A Kubernetes Operator for etcd
Open Source Tools
Enhancing the Kubernetes experience with open source tooling. Includes provisioning, stateful services and security related projects

Subscription
Reference architecture, online training and SLA support 24x7 for your production Kubernetes deployment

Consulting
Consulting and engineering to make the most of Kubernetes and move you to production quickly

Training
Learn and engage directly alongside our team, with courses for all stages of your Kubernetes journey
“We need to run etcd in Kubernetes.”
“What about a stateful set?”
etcd-0.example.com
etcd-1.example.com

etcd-0.example.com
etcd-1.example.com
Init container:
if !bootstrap and firstLaunch {
  err := addPeerToEtcd(peer)
  if err != nil {
    // Comms failure?
    fail()
  }
  markAsLaunched()
}

Pre-stop hook:
err := removePeerFromEtcd(peer)
if err != nil {
  // Uhh...
  // Shut down anyway?
}
Init container:
if !bootstrap and firstLaunch {
  err := addPeerToEtcd(peer)
  if err != nil {
    // Comms failure?
    fail()
  }
  markAsLaunched()
}

Pre-stop hook:
err := removePeerFromEtcd(peer)
if err != nil {
  // Uhh...
  // Shut down anyway?
}

Only if we’re really removing a peer, not if we just move the Pod...
Init container:
if !bootstrap and firstLaunch {
  err := addPeerToEtcd(peer)
  if err != nil {
    // Comms failure?
    fail()
  }
  markAsLaunched()
}

Pre-stop hook:
err := removePeerFromEtcd(peer)
if err != nil {
  // Uhh...
  // Shut down anyway?
}

Only if we’re really removing a peer, not if we just move the Pod...
Init container:
if !bootstrap and firstLaunch {
    err := addPeerToEtcd(peer)
    if err != nil {
        // Comms failure?
        fail()
    }
    markAsLaunched()
}

Detect bootstrap by trying to dial the cluster? What if it’s just a comms failure? How do we recover from that?

Detect firstLaunch by presence of file on the filesystem? What if we crash halfway through adding ourselves to etcd? Can we retry safely?

Pre-stop hook:
err := removePeerFromEtcd(peer)
if err != nil {
    // Uhh...
    // Shut down anyway?
}

If we scale to zero do we delete all the PVCs and all the data?

If we are bootstrapping, how do we determine the domain names of all the other peers? Query the STS to check the size? What if someone scales it while we’re bootstrapping? Means the etcd pod needs permissions to list stateful sets...

Only if we’re really removing a peer, not if we just move the Pod...
We need an Operator.
“Operators are software extensions to Kubernetes that make use of custom resources to manage applications and their components.”
“The Operator pattern aims to capture the key aim of a human operator [...] Human operators [...] have deep knowledge of how the system ought to behave, how to deploy it, and how to react if there are problems.”
An Operator encodes knowledge.
And many more
An Operator represents human operational knowledge in software, to reliably manage an application.

To demonstrate the Operator concept in running code, we have two concrete examples to announce as open source projects today:

1. The etcd Operator creates, configures, and manages etcd clusters. etcd is a reliable, distributed key-value store introduced by CoreOS for sustaining the most critical data in a distributed system, and is the primary configuration datastore of Kubernetes itself.

2. The Prometheus Operator creates, configures, and manages Prometheus monitoring instances. Prometheus is a powerful monitoring, metrics, and alerting tool, and a Cloud Native Computing Foundation (CNCF) project supported by the CoreOS team.
“How do you actually build an Operator?”
```
$ kubectl api-resources
<table>
<thead>
<tr>
<th>NAME</th>
<th>SHORTNAMES</th>
<th>APIGROUP</th>
<th>NAMESPACED</th>
<th>KIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployments</td>
<td>deploy</td>
<td>apps</td>
<td>true</td>
<td>Deployment</td>
</tr>
<tr>
<td>replicaset</td>
<td>rs</td>
<td>apps</td>
<td>true</td>
<td>ReplicaSet</td>
</tr>
<tr>
<td>statefulset</td>
<td>sts</td>
<td>apps</td>
<td>true</td>
<td>StatefulSet</td>
</tr>
<tr>
<td>pods</td>
<td>po</td>
<td></td>
<td>true</td>
<td>Pod</td>
</tr>
<tr>
<td>etcdclusters</td>
<td></td>
<td>etcd.improbable.io</td>
<td>true</td>
<td>EtcldCluster</td>
</tr>
</tbody>
</table>
```
< many lines omitted >
- apiGroups:
  - etcd.improbable.io
    resources:
    - etcdclusters
    verbs:
    - get
    - list
    - watch

- apiGroups:
  - apps
    resources:
    - replicaset
    verbs:
    - create
    - get
    - list
    - watch

- apiGroups:
  - ""
    resources:
    - services
    verbs:
    - create
    - get
    - list
    - watch
Kubebuilder - SDK for building Kubernetes APIs using CRDs  http://book.kubebuilder.io

k8s-sig-api-machinery

1,432 commits  8 branches  0 packages  37 releases  133 contributors  Apache-2.0
Operator logic
etcd-cluster-operator

1. EtcDCluster

2. Replica Set, Persistent Volume Claim, Service, etc.

etcd-cluster-operator
Design considerations
Be level-triggered.
Do one thing at a time.
The cache might lie to you.
Deploying an etcd Cluster
apiVersion: etcd.improbable.io/v1alpha1
kind: EtcdCluster
metadata:
  name: my-etcd
  namespace: foo
spec:
  replicas: 3
apiVersion: etcd.improbable.io/v1alpha1
kind: EtcdPeer
metadata:
  name: my-etcd-0
  Namespace: foo
spec:
  clusterName: my-etcd
  bootstrap:
    initialClusterState: New
  static:
    initialCluster:
    - name: my-etcd-0
      host: my-etcd-0.my-etcd.foo.svc
    - name: my-etcd-1
      host: my-etcd-1.my-etcd.foo.svc
    - name: my-etcd-2
      host: my-etcd-2.my-etcd.foo.svc
my-etcd-0.my-etcd.foo.svc
my-etcd-1.my-etcd.foo.svc
my-etcd-2.my-etcd.foo.svc

Kubernetes Icons Set — The Linux Foundation — CC-BY-4.0
Scale up
apiVersion: etcd.improbable.io/v1alpha1
kind: EtcdCluster
metadata:
  name: my-etcd
spec:
  replicas: 3 5
$ kubectl scale etcdcluster my-etcd --replicas 5
my-etcd-0.my-etcd.foo.svc
my-etcd-1.my-etcd.foo.svc
my-etcd-2.my-etcd.foo.svc
my-etcd-3.my-etcd.foo.svc
apiVersion: etcd.improbable.io/v1alpha1
kind: EtcdPeer
metadata:
  name: my-etcd-3
  namespace: foo
spec:
  clusterName: my-etcd
  bootstrap:
    initialClusterState: Existing
    static:
      initialCluster:
      - name: my-etcd-0
        host: my-etcd-0.my-etcd.foo.svc
      - name: my-etcd-1
        host: my-etcd-1.my-etcd.foo.svc
      - name: my-etcd-2
        host: my-etcd-2.my-etcd.foo.svc
      - name: my-etcd-3
        host: my-etcd-3.my-etcd.foo.svc
Scale down
$ kubectl scale etcdcluster my-etcd --replicas 1
my-etcd-0.my-etcd.foo.svc
my-etcd-1.my-etcd.foo.svc
my-etcd-2.my-etcd.foo.svc

Jetstack  
my-etcd-0.my-etcd.foo.svc
my-etcd-1.my-etcd.foo.svc
my-etcd-2.my-etcd.foo.svc

Kubernetes Icons Set — The Linux Foundation — CC-BY-4.0
Other features

- Version upgrade
- Backup
- Restore
Testing
$\text{kind create cluster}$

1. $\text{kind create cluster}$

2. $\text{docker build .}$

3. Load images & deploy operator

4. Deploy an EtcdCluster and assert on behaviour
What did we learn?
Operators provide value for applications with complex run-books.
Operators expose application specific APIs that work with existing tooling.
You can build an Operator in any stack, but Go was right for us.
You can end-to-end test Operators on your laptop with kind.
Thank you.

james.laverack@jetstack.io
@JamesLaverack
@JetstackHQ

jetstack.io

We’re hiring! — jetstack.io/careers