Operator Workshop @ DevOps Gathering

Marcel Müller
@muemarcel

Giant Swarm

Schedule

09:00 - 10:30 - First half of the workshop

10:30 - 11:00 - Coffee Break?

11:00 - 12:30 - Second half of the workshop

12:30 - 13:30 - Lunch Break

13:30 - 17:00 - Other Workshops :)

Who am I?

Marcel Müller

Platform Engineer @ Giant Swarm

Focus on Operators & Release Engineering

Working with Kubernetes for 2.5 years now

Giant Swarm

Who are you?

(First) Name?

Job description?

Kubernetes experience?

Stuff to install

KIND is already there?

1. Golang > v1.12
2. Kubectl > v1.15
3. Kubectl kustomize in place
4. Executing make files is possible

Giant Swarm

Agenda

1. Introduction & Motivation
2. Custom Resource Definitions
3. Controller Pattern
4. Kubernetes Operators
5. Kubebuilder
6. Hands On!

Giant Swarm

Why use operators?

What do you have in mind?

Open projects?

Why others use operators

- Packaging applications in easier to manage interfaces
    - Prometheus operator
    - Cassandra operator(s)
    - MySQL operator(s)

- Managing infrastructure
    - Aws-operator
    - Cluster-api operator(s)

How does a user interact with an operator?

Custom Resource Definition (CRD)

- Extension of the Kubernetes API
- Registered to Kubernetes at runtime
- Supplied to Kubernetes from the outside
- Simply let you store and retrieve **structured** data
- Offers a **declarative** API for interactions

Source: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

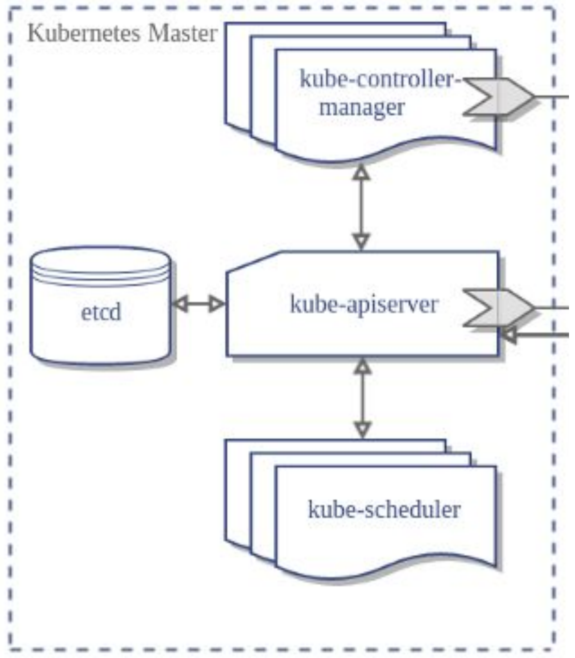
Custom Resource (CR)

- Instance of a CRD
- Comes with Spec / Status / Metadata
- Supports interaction like other kubernetes objects
- Is validated and defaulted against CRD

Source: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

Giant Swarm

Where are CRDs stored?



Source: https://kubernetes.io/blog/2019/04/17/the-future-of-cloud-providers-in-kubernetes/

Example CRD

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                  pattern: '^(\d+|\*)(/\d+)?(\s+(\d+|\*)(/\d+)?){4}$'
                  default: "5 0 * * *"
                image:
                  type: string
...
```

Example CRD

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                  pattern: '^(\d+|\*)(/\d+)?(\s+(\d+|\*)(/\d+)?){4}$'
                  default: "5 0 * * *"
                image:
                  type: string
...
```

Giant Swarm

Example CRD

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                  pattern: '^(\d+|\*)(/\d+)?(\s+(\d+|\*)(/\d+)?){4}$'
                  default: "5 0 * * *"
                image:
                  type: string
...
```

Example CRD

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                  pattern: '^(\d+|\*)(/\d+)?(\s+(\d+|\*)(/\d+)?){4}$'
                  default: "5 0 * * *"
                image:
                  type: string
...
```

Example CRD

```
      ...
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              ...
            status:
              type: object
              properties:
                replicas:
                  type: integer
                labelSelector:
                  type: string
      subresources:
        status: {}
  scope: Namespaced
   names:
     plural: crontabs
     singular: crontab
     kind: CronTab
     shortNames:
       - ct
```

Giant Swarm

Example CRD

```
     ...
     schema:
       openAPIV3Schema:
         type: object
         properties:
           spec:
             ...
           status:
             type: object
             properties:
               replicas:
                 type: integer
               labelSelector:
                 type: string
     subresources:
       status: {}
scope: Namespaced
 names:
   plural: crontabs
   singular: crontab
   kind: CronTab
   shortNames:
     - ct
```

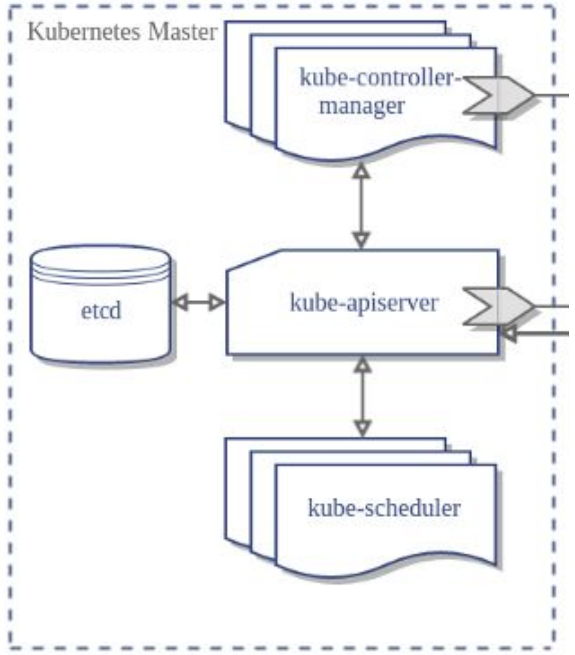
Example CR

```yaml
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
```

Who takes action and when?!

Controllers?

Operators?

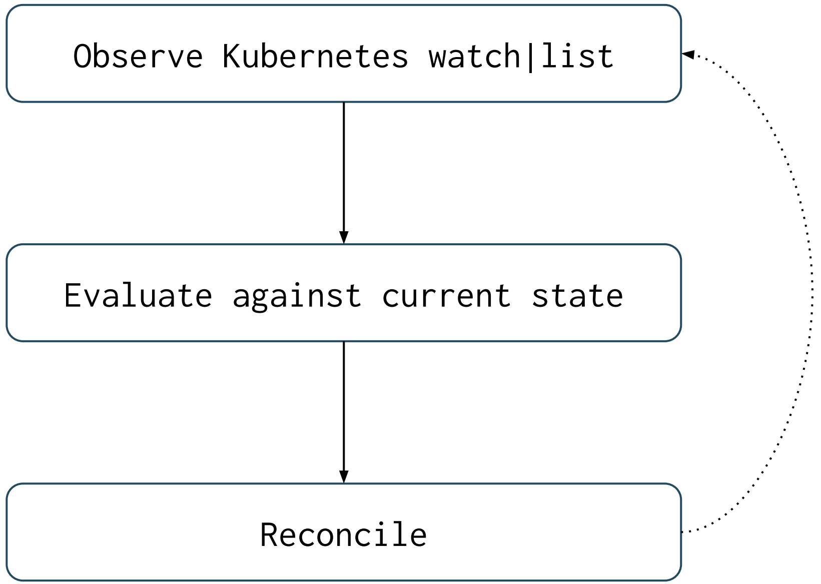


Source: https://kubernetes.io/blog/2019/04/17/the-future-of-cloud-providers-in-kubernetes/

Controller Definition

- A controller watches at least one Kubernetes resource type.
- Objects of this resource type have a spec field that represents the desired state.
- The controller(s) for that resource are responsible for making the current state come closer to that desired state.

Source: https://kubernetes.io/docs/concepts/architecture/controller/



- Desired state in CR Spec
- Current state as reality
- Reconcile by applying diff to current state
- Periodically get desired state through list

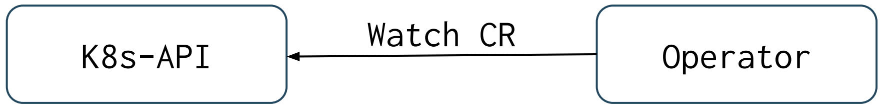
Giant Swarm

Operator Definition

- Kubernetes' controllers concept lets you extend the cluster's behaviour without modifying the code of Kubernetes itself.

- Operators are clients of the Kubernetes API that act as controllers for a Custom Resource.

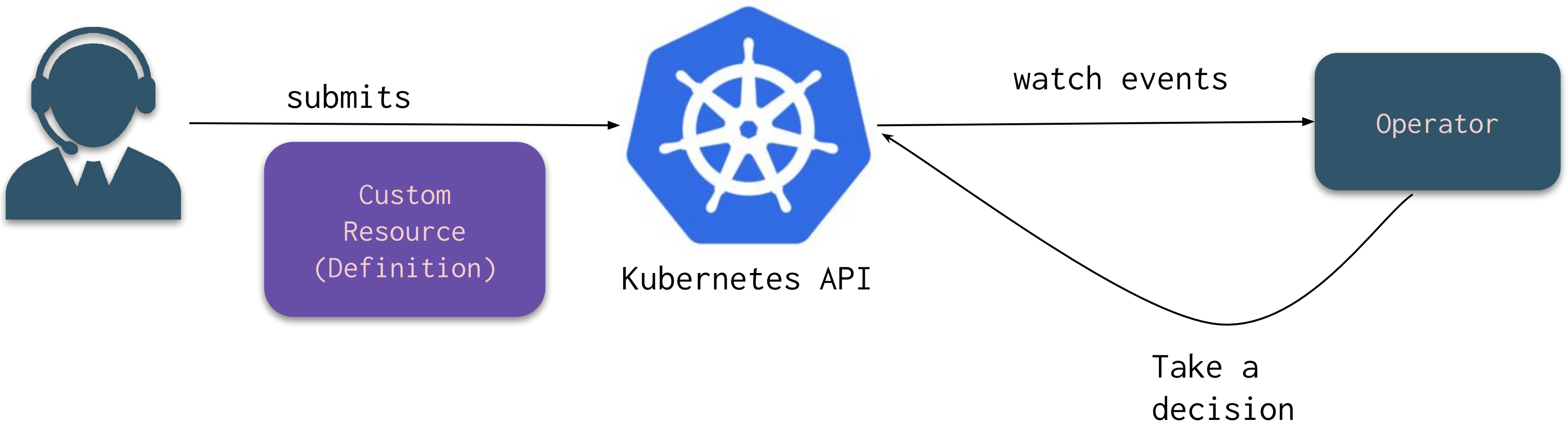
Source: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/



- Operators act like controllers

- Operators are clients of the Kubernetes API

Operator Definition



Example:
Prometheus-Operator

- Watches Prometheus CR
- Creates prometheus pod deployments
- Continuously reconciles desired configuration with actual deployment

https://github.com/coreos/prometheus-operator

Example:
AWS-Operator

- Watches Cluster CR
- Creates kubernetes clusters on AWS matching CR Spec
- Continuously reconciles desired configuration with actual cluster

https://github.com/kubernetes-sigs/cluster-api
https://github.com/giantswarm/aws-operator

Giant Swarm

Okay cool, but how do I build one?

CoreOS Operator Framework

(https://github.com/operator-framework)

Giant Swarm Operatorkit

(https://github.com/giantswarm/operatorkit)

Kubebuilder

(https://github.com/kubernetes-sigs/kubebuilder)

Kudo (https://github.com/kudobuilder/kudo)

...

Kubebuilder

- Code generation for CRDs
- Easy to use reconcile() function
- Very good documentation!
- Golang

https://book.kubebuilder.io/

Giant Swarm

What is the operator idea for this workshop?

codimd-operator

- Read markdown from url
- Public source for markdown https://hackmd.okfn.de/
- Try to parse code snippets as kubernetes deployments
- Create deployments in kubernetes cluster!

https://github.com/giantswarm/codimd-operator

Giant Swarm

codimd-operator

1. Clone the repository

2. Have kind create a local cluster

3. Check again if local requirements are met
   a. Golang > v1.12
   b. Kubectl > v1.15
   c. Kubectl kustomize in place
   d. Executing make files is possible

4. We walk through the existing code together!

codimd-operator - interacting

1. `make install`

2. `kubectl apply -f config/samples/`
    a. Check which CRs and CRDs exist now
    b. Check the content of those CRs
    c. Add your own CR!

3. `make run`
    a. Check how the CR status gets written
    b. Check the created Deployment
    c. Check what happens when you edit the codimd markdown

codimd-operator - finalizers

1. `make install`

2. `kubectl apply -f config/samples/`

3. `make run`
   a. Observe the finalizer addition in CR metadata
   b. Check code adding the finalizer
   c. Manually add finalizers
   d. Check how operator reconciles on deletion

4. Check out deletion of a CR without running operator

codimd-operator - adding fields

1. Add a spec field and use it in the operator!
   a. Base url?
   b. Suffix?

2. Add a status field and write to it in the operator!
   a. `resolves` field?

3. Add validation rules for the spec field

4. Explore the CRD with `kubectl explain`

Giant Swarm

codimd-operator - adding events

1. Check the existing events for creation
2. Add an event for deployment update!
3. Check if your added events take effect by rerunning the operator
4. Describe the CR to see events being written on them

Thank you!

Giant Swarm

Questions?

Stay in touch

- Twitter @muemarcel
- Github MarcelMue
- Meet me at the conference!