Kubernetes in Production: Operating etcd with etcdadm

Daniel Lipovetsky
Software Engineer, Platform9 Systems

April 16, 2019

etcdadm

- CLI to simplify etcd operation, including disaster recovery
- Inspired by lessons learned running Kubernetes in production
- An open-source, community project:
  - https://sigs.k8s.io/etcdadm
- Easy to install
  - `go get sigs.k8s.io/etcdadm`
  - Binary releases coming soon

Lessons Learned in Production

Some definitions

- *Control plane*
  - Group of stateless components
    - apiserver, controller-manager, scheduler
  - One stateful component
    - etcd

Lessons Learned in Production

1. API uptime is critical.
    - Without the API, the cluster is a zombie.
    - All CRD-based services need the API.
2. Many API outages are due to etcd failure.
    - Check component statuses, or apiserver log.
3. Complete etcd failure does happen.
4. Have a **manual** etcd recovery process.
5. Periodic etcd backups are important, but try to recover the latest state, if possible.

How to ensure API uptime

- There are two strategies:
  - Tolerate partial failure.
  - Reduce recovery time.
- You need both.

How to tolerate partial failure

- Deploy multiple control plane replicas.
  - Easier
  - No performance penalty
- Deploy multiple etcd members.
  - Harder
  - Performance penalty

How to reduce recovery time

- Write a service to automate recovery.
  - More complex and less flexible
  - Depends on external APIs
  - Hard to debug and patch
  - Deja vu: *You have to ensure it tolerates partial failure and have a plan to recover from a complete failure.*
- Have a manual recovery process.
  - Can be made simple with tooling
  - Has no dependencies
  - Easy to debug and patch

etcdadm

- Goals:
  - Make it easy to tolerate partial etcd failure
  - Make it easy to have a manual etcd recovery process
  - Work without dependencies on external services like DNS, or networked storage
  - Compose well with other tools
    - Use kubeadm to deploy control plane replicas
- Let's demo!
  - How to deploy a multi-member cluster
  - How to scale the cluster
  - How to recover from failure

How to deploy a multi-member cluster

- Deploy all members atomically
  - Discovery service
  - DNS
  - Static
- Deploy one member, then scale up

How to deploy a multi-member cluster

- etcdadm is designed to deploy one member, then scale up
  - One mechanism to understand
  - No dependencies on DNS or discovery service
  - Easily understood failure
  - Must deploy members sequentially

How to deploy a multi-member cluster

Create the first member

```
172.0.0.1> etcdadm init
```

> Behind the scenes
>
> 1. Generates CA, server and client certificates
> 2. Writes configuration
> 3. Creates and starts systemd service

How to deploy a multi-member cluster

Scale up

1. Copy CA cert/key

```
172.0.0.1> rsync /etc/etcd/pki/ca.* 172.0.0.2:/etc/etcd/pki
```

2. Join the cluster

```
172.0.0.2> etcdadm join https://172.0.0.2:2379
```

> Behind the scenes:
>
> 1. Adds member using etcd API
> 2. Discovers all members using etcd API
> 3. Writes configuration
> 4. Creates and starts systemd service

How to scale down

1. Leave the cluster

```
172.0.0.2> etcdadm reset
```

> Behind the scenes:
>
> 1. Discovers identity of local member
> 2. Removes member using etcd API
> 3. Stops and removes systemd service
> 4. Removes configuration and data

How to handle etcd failure

Some definitions

- *Minority failure*: A partial failure where a majority of members are available
  - Examples: Planned maintenance, network partition, hard disk failure
- *Majority failure*: A partial (or complete) failure where a majority of members are not available
  - Examples: Data center outage, networked storage failure

How to prepare for a planned minority failure

First, consider how many failures your cluster can tolerate.

Then, choose how to prepare:

- Do nothing.
  - High risk.
- Migrate the member.
  - A special procedure.
  - Less data to catch up on after migrating.
- Replace the member.
  - Reuses the scaling procedure: Scale up, then down.
  - More data to catch up on after scaling up.

How to prepare for a planned minority failure

Replace the member prior to maintenance; etcdadm makes this easy.

1. Copy CA cert/key

```
172.0.0.2> rsync /etc/etcd/pki/ca.* 172.0.0.3:/etc/etcd/pki
```

2. Remove the member

```
172.0.0.2> etcdadm reset
```

3. Add its replacement

```
172.0.0.3> etcdadm join https://172.0.0.1:2379
```

How to recover from an unplanned minority failure

If the data is on disk and the member is reachable:

- Tail the etcd log.
- Check for a changed IP. If the IP changed, update the member's peer and client URLs. Then start the etcd service.
- Check for insufficient disk space `df -h /var/lib/etcd`
- Something else? See this great KubeCon talk on debugging etcd.

How to recover from an unplanned minority failure

If the data is not on disk, the member is unreachable, or you don't have time to investigate:

1. Identify the failed member.

```
172.0.0.3> etcdctl.sh member list
7675368186969f2a, started, member1, https://172.0.0.1:2380, https://172.0.0.1:2379
7a085789484825b5, started, member2, https://172.0.0.2:2380, https://172.0.0.2:2379
ffe8a15189b30b53, started, member3, https://172.0.0.3:2380, https://172.0.0.3:2379
```

2. Remove the member.

```
172.0.0.3> etcdctl.sh member remove 7a085789484825b5
```

3. Add its replacement

How to recover from an unplanned majority failure

Fetch a backed-up snapshot, or take a snapshot of some available member.

```
etcdctl.sh snapshot save /tmp/etcd.snapshot
```

Create a new one-member cluster from a snapshot.

```
etcdadm init --snapshot /tmp/etcd.snapshot
```

> Behind the scenes
>
> 1. Generates CA, server and client certificates
> 2. Initializes data directory from snapshot
> 3. Writes configuration
> 4. Creates and starts systemd service

Finally, scale up.

2019 Roadmap

- Implement automation that invokes the etcdadm CLI
- Implement periodic backups
- Improve upgrade support
- What feature would **you** like to see? File an issue in github.com/kubernetes-sigs/etcdadm/issues
- Find us in **#etcdadm** in kubernetes slack

Thank you!

Thanks to everyone at Platform9 Systems, the Cluster Lifecycle Special Interest Group, and the authors of etcd and kubeadm.

Q&A