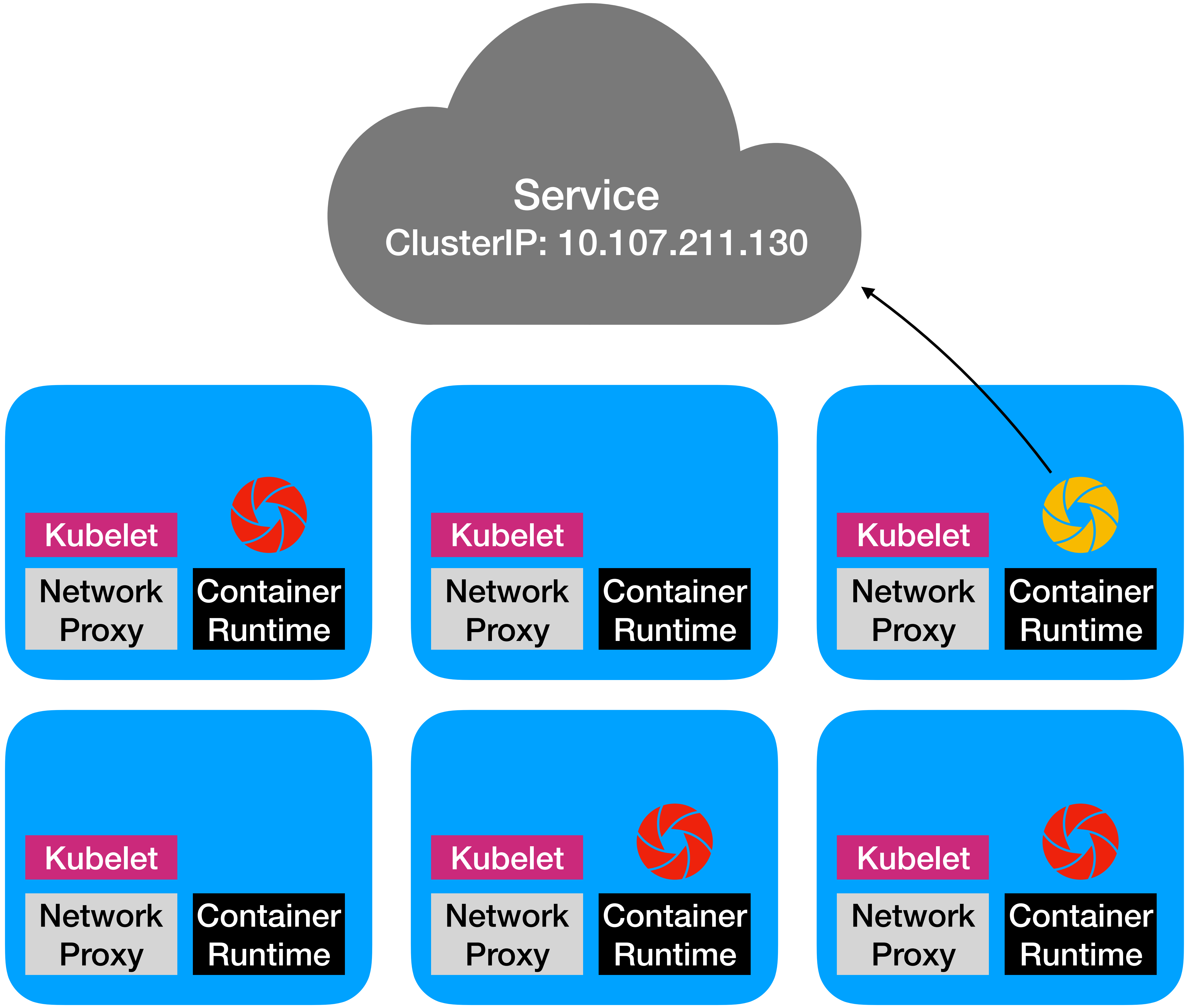
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
    name: kubernetes-bootcamp
spec:
    replicas: 1
    selector:
        matchLabels:
            run: kubernetes-bootcamp
    template:
        metadata:
            labels:
                run: kubernetes-bootcamp
        spec:
            containers:
            - image: gcr.io/google-samples/kubernetes-bootcamp:v1
                name: kubernetes-bootcamp
                ports:
                - containerPort: 8080
```

Exposing the container

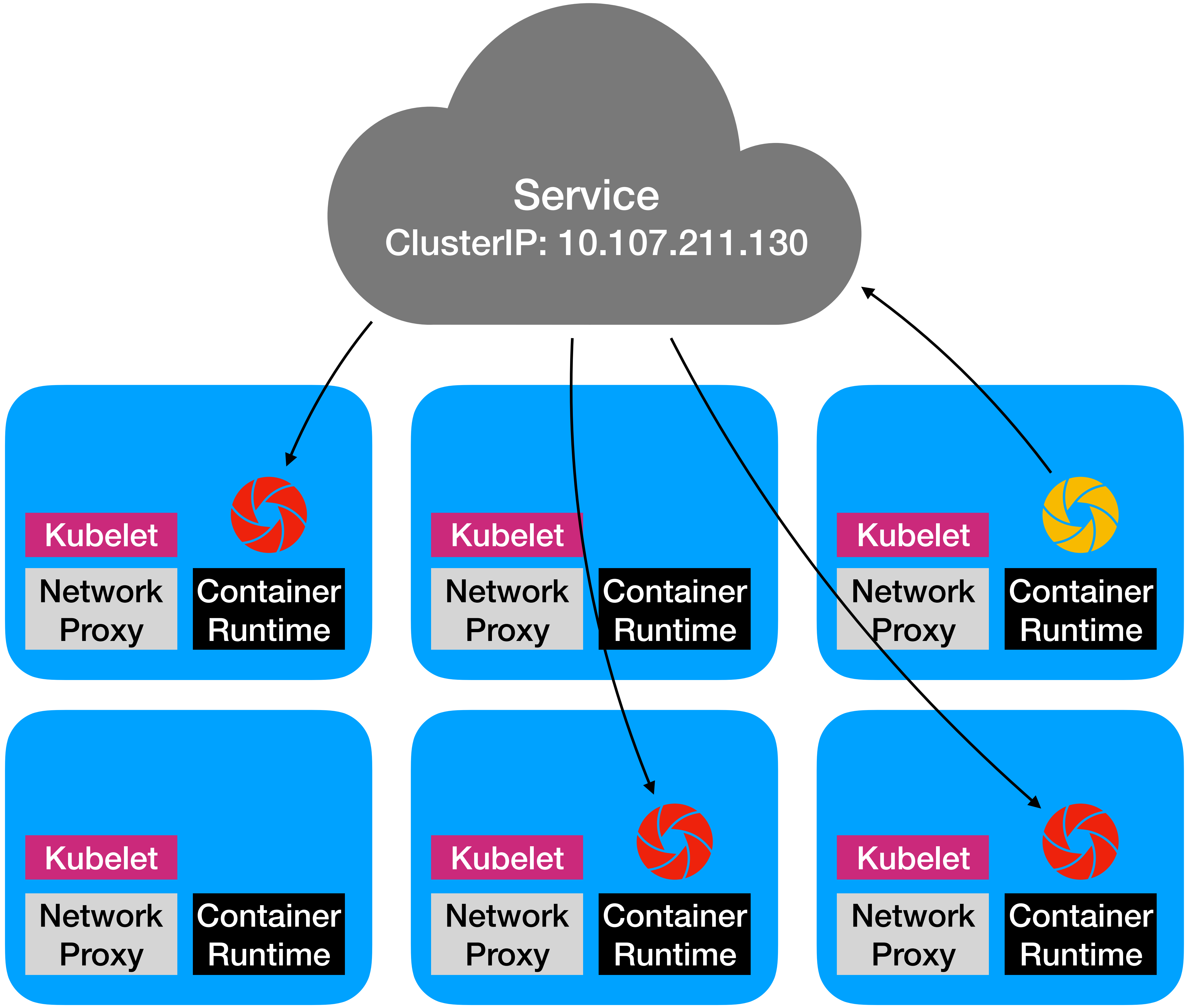


```
$ kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port 8080
service "kubernetes-bootcamp" exposed

$ kubectl get services
NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes-bootcamp   NodePort    10.107.211.130   <none>        8080:32437/TCP   46s
$ kubectl describe services/kubernetes-bootcamp
Name:                     kubernetes-bootcamp
Namespace:                default
Labels:                   run=kubernetes-bootcamp
Annotations:              <none>
Selector:                 run=kubernetes-bootcamp
Type:                     NodePort
IP:                       10.107.211.130
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  32437/TCP
Endpoints:                172.18.0.4:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

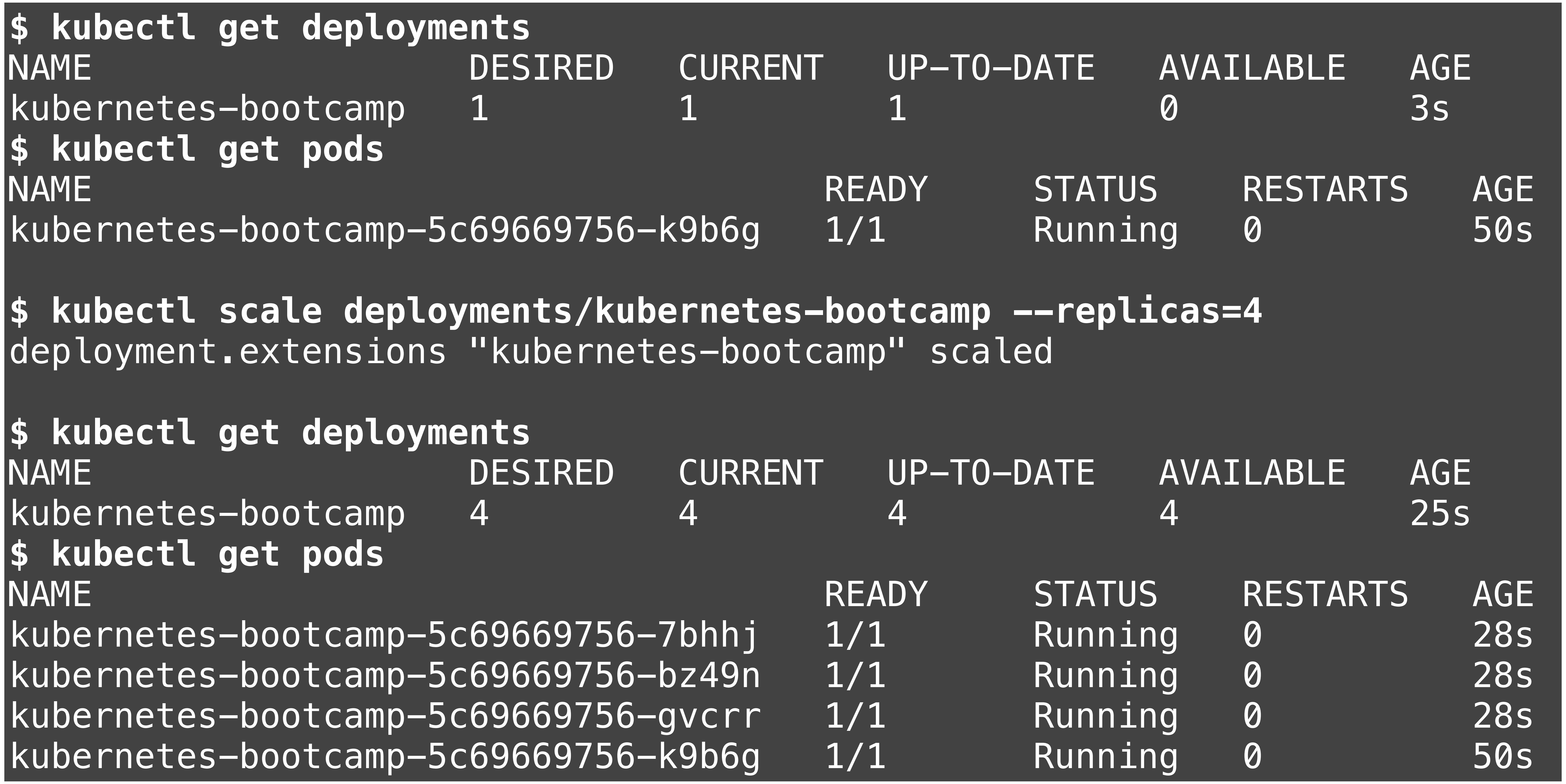
Exposing the container



Exposing the container

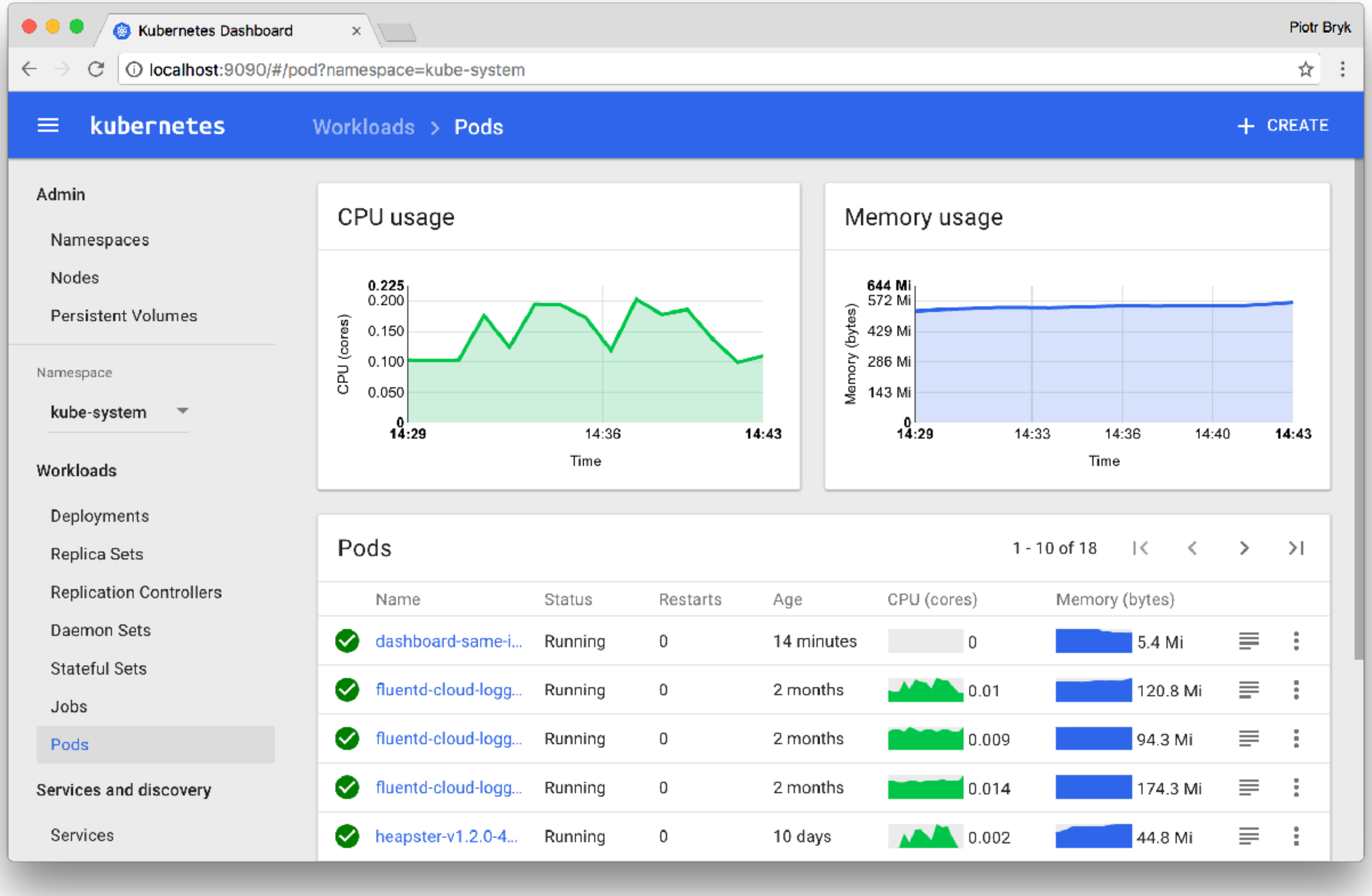


Scaling the application



```
$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1         1         1            0           3s
$ kubectl get pods
NAME                                   READY     STATUS    RESTARTS   AGE
kubernetes-bootcamp-5c69669756-k9b6g   1/1       Running   0          50s

$ kubectl scale deployments/kubernetes-bootcamp --replicas=4
deployment.extensions "kubernetes-bootcamp" scaled

$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   4         4         4            4           25s
$ kubectl get pods
NAME                                   READY     STATUS    RESTARTS   AGE
kubernetes-bootcamp-5c69669756-7bhhj   1/1       Running   0          28s
kubernetes-bootcamp-5c69669756-bz49n   1/1       Running   0          28s
kubernetes-bootcamp-5c69669756-gvcrr   1/1       Running   0          28s
kubernetes-bootcamp-5c69669756-k9b6g   1/1       Running   0          50s
```

Dashboard



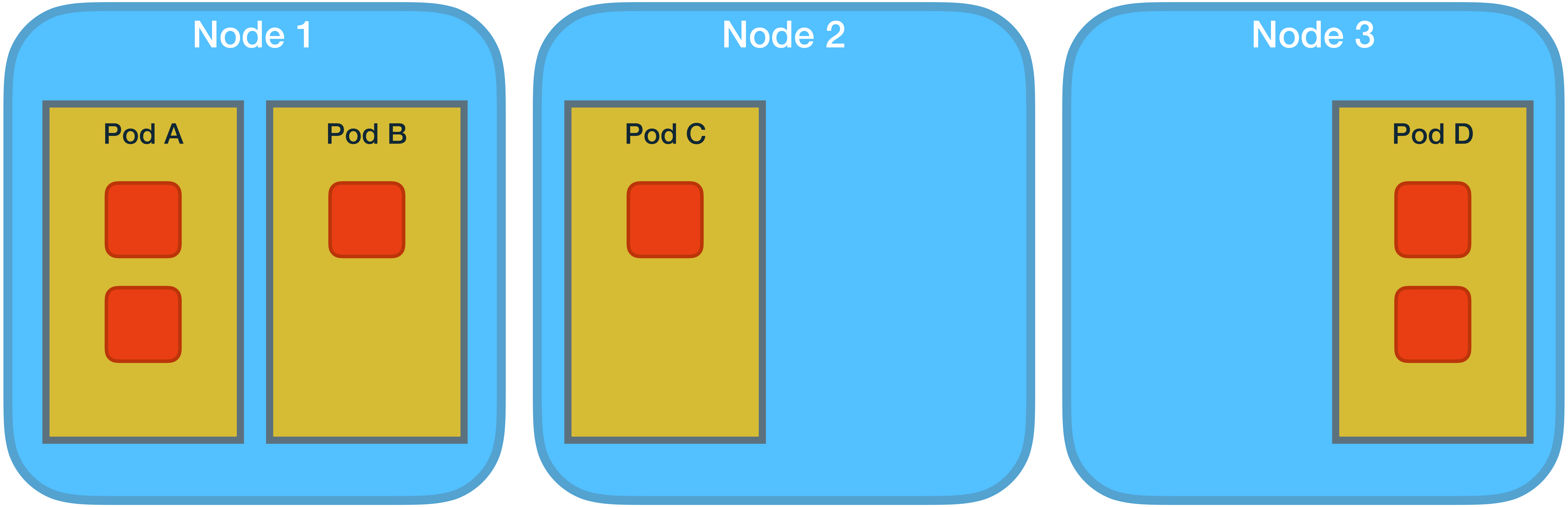
Pods

1 - 10 of 18

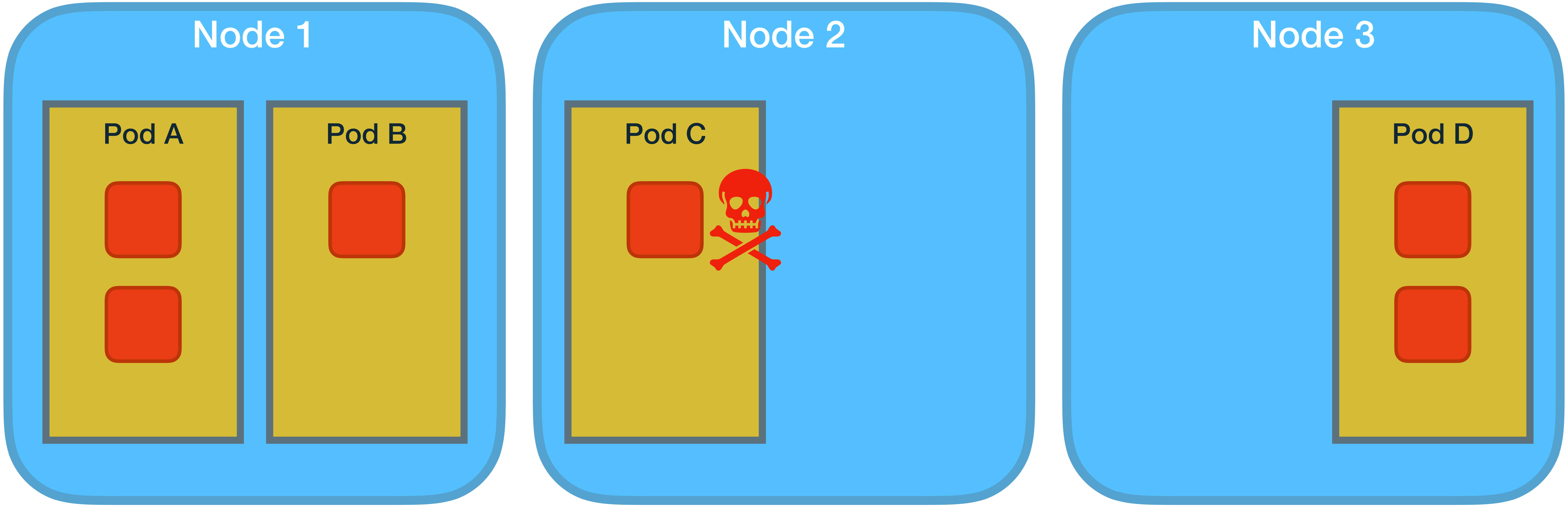
	Name	Status	Restarts	Age	CPU (cores)	Memory (bytes)
✓	dashboard-same-i...	Running	0	14 minutes	0	5.4 Mi
✓	fluentd-cloud-logg...	Running	0	2 months	0.01	120.8 Mi
✓	fluentd-cloud-logg...	Running	0	2 months	0.009	94.3 Mi
✓	fluentd-cloud-logg...	Running	0	2 months	0.014	174.3 Mi
✓	heapster-v1.2.0-4...	Running	0	10 days	0.002	44.8 Mi

Kubernetes Concepts

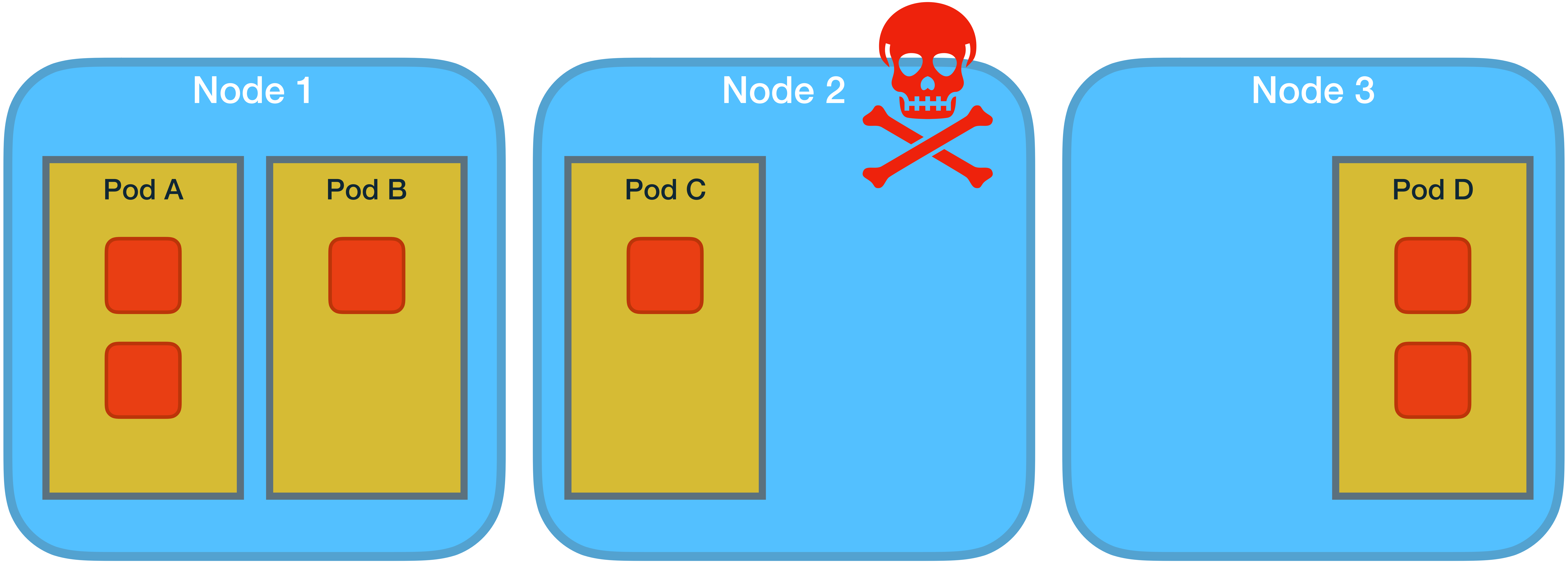
Concept #1 — Pod



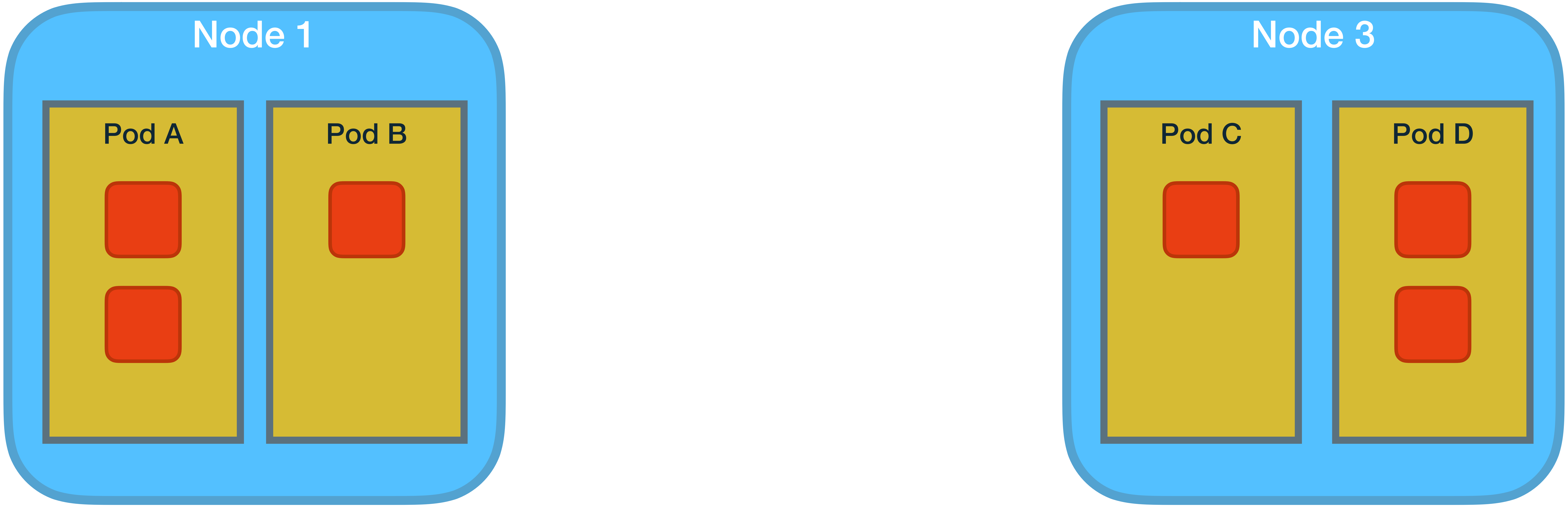
Concept #1 — Pod



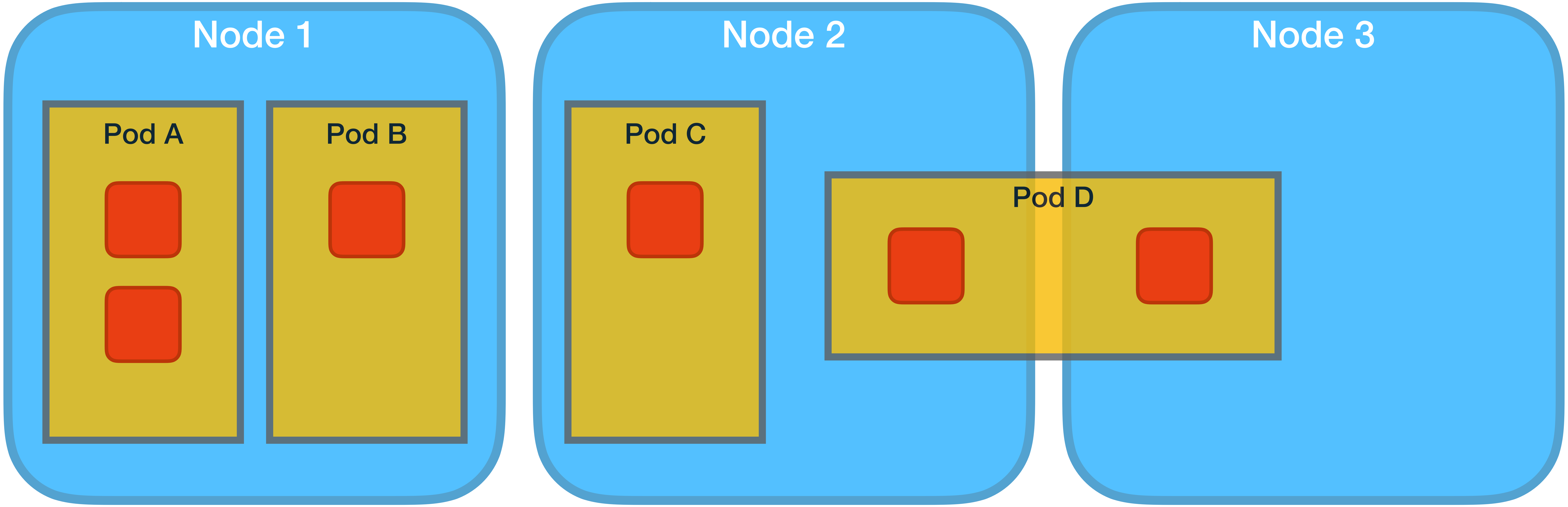
Concept #1 — Pod



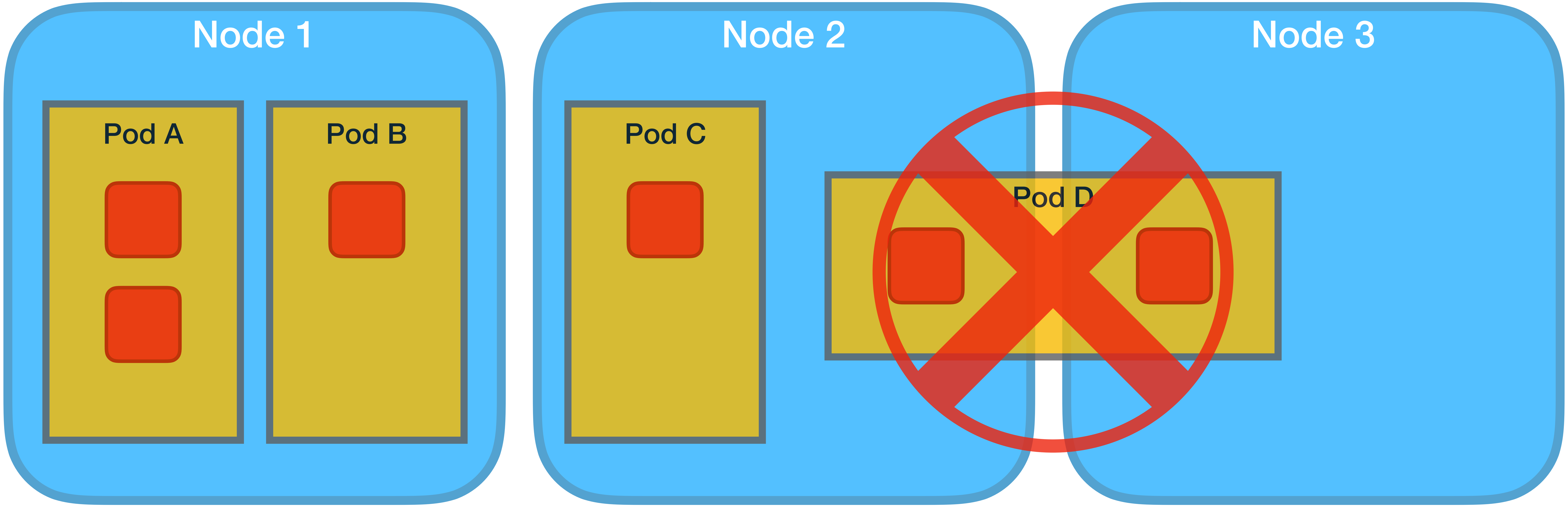
Concept #1 — Pod



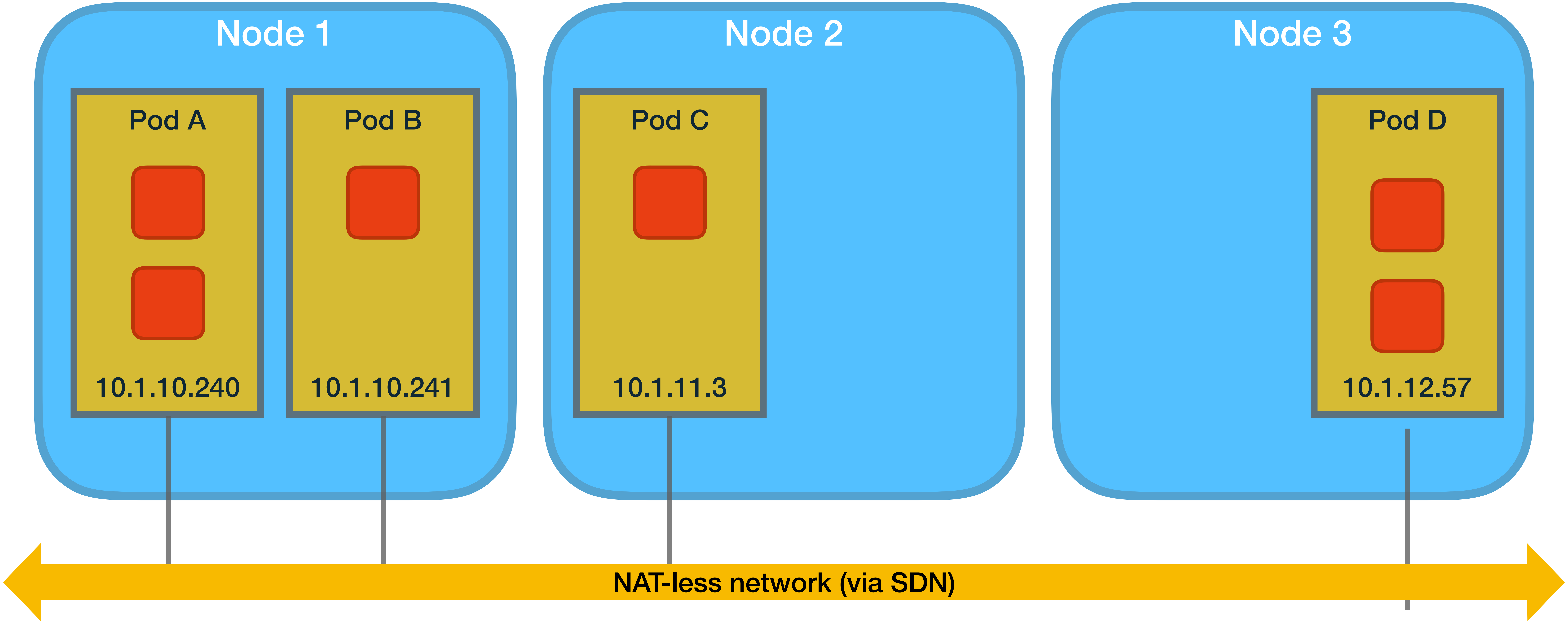
Concept #1 — Pod



Concept #1 — Pod



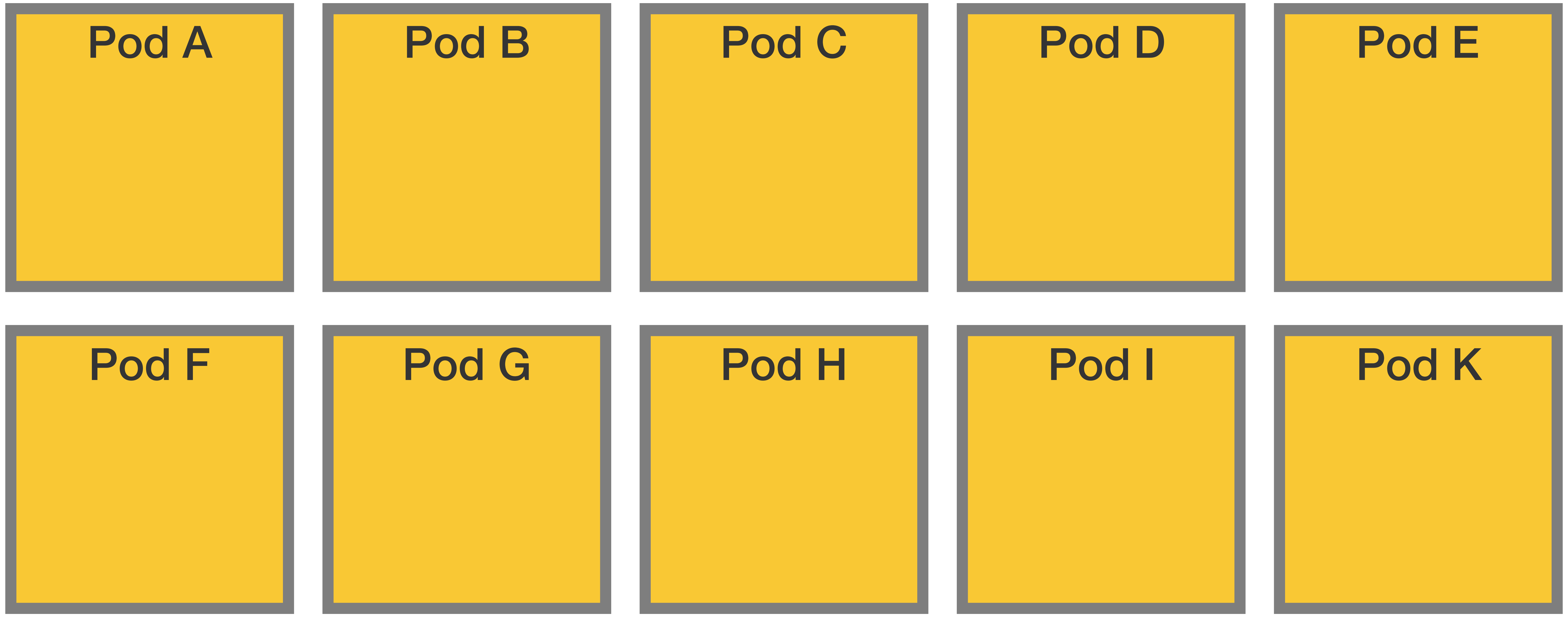
Concept #1 — Pod



Concept #1 — Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ecf-k8s-101
spec:
  containers:
  - image: nginx
    name: static-nginx
    ports:
    - containerPort: 8080
      protocol: TCP
```

Concept #2 — Labels



Pod A

Pod B

Pod C

Pod D

Pod E

Pod F

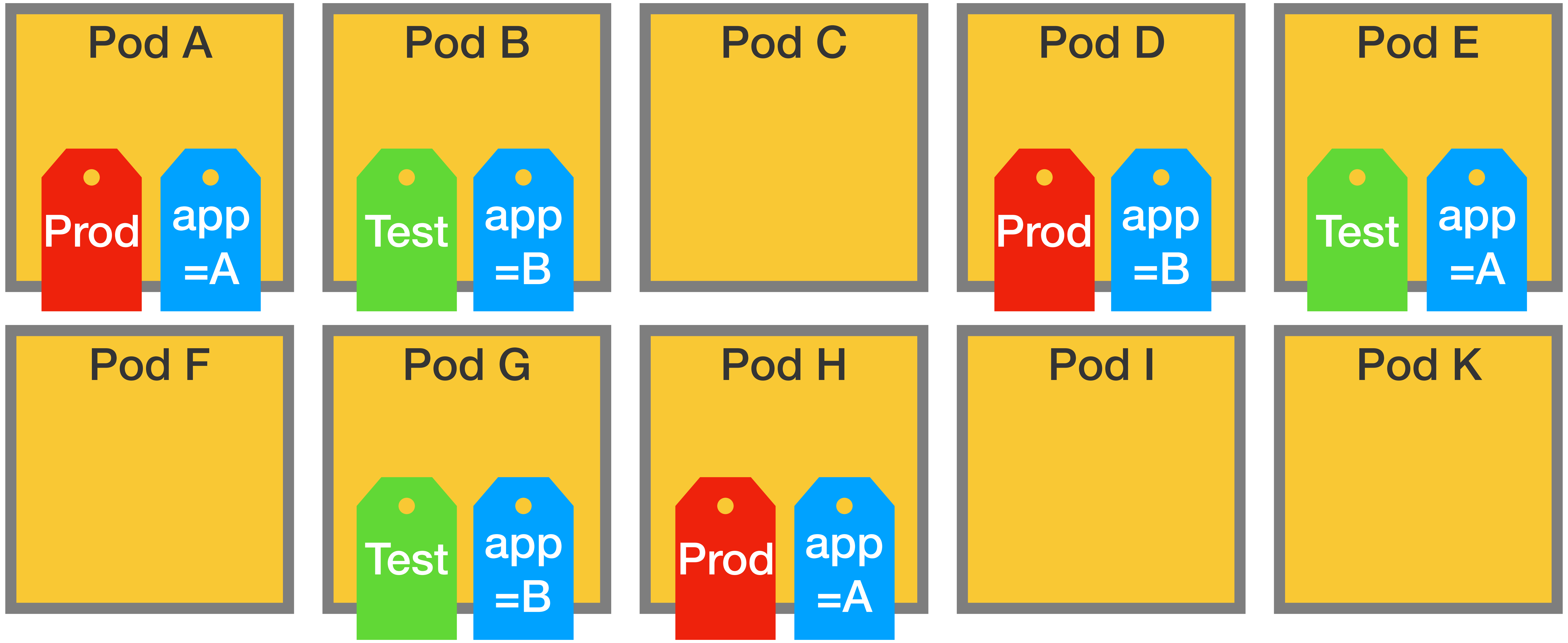
Pod G

Pod H

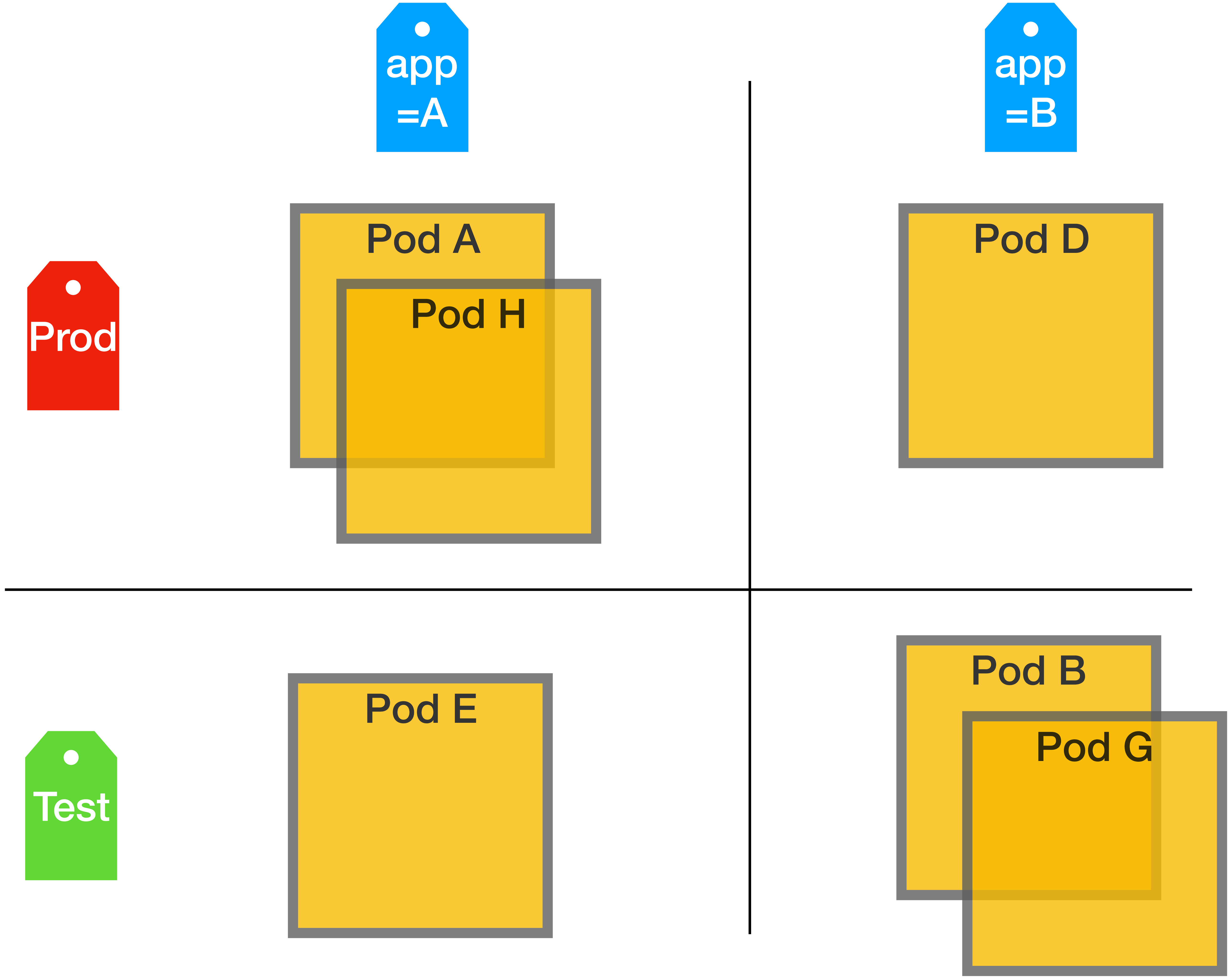
Pod I

Pod K

Concept #2 — Labels



Concept #2 — Labels



Concept #3 — Namespaces

- They are not Linux namespace (used by container runtimes to isolate containers)
- Separate resources into non-overlapping groups
- Can define permissions on namespaces
- Can even limit the amount of computational resources available
- Network policies use namespaces to isolate pods

custom-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: custom-ns
```

```
$ kubectl create -f custom-ns.yaml
Namespace “custom-ns” created

$ kubectl create namespace custom-ns
namespace "custom-ns" created
```

ECLIPSE FOUNDATION

Concept #4 — ReplicaSet

Ensures that a specified number of pod replicas are running at any given time.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
 template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 80
```

Concept #5 — DaemonSet

- A DaemonSet ensures that all (or some) Nodes run a copy of a Pod.
- As nodes are added to the cluster, Pods are added to them.
- As nodes are removed from the cluster, those Pods are garbage collected.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: k8s.gcr.io/fluentd-elasticsearch:1.20
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

Concept #6 — Job and CronJob

- A job creates one or more pods and ensures that a specified number of them successfully terminate.

- As pods successfully complete, the job tracks the successful completions. When a specified number of successful completions is reached, the job itself is complete. Deleting a Job will cleanup the pods it created.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print 
bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

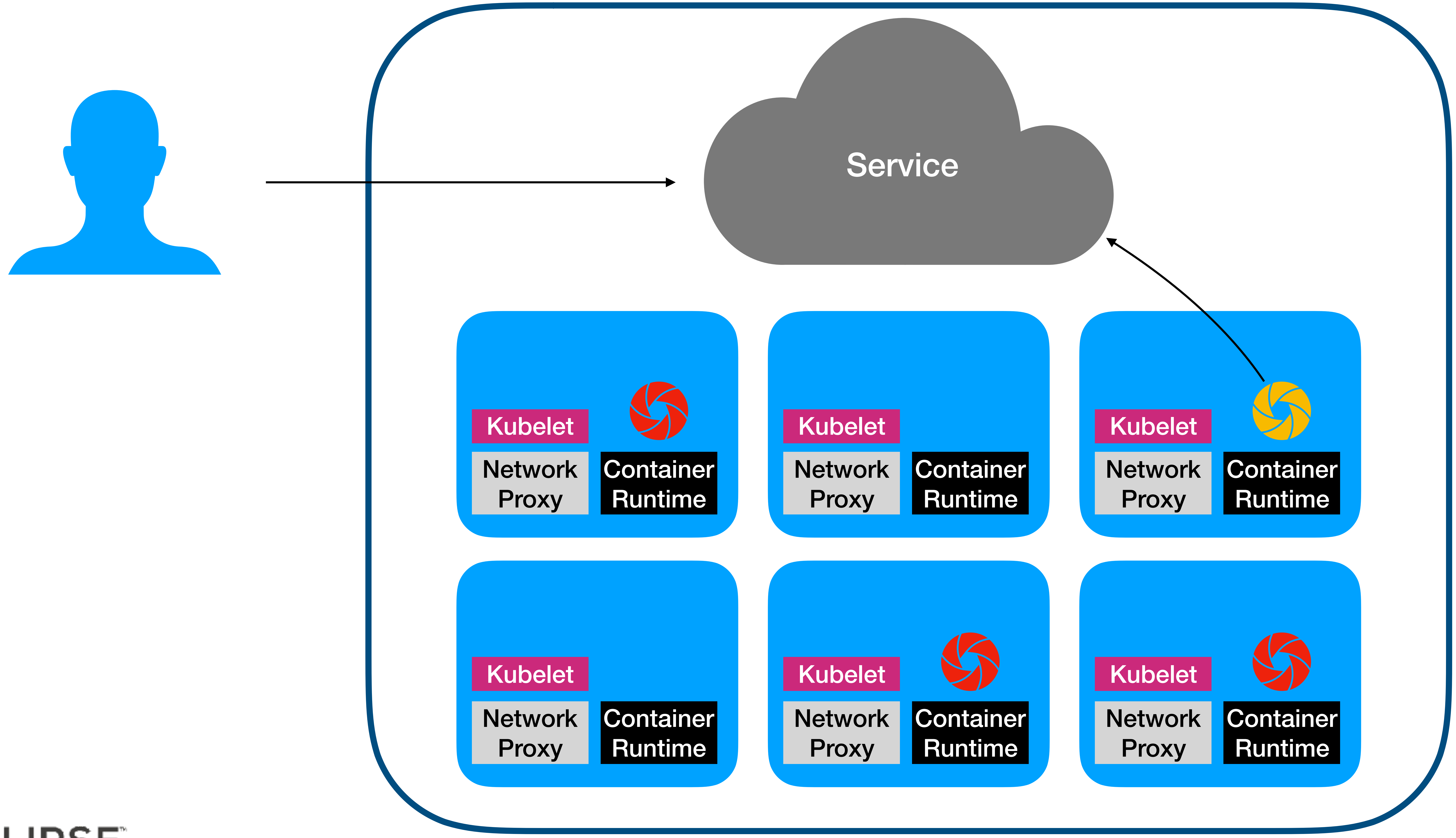
Concept #7 — Service

- A service is a stable address for a pod (or a bunch of pods)
  - ClusterIP: cluster-private IP
  - NodePort: ClusterIP + any node IP on specific port (thus available to the "outside" of the cluster)
  - LoadBalancer: NodePort + load balancer frontend (provided by the cloud provider).
  - Ingress: kind of reverse proxy in front of the cluster

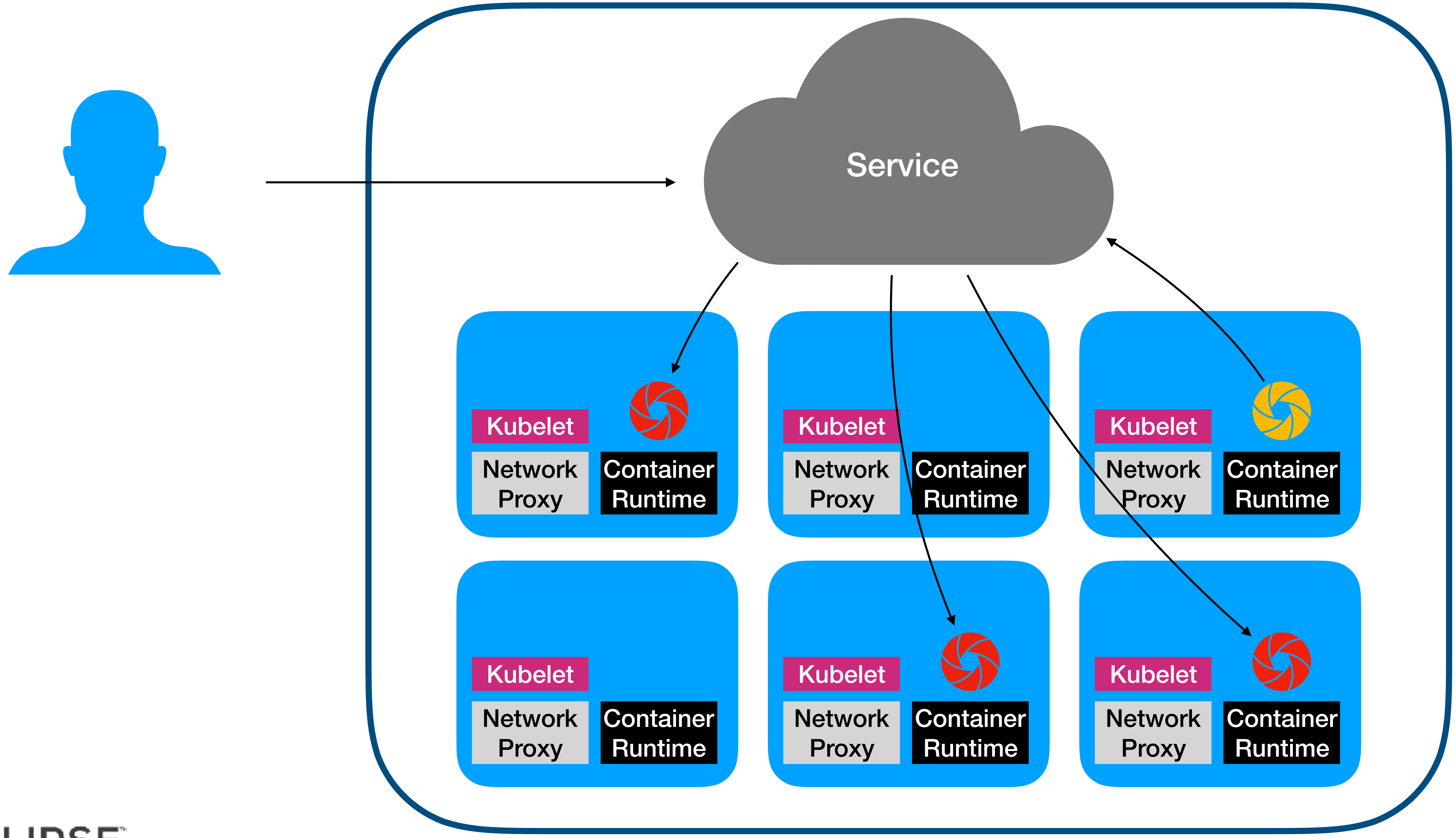
```
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
```

```
$ kubectl get svc my-nginx
NAME        CLUSTER-IP     EXTERNAL-IP   PORT(S)
my-nginx    10.0.162.149   <none>        80/TCP
```

Concept #7 — Service



Concept #7 — Service

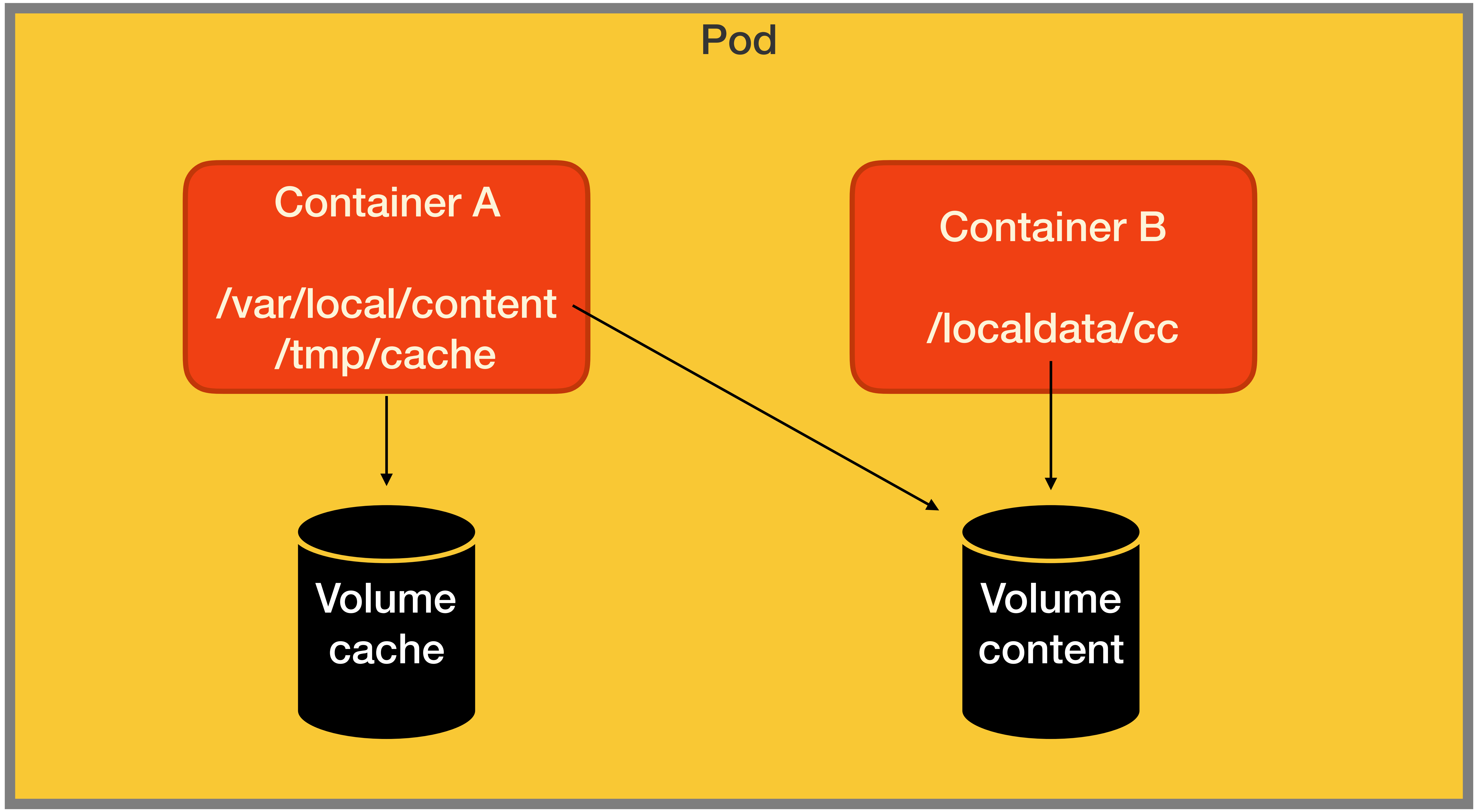


ECLIPSE FOUNDATION

Concept #7 — Service

- Service Discovery
  - Via environment variables. Each pod is initialized with environments variables for each service available at that moment (e.g. MY-NGINX_SERVICE_HOST=10.0.162.149 and MY-NGINX_SERVICE_PORT=80)
  - Via DNS. The control plane runs a DNS server and modify each container's /etc/resolve.conf to use it. Each service gets a DNS entry in the form <service_name>.<namespace>.<cluster_domain_suffix> (e.g.,my-ginx.default.svc.eclipsefnd-cluster1.local).
  - Access to external resources is also possible via Endpoints

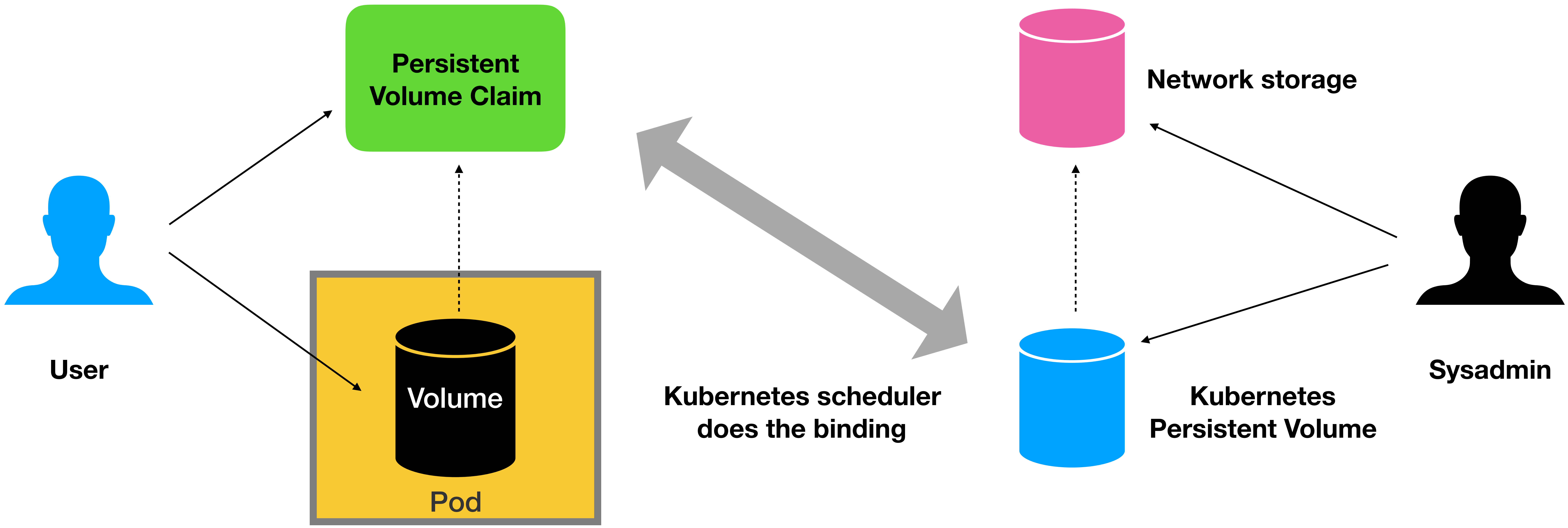
Concept #8 — Volumes



Concept #8 — Volumes

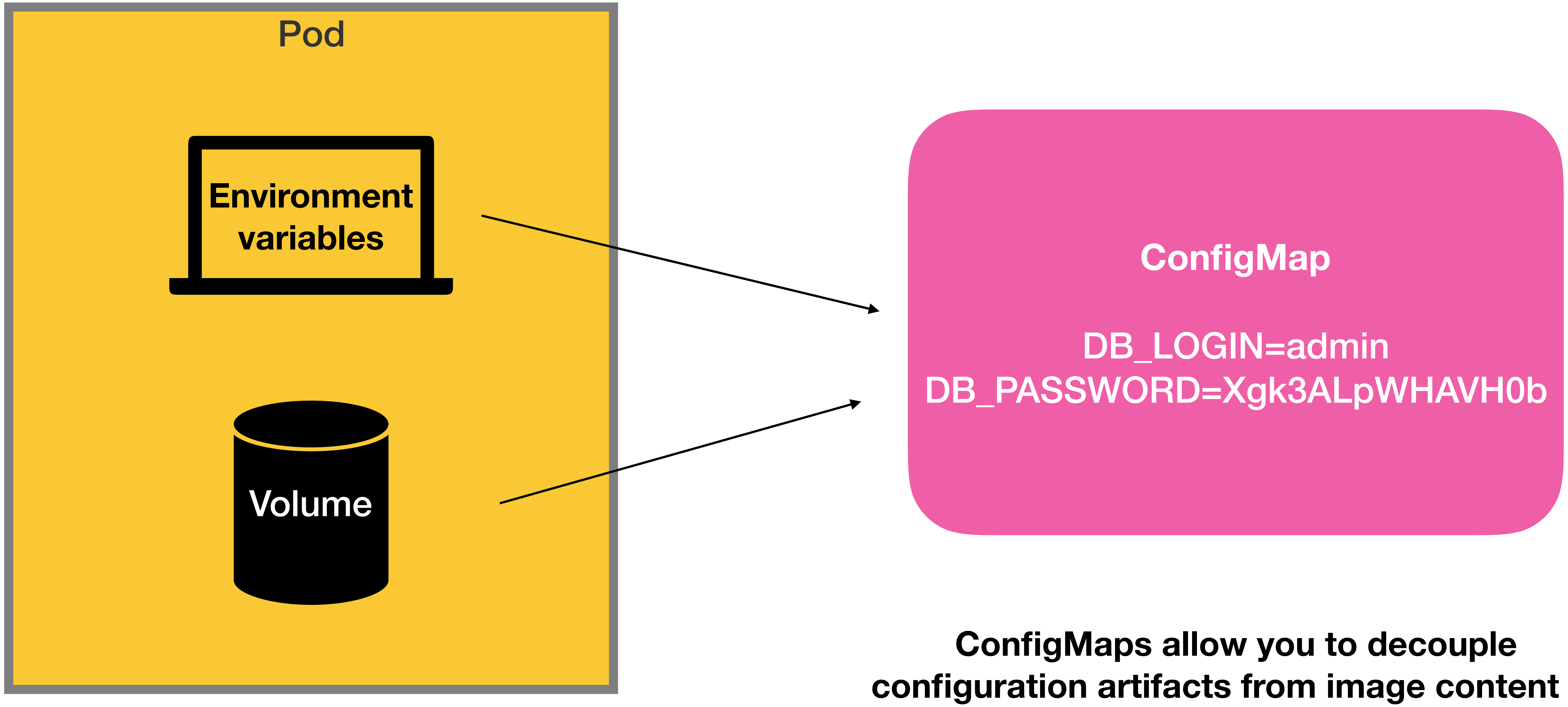
- Empty directory used for storing transient data (lifecycle is tied to its pod).
- Worker node's filesystem path (hostPath).
- Git repository
- Cloud provider-specific storage (Google Compute Engine Persistent Disk, Amazon Web Services Elastic Block Store Volume, Microsoft Azure Disk Volume.
- Various network storage (nfs, cinder, cephfs, iscsi, flocker, glusterfs, ...)
- Volumes used to expose certain Kubernetes resources and cluster information to the pod (configMap, secret, downwardAPI)
- persistentVolumeClaim A way to use a pre- or dynamically provisioned persistent storage.

Concept #8 — Volumes


Persistent Volume Claim
User
Volume
Pod
Kubernetes scheduler does the binding
Network storage
Kubernetes Persistent Volume
Sysadmin

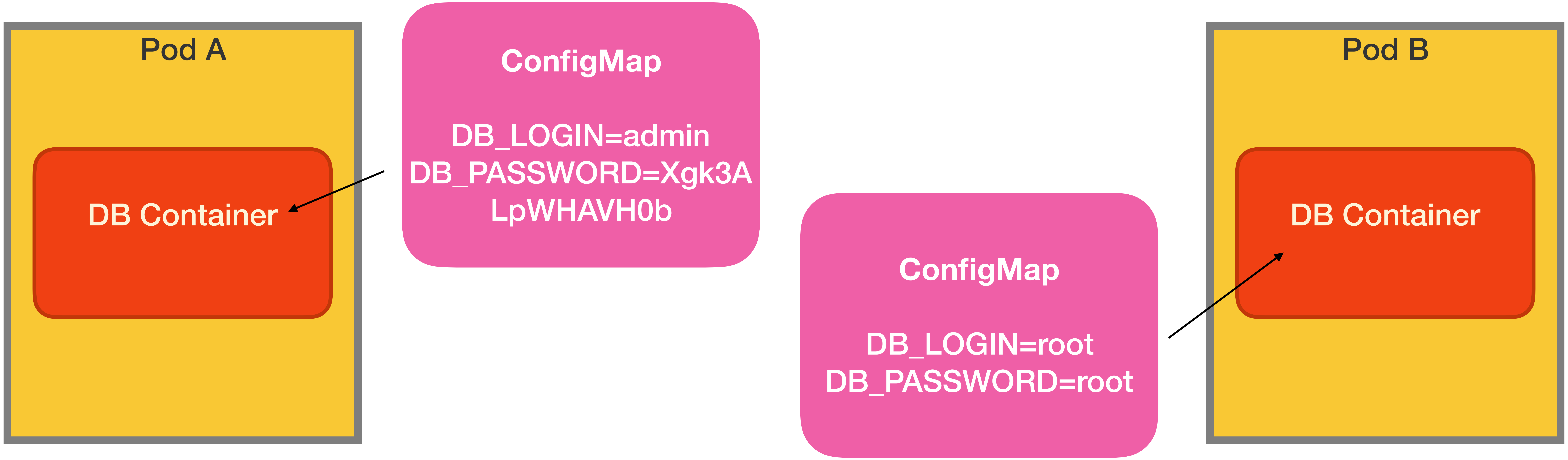
ECLIPSE FOUNDATION

Concept #9 — ConfigsMap & Secret



ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable

Concept #9 — ConfigsMap & Secret



Pod A

DB Container

ConfigMap

DB_LOGIN=admin
DB_PASSWORD=Xgk3A
LpWHAVH0b

ConfigMap

DB_LOGIN=root
DB_PASSWORD=root

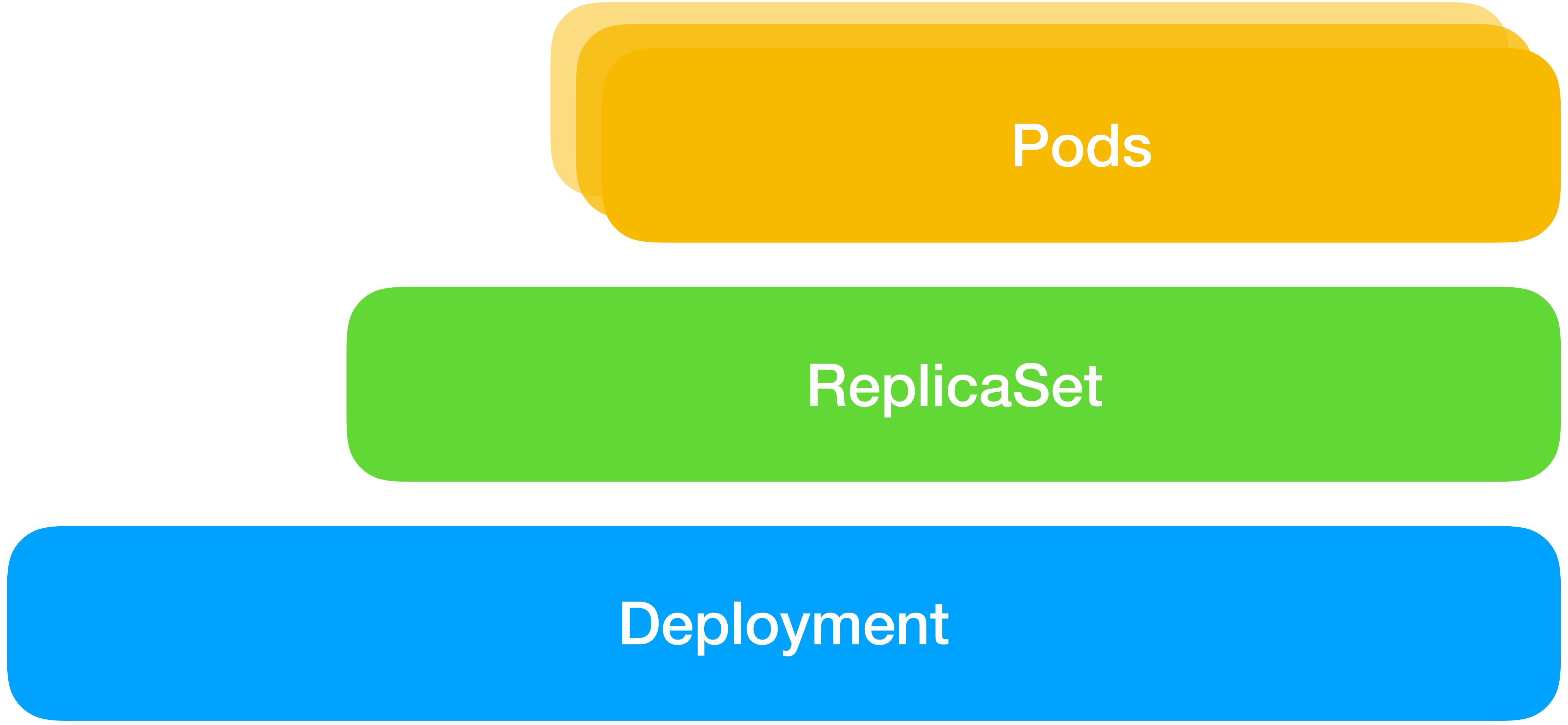
Pod B

DB Container

Concept #9 — ConfigsMap & Secret

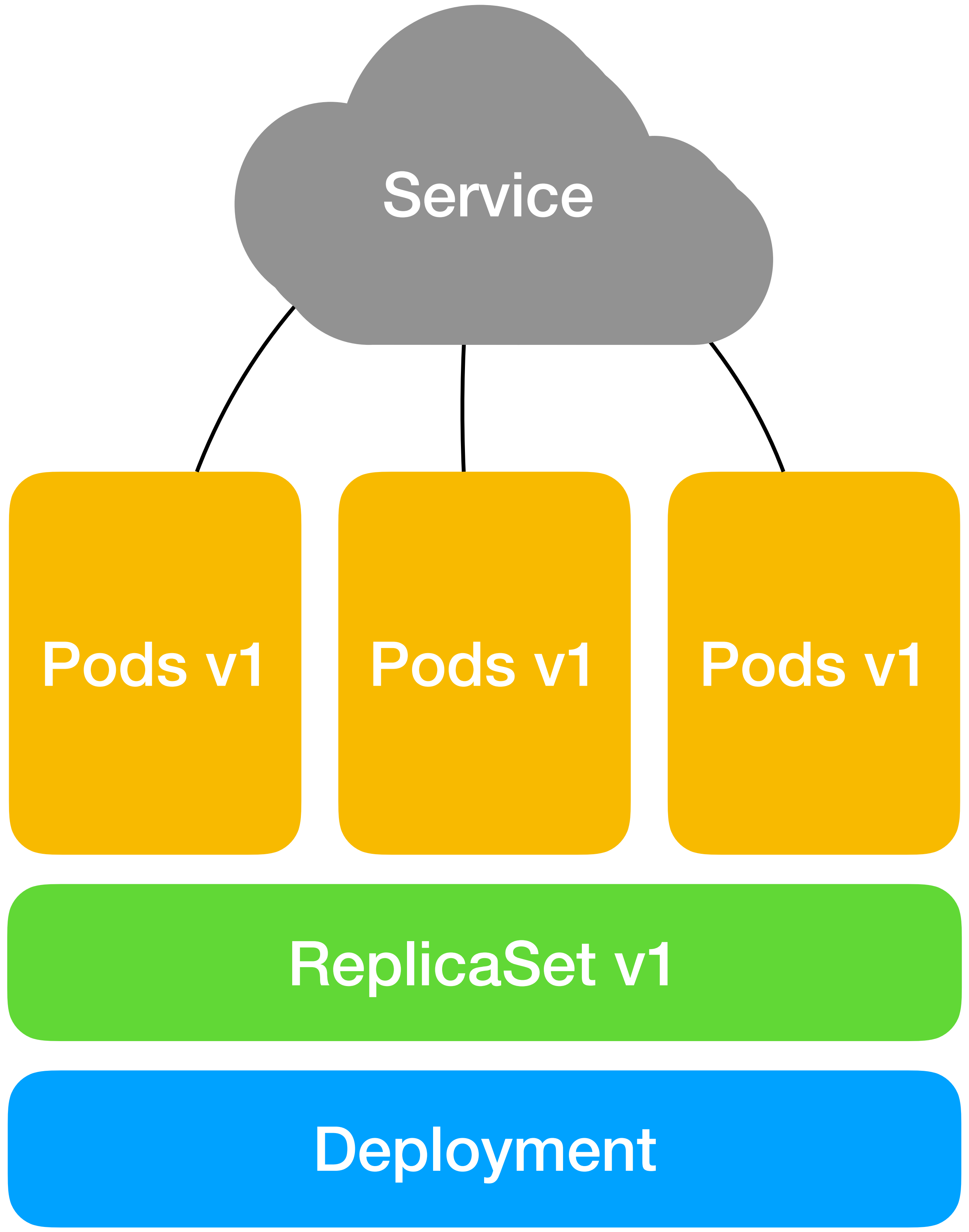
- Secrets are much like ConfigMaps but to store certs, password, etc.
- Kubernetes makes sure each Secret is only distributed to the nodes that run the pods that need access to the Secret.
- Secrets are always stored in memory (tmpfs) on the nodes and never written to physical storage.
- Secrets are stored encrypted in etcd

Concept #10 — Deployment

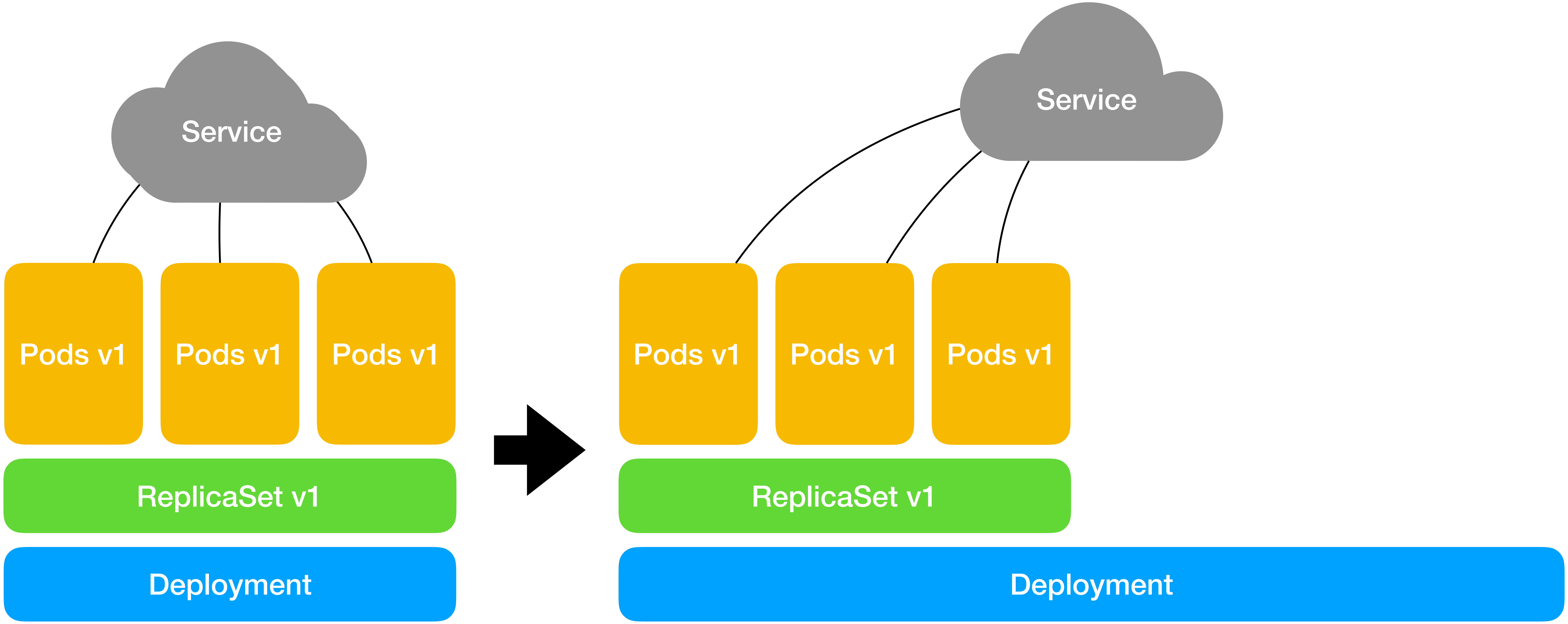


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

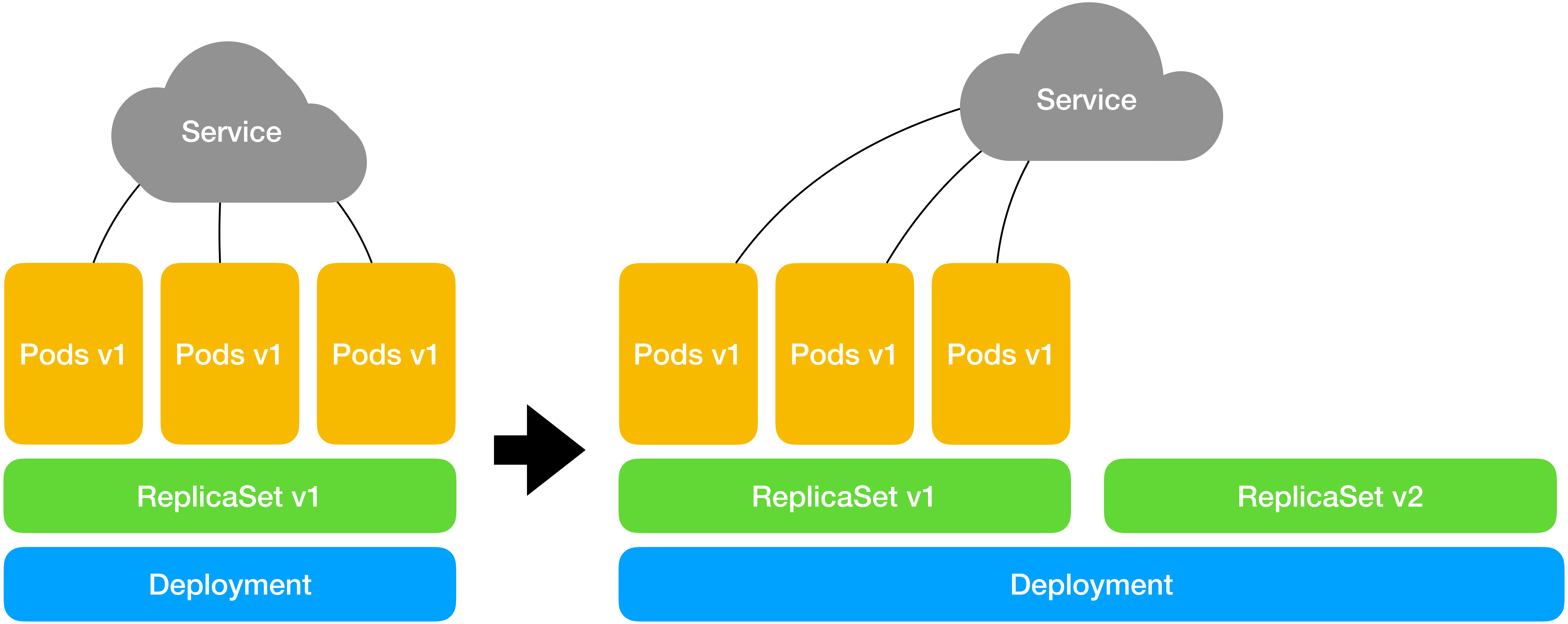
Concept #10 — Deployment



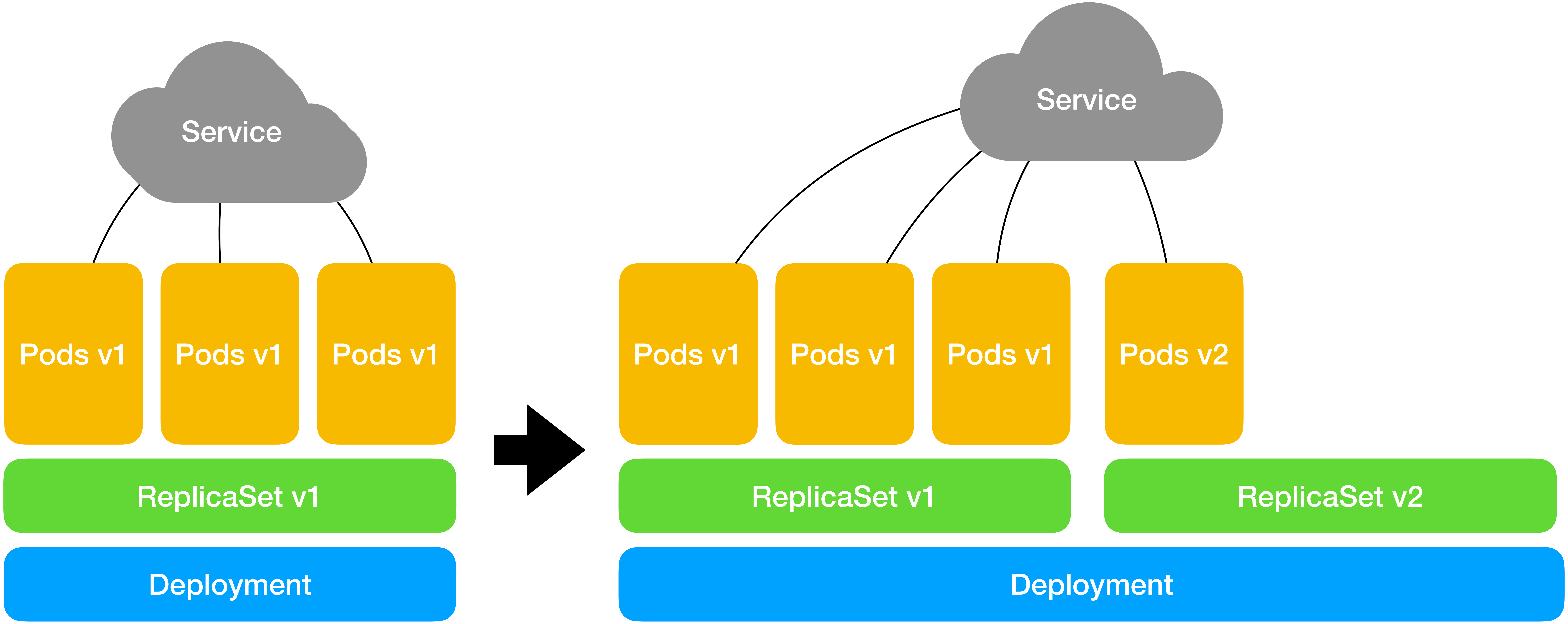
Concept #10 — Deployment



Concept #10 — Deployment

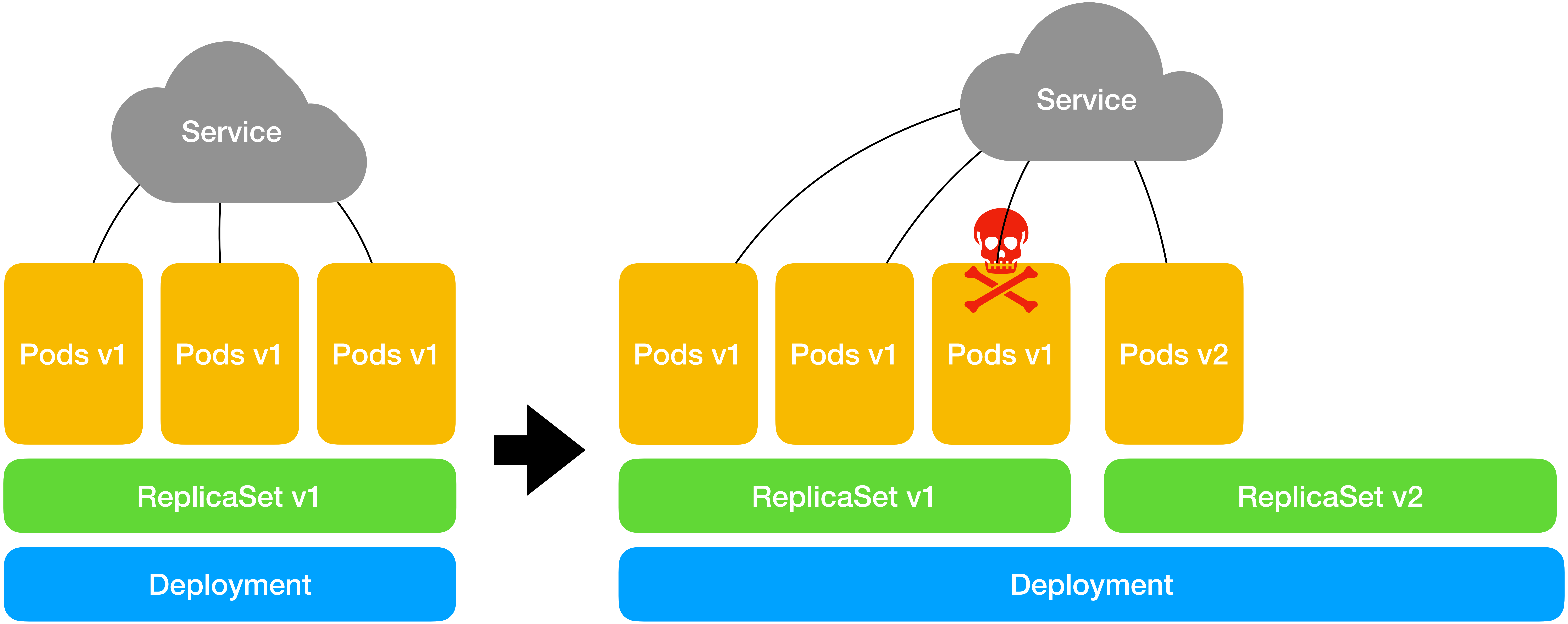


Concept #10 — Deployment

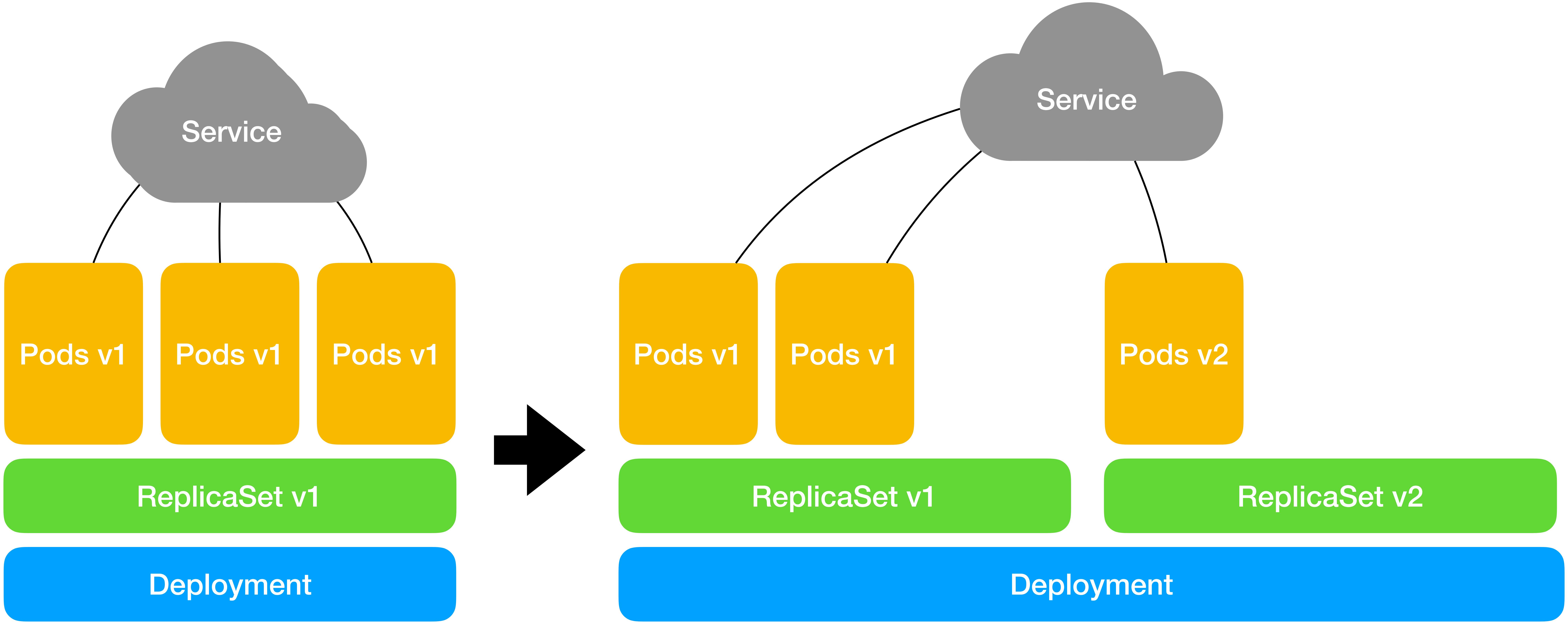


ECLIPSE FOUNDATION

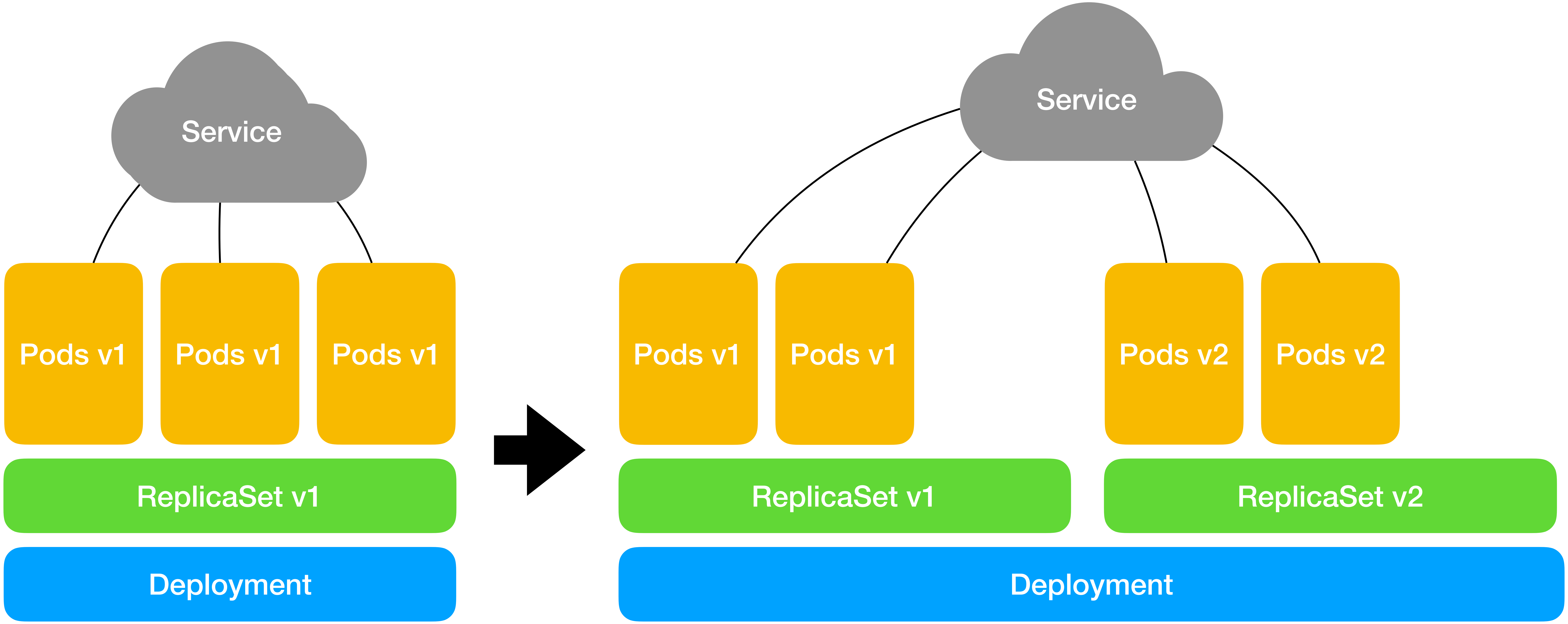
Concept #10 — Deployment



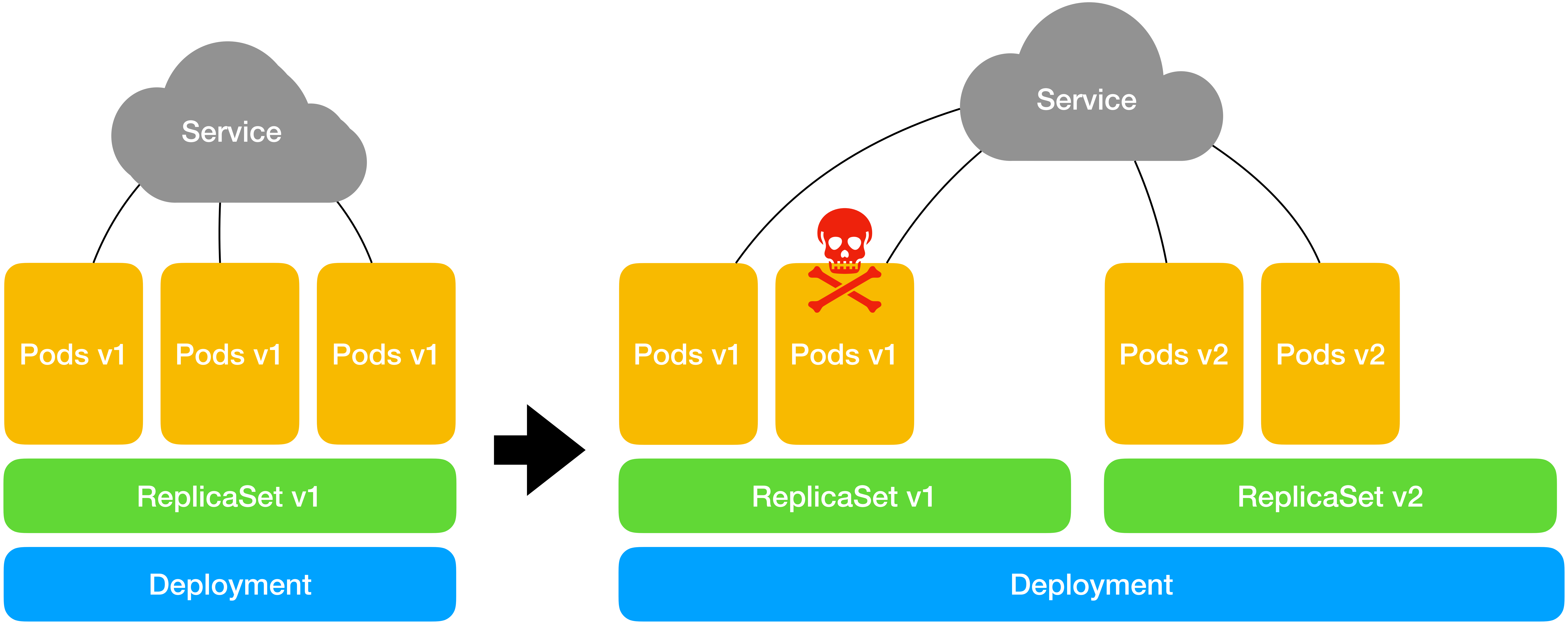
Concept #10 — Deployment



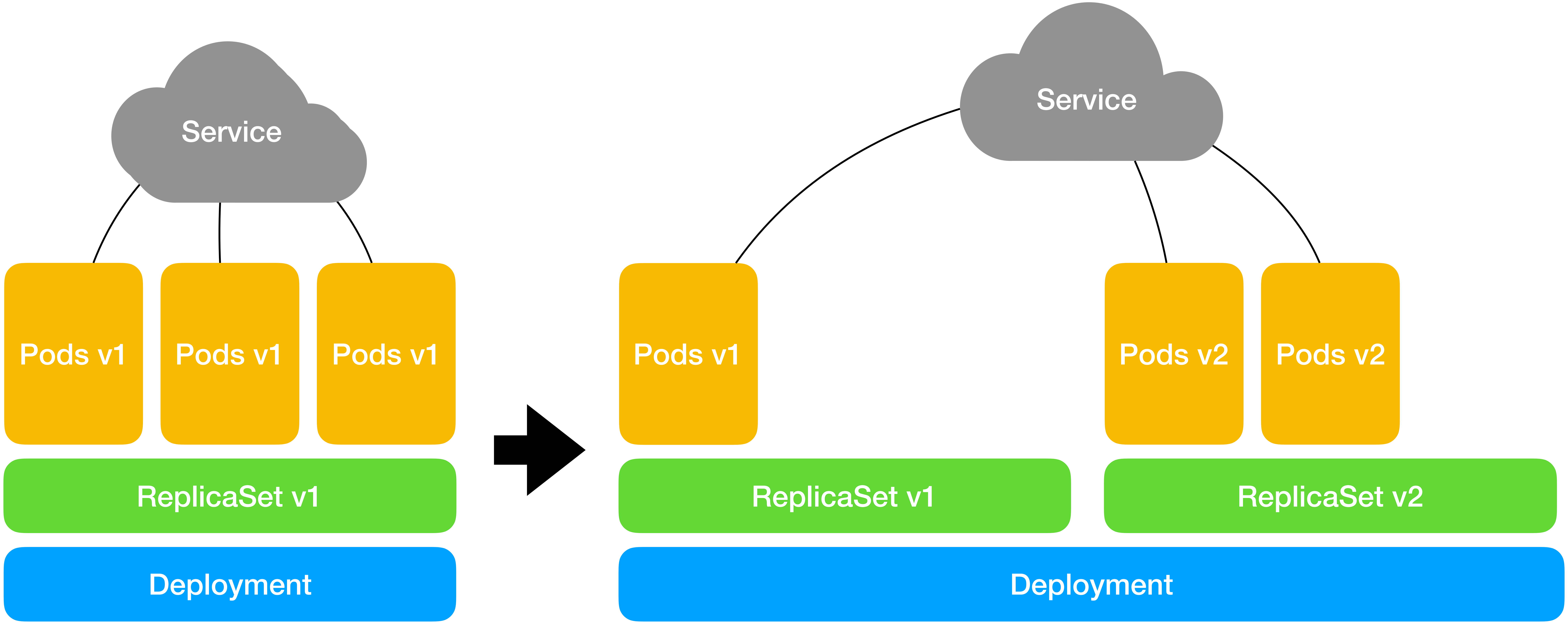
Concept #10 — Deployment



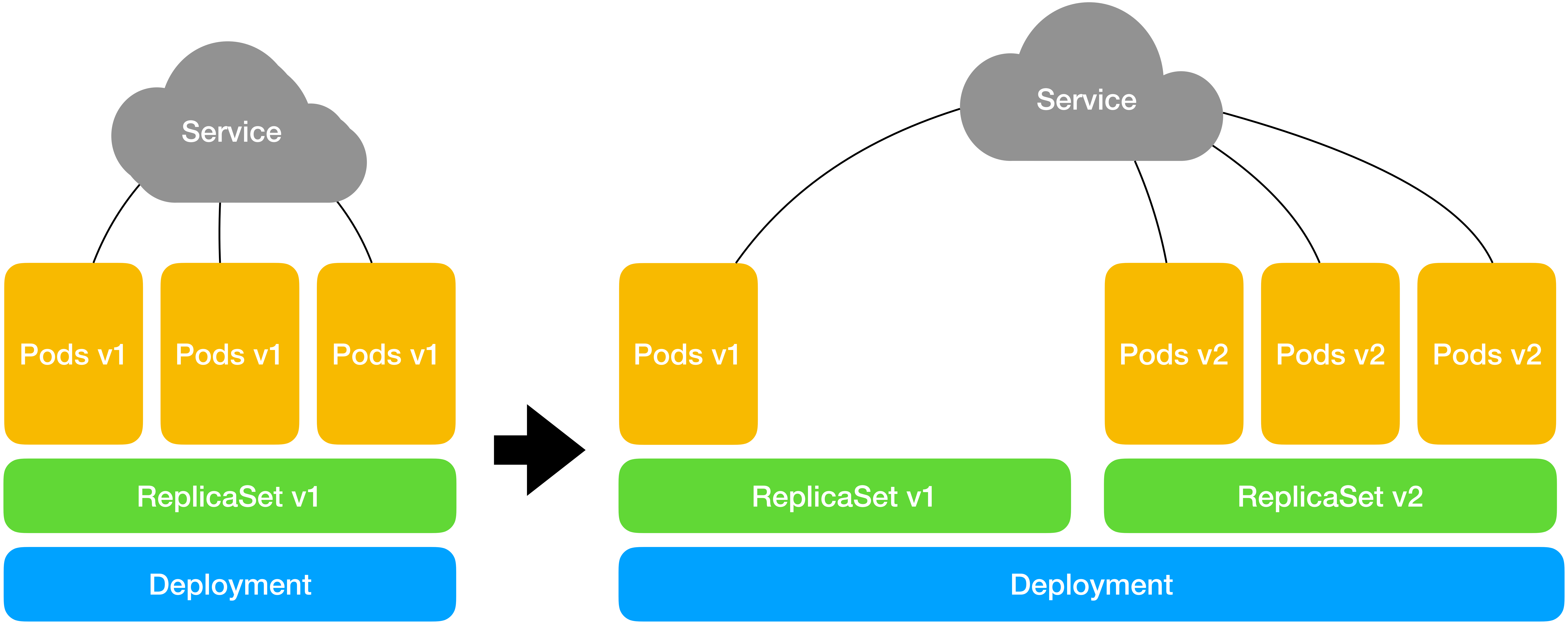
Concept #10 — Deployment



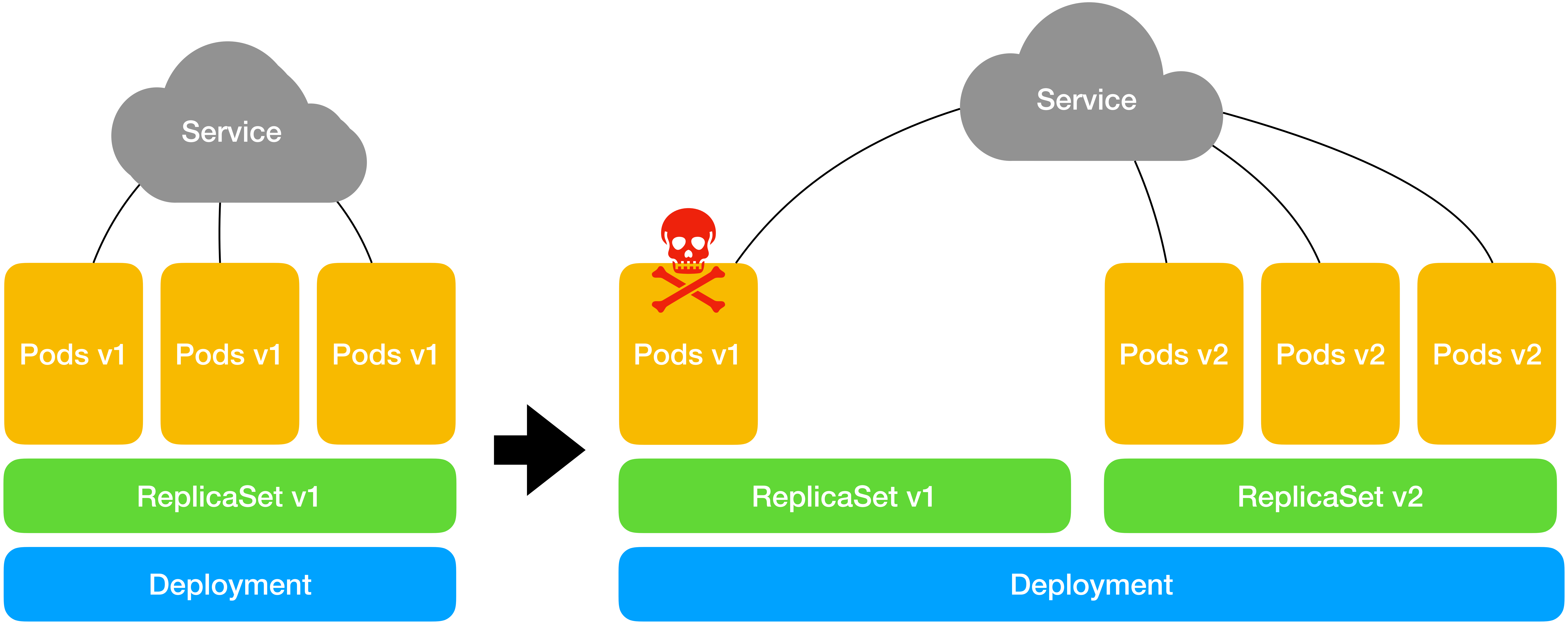
Concept #10 — Deployment



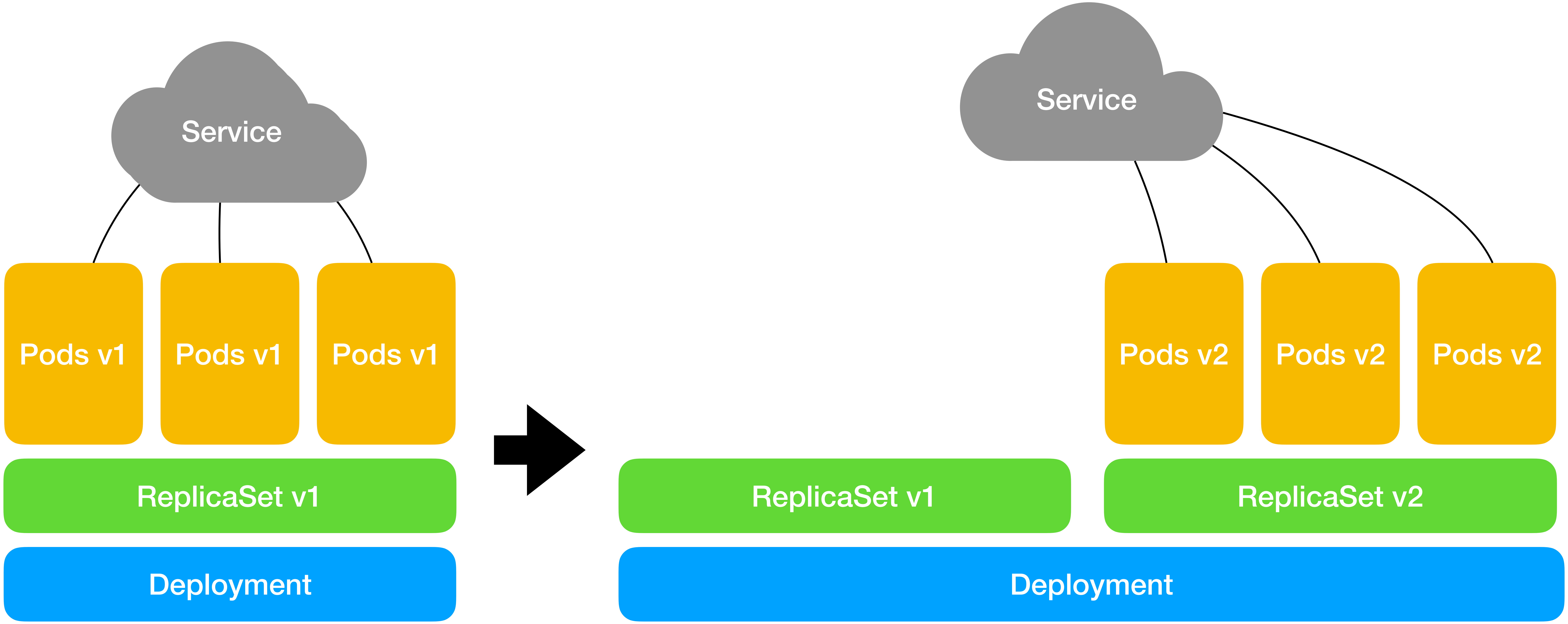
Concept #10 — Deployment



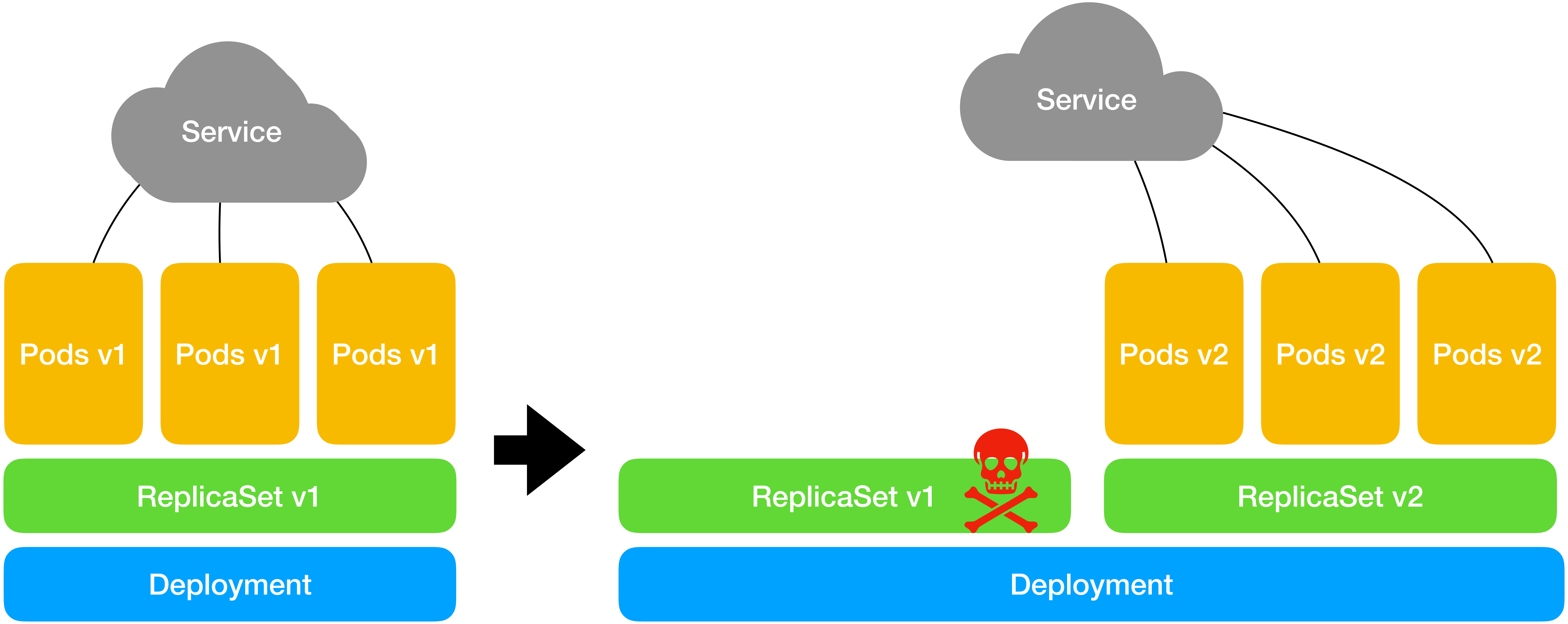
Concept #10 — Deployment



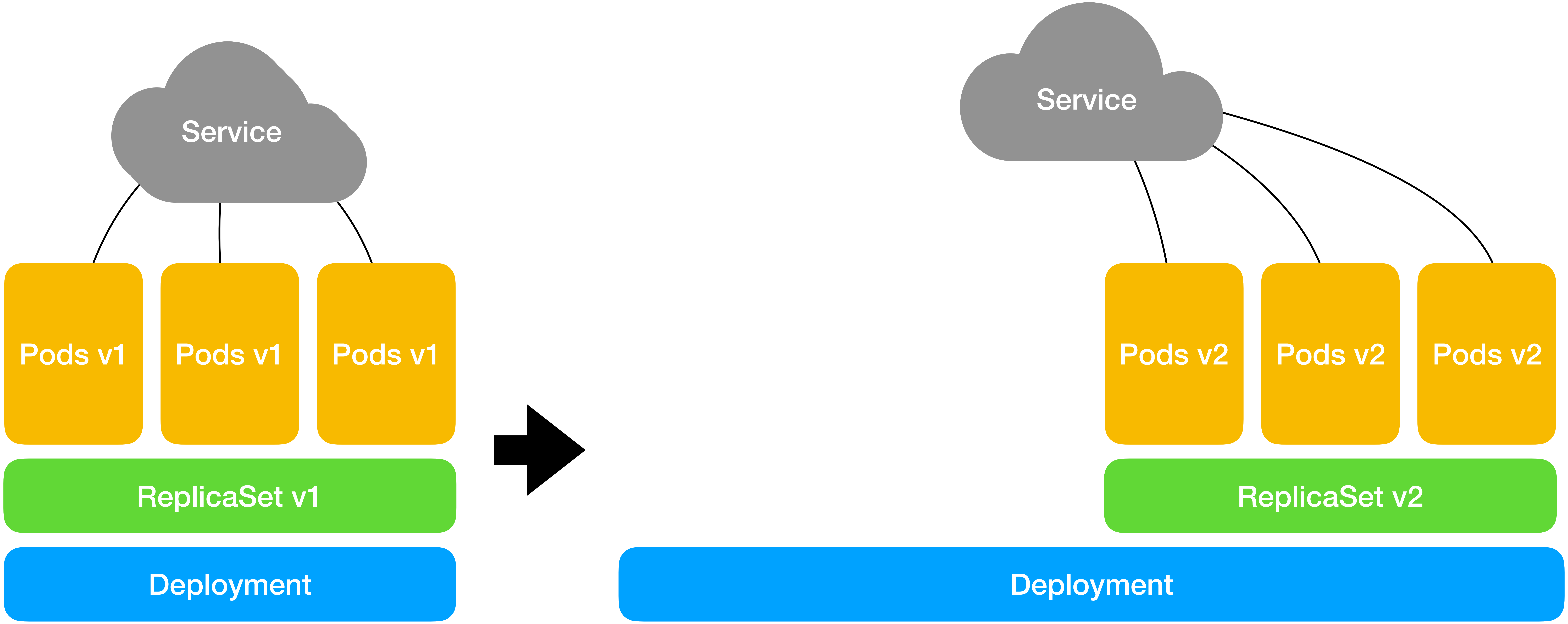
Concept #10 — Deployment



Concept #10 — Deployment



Concept #10 — Deployment



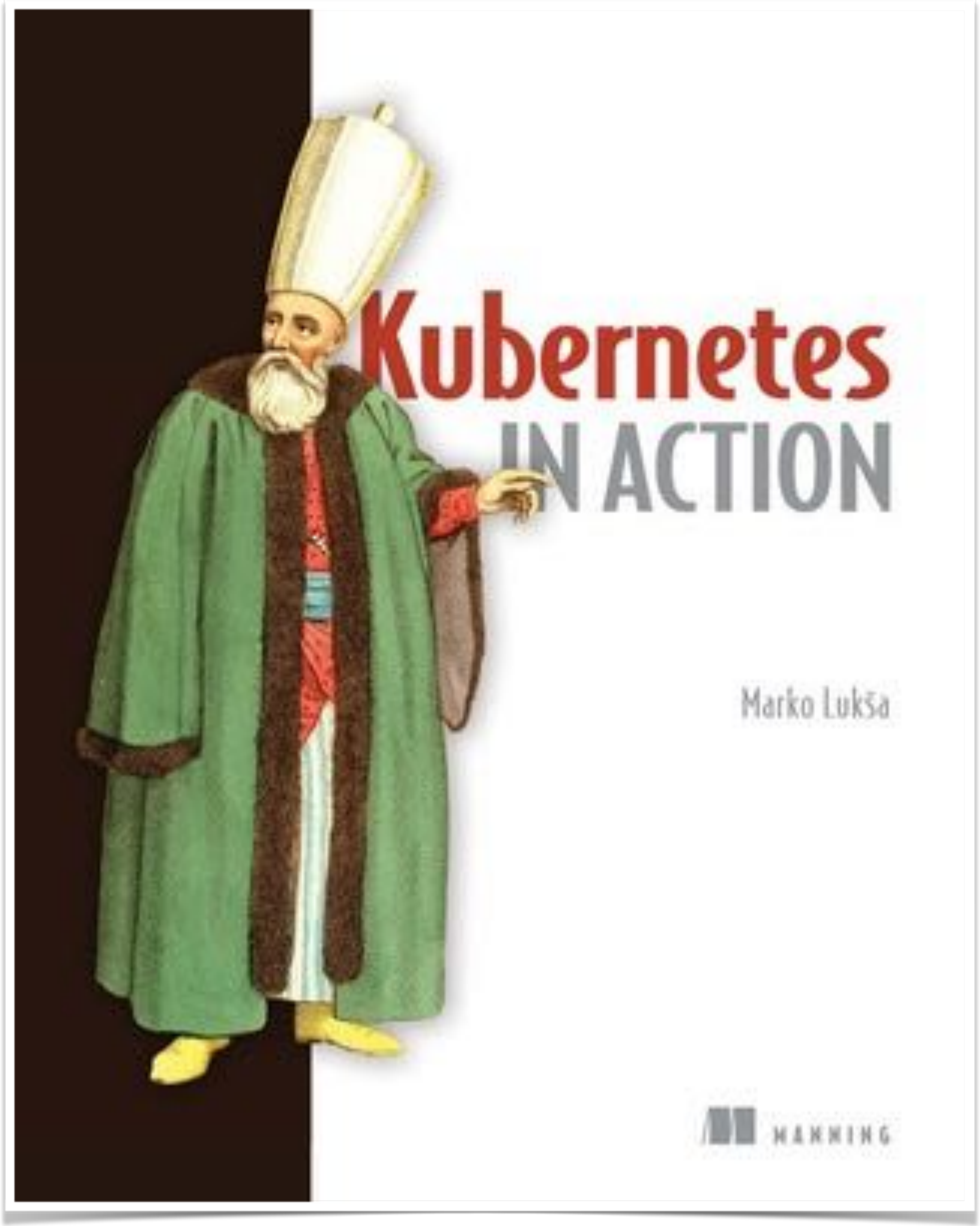
And More...

- Requesting and limiting resources of pod's containers (cpu, memory), QoS
- StatefulSets, e.g. to run a replicated mySQL DB with automatic failover (cold standby).
- HA for K8s Control Plane
- RBAC to secure API server, nodes and the network
- Autoscaling of pods and nodes
- Cluster federation (multi-zone, multi-cloud provider K8s clusters)

Questions?

Kubernetes In Action

Marko Luksa, Manning Publications,
ISBN: 9781617293726



Key takeaways

- Kubernetes is a cluster operating system
- Defines a sound set of concepts to deploy and manage resilient container-based applications
- Enables better usage of hardware resources
- Very dynamic community, which uses Kubernetes extensibility to add even more values
  - e.g., service mesh management platform Istio (https://istio.io/)

Thank You!

@mikbarbero