Cluster Reconciliation
Managing Resources Across Multiple Clusters

Vallery Lancey (@VLLRY)

Why Multicluster

- Regional POPs.
- Cluster-level redundancy.
- Cluster-level security isolation.
- Blue-green cluster upgrades.

What Kinds of Things

- Core cluster setup.
  - E.G. admin RBAC, cluster monitoring stack.
- Workload enablers and middleware
  - E.G. storage providers, ingress controllers.
- Applications.

What Problems Are We Trying to Solve

- Deploy common resources to multiple clusters.
- Loosely couple clusters and resources.
- Clearly map clusters and resources.
- Garbage collect old resources.

What Does Failure Look Like?

- Dependency tangle between resources and versions.
- Orphaned resources on clusters.
- Wonky workload : cluster mapping.
- Hard to bootstrap clusters.

What Does Deploying Something Mean?

The Kubernetes Object Model

- All objects have a group, version, and kind.
- All objects have an identifier.
  - Name
  - Namespace (if a namespaces object).
- Combine to give a URL like:
  - /apis/apps/v1/namespaces/default/deployments/demo

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: demo
namespace: default
```

Kubernetes Verbs

- CREATE: creates a new object.
- UPDATE: replaces an object with the new state.
  - Has a version to check to avoid race conditions.
- PATCH: inserts/replaces only the specificed fields of an object.
  - Uses custom semantics
- DELETE: deletes the object, after finalizers complete.

(Serverside) Apply

New-ish API

- Kubectl apply (and the apply API) have patch-like behavior.
  - “Ownership” of fields is tracked in object metadata.
- Fields in the desired state replace the actual state.
  - Fields *not* present in the desired state are left as-is.

(Serverside) Apply

- When apply results in a conflict, there are 3 options to remediate:
  - Re-apply without those fields.
  - Re-apply with the existing field values, and become a shared owner.
  - Force apply, updating the fields, and becoming the sole owner.

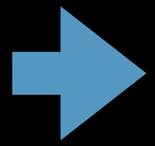
Gotchas

A Service in Kubernetes

- Let’s say an archetypical service on Kubernetes contains:
  - Deployment.
  - Service (it’s a bad name, we know).
  - Ingress.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    ...
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
    ...
    spec:
      containers:
      - name: nginx
        image: nginx:1.15
```

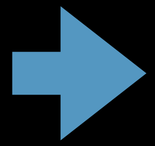


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    ...
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

Next File

No port!

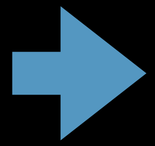
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver
  …
```



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  …
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver
  …
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: demo
spec:
  selector:
    app: demo
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```



```yaml
apiVersion: v1
kind: Service
metadata:
  name: demo
spec:
  selector:
    app: demo
  type: ClusterIP
  clusterIP: 1.2.3.4
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Reconciliation Challenges

- Apply & variants may leave undesired fields untouched.
- Some fields will only exist at runtime.
  - E.G. service.spec.clusterIP.
- Multiple tools may try to maintain different fields.
  - E.G. sidecar container pipelines may deploy image updates independently.
- Some fields are immutable.
  - E.G. secrets can’t switch between using secrets.data and secrets.binaryData.

The Multicluster Journey

Evolution of Cluster Management with Scale

- Trends from tighter to looser coupling between apps and infra.
- Trends from manual to automated management.

Reminder: The Goal is to Reduce Uncertainty and Toil

A Beautiful System is One that Works (Well)

Model 1: Hardcoded Clusters

- Deploy tooling explicitly targets specific clusters.
  - kubectl config use-context foo && kubectl apply -f state.yaml
- Lots of duplication across config & workloads.
- Configuration lags as cluster topology changes.
  - Removing clusters breaks many assumptions.
  - Workloads must explicitly opt in (or out) of new clusters.

Model 2: Cluster Groups

Typical in Orgs with Many Clusters and/or Workloads

- Curate named groups of clusters.
  - “production”, “foo-prod”, etc.
- Map workloads to a group.
- Deploy tooling targets each cluster in the group, at deploy-time.

Model 2: Cluster Groups

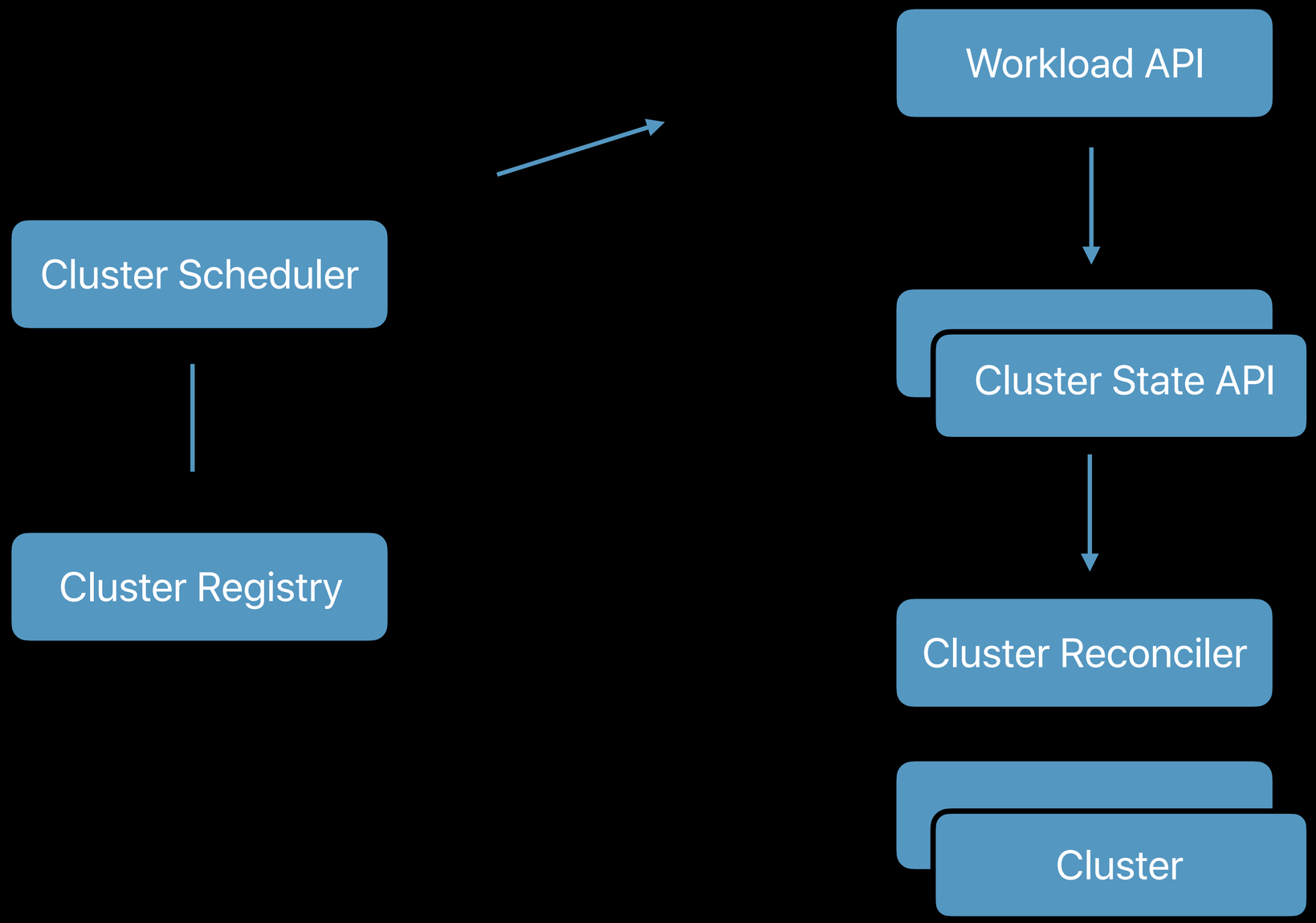
- Any workload needs a cluster group to target.
  - Still coupling workloads and underlying infra details.
  - E.G. batch jobs that should only exist in 1 cluster would need an explicitly designated “batch cluster”.

Model 3: Cluster Scheduling

- Specify workload constraints without specifying clusters.
  - E.G. “three geo-distributed clusters in the US”
  - E.G. “any 1 cluster that runs workload <x>”
- Allows workload owners to capture topology *intent* without hard coupling or knowledge of specific infra topology.

Dynamically Scheduling Workloads

The Endgame of Multicluster Kubernetes is to Treat Clusters like Nodes


Cluster Scheduler
Cluster Registry
Workload API
Cluster State API
Cluster Reconciler
Cluster

Learn More

- Blog: A Model for Multicluster Workloads (in Kubernetes and Beyond)

The Cluster Inventory API

- Defines what objects should exist in each cluster.
  - Excludes higher-level ambiguity.

The Cluster Reconciler

- The cluster reconciler syncs objects into the cluster.
  - Creates missing objects, updates objects that differ, deletes previously managed objects.
- Pulls state from the cluster inventory API.

We’re Prototyping a Reconciler and Inventory API Design Now in SIG-Multicluster

The Cluster Registry

- Lists clusters, status, and credentials.
- Enables tooling to inventory and authenticate to available clusters.
- An “official” one exists, but it has an uncertain future.

Conjecture Ahead

The Workload API

- The workload API is by far the most ambiguous part.
  - How are workloads gracefully rolled out between clusters?
  - What templating options are available?
  - What scheduling options are available?
  - How sticky is cluster scheduling?

The Cluster Scheduler

- Decides which cluster(s) to assign a workload to.
  - Reads the cluster registry to choose from available clusters
  - Updates the cluster inventory.

We’re Hiring

Stop by the Apple Booth

TM and © 2020 Apple Inc. All rights reserved.