Kubernetes SIG MultiCluster
Intro + Deep Dive

shashidharatd@
XunPan@

Tuesday, June 25 • 11:45
Agenda

- Mission
- Active projects
  - Cluster Identity
  - Kubernetes Cluster Federation
- What have we been doing
- Deep dive into Kubernetes Cluster Federation concepts
- Q&A
SIG MultiCluster mission

- Solving common challenges related to the management of multiple Kubernetes clusters, and applications that exist therein
- Designing, discussing, implementing and maintaining
  - API’s, tools and documentation
  - related to multi-cluster administration and application management
- Includes **not only active automated approaches** such as Kubernetes Cluster Federation
  - also those that employ batch workflow-style continuous deployment systems
- Includes:
  - standalone building blocks (for example a cluster registry), and
  - proposed changes to kubernetes core where appropriate
- See more at https://github.com/kubernetes/community/blob/master/sig-multicluster/README.md
Active work efforts

- Cluster Identity
- Kubernetes Cluster Federation (KubeFed)
Sub Project: Cluster Identity

● Goals:
  ○ Develop a cluster identifier concept more durable than API endpoint addresses, certs, etc.
  ○ Use cases:
    ■ Identifier that persists longer than currently used identifiers
    ■ Verify identity of a cluster

● Status:
  ○ Recently started; discussions happening in SIG multicloud cluster biweekly sync
KubeFed?
Federation v2 is dead; long live KubeFed
We are now officially KubeFed

- We found federation ambiguous, including references to stuff outside kubernetes.
- **KUBErnetes Cluster FEDeration** established as the official term.
- `kubefed2` tool renamed to `kubefedctl`.
- Project now lives at:
  - http://sigs.k8s.io/kubefed
Whats KubeFed for?

- Coordinate configuration in multiple Kubernetes clusters from a single API surface.
- Active reconciliation to the desired state of the configuration over the lifecycle of the app.
- To be used as foundation blocks for building higher level multicluster use cases e.g. multi geography applications and disaster recovery.
Where we are at?

- Beta release candidate is out.
  - https://github.com/kubernetes-sigs/kubefed/releases/tag/v0.1.0-rc2
- Few outstanding issues, enhancing documentation before final beta, very soon.
What have we been doing?
Let’s start with a demo

- Run an application/Already have an application
- Deploy KubeFed
- Enable needed types
- Federate application
- Change application in specific clusters
- Control placement of the application in clusters
Demo
Sub Project: KubeFed

Type Configuration
- Configures federation for a single API type

Cluster Configuration
- Configures federation with a set of target clusters

Federated Type
- Template: Declaration of resources to be distributed
- Placement
- Overrides

Propagation

Higher level APIs and controllers
- DNS: Maintains DNS entries for federated services and ingresses
- Scheduling: Makes decisions using status of deployed resources

Status
- Collects status of distributed resources

Cluster 1

Cluster N
We now have useful tooling via **kubefedctl**
- **kubefedctl enable** to enable federation of API Types.
- **kubefedctl federate** to convert kubernetes resources to federated equivalents.
API Overhaul

Federated<type>

Federated<type>Placement

Federated<type>Override
API Overhaul

Unified

Template
Placement
Override
Status
API Overhaul

- Unified **template, placement and overrides**
  - Expressed in a single API resource now.

- This API resource now has a meaningful **status**
  - Provides consolidated propagation status.
  - Lists propagation problems, if any.
API Overhaul

FederatedType

- Template
- Placement
- Override
- Status

FederatedTypeConfig

KubefedCluster
API Overhaul

• Core APIs are now compliant with Kubernetes API conventions.
Future work

• Improve usability and documentation.
  • Further easing API translation from Kubernetes resources to KubeFed resources.
  • Helm plugin to auto-convert manifests while installing.
• Higher level user facing API, akin to Apps CRD in Kubernetes.
• Pull reconciliation
Deep dive

• How does KubeFed manage resources in multiple clusters?
KubeFed Beta Features

- Register a cluster to KubeFed control plane
  - `kubefedctl join`
- Enable management of a type of resource
  - `kubefedctl enable`
- Convert resources to federated form
  - `kubefedctl federate`
- Manage member cluster API state
  - Control plane + federated resources
Registering a cluster

- **kubefedctl join**
  - Registers a cluster as a ‘member’ cluster
- Creates service account (**SA**) in member cluster
- Creates a **KubeFedCluster** in the host cluster
  - API endpoint of member cluster
  - CA bundle of member cluster
  - Reference to secret containing **SA** token
Enabling federation of a type

- `kubefedctl enable`
- Creates the federated type CRD
  - e.g. `FederatedDeployment`
- Creates a `FederatedTypeConfig`
  - Includes group, version, resource name (GVR) for 2 API types
  - Target type (e.g. `Deployment`)
  - Federated type (e.g. `FederatedDeployment`)
Federated Types

- A federated type defines how a resource should appear in multiple clusters
  - e.g. **FederatedDeployment**
- 3 fields:
  - template: common form of the resource
    - Embeds the target resource
  - placement: the clusters it should appear in
  - overrides: per-cluster variation
• Namespaces are special (only container)
  ○ So is the federated equivalent!

• **FederatedNamespace**
  ○ is namespace-scoped (simpler permissioning)
  ○ Placement for a contained federated resource is the intersection of the federated resource and federated namespace placement
Converting existing resources

- `kubefedctl federate`
- Converts Kubernetes resources to their federated equivalent
- Resources can exist in API or be provided via yaml
- `kubefedctl federate ns my-ns --contents`
  - Convert the namespace and its contents to federated equivalents
Configuring propagation

- **propagation**: reconciling resources in member clusters with a federated resource
- **FederatedTypeConfig (FTC)** configures propagation for a federated type
- For each FTC, a ‘sync’ controller is launched:
  - Watches federated resources in host cluster
  - Watches target resources in member clusters
  - Ensures target state matches federated state
Sync controller: API interaction

- No explicit type support compiled in
- **FTC** provides *group, version, & resource name* (GVR) for target and federated API types
- **GVR** enables API calls (e.g. Get, Watch)
- **unstructured** (maps of interfaces) used in place of golang structs
  - No type safety
  - But federated resources are consistent
Sync controller: Updates

- Sync controller lacks type details
- How to decide if a target resource is up-to-date?
- Track what was used as input
  - Record hashes of the template and overrides fields
- Track the version of the resulting resource
  - Record resource version or generation
Enough detail?
• Common questions
  ○ Is it necessary to grant KubeFed cluster-admin in member clusters?
  ○ What happens when KubeFed goes down?
  ○ How do I enable cross-cluster service discovery?
  ○ How would I decide between KubeFed and gitops?
Getting involved

• Github:
  ○ [https://sigs.k8s.io/kubefed](https://sigs.k8s.io/kubefed)

• Kube Slack:
  ○ #sig-multicluster on [https://kubernetes.slack.com/](https://kubernetes.slack.com/)

• Mailing list:
  ○ [https://groups.google.com/forum/#!forum/kubernetes-sig-multicluster](https://groups.google.com/forum/#!forum/kubernetes-sig-multicluster)

• Community page:
  ○ [https://github.com/kubernetes/community/tree/master/sig-multicluster](https://github.com/kubernetes/community/tree/master/sig-multicluster)