KUBECON + CLOUDNATIVECON NORTH AMERICA 2026

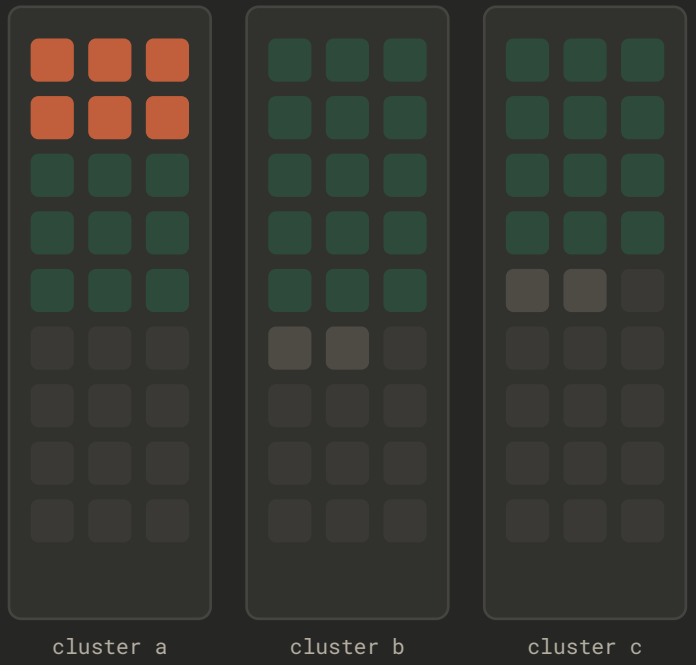
Beyond Stateless vs Stateful

A Unified Workload API for Multi-Cluster Kubernetes



Ronak Nathani

Principal Staff Software Engineer, LinkedIn



BACKGROUND

LinkedIn runs on Kubernetes

500K+
bare-metal servers

5M+
pods

3,000+
services

110K+
deployments / week

BACKGROUND

Kubernetes Offerings

Stateless services

`LiDeployment`

for app teams

Stateful services

`LiStatefulSet`

for app teams

Jobs

AI training · Spark · CI

own schedulers and quotas

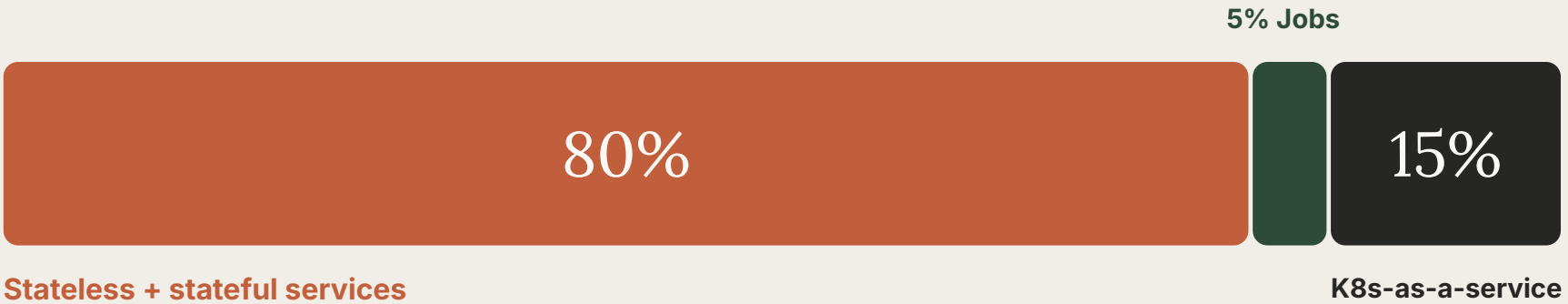
K8s-as-a-service

Flink · DB-as-a-service

for platform teams

BACKGROUND

80% of our compute runs via two CRDs

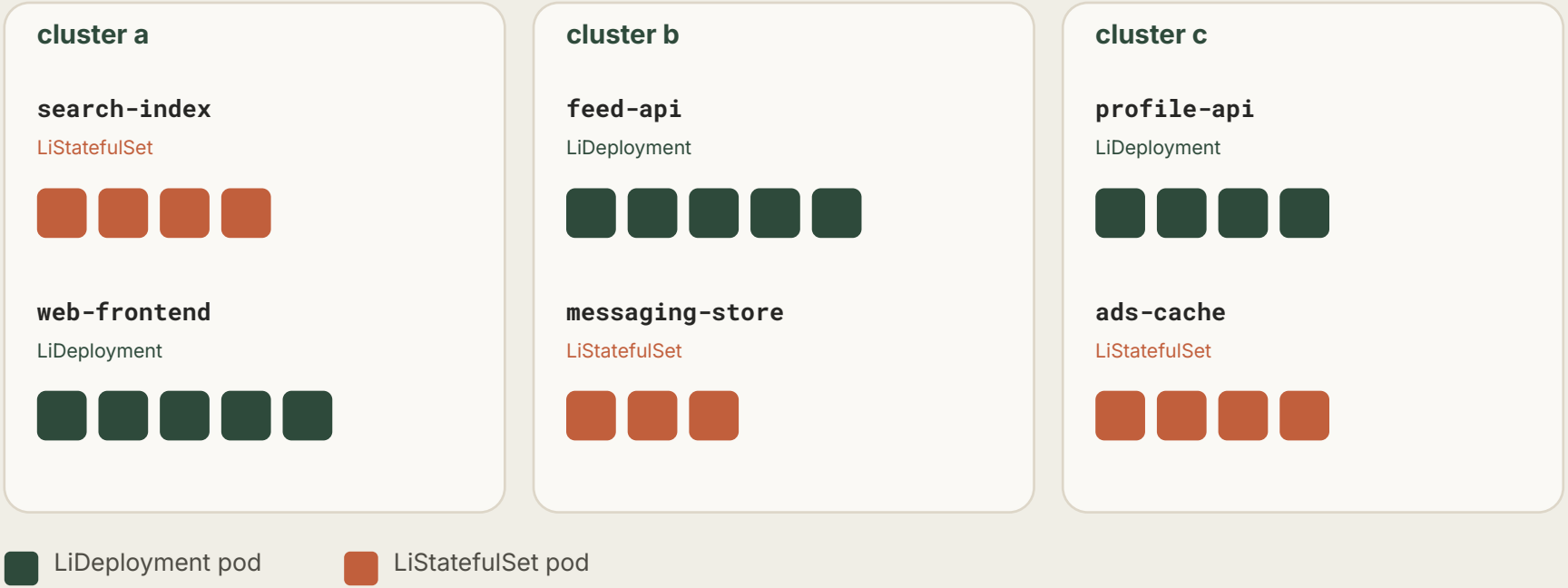


LiDeployment

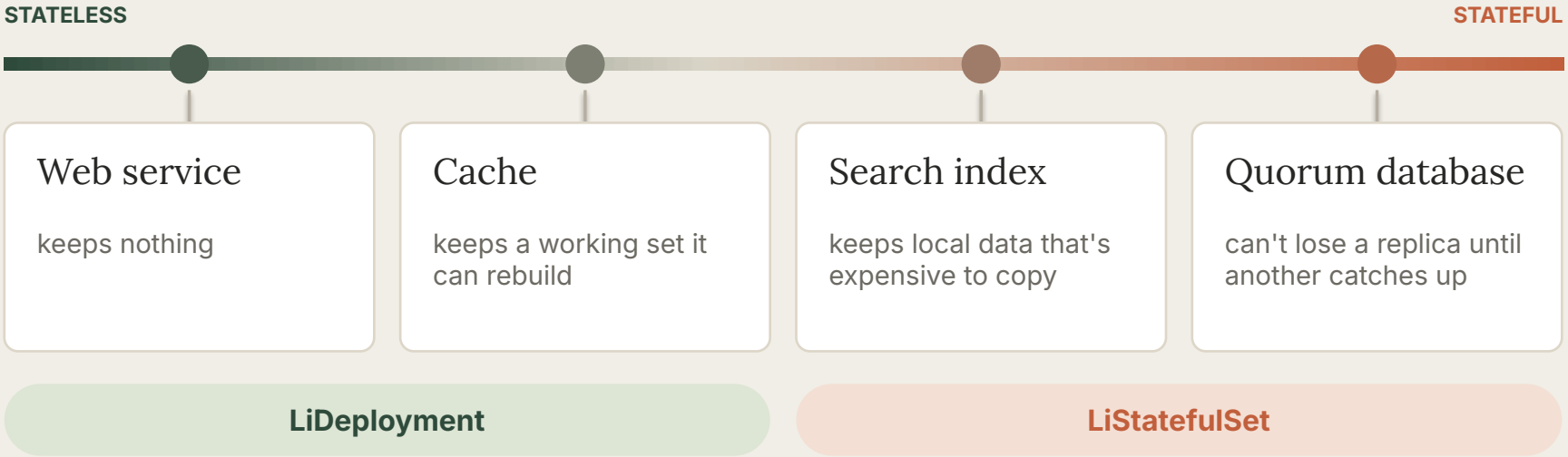
LiStatefulSet

BACKGROUND

Clusters are shared, but each app lives in exactly one



Moving along the spectrum means switching APIs



NOT JUST OUR CRDS

Upstream APIs bundle updates and persistence too

`Deployment` — Rolling update + No PVC per pod

`StatefulSet` — One pod at a time + PVC per pod

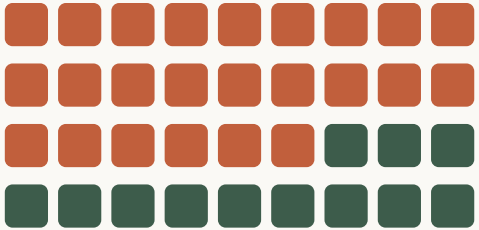
Search index wants — Rolling update + PVC per pod

neither API offers this

CHALLENGE 2 OF 4

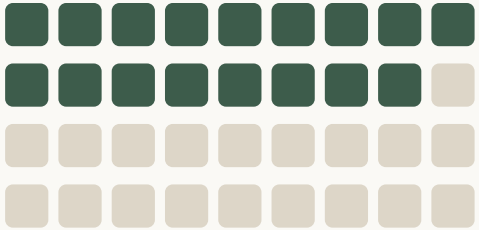
An app can only grow as big as one cluster

cluster a FULL



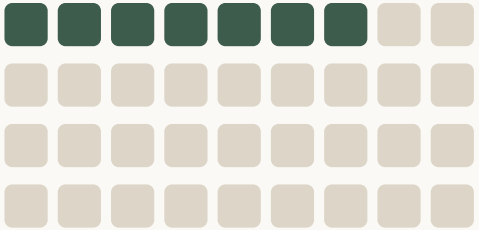
search-index can't grow here

cluster b



room to spare

cluster c



room to spare

CHALLENGE 3 OF 4

Moving an app between clusters happens out of band



CHALLENGE 4 OF 4

Teams want to run more than two versions at once

TODAY



WHAT TEAMS WANT



TAKING A STEP BACK

What a workload is made of

Pod spec

the containers to run

Update strategy

how pods get replaced

Persistence

whether data survives an update

Placement

which clusters, and how many replicas in each

A composable workload API

`Revision`

written by the app team

- Pod spec
- PVC templates

`Workload`

written by the app team

- Replicas
- Revision distribution
- Update strategy

`WorkloadPlacement`

written by the cluster admin

- Clusters
- Replicas per cluster

A Revision holds a complete pod spec

Revision
written by the app team

Workload
written by the app team

WorkloadPlacement
written by the cluster admin

```yaml
kind: Revision
metadata: {name: v2}
spec:
  template:
    spec:
      containers:
      - name: store
        image: registry.example.com/store:2.0
  volumeClaimTemplates:
  - metadata: {name: data}
    spec:
      storageClassName: local-host
      resources: {requests: {storage: 500Gi}}
```

Revisions are immutable. A change is a new Revision.

It can also give every pod its own PVC

Revision
written by the app team

Workload
written by the app team

WorkloadPlacement
written by the cluster admin

```
kind: Revision
metadata: {name: v2}
spec:
  template:
    spec:
      containers:
      - name: store
        image: registry.example.com/store:2.0
  volumeClaimTemplates:
  - metadata: {name: data}
    spec:
      storageClassName: local-host
      resources: {requests: {storage: 500Gi}}
```

Persistence is a field, not a different API.

A Workload sets how many replicas to run

Revision	**Workload**	WorkloadPlacement
written by the app team	written by the app team	written by the cluster admin

```
kind: Workload
metadata: {name: store}
spec:
  replicas: 9
  replicaDistribution:
  - revision: v1
    allocation: 75%
    tag: stable
  - revision: v2
    allocation: 25%
    tag: canary
  updateStrategy:
    rollingUpdate: {maxSurge: 0, maxUnavailable: 1}
```

The count is for the whole workload, not one cluster.

It splits those replicas across revisions

Revision
written by the app team

Workload
written by the app team

WorkloadPlacement
written by the cluster admin

```yaml
kind: Workload
metadata: {name: store}
spec:
  replicas: 9
  replicaDistribution:
  - revision: v1
    allocation: 75%
    tag: stable
  - revision: v2
    allocation: 25%
    tag: canary
  updateStrategy:
    rollingUpdate: {maxSurge: 0, maxUnavailable: 1}
```

Any number of revisions, each with its own share.

It also says how pods get updated

Revision	**Workload**	WorkloadPlacement
written by the app team	written by the app team	written by the cluster admin

```yaml
kind: Workload
metadata: {name: store}
spec:
  replicas: 9
  replicaDistribution:
  - revision: v1
    allocation: 75%
    tag: stable
  - revision: v2
    allocation: 25%
    tag: canary
  updateStrategy:
    rollingUpdate: {maxSurge: 0, maxUnavailable: 1}
```

OR, FOR STATEFUL WORKLOADS

```yaml
updateStrategy:
  coordinator:
    name: store-coordinator
```

A WorkloadPlacement says which clusters run it

Revision
written by the app team

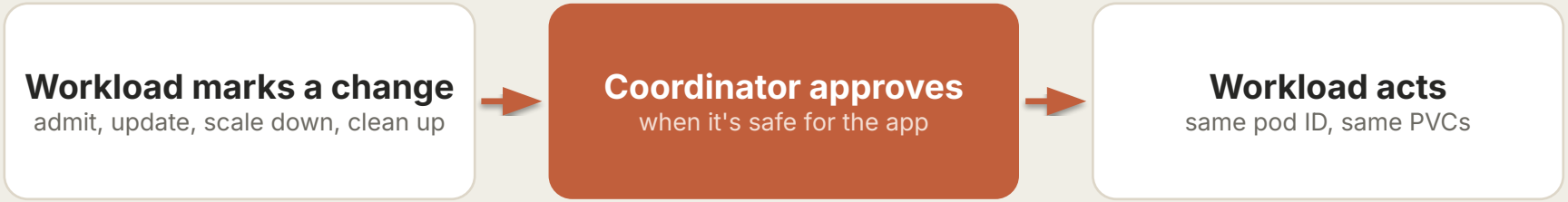
Workload
written by the app team

WorkloadPlacement
written by the cluster admin

```yaml
kind: WorkloadPlacement
metadata: {name: store}   # same name as the Workload
spec:
  allocationStrategy: Explicit
  clusterPlacements:
  - {clusterName: a, allocation: 50%}
  - {clusterName: b, allocation: 30%}
  - {clusterName: c, allocation: 20%}
```

Cluster admins write this, not app teams.

Supporting stateful workloads



HOW WE RUN STATEFUL WORKLOADS ON KUBERNETES

TALK · KUBECON NA 2024

Cooperative Scheduling for Stateful Systems

Michael Youssef & Zhantong Shang

`youtu.be/jB2MI7ZKw40`

BLOG · LINKEDIN ENGINEERING

Stateful workload operator: stateful systems on Kubernetes at LinkedIn

Michael Youssef, Zheyi Y. & Daniel Cheng

`linkedin.com/blog/engineering`

COORDINATOR

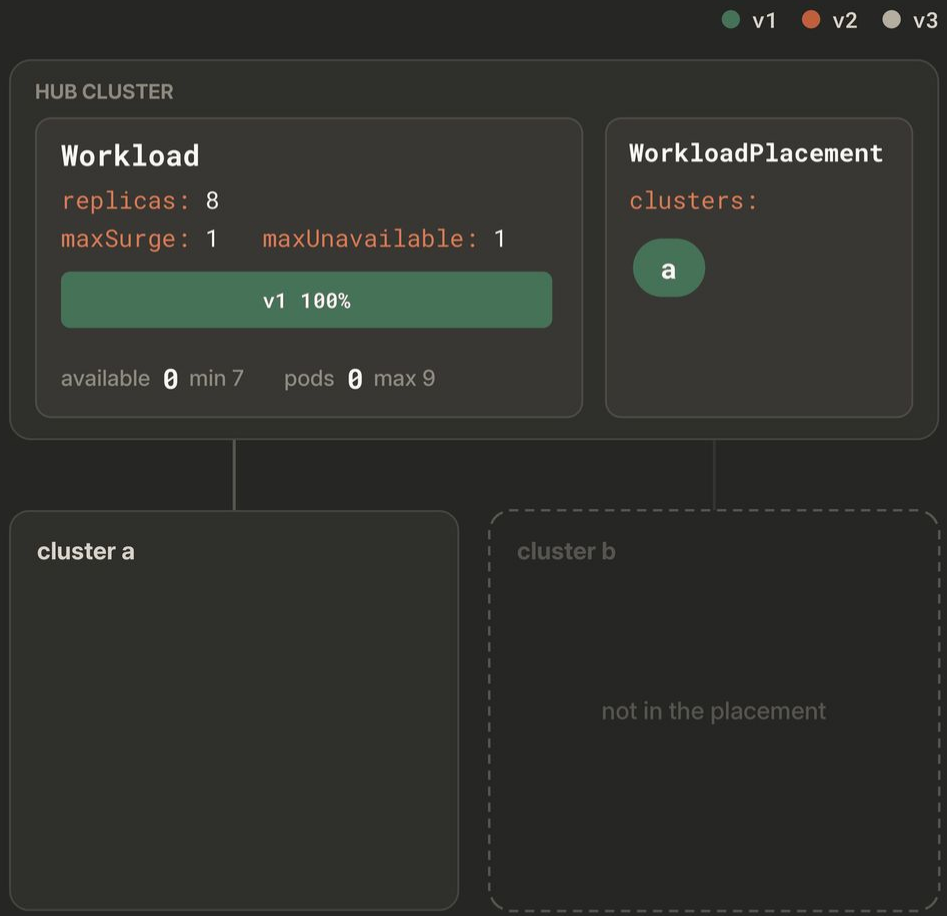
The coordinator contract is ordinary pod metadata

OPERATION	WORKLOAD MARKS IT WITH	COORDINATOR APPROVES BY
Scale up	a scheduling gate	removing the gate
Confirm readiness	a readiness gate	setting the condition
Update to a new revision	`update-pending` label	`update-approved` label
Scale down	`scale-down-pending` label	`delete-approved` label
Clean up after deletion	a finalizer	removing the finalizer

KUBECON + CLOUDNATIVECON NA 2026

One Workload, many clusters

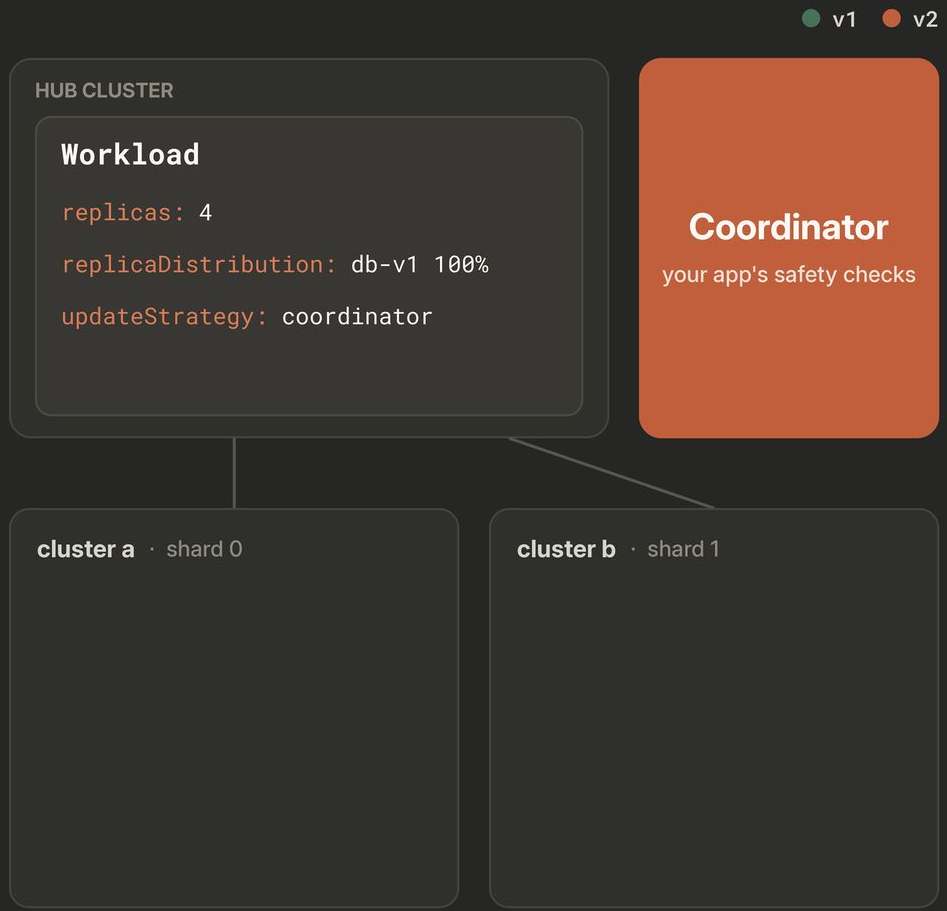
1. Start in one cluster
2. Add cluster b to the placement
3. Add revision v2 at 25%
4. Ramp v3 to 50%



KUBECON + CLOUDNATIVECON NA 2026

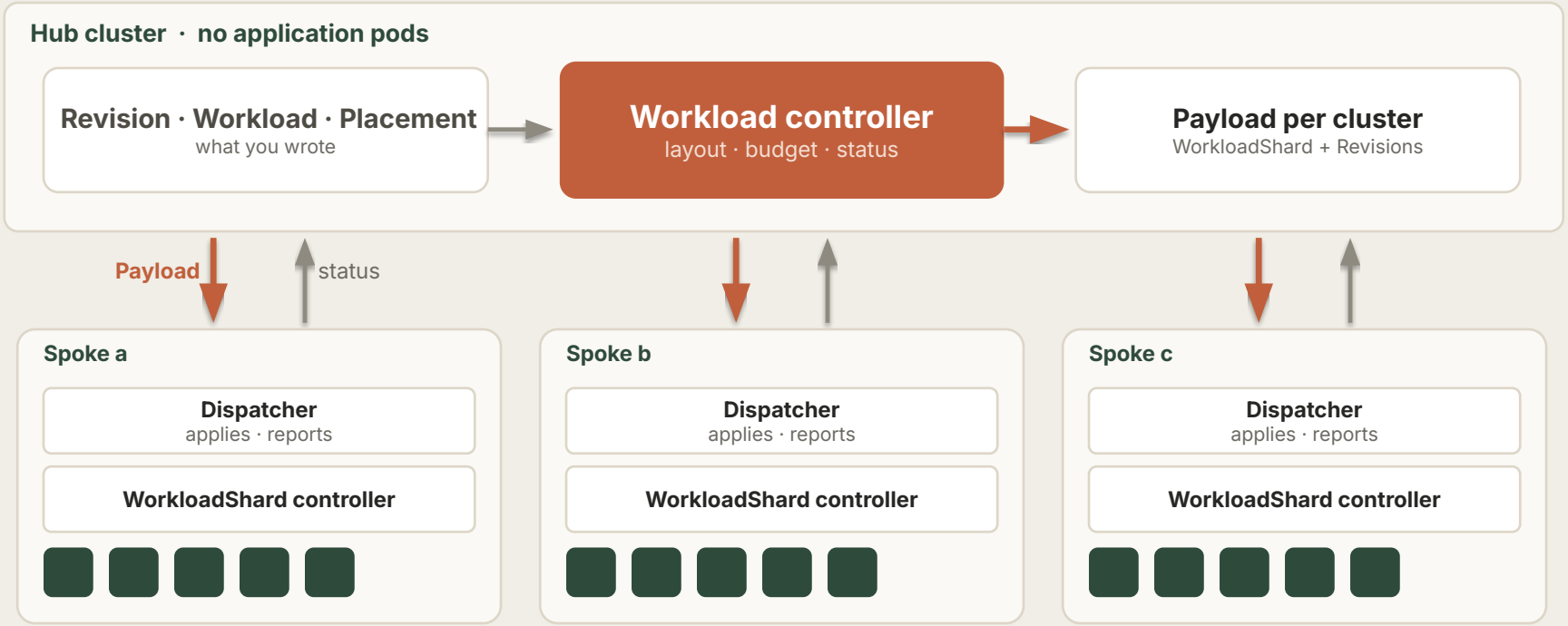
Coordinated updates across clusters

1. Scale up
2. Roll out a new revision
3. Scale down



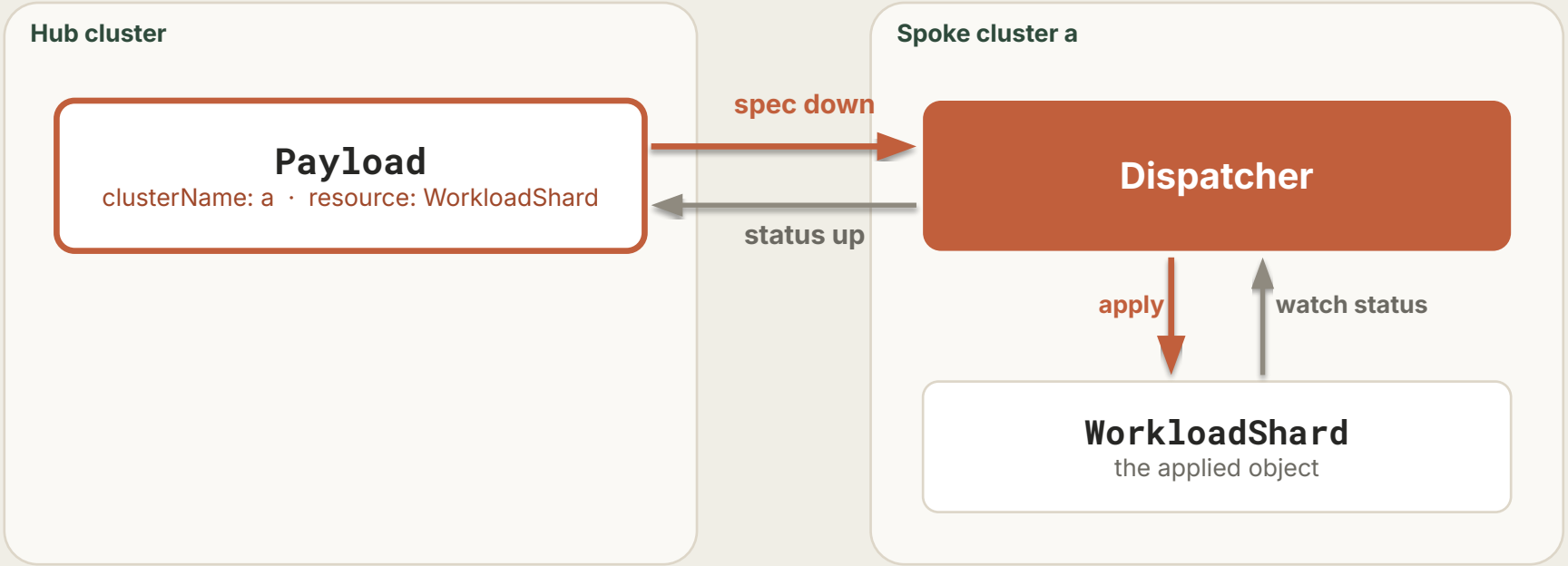
HOW IT WORKS

A hub cluster stores intent and spoke clusters run pods



DISPATCHER

A generic dispatcher sends spec down and status back up

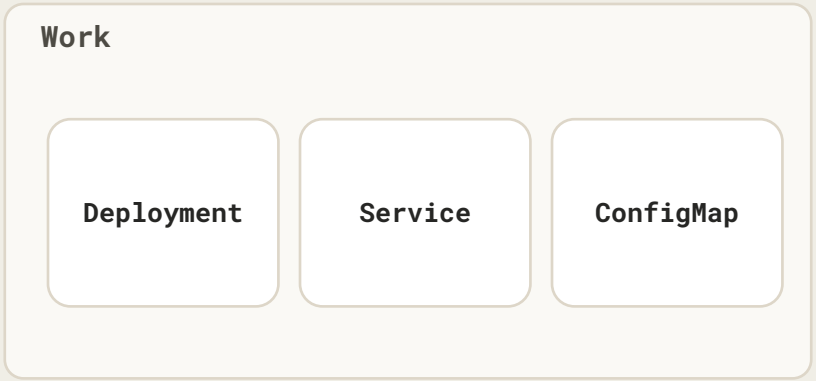


DISPATCHER

A Payload carries exactly one object

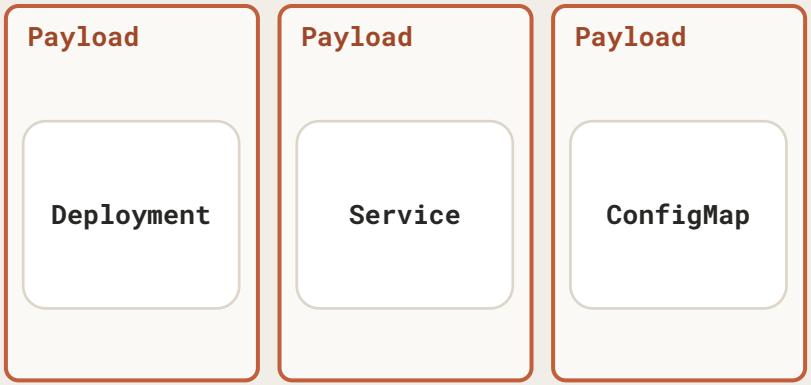
Work API

one Work, many objects



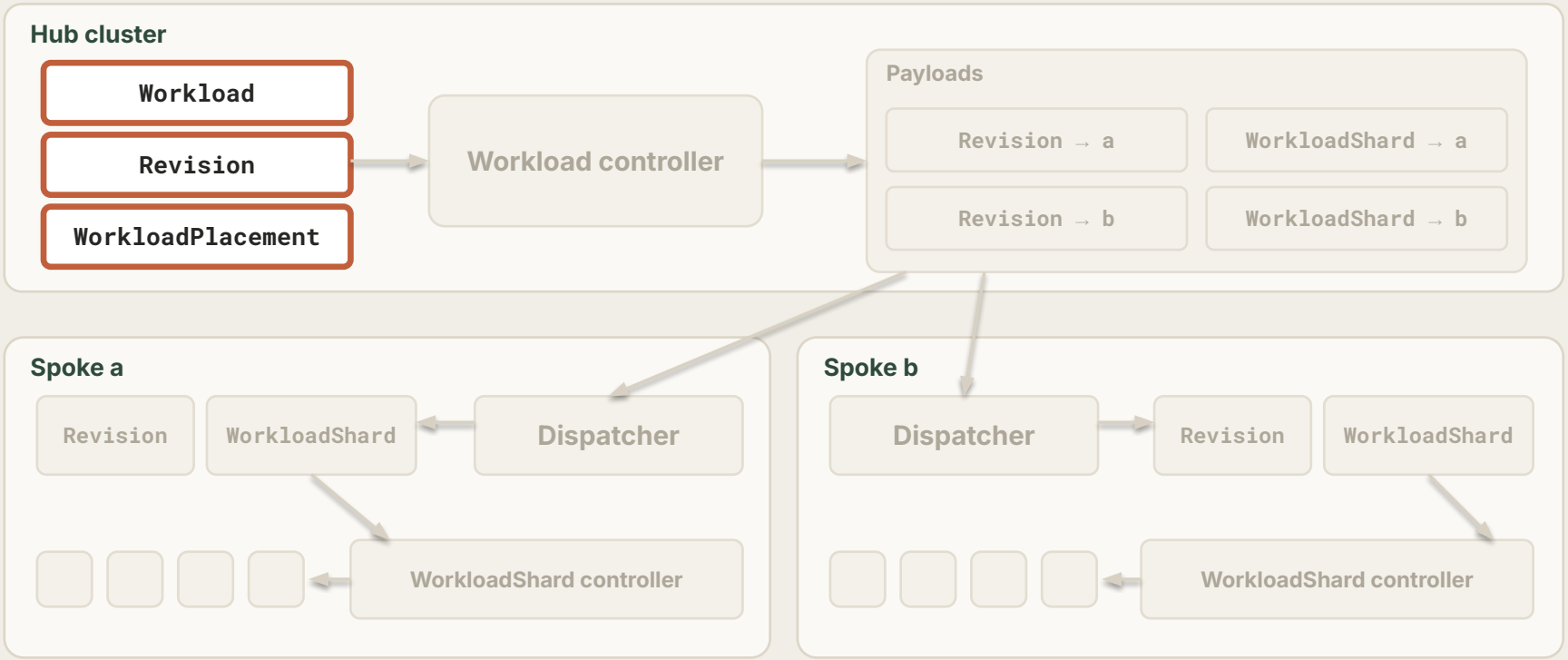
Payload

one Payload, one object

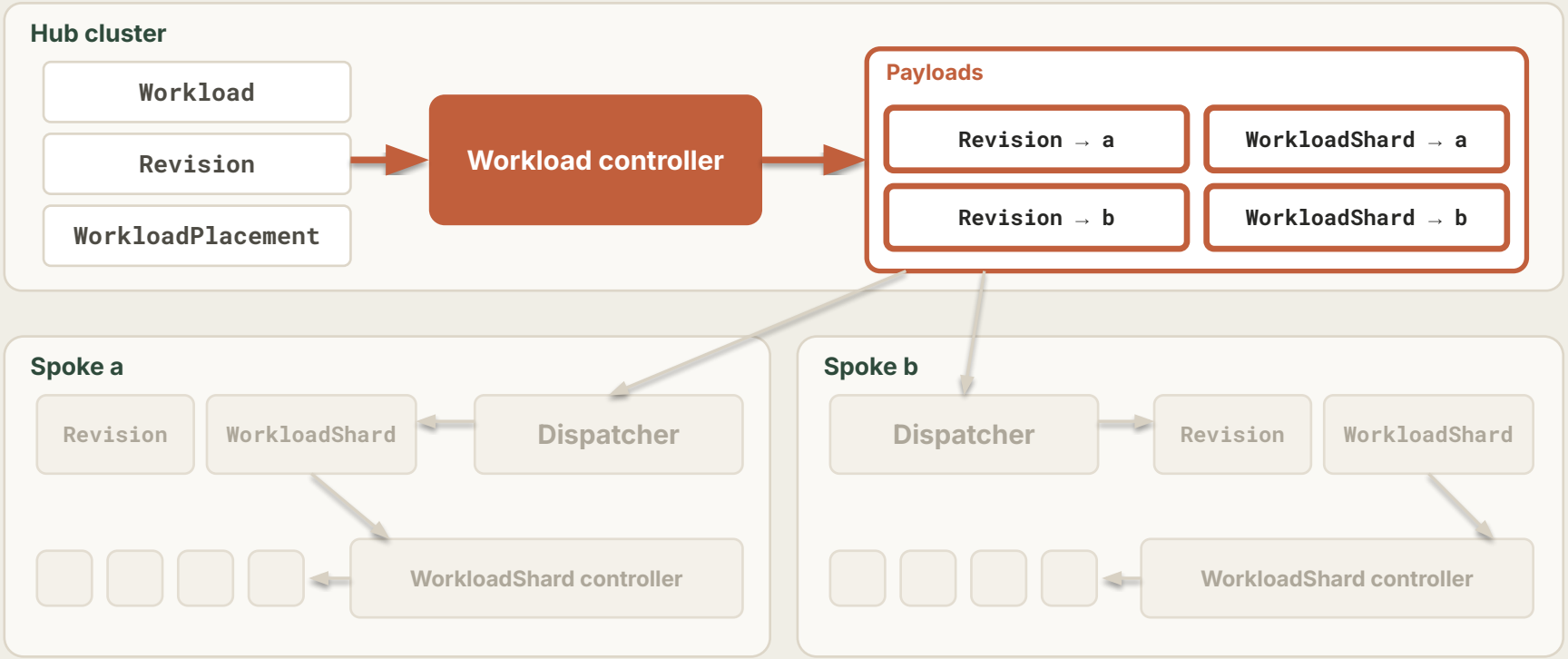


Each object has its own lifecycle, and the API enforces it instead of a webhook.

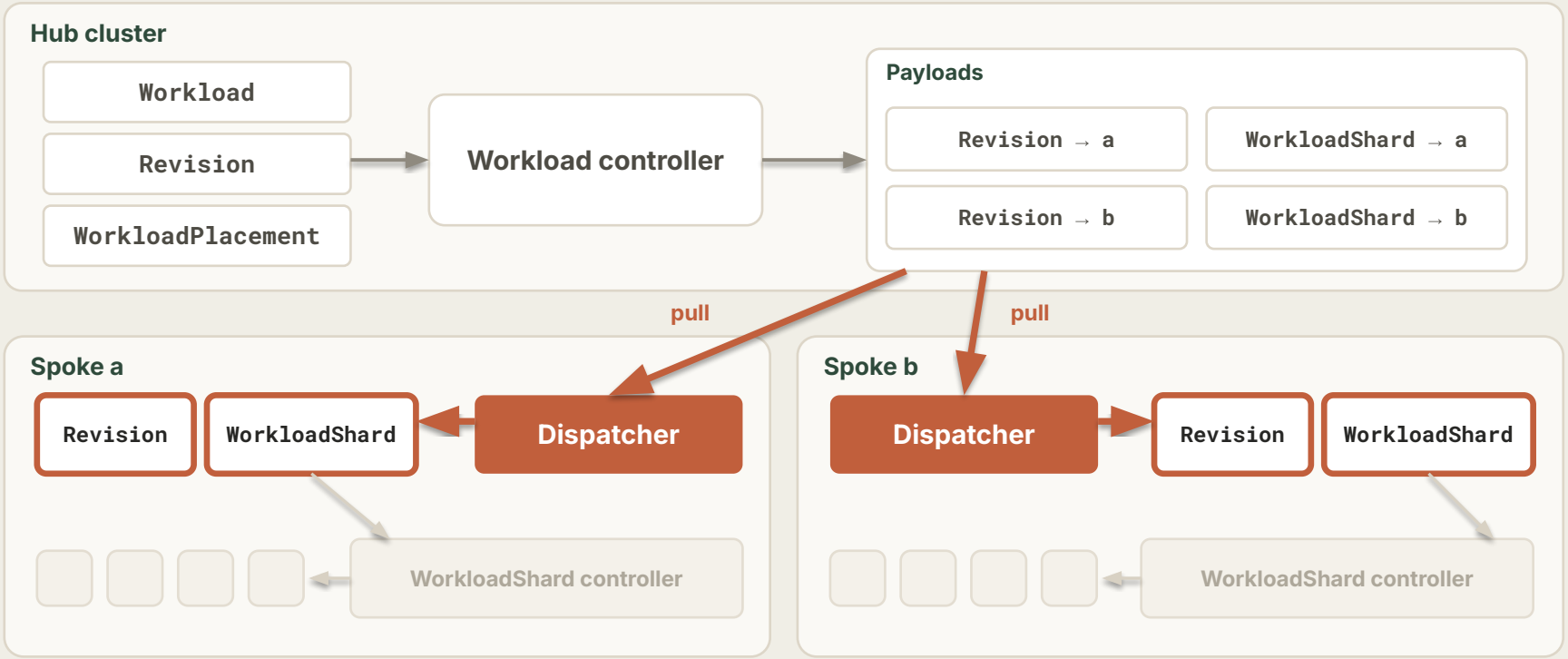
The app team and a cluster admin write to the hub



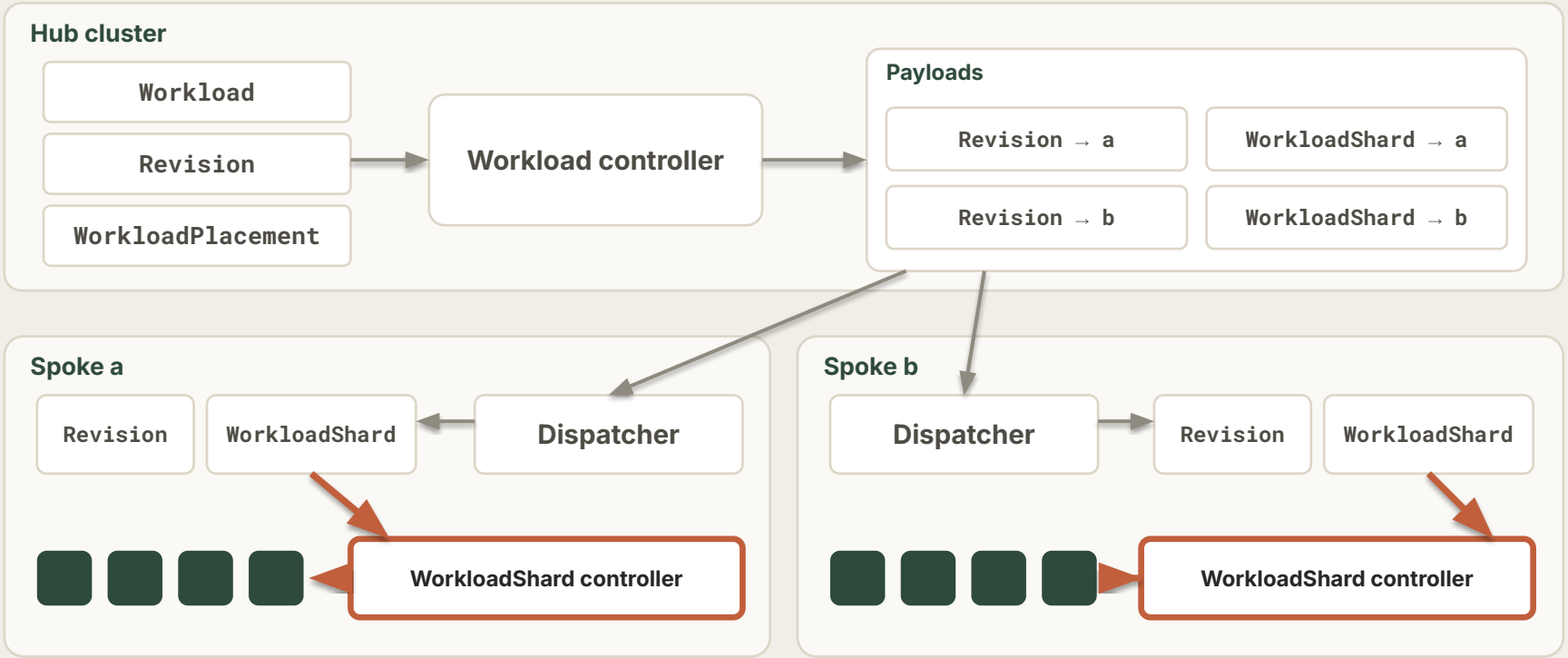
The hub writes one Payload per object, per cluster



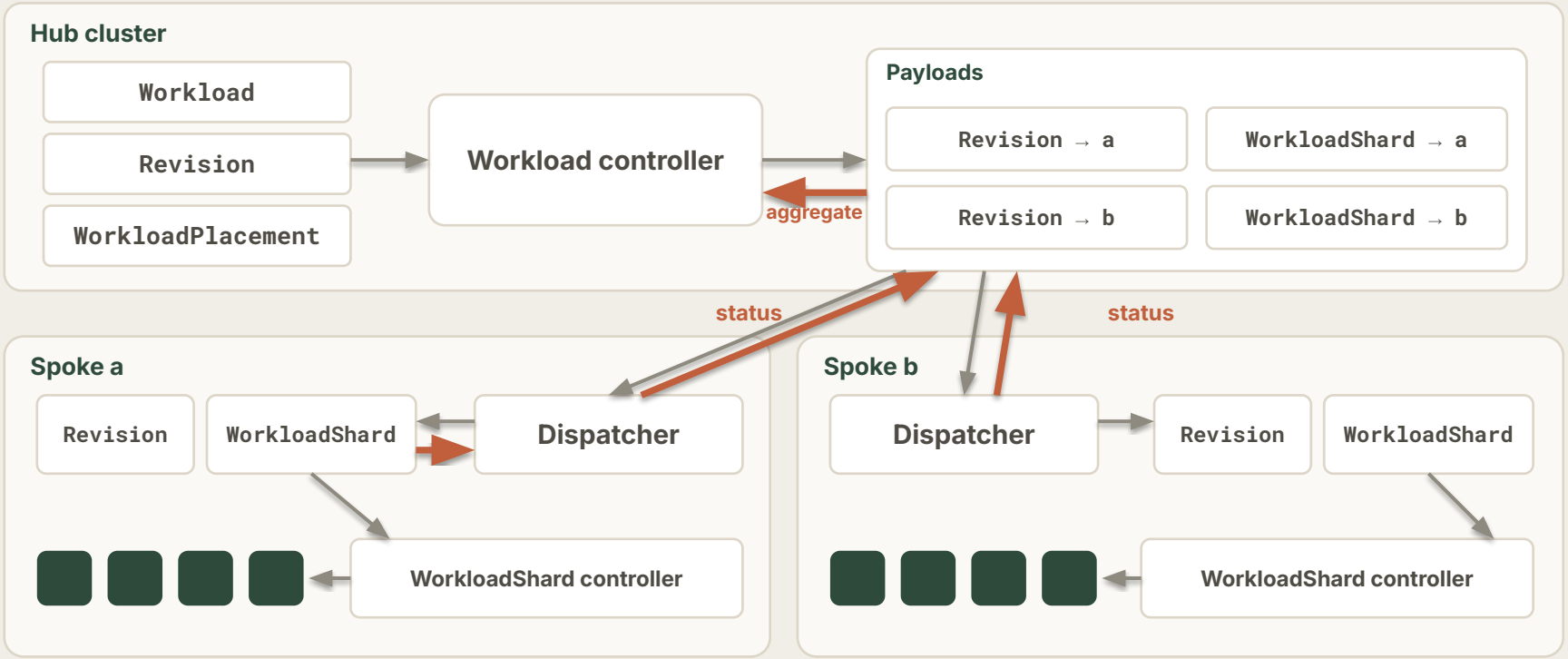
Dispatcher pulls its Payloads and applies them



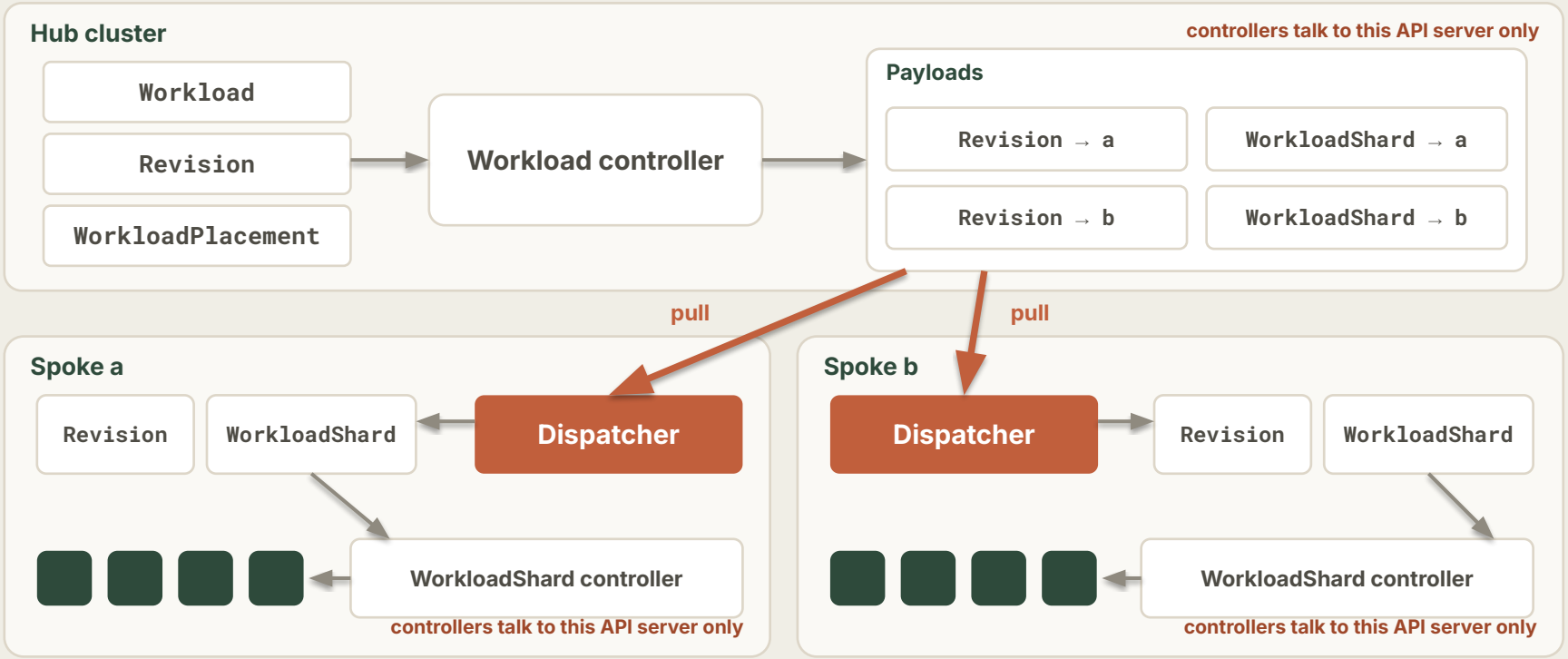
The WorkloadShard controller creates the pods



Status flows back up the same way

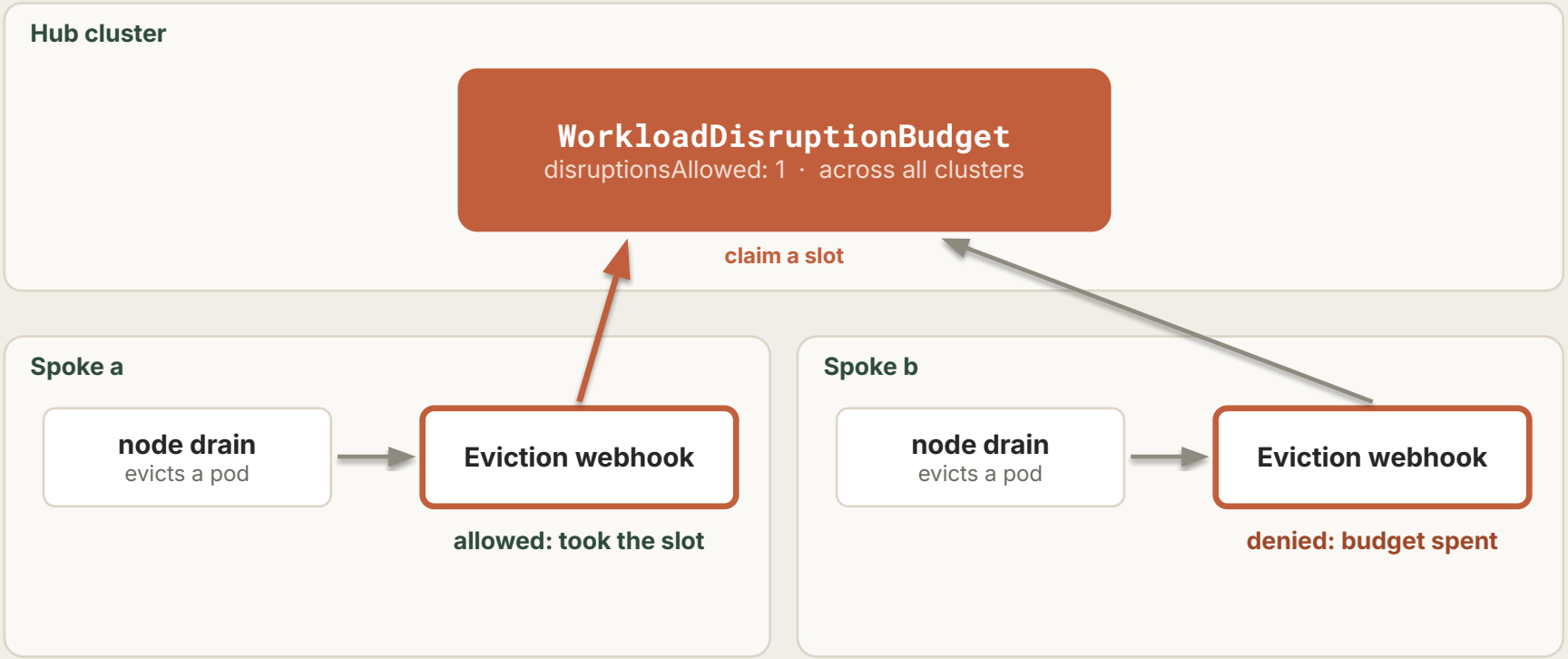


Each controller is cluster local, except Dispatcher



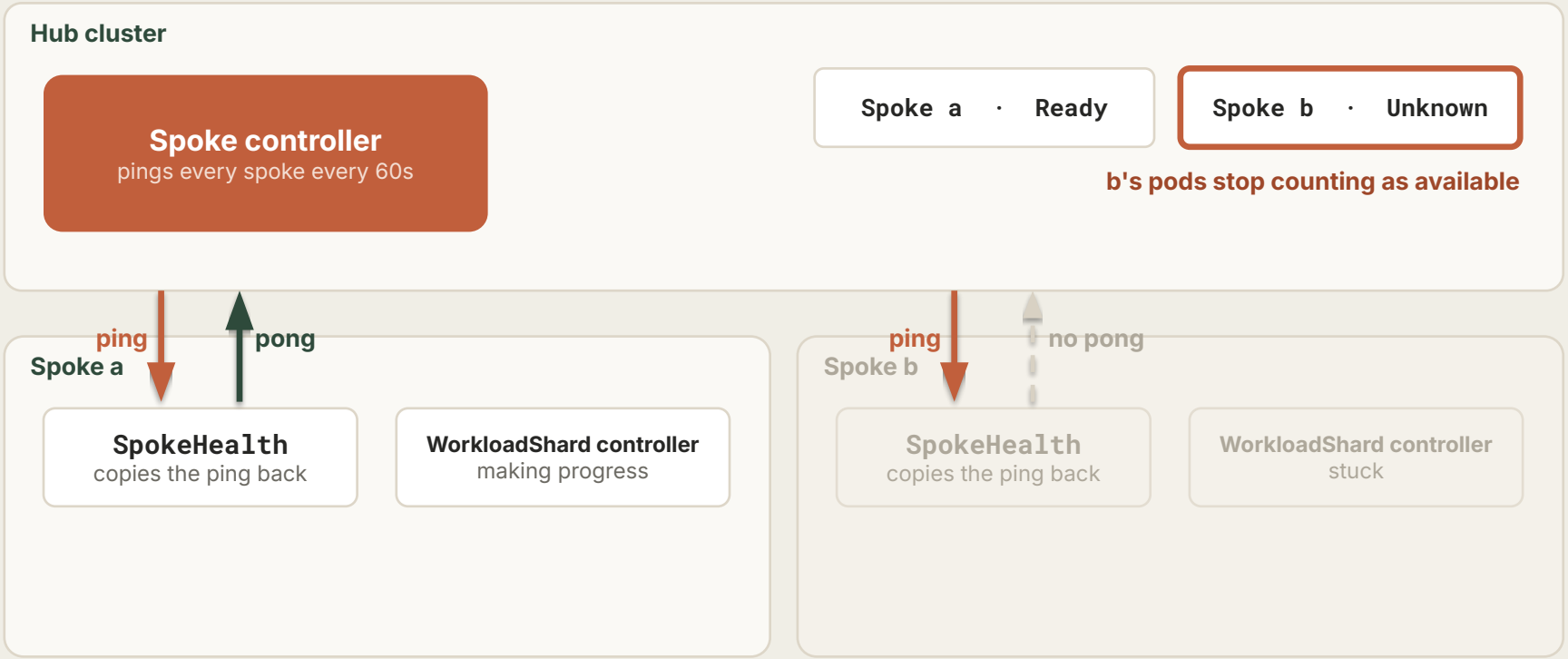
DISRUPTIONS

Every spoke's evictions check one budget on the hub



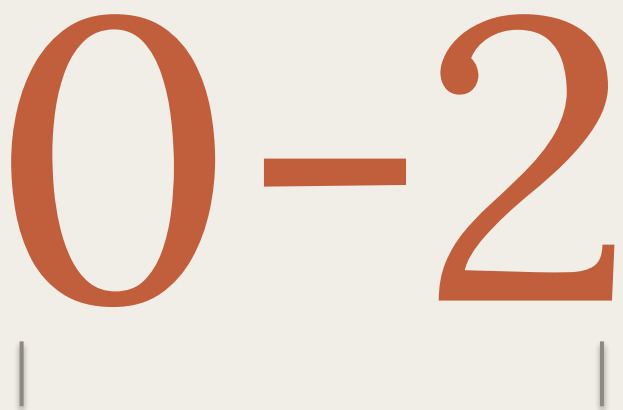
STALE STATUS

Spokes heartbeat to the hub, the way kubelets do



STABLE POD IDS

Every pod gets a stable ID



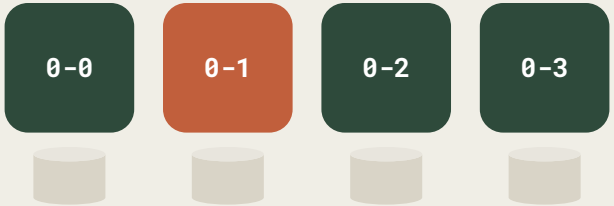
shard index
one per cluster, never reused

ordinal
lowest free one in the cluster

STABLE POD IDS

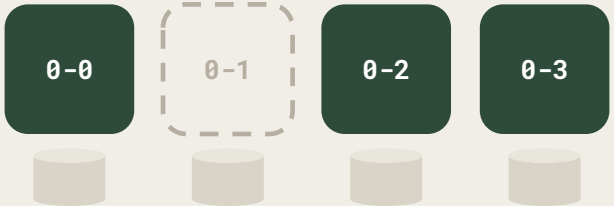
Scaling fills gaps, and PVCs follow the ID

Start



0-1 is failing

4 → 3



0-1 goes. Its PVC stays

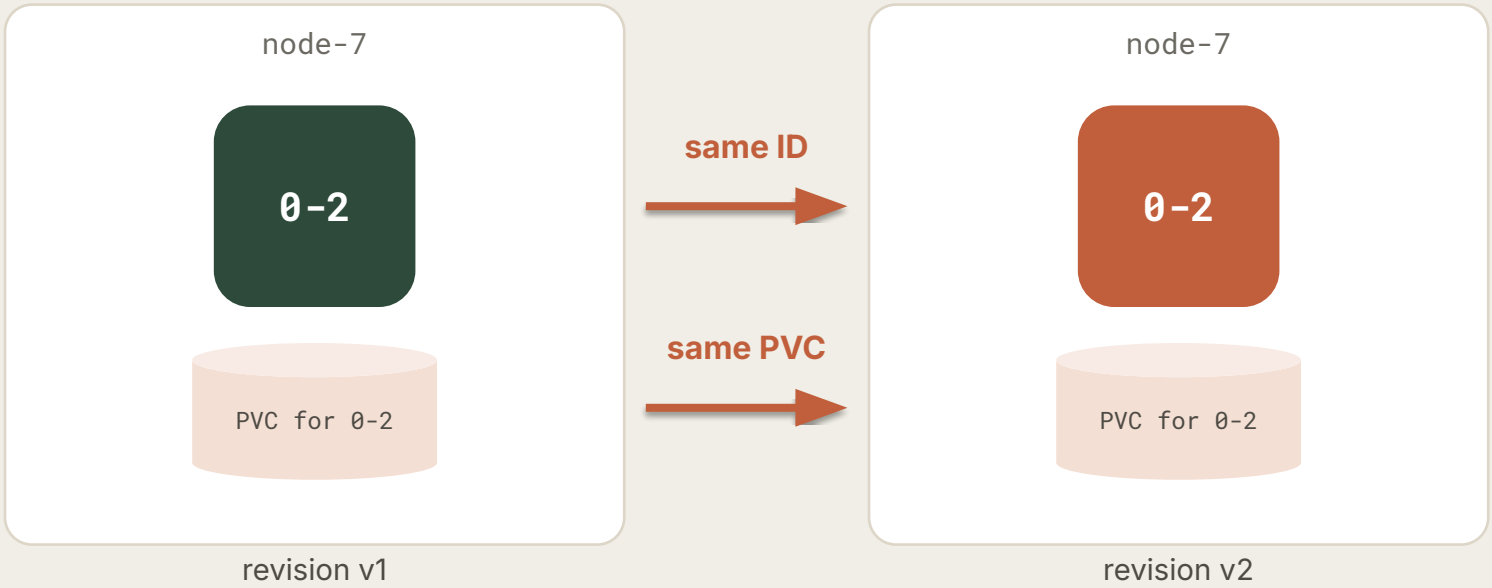
3 → 5



0-1 mounts its old PVC

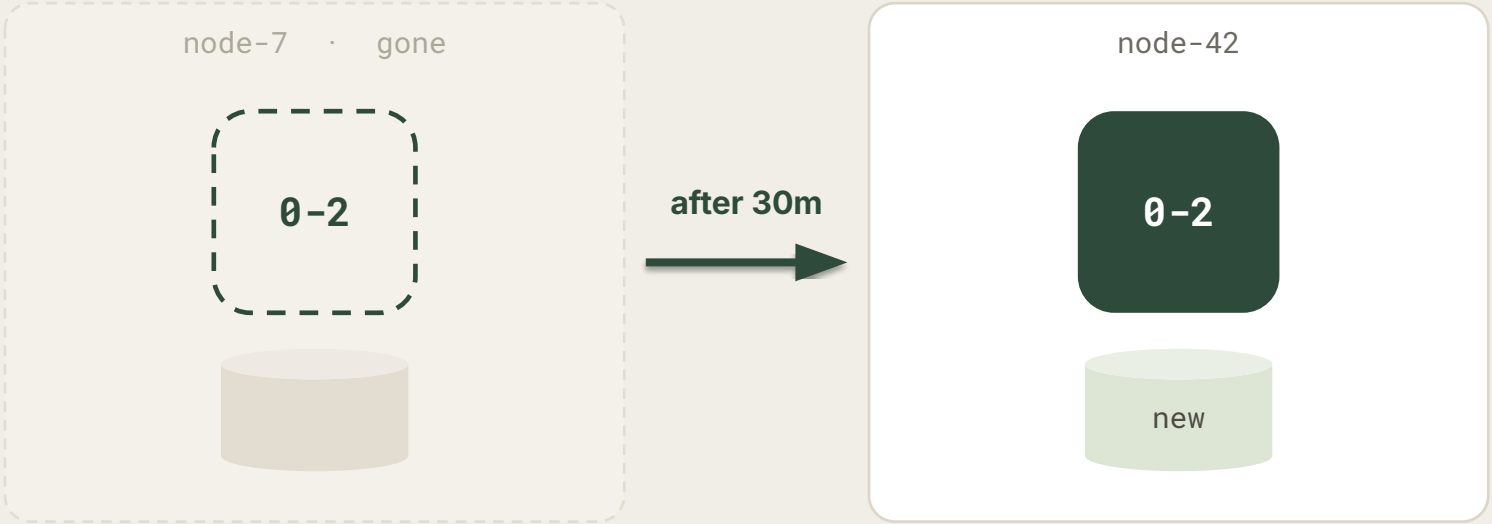
STABLE POD IDS

The stable ID is what lets a pod keep its PVC



PERSISTENCE

If a pod's node is gone, it can get a new PVC elsewhere



`recycleAfterUnschedulable: 30m`

PATTERNS

Multiple patterns, one API

Stateless service

web API, model server

Revision
Pod spec, no PVCs

Workload
Rolling update, with a canary revision

WorkloadPlacement
Any clusters

Cache on local disk

search cache, model weights

Revision
PVC per pod, recycled if its node is gone

Workload
Rolling update, a replacement reuses the PVC

WorkloadPlacement
Any clusters

Sharded store

Kafka, database, vector index

Revision
PVC per pod that holds data

Workload
A coordinator approves each pod change

WorkloadPlacement
Any clusters

None of these fits a stateless or stateful box.

TAKEAWAYS

What we learned

Stateless and stateful are useful characteristics, but a bad API boundary

Keep clusters out of the app's spec

A multi-cluster workload needs more than a generic dispatcher for single-cluster APIs

OPEN SOURCE

We're thinking about open sourcing this

> If this would be useful to you, please reach out.

Thank you

Questions?

Ronak Nathani
Principal Staff Software Engineer, LinkedIn

ronaknathani.com

linkedin.com/in/ronaknnathani

@RonakNathani



ronaknathani.com