Kubernetes Operator-Framework Workshop

Presented by:

Michael Hrivnak - Principal Software Engineer
Edmund Ochieng - Senior Service Reliability Engineer
Matt Dorn - Principal Software Engineer
HOW MANY PEOPLE HERE USE KUBERNETES/OPENSSHIFT REGULARLY?

DEVELOP IN GOLANG REGULARLY?

USE ANSIBLE REGULARLY?

ANYONE ATTEMPTED TO BUILD A CUSTOM KUBERNETES CONTROLLER FROM SCRATCH?

ANYONE TRIED THE OPERATOR-SDK, ANSIBLE-OPERATOR, OR HELM-APP OPERATOR YET?
OUR GOAL IS TO HELP YOU
SUCCEED WITH OUR TOOLS.
IN THIS ENTRY-LEVEL SESSION WE WILL BE EXPLORING...
Support Channels

Google Groups
https://groups.google.com/forum/#!forum/operator-framework

Kubernetes Slack (slack.kubernetes.io)
#kubernetes-operators

OpenShift Commons - Operator Framework
https://commons.openshift.org/sig/OpenshiftOperators.html
Every third Tuesday of the month at 9am Pacific
So...what is an Operator?
Operators

An Operator represents human operational knowledge in software, to reliably manage an application.
LET'S GO BACK A FEW YEARS.
Introducing Operators: Putting Operational Knowledge into Software

November 03, 2016 • By Brandon Phillips

A Site Reliability Engineer (SRE) is a person that operates an application by writing software. They are an engineer, a developer, who knows how to develop software specifically for a particular application domain. The resulting piece of software has an application’s operational domain knowledge programmed into it.

Our team has been busy in the Kubernetes community designing and implementing this concept to reliably create, configure, and manage complex application instances atop Kubernetes.

We call this new class of software Operators. An Operator is an application-specific controller that extends the Kubernetes API to create, configure, and manage instances of complex stateful applications on behalf of a Kubernetes user. It builds upon the basic Kubernetes resource and controller concepts but includes domain or application-specific knowledge to automate common tasks.
Resources

- Pod
- ConfigMap
- Route

Controllers

- ReplicaSet
- Deployment
- DaemonSet
Domain or Application Specific Knowledge?

- Installing
- Scale (properly)
- Update
- Backup
- Restore
- Self-Heal
- Clean Up
- etc.
An “Operator” takes advantage of what Kubernetes does best...
$ oc proxy

$ curl localhost:8001
$ curl http://localhost:8001/api/v1/ | jq .resources[].name
"bindings"
"componentstatuses"
"configmaps"
"endpoints"
"events"
"limitranges"
"namespaces"
"namespaces/finalize"
"namespaces/status"
"nodes"
"nodes/proxy"
"nodes/status"
"persistentvolumeclaims"
"persistentvolumeclaims/status"
"persistentvolumes"
"persistentvolumes/status"
"Pods"

$ kubectl get configmaps

$ kubectl get endpoints

$ kubectl get namespaces
$ kubectl get pod kube-dns-1187388186-rr1jb -n kube-system -o yaml

(curl -XGET ../api/v1/namespaces/kube-system/pods/kube-dns-1187388186-rr1jb)

apiVersion: v1
group: 
kind: Pod
metadata:
  name: kube-dns-1187388186-rr1jb
  namespace: kube-system
  ownerReferences:...
Spec:
  Containers:
    name: kubedns
    image: gcr.io/google_containers/k8s-dns-kube-dns-amd64:1.14.4
Operators take advantage of Custom Resource Definitions.
CRDs allow us to extend the Kubernetes API.
Let’s extend the Kubernetes API by creating our very own object/resource via CRDs.
Create the CRD

$ cat my-new-crd.yaml

apiVersion: apiextensions.k8s.io/v1beta1
group: db.example.com
kind: CustomResourceDefinition
metadata:
  name: mysql.db.example.com
Spec:
  version: v1
  scope: Namespaced
  names:
    plural: mysqls
    singular: mysql
    kind: MySql
    shortNames:
      - ms

$ kubectl create -f my-new-crd.yaml
Let’s first verify the creation of the CRD object/resource.
## Verify CRD Creation via CLI

```bash
$ kubectl get crd
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>KIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql.db.example.com</td>
<td>CustomResourceDefinition.v1beta1.apiextensions.k8s.io</td>
</tr>
</tbody>
</table>
Verify CRD Creation via API

curl -XGET localhost:8001/apis/apiextensions.k8s.io/v1beta1/customresourcedefinitions

```json
{
    "kind": "CustomResourceDefinitionList",
    "apiVersion": "apiextensions.k8s.io/v1beta1",
    "metadata": {
        "selfLink": "/apis/apiextensions.k8s.io/v1beta1/customresourcedefinitions",
        "resourceVersion": "229273"
    },
    "items": [
        {
            "metadata": {
                "name": "mysql.db.example.com",
                "selfLink": "/apis/apiextensions.k8s.io/v1beta1/customresourcedefinitions/mysql.db.example.com",
                "uid": "8e4d17df-b085-11e7-9176-080027b424ef",
                "resourceVersion": "228836",
                "creationTimestamp": "2017-10-14T02:15:32Z"
            }
        }
    ]
}
```
Let’s now actually verify our new **mysql** resource/object!
Verify New **Database** Resource via CLI

$ kubectl get mysql

No resources found.
Verify New **Database** Resource via API

curl -XGET localhost:8001/apis/db.example.com/v1/namespaces/default/mysqls

```json
{
    "apiVersion": "db.example.com/v1",
    "items": [],
    "kind": "MySqlList",
    "metadata": {
        "resourceVersion": "240591",
        "selfLink": "/apis/stable.example.com/v1/namespaces/default/mysqls"
    }
}
```
Let’s create a new **database** object.
Create a new **mysql** object

```
$ cat new-mysql-object.yaml

apiVersion: "db.example.com/v1"
kind: MySql
metadata:
  name: wordpress
spec:
  user: wp
  password: secret
  foo: bar

$ kubectl create -f new-mysql-object.yaml
```
Let’s verify the creation of the `mysql` object.
Verify **database** object via CLI

```
$ kubectl get mysql
NAME    AGE
wordpress 5s
```

```
$ kubectl get mysql wordpress -o yaml

apiVersion: db.example.com/v1
kind: MySql
metadata:
 .clusterName: ""
 .creationTimestamp: 2017-10-14T03:23:26Z
 .deletionGracePeriodSeconds: null
 .deletionTimestamp: null
 .name: wordpress
 .namespace: default
 .resourceVersion: "238701"
 .selfLink: /apis/db.example.com/v1/namespaces/default/mysqls/wordpress
 .uid: 0afd1584-b08f-11e7-9176-080027b424ef
spec:
 .foo: bar
 .password: secret
 .user: wp
```
A Custom Resource needs a controller to **ACT** upon its presence.
Kubernetes Controllers

- **Observe**: Current state of the cluster.
- **Analyze**: Compare current state to desired state.
- **Act**: Perform all the actions necessary to make current state meet desired state.
apiVersion: db.example.com/v1
kind: MySql
metadata:
  clusterName: ""
  creationTimestamp: 2017-10-14T03:47:21Z
deletionGracePeriodSeconds: null
deletionTimestamp: null
name: wordpress
namespace: default
resourceVersion: "242282"
selfLink: /apis/db.example.com/v1/namespaces/default/mysqls/wordpress
uid: 6228add3-b092-11e7-9176-080027b424ef
spec:
  foo: bar
  password: secret
  user: wp
We need a custom controller to notice the new database object and **ACT**!
ACT?
CREATE.
READ.
UPDATE.
DELETE.
But that’s probably not enough..

- Server startup/shutdown
- Mastering the mysqladmin administrative client
- Using the mysql interactive client
- User account maintenance
- Log file maintenance
- Database backup/copying
- Hardware tuning
- Multiple server setups
- Software updates and upgrades
- File system security
- Server security
- Repair and maintenance
- Crash recovery
- Preventive maintenance
- Understanding the mysqld server daemon
- Performance analysis
- Choosing what else to install (e.g. Apache, Perl +modules, PHP)
- Which version of MySQL (stable, developer, source, binary)
- Creating a user account for the mysql user and group
- Download and unpack a distribution
- Compile source code and install (or rpm)
- Initialize the data directory and grant tables with mysql_install_db
- Starting the server
- Installing Perl DBI support
- Installing PHP
- Installing Apache
- Obtaining and installing the samp_db sample database

- Securing a new MySQL installation
- Running mysqld as an unprivileged user
- Methods of starting the server
- Invoking mysqld directly
- Invoking safe_mysqld
- Invoking mysql.server
- Specifying startup options
- Checking tables at startup
- Shutting down the server
- Regaining control of the server if you can’t connect

- Creating new users and granting privileges
- Determining who can connect from where
- Who should have what privileges?
- Administrator privileges
- Revoking privileges
- Removing users
- Methods: mysqldump vs. direct copying
- Backup policies
- Scheduled cycles
- Update logging
- Consistent and comprehensible file-naming
- Backing up the backup files
- Off-site / off-system backups
- Backing up an entire database with mysqldump
- Compressed backup files
- Backing up individual tables
- Using mysqldump to transfer databases to another server
- mysqldump options (flush-logs, lock-tables, quick, opt)
- Direct copying methods
- Database replication (live and off-line copying)
- Recovering an entire database
- Recovering grant tables
- Recovering from mysqldump vs. tar/cpio files
- Using update logs to replay post-backup queries
- Editing update logs to avoid replaying erroneous queries
- Recovering individual tables
- Default parameters
To recap...
Custom Resource Definitions (CRD)
Third Party Resources (TPR)
Custom Controller
Your Knowledge!
Operators!
Why do Operators matter?
We want an “as-a-service” platform experience!
Build an ecosystem of software on Kubernetes that can be as easy, safe, and reliable to use and operate as a Cloud Service.
Low-touch, remotely managed, one-click-updates.
## Operator Examples

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>Show namespace</th>
<th>Enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>etcd 0.6.1 by CoreOS, Inc</td>
<td>Enabled (1 namespace)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prometheus 0.14.0 by CoreOS, Inc</td>
<td>Enabled (1 namespace)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vault 0.1.3 by CoreOS, Inc</td>
<td>Enabled (1 namespace)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Super easy to deploy an Operator in a Kubernetes environment.
Create the CRD

```yaml
apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: etcdclusters.etcd.database.coreos.com
spec:
  group: etcd.database.coreos.com
  names:
    kind: EtcdCluster
    listKind: EtcdClusterList
    plural: etcdclusters
    shortNames:
      - etcdclus
      - etcd
    singular: etcdcluster
  scope: Namespaced
  version: v1beta2
  versions:
    - name: v1beta2
      served: true
      storage: true
```
Deploy etcd Operator

$ cat etcd-operator.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: etcd-operator
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: etcd-operator
    spec:
      containers:
      - name: etcd-operator
        image: quay.io/coreos/etcd-operator:v0.9.2
        command:
        - etcd-operator
        env:
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        # Uncomment to act for resources in all namespaces. More information in doc/clusterwide.md
        # - --cluster-wide
```
Deploy etcd Operator

$ kubectl create -f etcd-operator.yaml

$ kubectl get pods

<table>
<thead>
<tr>
<th>NAMESPACE</th>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>etcd-operator-67666dc65f-xwfvq</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>1s</td>
</tr>
</tbody>
</table>
$ cat etcd-instance.yaml

apiVersion: "etcd.database.coreos.com/v1beta2"
kind: "EtcdCluster"
metadata:
  name: "example-etcd-cluster"
spec:
  size: 3
  version: "3.2.13"
Deploy etcdCluster

$ kubectl create -f etcd-instance.yaml

$ kubectl get etcdcluster

<table>
<thead>
<tr>
<th>NAMESPACE</th>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>myetcdcluster</td>
<td>1s</td>
</tr>
</tbody>
</table>

$ kubectl get pods

<table>
<thead>
<tr>
<th>NAMESPACE</th>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>etcd-member-84cc6dfbb-b-rsw79</td>
<td>1/1</td>
<td>Running</td>
<td>1</td>
<td>60s</td>
</tr>
<tr>
<td>default</td>
<td>etcd-member-84cc6dfccc-skw29</td>
<td>1/1</td>
<td>Running</td>
<td>1</td>
<td>30s</td>
</tr>
<tr>
<td>default</td>
<td>etcd-member-84cc6dfccc-skw29</td>
<td>1/1</td>
<td>Running</td>
<td>1</td>
<td>15s</td>
</tr>
</tbody>
</table>
How do you create your own Operator?
Life before the Operator SDK...
If only it were a simple as....
Resources

type MyCustomResourceDefinition struct
{
    // API obj kind & schema version
    metav1.TypeMeta

    // Standard object metadata (optional)
    Metadata api.ObjectMeta

    // Describe how the resource appears
    Spec v1beta1.CustomResourceDefinitionSpec

    // State of the CRD
    Status CustomResourceDefinitionStatus
}

Controllers

for {
    current := getCurrentState()
    desired := getDesiredState()
    makeChanges(current, desired)
}
Custom Operators require many building blocks and boilerplate code.
...research/download tools to interact with the API.
# Kubernetes Client Libraries

## Officially-supported Kubernetes client libraries

The following client libraries are officially maintained by Kubernetes SIG API Machinery.

<table>
<thead>
<tr>
<th>Language</th>
<th>Client Library</th>
<th>Sample Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go</td>
<td><a href="https://github.com/kubernetes/client-go/">github.com/kubernetes/client-go/</a></td>
<td>browse</td>
</tr>
<tr>
<td>Python</td>
<td><a href="https://github.com/kubernetes/client/python/">github.com/kubernetes/client/python/</a></td>
<td>browse</td>
</tr>
<tr>
<td>Java</td>
<td><a href="https://github.com/kubernetes-client/java">github.com/kubernetes-client/java</a></td>
<td>browse</td>
</tr>
<tr>
<td>dotnet</td>
<td><a href="https://github.com/kubernetes-client/coleh">github.com/kubernetes-client/coleh</a></td>
<td>browse</td>
</tr>
<tr>
<td>JavaScript</td>
<td><a href="https://github.com/kubernetes-client/javascript">github.com/kubernetes-client/javascript</a></td>
<td>browse</td>
</tr>
</tbody>
</table>

## Community-maintained client libraries

The following Kubernetes API client libraries are provided and maintained by their authors, not the Kubernetes team.

<table>
<thead>
<tr>
<th>Language</th>
<th>Client Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clojure</td>
<td><a href="https://github.com/avostan/clojure-kubernetes">github.com/avostan/clojure-kubernetes</a></td>
</tr>
<tr>
<td>Go</td>
<td><a href="https://github.com/enchang/tk8s">github.com/enchang/tk8s</a></td>
</tr>
<tr>
<td>Java (0.5G)</td>
<td><a href="https://bitbucket.org/mandalaboe/mandalaboe-kubernetes">bitbucket.org/mandalaboe/mandalaboe-kubernetes</a></td>
</tr>
<tr>
<td>Java (0.9G)</td>
<td><a href="https://github.com/fabric8io/kubernetes-client">github.com/fabric8io/kubernetes-client</a></td>
</tr>
<tr>
<td>Lisp</td>
<td><a href="https://github.com/secondoftware/cl-k8s">github.com/secondoftware/cl-k8s</a></td>
</tr>
<tr>
<td>Node.js (TypeScript)</td>
<td><a href="https://github.com/Cypress/node-k8s-client">github.com/Cypress/node-k8s-client</a></td>
</tr>
</tbody>
</table>
Knowledge of informers/shared informers for object cache and event handling.
Communicating desired state/actual state via annotations.
Tracking kube-related resources.
Test scaffolding & repo organization.
package tpr

import {
   "fmt"
}

func CreateTPR(clientSet kubernetes.Interface, name, version, desc string) (*v1beta1.ThirdPartyResource, error) {
   // Initialize third party resource if it does not exist
   tpr, err := clientSet.Extensions().ThirdPartyResources().Get(name)
   if err != nil {
      if errors.IsNotFound(err) {
         tpr := &v1beta1.ThirdPartyResource{
            ObjectMeta: v1.ObjectMeta{
               Name: name,
            },
            Versions: []v1beta1.APIVersion{
               APIVersion: version,
               Name: name,
            },
            Description: desc,
         }
         result, err := clientSet.Extensions().ThirdPartyResources().Create(tpr)
         if err != nil {
            return nil, err
         }
         fmt.Printf("CREATED: \%v\%v\%v\%v\%v", result, tpr)
         return nil, err
      } else {
         fmt.Printf("SKIPPING: already exists \%v\%v", name, tpr)
      }
   } else {
      return tpr, nil
   }
}
We need an easier way to create Operators.

We need an easier way to manage Operators.
Operator Framework

The Operator Framework is an open source toolkit to manage Kubernetes native applications, called Operators, in an effective, automated, and scalable way.

Repositories

- **operator-sdk**
  - SDK for building Kubernetes applications.
  - Provides high level APIs, useful abstractions, and project scaffolding.
  - Go 1.8k 404

- **operator-lifecycle-manager**
  - A management framework for extending Kubernetes with Operators
  - Go 332 141

- **operator-metering**
  - Operator metering is responsible for collecting metrics and other information about what's happening in a Kubernetes cluster, and providing a way to create reports on the collected data.
  - Go 165 37

Pinned repositories

**community-operators**

The canonical source for Kubernetes Operators that appear on OperatorHub.io, OpenShift Container Platform and OKD.

- **operator-metering**
  - Shell 120 149 Updated 6 hours ago

Top languages

- Go
- Java
- Shell
- JavaScript
- Python

Most used topics

- kubernetes
- operator
Operator Lifecycle Manager (OLM)

Enable cluster admins to manage Operators on any Kubernetes cluster (dependency management).
Welcome to OperatorHub.io

OperatorHub.io is a new home for the Kubernetes community to share Operators. Find an existing Operator or list your own today.
Katacoda!

https://learn.openshift.com/training
Bonus Material!

workshop.coreostrain.me