Developing Operators for Kubernetes

Paweł Kopiczko & Ross Fairbanks

Giant Swarm

Tools

- Minikube 0.25

```
$ minikube start --kubernetes-version 'v1.10.0'
```

- Go 1.10
- Git

```
$ git clone
https://github.com/giantswarm/cll-operator-workshop
$GOPATH/src/github.com/giantswarm/cll-operator-workshop
```

Giant Swarm

Introductions

Agenda

- Deployments, Services & CRDs
- Operators and OperatorKit
- Exercise 1: Generating CR clients
- Exercise 2: Operator Structure
- Exercise 3: Operator Resources
- Exercise 4: Deployment to Kubernetes

Giant Swarm

Kubernetes Resources

- Pods
- Deployments
- Services
- Custom Resource Definitions (CRDs)

Custom Resource Definitions

- CRD is a Kubernetes resource that extends the Kubernetes API.
- Custom Resource or a CR is an instance of a CRD.
- When using client-go code generation is used to generate a CRD client.

API endpoints

- Core group

```
/api/v1/pods
```

- Named group

```
/apis/apps/v1/deployments
```

- CRD

```
/apis/GROUP/API_VERSION/memcachedconfigs
```

Giant Swarm

Example CRD

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: memcachedconfigs.workshop.continouslifecycle.london
spec:
  group: workshop.continouslifecycle.london
  version: v1alpha1
  scope: Namespaced
  names:
    plural: memcachedconfigs
    singular: memcachedconfig
    kind: MemcachedConfig
```

Giant Swarm

Example Custom Resource

- Instance of a CRD

```yaml
apiVersion: "workshop.continouslifecycle.london/v1alpha1"
kind: MemcachedConfig
metadata:
  name: my-memcached
spec:
  memory: “4Gi"
  replicas: 3
```

Giant Swarm

Demo

Operators

- Operator is a custom controller combined with a CRD.
- Operators let you manage complex stateful applications on Kubernetes.
- Pattern first proposed by CoreOS with the etcd-operator and prometheus-operator.

Giant Swarm

OperatorKit

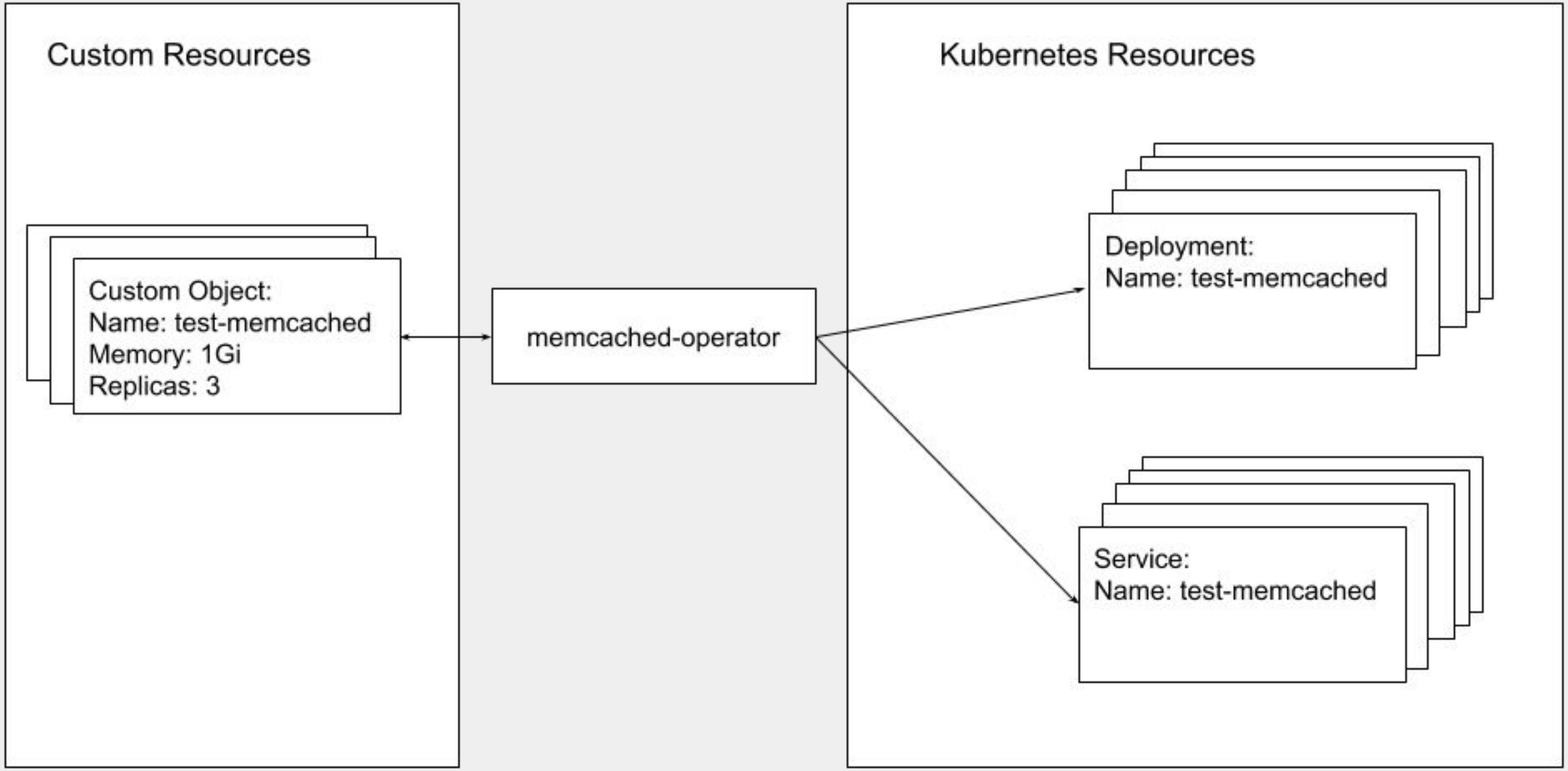
- Library we developed at Giant Swarm to help us develop operators.
- 16 operators in production using it.
- Provides shared logic such as
  - > Resource Framework
  - > Metrics
  - > Finalizer support

Task

memcached-operator

memcached-operator

- Why? Easy to scale as sharding is done client side.
- A complete example but we focus on the operator structure.
- An OperatorKit controller with Services and Deployment resources


Custom Resources
Custom Object:
Name: test-memcached
Memory: 1Gi
Replicas: 3
memcached-operator
Kubernetes Resources
Deployment:
Name: test-memcached
Service:
Name: test-memcached

Giant Swarm

Exercise 1: Generating clients

CLL = Continuous Lifecycle London

Exercise 1: Generating clients

- https://github.com/giantswarm/apiextensions#adding-a-new-group-andor-version
- doc.go
- register.go

Giant Swarm

Exercise 1: Generating clients

```
var MemcachedConfigCRD = &apiextensionsv1beta1.CustomResourceDefinition{
	TypeMeta: metav1.TypeMeta{
		APIVersion: apiextensionsv1beta1.SchemeGroupVersion.String(),
		Kind:       "CustomResourceDefinition",
	},
	ObjectMeta: metav1.ObjectMeta{
		Name: "memcachedconfigs.workshop.continuouslifecycle.london",
	},
	Spec: apiextensionsv1beta1.CustomResourceDefinitionSpec{
		Group:   "workshop.continuouslifecycle.london",
		Scope:   "Namespaced",
		Version: "v1alpha1",
		Names: apiextensionsv1beta1.CustomResourceDefinitionNames{
			Kind:     "MemcachedConfig",
			Plural:   "memcachedconfigs",
			Singular: "memcachedconfig",
```

Giant Swarm

Exercise 1: Generating clients

```
import (
	apiextensionsv1beta1 "k8s.io/apiextensions-apiserver/pkg/apis/apiextensions/v1beta1"
	metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)
```

Exercise 1: Generating clients

```
// +genclient
// +genclient:noStatus
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

type MemcachedConfig struct {
	metav1.TypeMeta   `json:",inline"`
	metav1.ObjectMeta `json:"metadata"`
	Spec              MemcachedConfigSpec `json:"spec"`
}
```

Exercise 1: Generating clients

```
type MemcachedConfigSpec struct {
      // ...
}
```

Giant Swarm

Exercise 1: Generating clients

```
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

type MemcachedConfigList struct {
	metav1.TypeMeta `json:",inline"`
	metav1.ListMeta `json:"metadata"`
	Items           []MemcachedConfig `json:"items"`
}
```

Giant Swarm

Exercise 2: Operator structure

Exercise 2: Operator structure

```
import "github.com/giantswarm/operatorkit/client/k8srestconfig"


k8sRestConfigConfig := k8srestconfig.Config{
        Logger: logger.Default,
        Address:   "os.Getenv("K8S_ADDR")", // export K8S_ADDR=$(minikube ip)
        InCluster: false,
        TLS: k8srestconfig.TLSClientConfig{
                CAFile:  os.Getenv("HOME") + "/.minikube/ca.crt",
                CrtFile: os.Getenv("HOME") + "/.minikube/apiserver.crt",
                KeyFile: os.Getenv("HOME") + "/.minikube/apiserver.key",
        },
}
```

Exercise 2: Operator structure

```
restConfig, err = k8srestconfig.New(k8sRestConfigConfig)
if err != nil {
	return err
}
```

Giant Swarm

Exercise 2: Operator structure

```
import "github.com/giantswarm/cll-operator-workshop/pkg/clientset/versioned"

import apiextensionsclient "k8s.io/apiextensions-apiserver/pkg/client/clientset/clientset"

import "k8s.io/client-go/kubernetes"


k8sClient, err := kubernetes.NewForConfig(restConfig)

if err != nil { ...


k8sExtClient, err := apiextensionsclient.NewForConfig(restConfig)

if err != nil { ...


cllClient, err := versioned.NewForConfig(restConfig)

if err != nil { ...
```

Exercise 2: Operator structure

```
import "github.com/giantswarm/operatorkit/client/k8scrdclient"

crdClientConfig := k8scrdclient.Config{
      Logger: logger.Default,
      K8sExtClient: config.K8sExtClient,
}

crdClient, err = k8scrdclient.New(c)
if err != nil {
      return err
}
```

Giant Swarm

Exercise 2: Operator structure

```
import "github.com/giantswarm/operatorkit/informer"

memcachedInformerConfig := informer.Config{
	Logger: logger.Default,
	Watcher: cllClient.GROUPVAPIVERSION().MemcachedConfigs(""),
}

memcachedInformer, err = informer.New(memcachedInformerConfig)
if err != nil {
	return err
}
```

Giant Swarm

Exercise 2: Operator structure

```go
import "github.com/giantswarm/operatorkit/controller"


// Explained in exercise 3.


resources := []controller.Resource{}


resourceRouter, err := newSimpleResourceRouter(resources)

if err != nil {

	return err

}
```

Exercise 2: Operator structure

```
operatorkitControllerConfig := controller.Config{
        Logger:         logger.Default,
        Name:           "memcached-operator",
        CRD:            NewMemcachedConfigCRD,
        CRDClient:      crdClient,
        Informer:       memcachedInformer,
        RESTClient:     cllClient.GROUPAPIVERSION().RESTClient(),
        ResourceRouter: resourceRouter,
}

operatorkitController, err = controller.New(operatorkitControllerConfig)
if err != nil { ...
```

Giant Swarm

Exercise 3:

Operator resources

Exercise 3: Operator resources

```
type Deployments struct {

        k8sClient kubernetes.Interface

}


func (d *Deployments) Name() string { return "deployments }


func (d *Deployments) EnsureCreated(ctx context.Context, obj interface{}) error {

        memcachedConfig := obj.(*workshopv1alpha1.MemcachedConfig).DeepCopy()

        d.k8sClient.AppsV1().Deployments(memcachedConfig.Namespace)

        ...

}


func (d *Deployments) EnsureDeleted(ctx context.Context, obj interface{}) error { ... }
```

Giant Swarm

Exercise 3: Operator resources

```
resources := []controller.Resource{
        Deployments{
                k8sClient: k8sClient,
        },
        Services{
                k8sClient: k8sClient,
        },
}
```

Exercise 4:

Deployment