

# Operator Workshop @ Containerconf



Marcel Müller  
@muemarcel

---

# Schedule

09:00 - 10:45 - Intro + Theory

10:45 - 11:15 - Coffee Break?

11:15 - 12:30 - Local Dev Environment

12:30 - 13:30 - Lunch Break

13:30 - 14:45 - Finalizer, Events, Status

14:45 - 15:15 - Coffee Break?

15:15 - 16:15 - Deployment, Monitoring

# Who am I?

Marcel Müller

Platform Engineer @ Giant Swarm

Focus on Operators & Release  
Engineering

Working with Kubernetes for 3.5  
years now

---

# Who are you?

(First) Name?

Job description?

Kubernetes experience?

# Stuff to install

KIND is already there?

1. Golang > v1.14
2. Kubectl > v1.18
3. Kubectl kustomize in place
4. Executing make files is possible

# 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

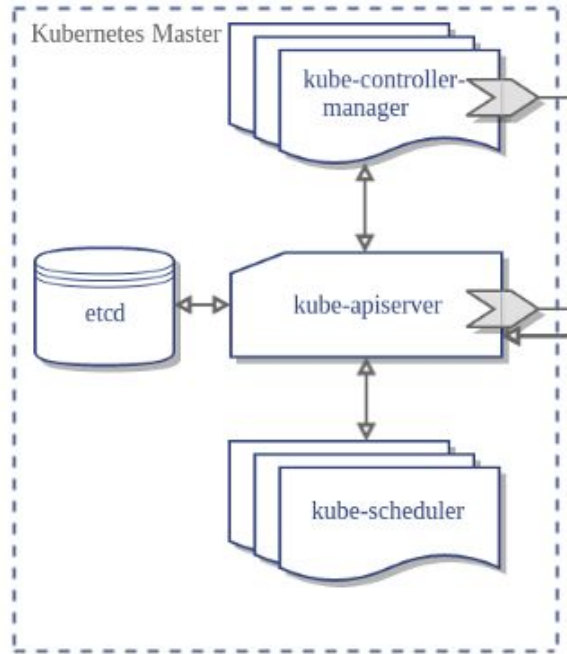
---

# 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/>

# Where are CRDs stored?



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

---

# Example CRD

```
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

```
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

```
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

```
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
```



---

# 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
```

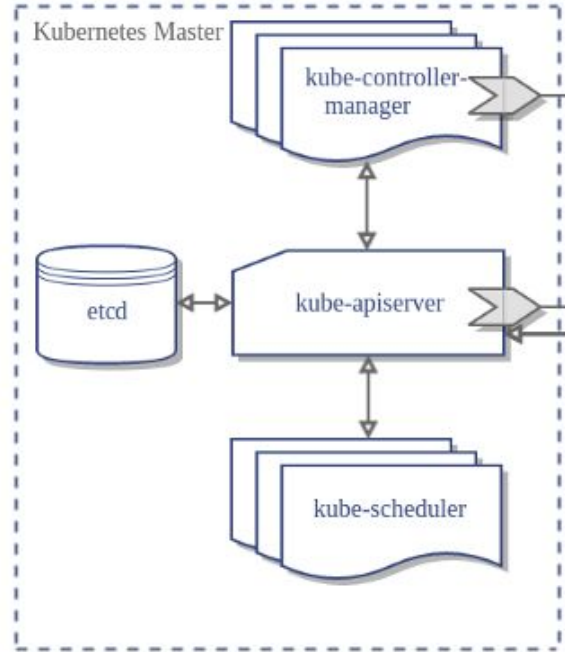
---

# Example CR

```
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?

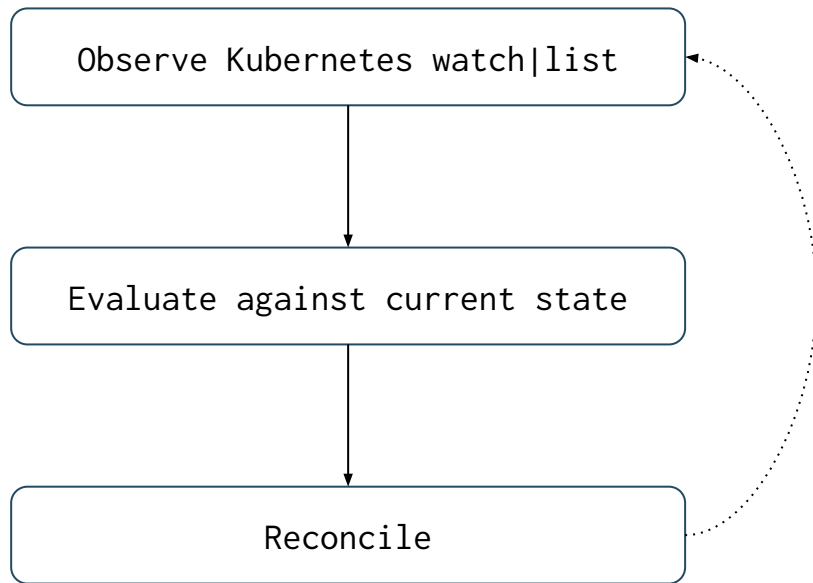


Source: <https://kubernetes.io/blog/2019/04/17/the-future-of-cloud-providers-in-kubernetes/>

---

# 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.



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

---

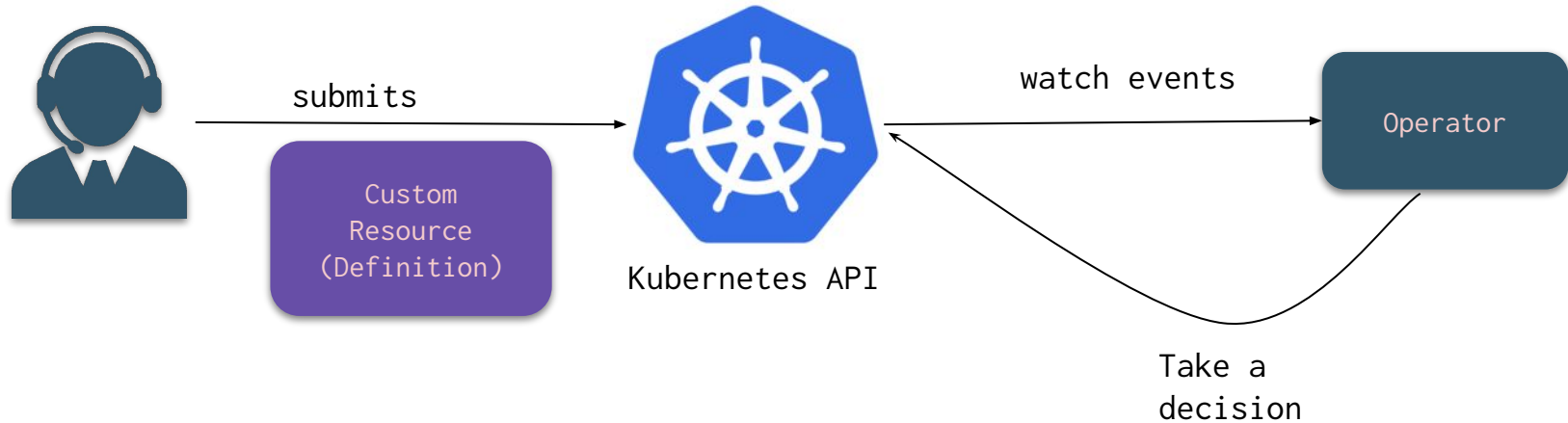
# Operator Definition

- 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.



- 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

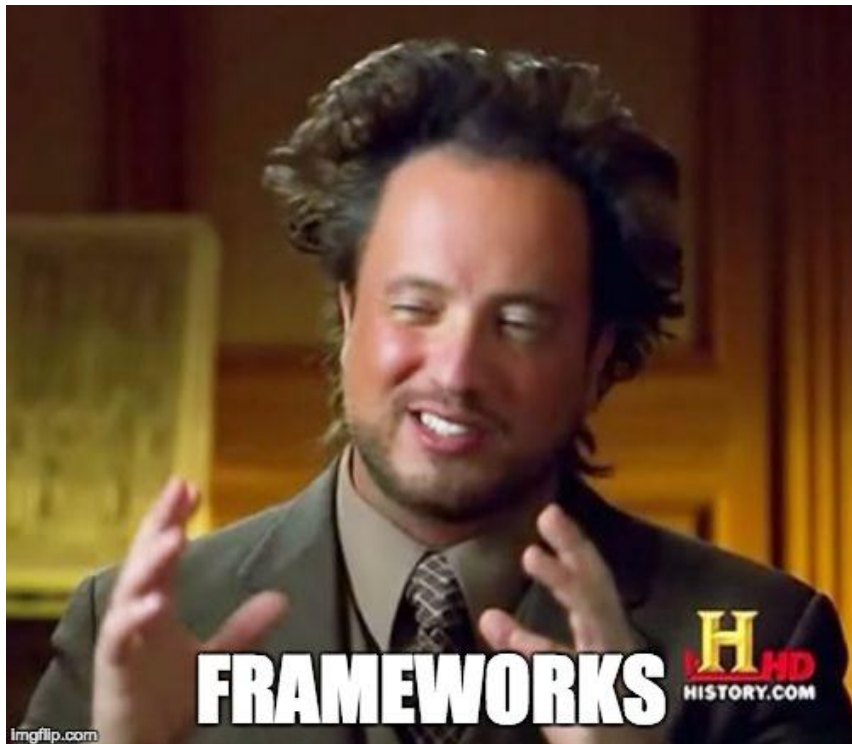
# Example:

## AWS-Operator

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

---

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/>

---

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>

---

# codimd-operator

1. Clone the repository
2. Have kind create a local cluster
3. Check again if local requirements are met
  - a. Golang > v1.14
  - b. Kubectl > v1.18
  - 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`

---

## 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!

## Questions?

### Stay in touch

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