

Core internal principles of Kubernetes

Vašek Šraier

Vašek Šraier

Who am I?

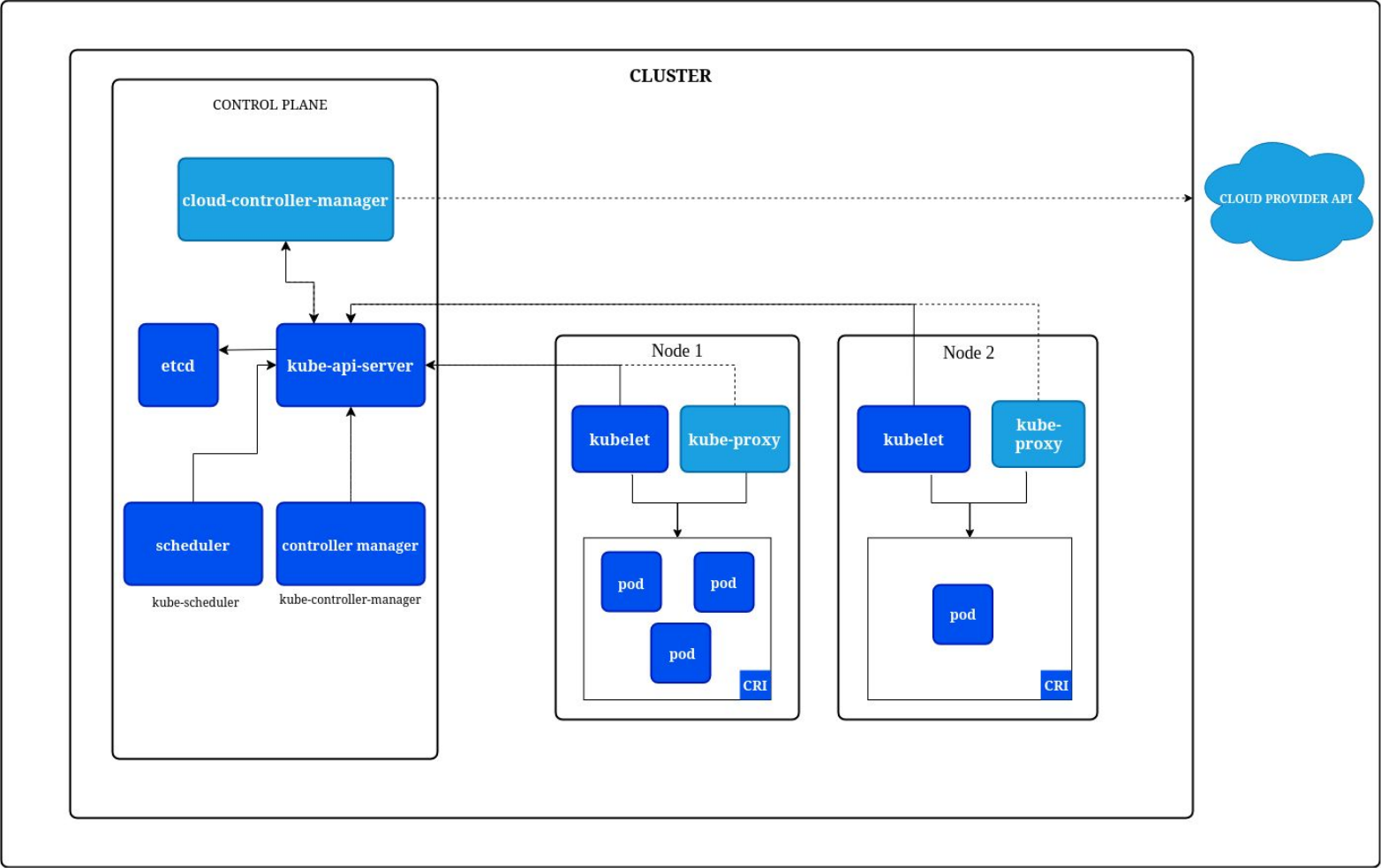
- software engineer @ Everpure
- organizer of programming summer camp Protab
- adventurer

My experience with Kubernetes?

- I've never used Kubernetes in production.
- At work, we are building a distributed system built on Kubernetes principles and architecture.

Contact? vsq.cz





Pod

- a set of containers sharing network namespace
- primitive that we are managing

Let's build Kubernetes from scratch

Goals

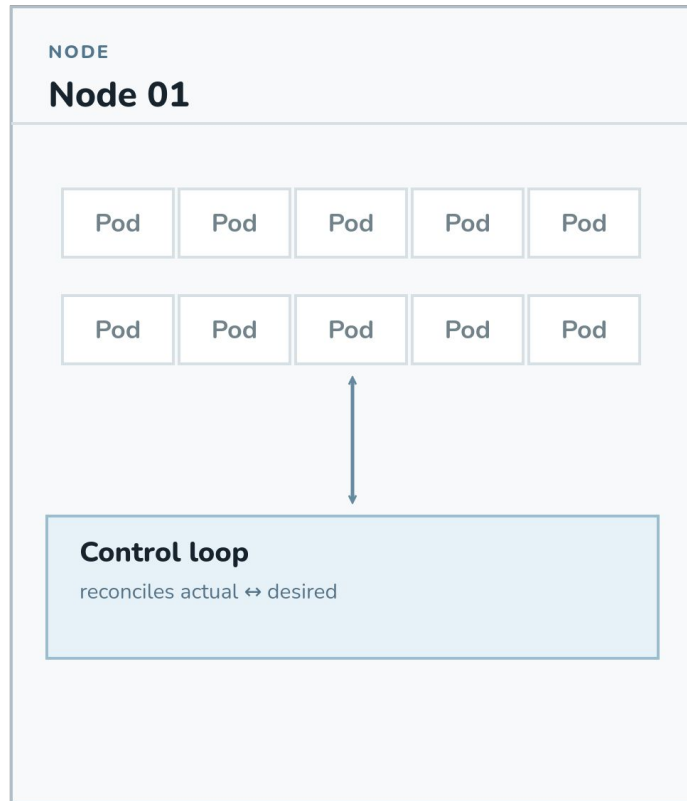
- Pod management
- Dynamically configurable
- Fault resistant

Non-goals

- Discuss networking
- Discuss container runtime

Control loop to keep 10 pods running

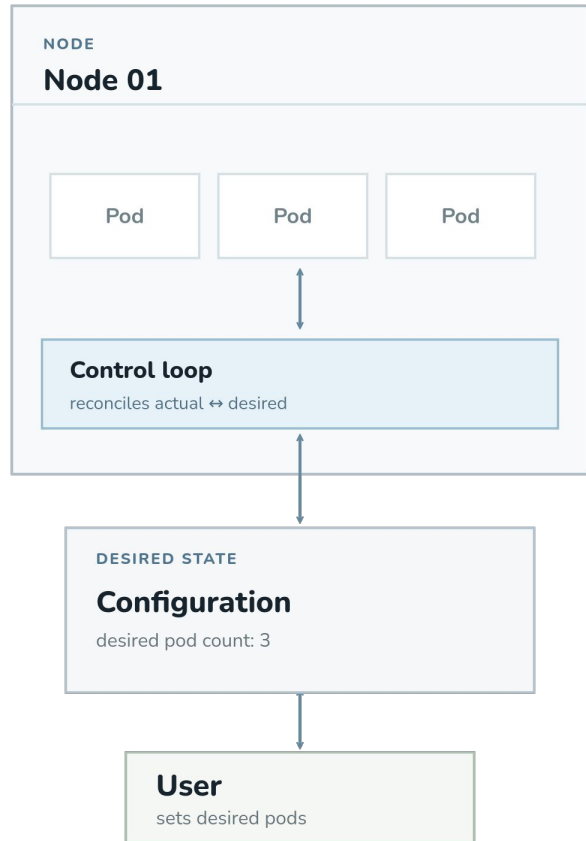
```
while True:
    running = get_running()
    missing = 10 - len(running)
    if missing > 0:
        spawn_pods(count= missing)
    elif missing < 0:
        destroy_pods(count= -missing)
```



Control loop to keep some number of pods running

```
while True:
    running = get_running()
    desired = config.get_desired_count()
    missing = desired - len(running)

    if missing > 0:
        spawn_pods(count= missing)
    elif missing < 0:
        destroy_pods(count= -missing)
```

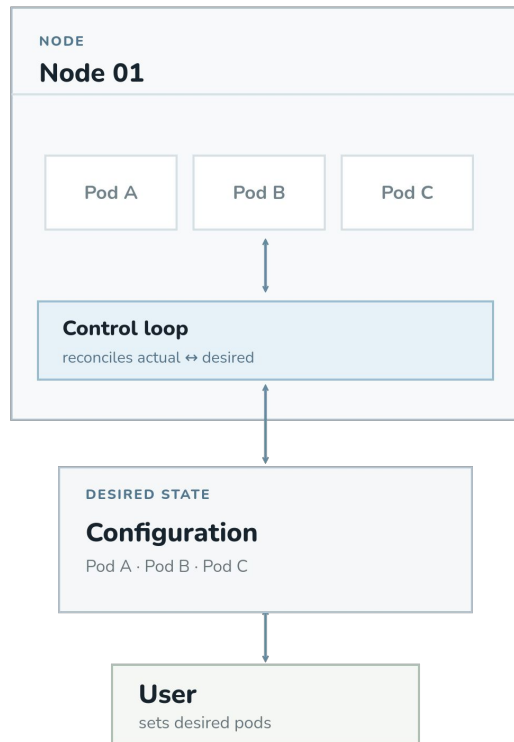


Control loop supporting different pods

```
while True:
    running = get_running()

    # start missing pods
    for pod in config.get_pods():
        if pod not in running:
            spawn_pod(pod)

    # destroy extra pods
    for pod in running:
        if pod not in config.get_pods():
            destroy_pod(pod)
```



Changes take time

- Kubernetes record what you want
- Control loops make it happen
- The real state may (or may not) catch up later

Desired and actual state **eventually** converge (or not)

Configuration format

- let's store configuration in objects
 - object ~ JSON-like document
- object format
 - kind
 - metadata
 - name
 - spec
 - status
- writing an object is an atomic operation with optimistic concurrency control
 - similar to compare-and-swap operations

A pod object example

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: nginx
```

Control loop configurable with objects

```
# reconciler
```

```
def reconcile_pods():  
    running = get_running()  
  
    # start pods if needed  
    for pod in store.List(Pod):  
        if pod not in running:  
            spawn_pod(pod)  
  
    # destroy extra pods  
    for pod in running:  
        if pod not in  
store.List(Pod):  
            destroy_pod(pod)
```

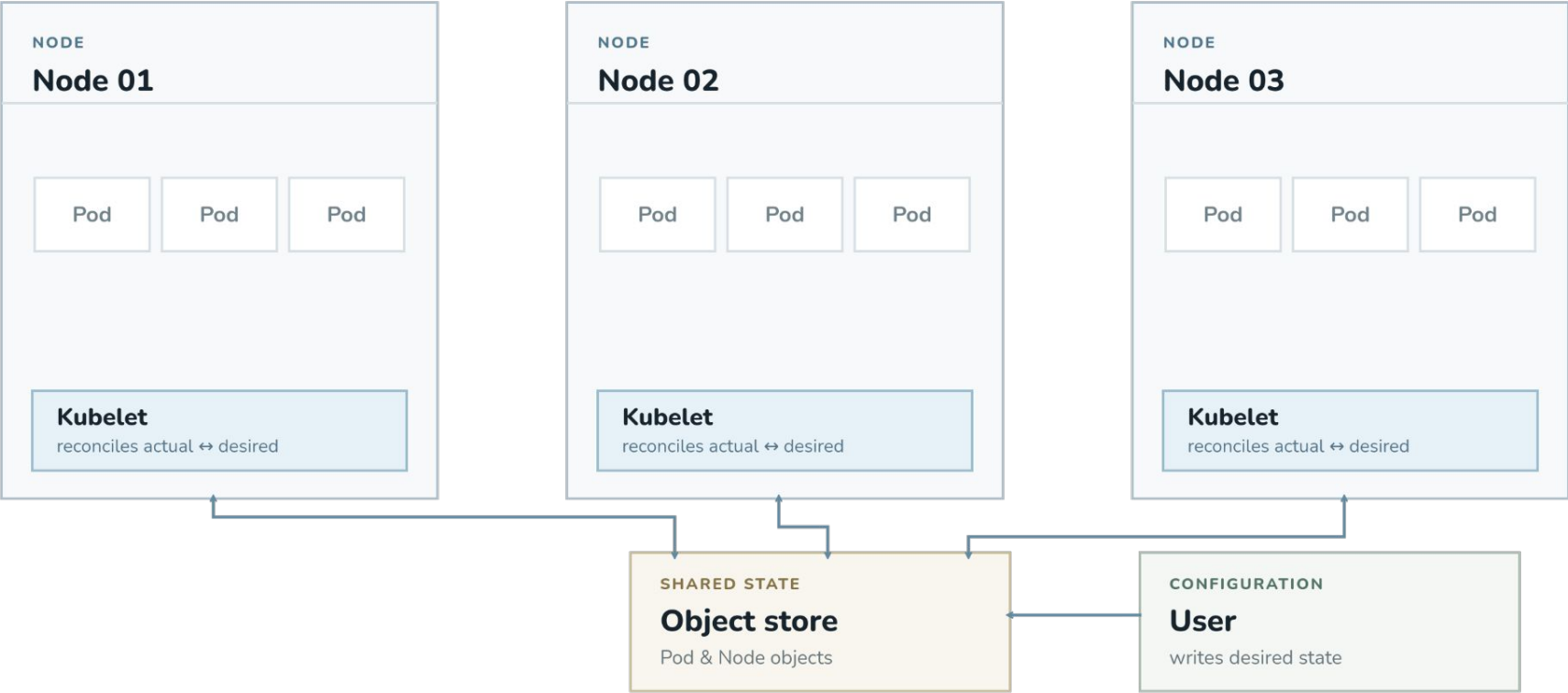
```
# controller
```

```
while True:  
    reconcile_pods()
```

Controller, reconciler and kubelet

- **reconciler**
 - a function that that adjusts the *actual state* to match the *desired state* (spec)
 - the inner part of the control loop
- **controller**
 - a group of reconcilers and the necessary scaffolding to run the efficiently (e.g. execute them on events, not in an infinite loop)
- **kubelet**
 - process running on each node in Kubernetes
 - ensures that Pods are running on the node
 - technically a specific type of controller

More machines



A pod object example

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  nodeName: worker-node-1
  containers:
    - name: my-container
      image: nginx
```

Observing the actual state

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  nodeName: worker-node-1
  containers:
    - name: my-container
      image: nginx
status:
  phase: Running
  podIP: 10.244.0.15
  hostIP: 192.168.1.10
  startTime: "2026-10-03T21:00:00Z"
  conditions:
    - type: Ready
      status: "True"
  containerStatuses:
    - name: my-container
      ready: true
      restartCount: 0
      image: nginx
      state:
        running:
          startedAt: "2026-10-03T21:00:02Z"
```

status section in the object contains description of the actual (observed) state. Status can become stale.

spec: desired state

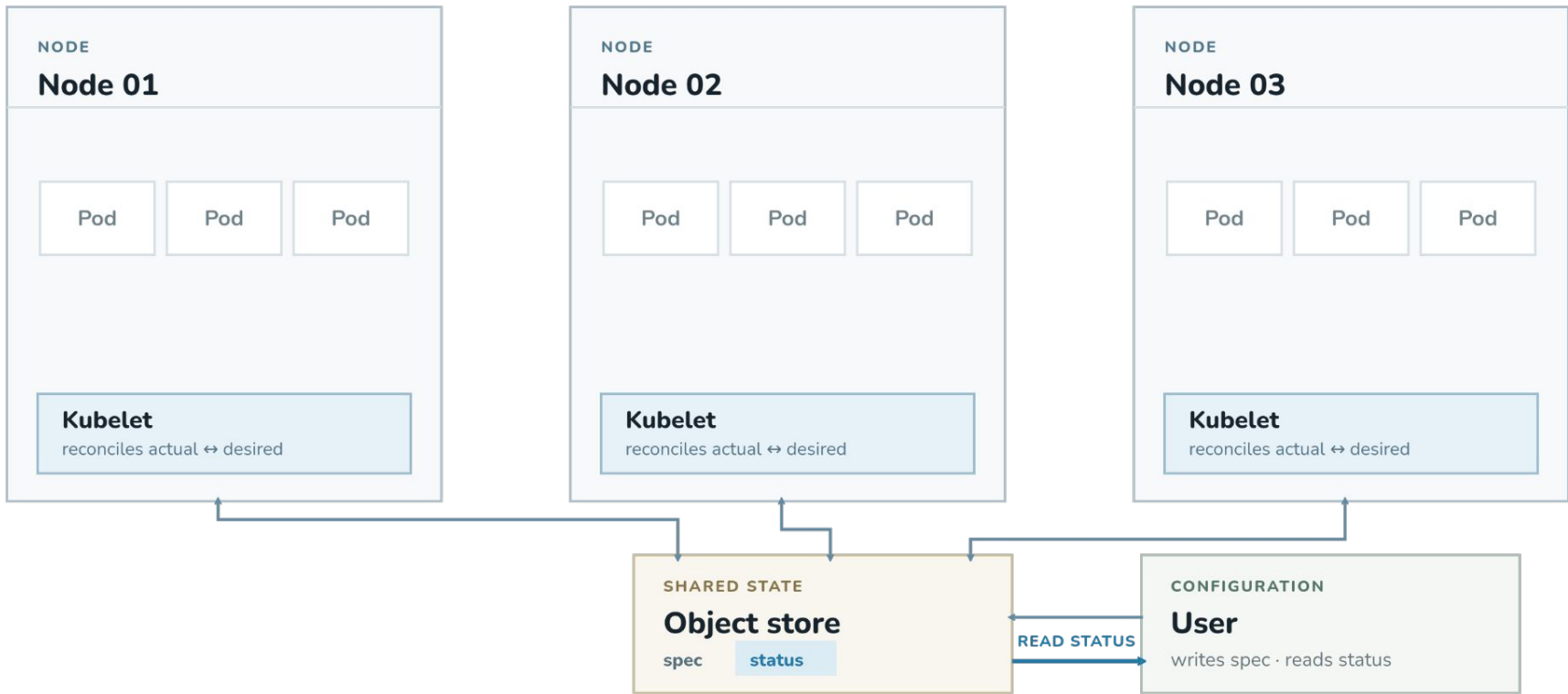
status: actual state

Node object

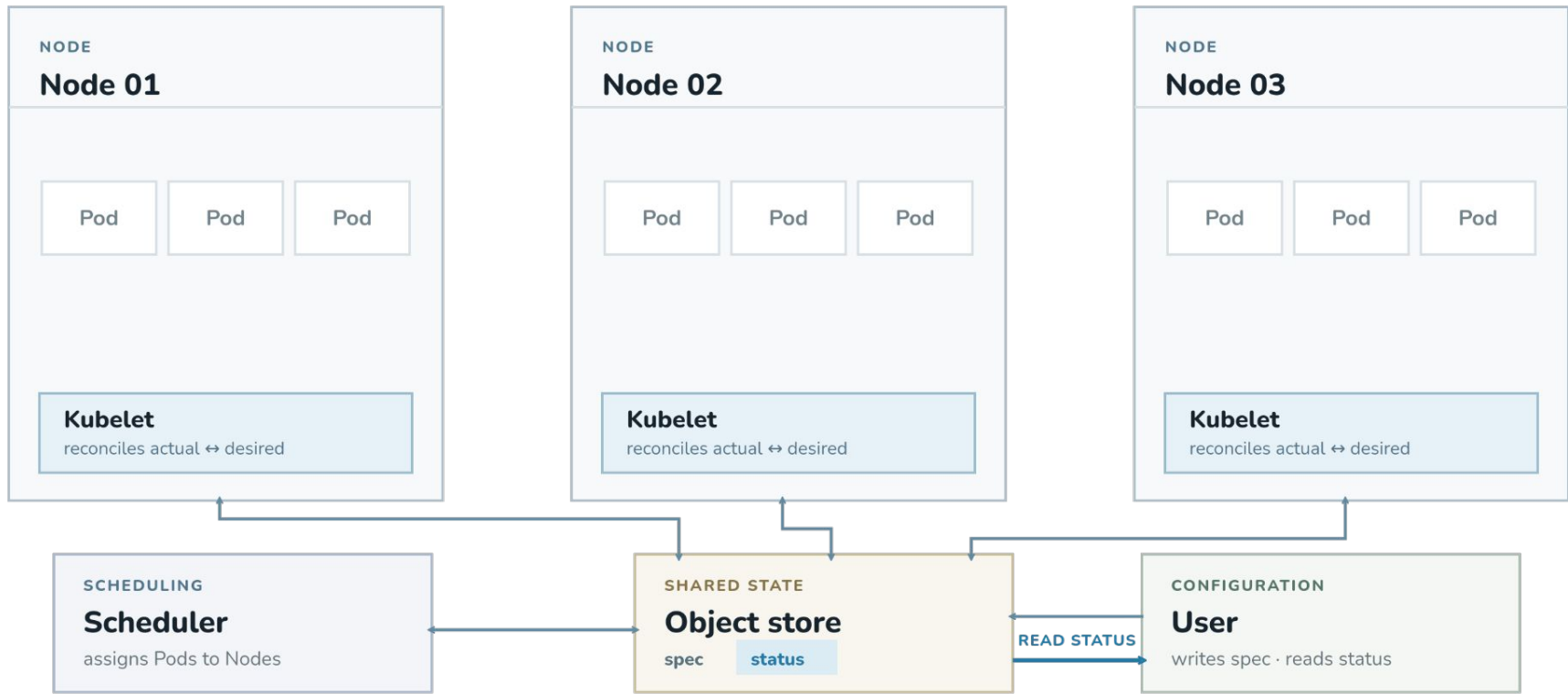
Node object contains information about the node capacity in the **status** section.

```
apiVersion: v1
kind: Node
metadata:
  name: worker-node-1
spec:
  podCIDR: 10.244.1.0/24
status:
  addresses:
    - type: InternalIP
      address: 192.168.1.10
    - type: Hostname
      address: worker-node-1
  conditions:
    - type: Ready
      status: "True"
  capacity:
    cpu: "4"
    memory: 16Gi
    pods: "110"
  nodeInfo:
    kubeletVersion: v1.31.0
    osImage: Ubuntu 24.04 LTS
    containerRuntimeVersion: containerd://1.7.20
```

Allocating pods to nodes becomes possible

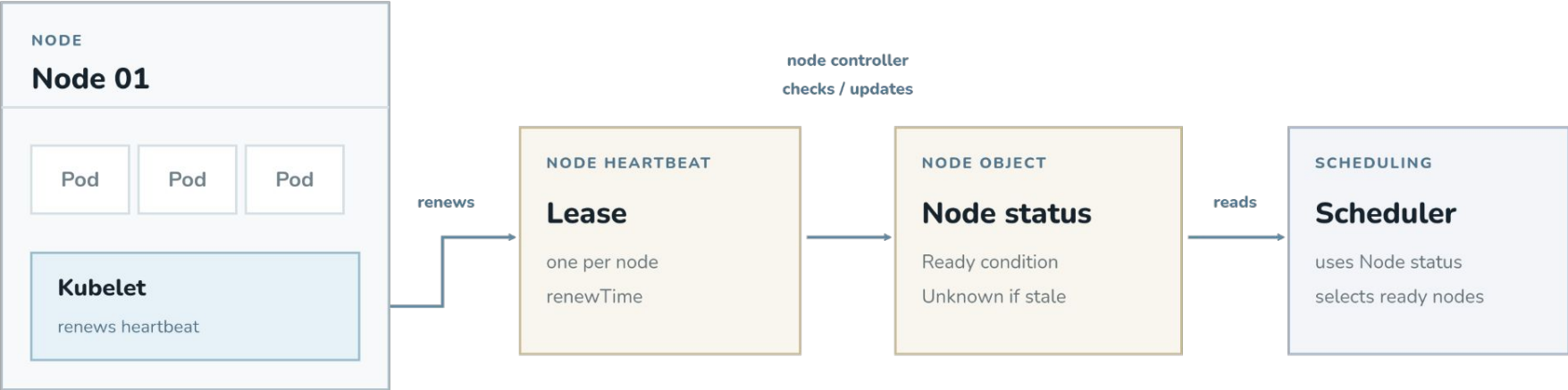


Automatic scheduling onto nodes

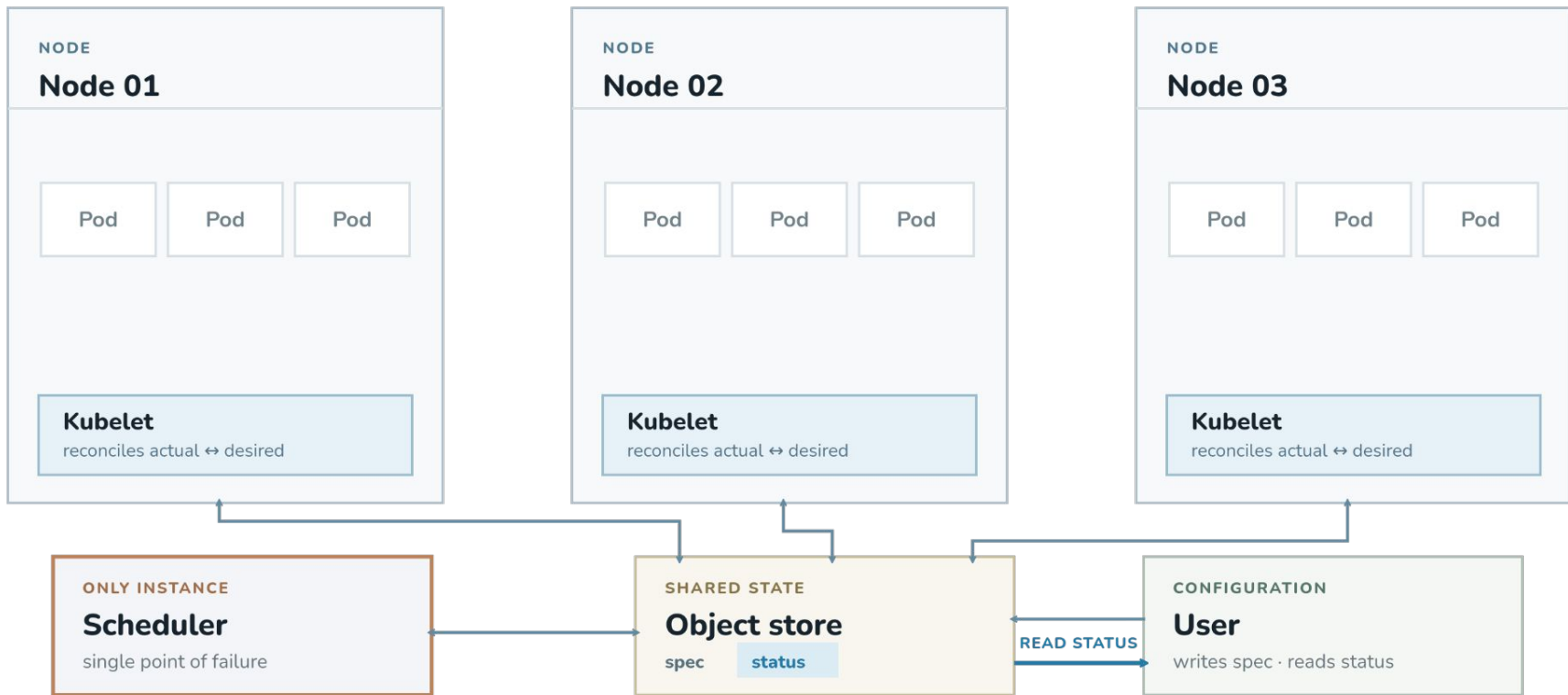


Note: I am mixing two components into one - the object store box contains API server in these diagrams.

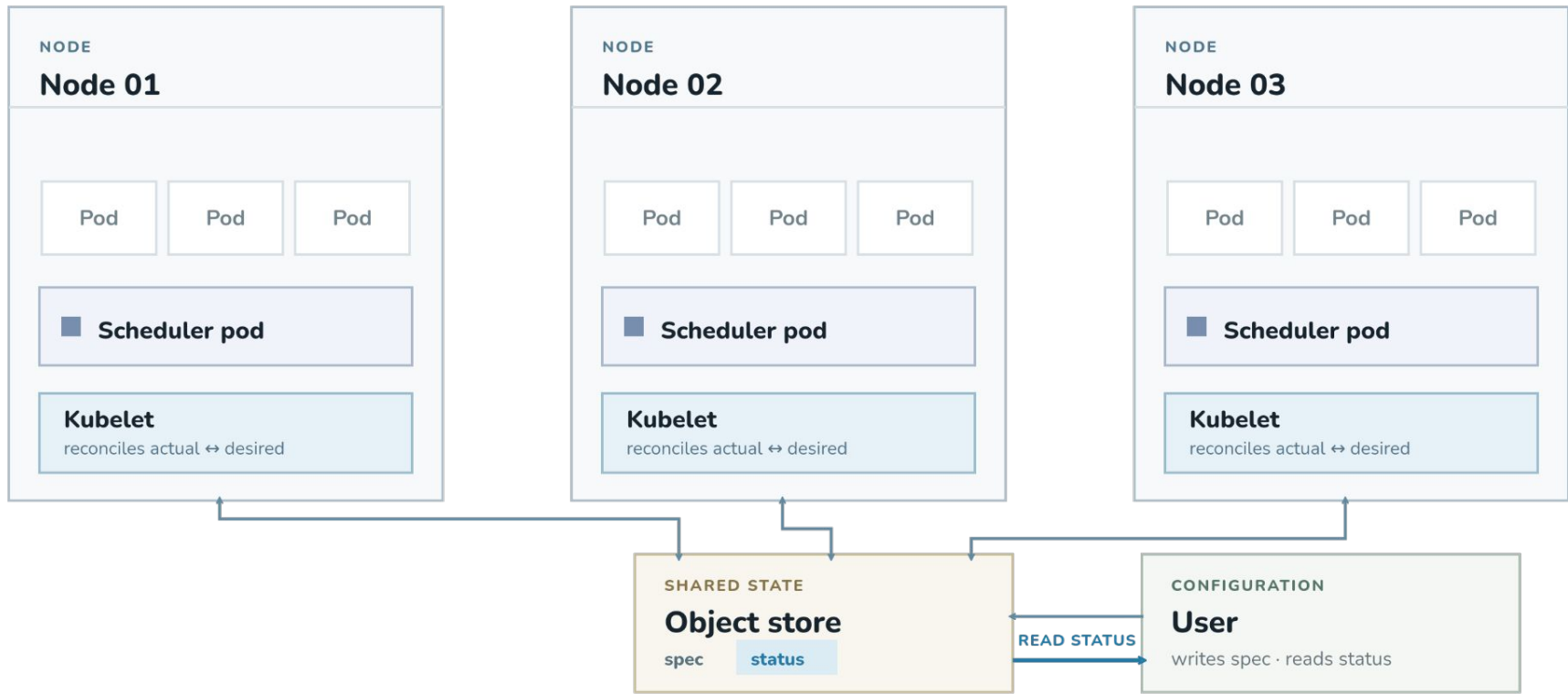
Node down?



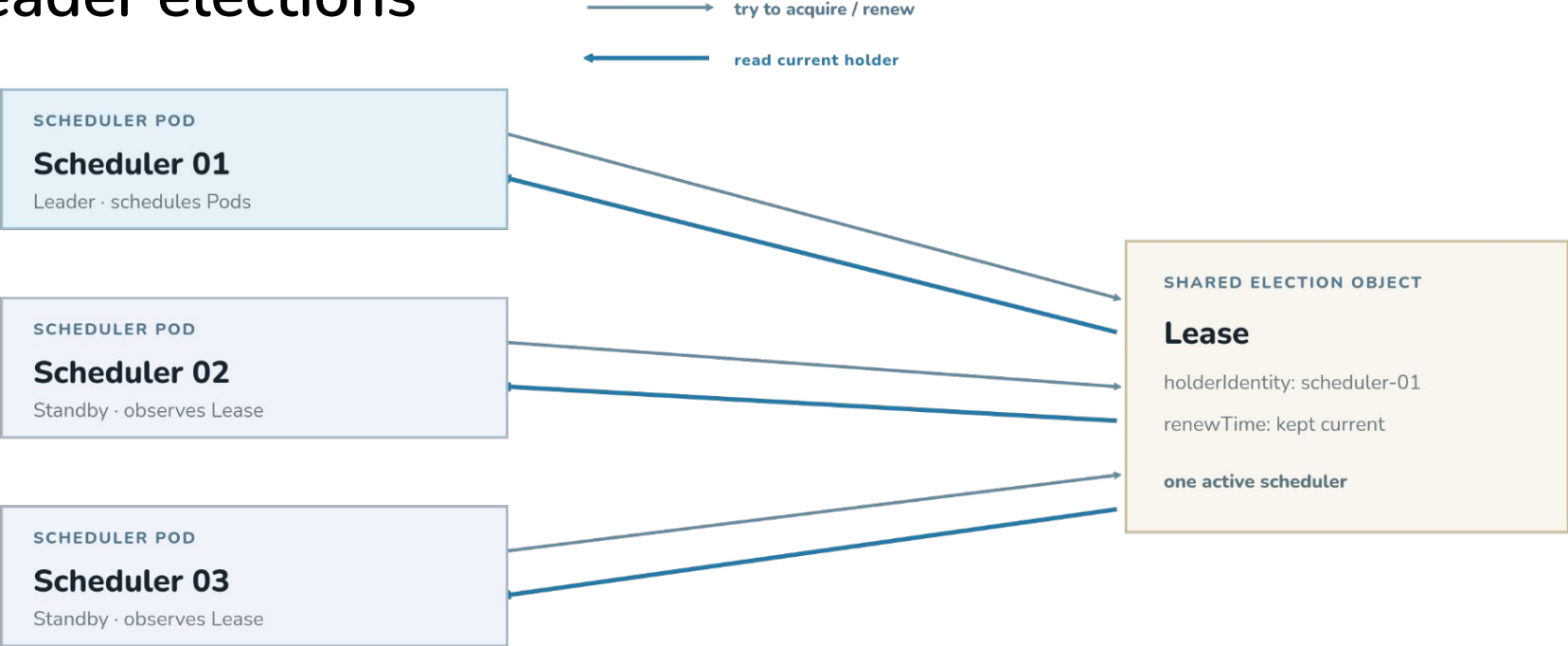
Eliminating SPoFs: Scheduler



Resilient scheduler



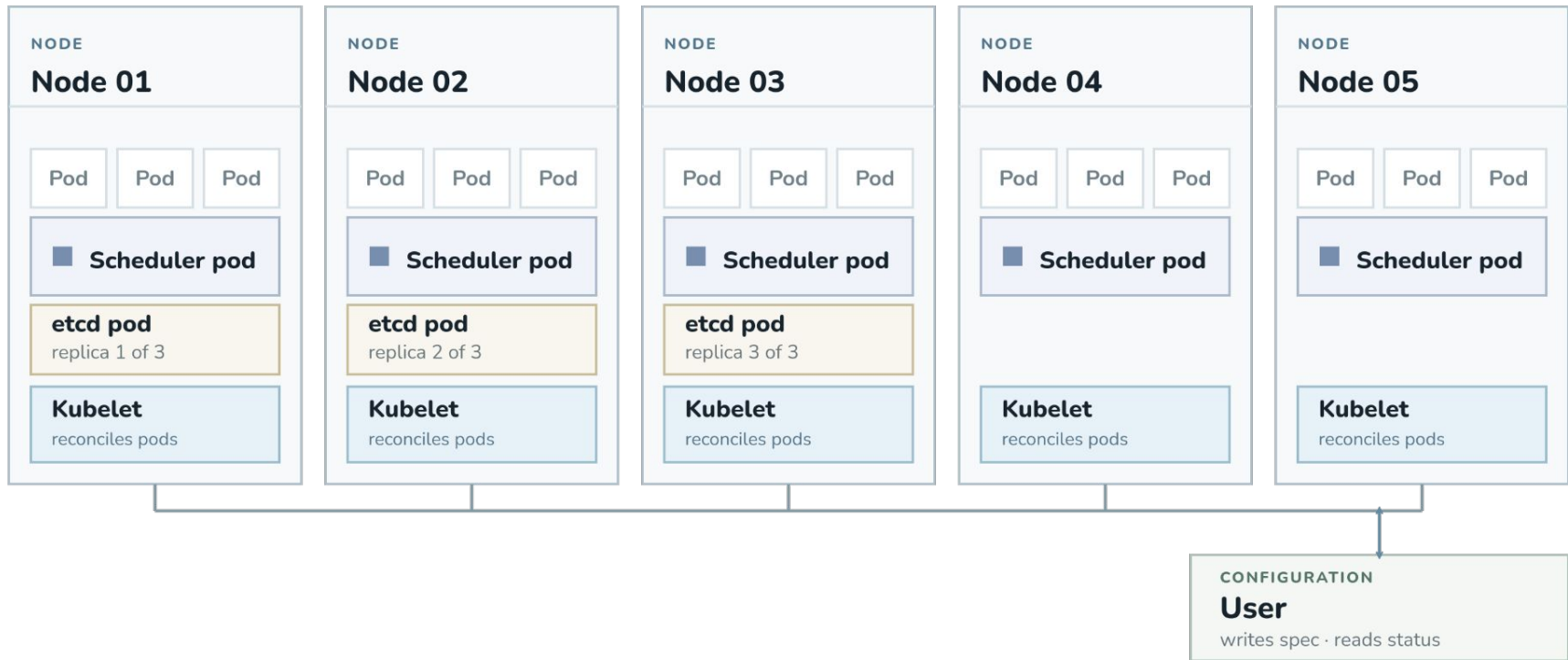
Leader elections



Highly-available control plane

- distributed database **etcd**
 - every write has to be confirmed by a quorum (majority)
 - requires quorum to continue operating
- etcd remains critical part of the cluster
 - higher number of etcd nodes → improved fault tolerance
 - less etcd nodes → better performance

Fault tolerant cluster



Recursive self-hosting of the control plane

- control plane components can run in containers
 - scheduler can be a pod
 - etcd can run in pods
- kubelet is the only host-level dependency

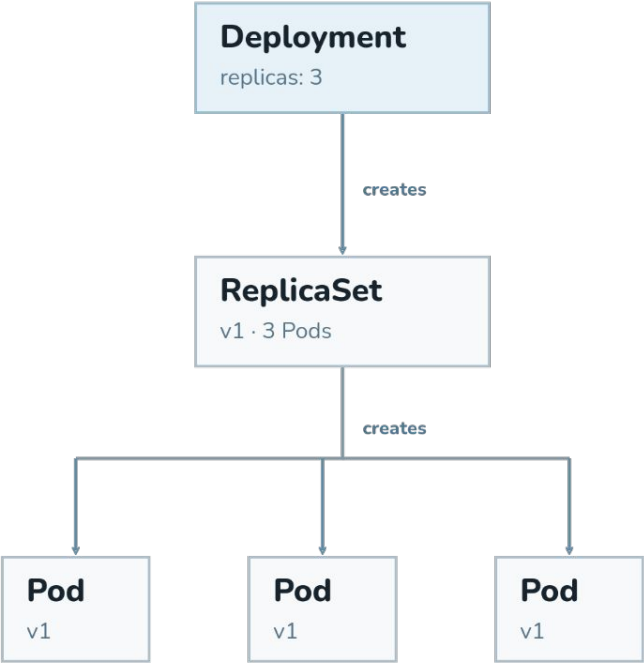
Self-hosting enables self-healing as long as quorum is intact.

Deployment and ReplicaSet

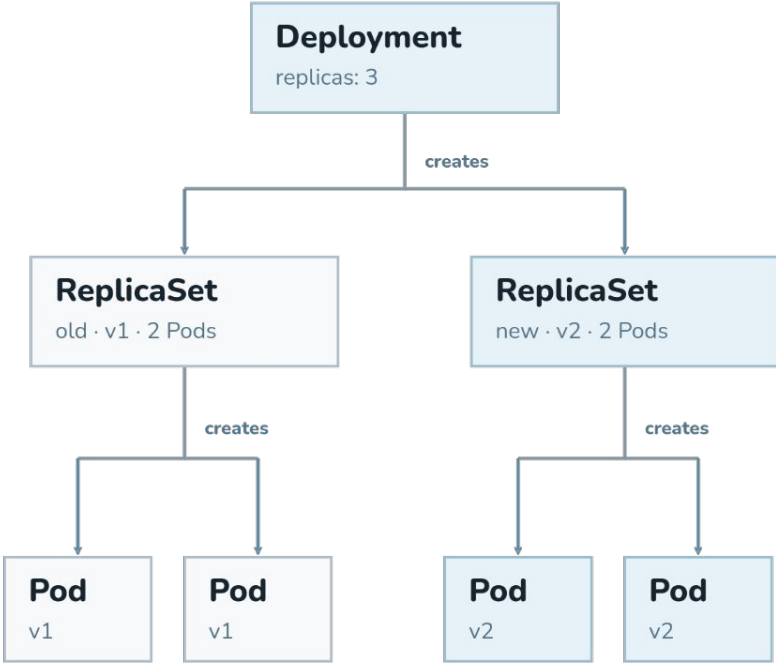
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-deployment-7c6b548b8
  labels:
    app: my-app
    pod-template-hash: 7c6b548b8
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
      pod-template-hash: 7c6b548b8
  template:
    metadata:
      labels:
        app: my-app
        pod-template-hash: 7c6b548b8
    spec:
      containers:
        - name: my-container
          image: nginx
```

Before upgrade



During rolling upgrade



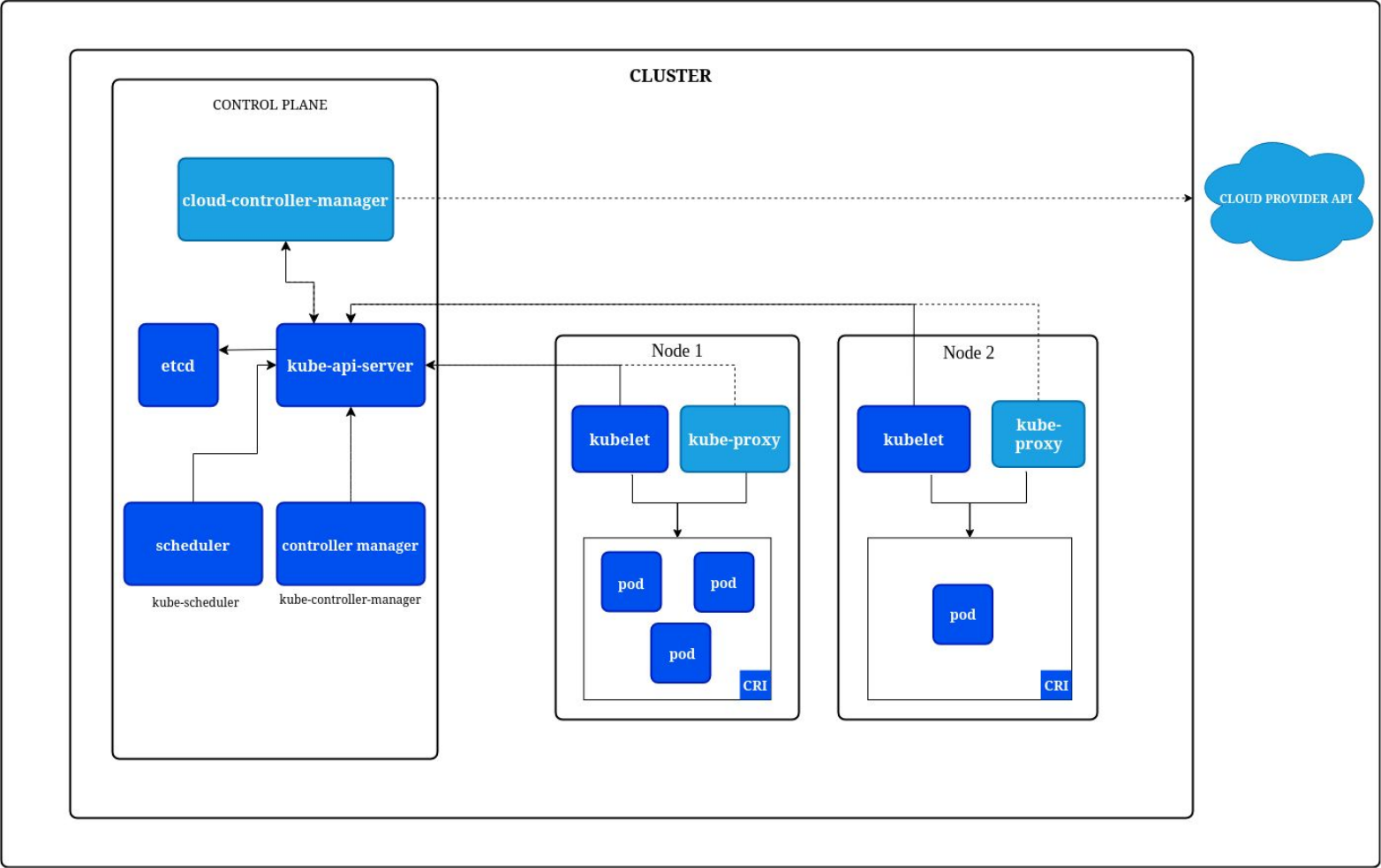
The core principles of Kubernetes

- **Declare** desired state in objects
- **Reconcile** with independent control loops.
- **Converge eventually** onto the desired state, if it is feasible.

The core principles of Kubernetes

- **Declare** desired state in objects
- **Reconcile** with independent control loops.
- **Converge eventually** onto the desired state, if it is feasible.

⇒ Kubernetes can accept a change even if it cannot make it happen.



Thank you!
Questions?