

# Developing Operators for Kubernetes

Paweł Kopiczko & Ross Fairbanks



# Tools

- Minikube 0.25

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

- Go 1.10

- Git

```
$ git clone  
https://github.com/giantswarm/c11-operator-workshop  
$GOPATH/src/github.com/giantswarm/c11-operator-workshop
```

# 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

# 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.

# Example CRD

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

# Example Custom Resource

- Instance of a CRD

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



# 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.

# 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

# API endpoints

- Core group

`/api/v1/pods`

- Named group

`/apis/apps/v1/deployments`

- CRD

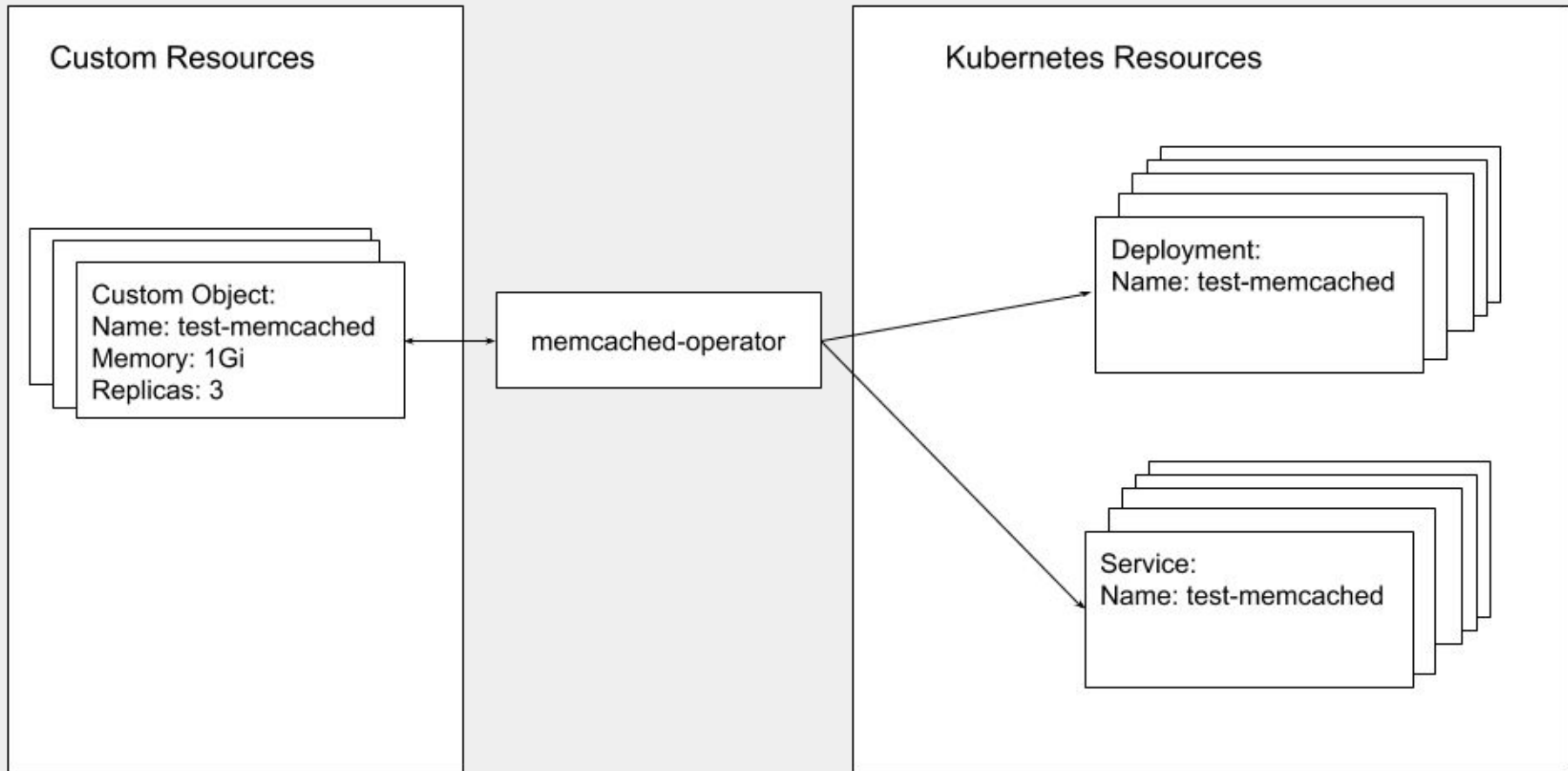
`/apis/GROUP/API_VERSION/memcachedconfigs`

# 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



# Exercise 1: Generating clients



CLL = Continuous Lifecycle London

# Exercise 1: Generating clients

- <https://github.com/giantswarm/apiextensions#adding-a-new-group-and-or-version>
- doc.go
- register.go

# 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",
```

# 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 {  
    // ...  
}
```

# 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"`  
}
```

# 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
}
```

# Exercise 2: Operator structure

```
import "github.com/giantswarm/c11-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 { ...

c11Client, err := versioned.NewForConfig(restConfig)

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

# Exercise 2: Operator structure

```
import "github.com/giantswarm/operatorkit/client/k8scredclient"
```

```
crdClientConfig := k8scredclient.Config{  
    Logger: logger.Default,  
    K8sExtClient: config.K8sExtClient,  
}
```

```
crdClient, err = k8scredclient.New(c)  
if err != nil {  
    return err  
}
```

# 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
}
```

# Exercise 2: Operator structure

```
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 { ...
```

# 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 { ... }
```

# Exercise 3: Operator resources

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

# Exercise 4:

## Deployment