Rage Against the API Machinery: Writing an Operator for Production

Ilya Kislenko
@depohmel ilya@kasten.io

Tom Manville
@tdmanv tom@kasten.io
● Kanister: Open-source Operator
  ○ Framework for application-level data management
  ○ Example Apps: MySQL, MongoDB, PostgreSQL, ElasticSearch
  ○ Generic Infra Support: Volumes, ObjectStore

● K10’s API
  ○ 2 CRD-controllers
  ○ 1 Aggregated API server
Motivation

You’ve decided an Operator solves your problems

- Centralize/automate operational experience
- Coordinate app operations w/ Kubernetes

What now?

- Bootstrapping is simple
- Production is hard
Rule of Thumb

Does your Operator feel like a native Kubernetes API?
Overview

- Extending the Kubernetes API
- Production Requirements
- Papercuts
Overview

- Extending the Kubernetes API
  - Operator Definition
  - Custom Resources
  - API Conventions
  - Bootstrapping Projects
- Production Requirements
- Papercuts
What is an Operator?

Domain
What is an Operator?

Domain

CustomResource
What is an Operator?

- Domain
- CustomResource
- Controller
What is an Operator?

- Domain
- CustomResource
- Controller
- Operator
Operator Design Pattern

Human ops knowledge → Software

- Support complex Operations
- Active reconciliation
- Extensible
CustomResource(Definition)s

CustomResourceDefinition (CRD)
- Defines new API
- Kubernetes API Object
- Scope: Cluster
- Predefined Schema
- In-tree API
- Adds a new endpoint to the API server

CustomResource (CR)
- Instance of CRD
- Also Kubernetes API Object
- Scope: Cluster or Namespace
- Arbitrary Schema
- Out-of-tree API
CustomResourceDefinition (CRD)

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
group: stable.example.com
  versions:
    - name: v1
      storage: true
  scope: Cluster
  names:
    plural: crontabs
    kind: CronTab
    shortNames:
      - ct
```

CustomResource (CR)

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

Follow best practices for:

- Restful
- Naming/Namespaceing
- ObjectMeta
  - Labels
  - Versioning
- Declarative vs. Imperative
- Spec vs. Status

https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md
War of the Operator Kits

- Rook
  https://github.com/rook/operator-kit

- Giant Swarm
  https://github.com/giantswarm/operatorkit
New School

- Operator SDK
  https://github.com/operator-framework/operator-sdk
- Kubebuilder
  https://github.com/kubernetes-sigs/kubebuilder
- Metacontroller
  https://github.com/GoogleCloudPlatform/metacontroller
Overview

- Extending the Kubernetes API
- Production-Ready Features
  - Clients
  - RBAC
  - Safety
  - Events
  - Testing
- Papercuts
**Clients: kubectl + SDKs**

**kubectl**

```bash
$ cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: widget-namespace
EOF
```

**Go SDK**

```go
cli := kubernetes.NewClient()
ns := &v1.Namespace{
    apiVersion: "v1",
    kind: "Namespace",
    metadata:
        name: "widget-namespace",
}
err := cli.CoreV1().Namespaces().Create(ns)
```
RBAC

- RBAC is a double-edged sword
- Users != ServiceAccounts
- CRD != CR
- Follow principle of least privilege

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: widget-operator
rules:
- apiGroups:
  - widgets.company.io
    resources:
    - "*"
    verbs:
    - "*"
```

RBAC is a double-edged sword

Users != ServiceAccounts

CRD != CR

Follow principle of least privilege
Graceful Changes

- Changes are usually what breaks production
- Operators can handle complex state transitions
- You should trust your Operator as you trust your DevOps
- Suggestion: Use one of Workload types
Code Generation

https://github.com/kubernetes/code-generator

- deepcopy-gen
- client-gen
- Informer-gen
- lister-gen

// +genclient
// +genclient:noStatus
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
$ kubectl describe actionset backup-84w5z
...

Events:

<table>
<thead>
<tr>
<th>Type</th>
<th>Reason</th>
<th>Age</th>
<th>From</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>Started Action</td>
<td>42s</td>
<td>Kanister Controller</td>
<td>Executing action backup</td>
</tr>
<tr>
<td>Normal</td>
<td>Started Phase</td>
<td>42s</td>
<td>Kanister Controller</td>
<td>Executing phase dumpToObjectStore</td>
</tr>
<tr>
<td>Normal</td>
<td>Ended Phase</td>
<td>41s</td>
<td>Kanister Controller</td>
<td>Completed phase dumpToObjectStore</td>
</tr>
<tr>
<td>Normal</td>
<td>Update Complete</td>
<td>41s</td>
<td>Kanister Controller</td>
<td>Updated ActionSet 'backup-84w5z'</td>
</tr>
</tbody>
</table>

$ kubectl get events

<table>
<thead>
<tr>
<th>LAST SEEN</th>
<th>FIRST SEEN</th>
<th>COUNT</th>
<th>NAME</th>
<th>KIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>54m</td>
<td>54m</td>
<td>1</td>
<td>backup-84w5z.1565461009ce6235</td>
<td>ActionSet ...</td>
</tr>
<tr>
<td>54m</td>
<td>54m</td>
<td>1</td>
<td>backup-84w5z.156546100ac72a84</td>
<td>ActionSet ...</td>
</tr>
<tr>
<td>54m</td>
<td>54m</td>
<td>1</td>
<td>backup-84w5z.15654610618cb419</td>
<td>ActionSet ...</td>
</tr>
<tr>
<td>54m</td>
<td>54m</td>
<td>1</td>
<td>backup-84w5z.15654610620c7952</td>
<td>ActionSet ...</td>
</tr>
</tbody>
</table>
// Initialize Event Recorder
broadcaster := record.NewBroadcaster()
broadcaster.StartEventWatcher(
    func(event *core.Event) {
        _, err := client.Core().Events(event.Namespace).Create(event)
    },
)
source := core.EventSource{Component: "Widget Controller"}
recorder := broadcaster.NewRecorder(scheme.Scheme, source)

// Record Event
recorder.Event(obj, corev1.EventTypeErrorNormal, "Started", "Started work on Widget!")
stdlib

$ go test .

import "testing"

func TestFail(t *testing.T) {
    t.Fail()
}

ginkgo

import "github.com/onsi/ginkgo"

var _ = ginkgo.Describe("Some behavior",
    func() {
        ginkgo.Context("My test", func() {
            ginkgo.It("may fail", func() {
                ginkgo.Fail("This test failed")
            })
        })
    })
### In-Cluster

```go
config, err := rest.InClusterConfig()
```

### Out-of-Cluster

```go
config, err := clientcmd.NewNonInteractiveDeferredLoadingClientConfig(
    clientcmd.NewDefaultClientConfigLoadingRules(),
    &clientcmd.ConfigOverrides{},
).ClientConfig()
```
// Always return a new Widget with the request name.
reaction := func(action testing.Action) (bool, runtime.Object, error) {
    get, _ := action.(testing.GetAction)
    ret := &v1.Widget{
        ObjectMeta: metav1.ObjectMeta{
            Name: get.GetName(),
        },
    }
    return true, w, nil
}

// Create fake Clientset
cli := fake.NewSimpleClientset()
cli.PrependerReactor("get", "widgets", reaction)
Overview

- Extending the Kubernetes API
- Production-Ready Features
- Papercuts
  - Validation
  - CRD Lifecycle
  - Object Versioning
  - Code Generation
Validation

- You are responsible for validating content
- OpenAPI v3.0 validation
  - Basic schema validation
- Validating Admission Webhook

Admission webhooks are essentially part of the cluster control-plane. You should write and deploy them with great caution.
CRD Lifecycle Options

Create CRD in the controller
- Self-contained dependency
- Controller requires additional permissions

Create CRD during deployment
- Custom logic is difficult
- Controller will fail w/o CRD
CustomResource Versioning

Kubernetes v1.11.0

- Multiple versions supported
- "Nop" Conversion

Future Improvements

- Mutating webhooks for conversion
- Separate validation for different versions
Closing Suggestions

- Follow API conventions
- Bootstrap with a go-based project
- Generate your code
- Setup RBAC for your CRs
- Create Kubernetes Events
- Invest in tests
- Beware of gaps between CRDs and core APIs
Keep the Conversation Going

www.kanister.io

https://github.com/kanisterio/kanister

https://twitter.com/tdmanv

https://twitter.com/depohmel

https://kasten.io
Questions?
Misc. Attributions

Go Gophers
● https://github.com/ashleymcnamara/gophers/blob/master/LICENSE

Ginkgo
● https://github.com/onsi/ginkgo/blob/master/LICENSE

Koala art:
● https://www.instagram.com/anaitsart/