Safely upgrading Kubernetes clusters
廖宇  Yu Liao (yliaog@github)

Staff Software Engineer at Google Kubernetes team. Active contributor to Kubernetes.
徐超  Chao Xu (caesarxuchao@github)

Software Engineer at Google Kubernetes team. Active contributor to Kubernetes.
Death, taxes, and upgrades
Death, taxes, and upgrades

Upgrades are necessary
Upgrades are necessary

Upgrades are hard to get right
Takeaways

To cluster admins:

- Known gotchas
- Best practices

To contributors:

- Areas that need improvements
- Code review tips
Scope of cluster upgrades

- Cluster networking
- Etcd
- Base OS
- Container runtime
- **Kubernetes binaries**

...
Agenda

- Status quo - upgrade best practices
- Foreseeable problems & solutions
- Unforeseeable problems & suggestions
Before you begin...

What’s the first step of cluster upgrade?
Before you begin...

What’s the *first* step of cluster upgrade?

`etcdctl snapshot save backup.db`
Before you begin...

What’s the *first* step of cluster upgrade?

```
etcdctl snapshot save backup.db
etcdctl snapshot status backup.db
```
Before you begin...

My cluster is of v1.9, can I upgrade to v1.12 directly?
My cluster is of v1.9, can I upgrade to v1.12 directly?

No, one minor version at a time
Before you begin...

Have you read the release notes?
Before you begin...

Have you read the release notes?

`CHANGELOG-version.md`

“Known Issues”
“Action Required”
“Deprecations and removals”
Tools

- Kops
- Kubeadm
- Kubespray
- `cluster-api` in the future
Upgrading master first:

- Upgrade kube-apiserver, controller manager, scheduler
- Upgrade addons
- (optional) Setting up RBAC rules for node bootstrap tokens
Upgrading each node:

- `kubectl drain`
  - use `pod disruption budget` to minimize disruptions
- Upgrade kubelet binary
- `kubectl uncordon`
In-place kubelet upgrade is *NOT* supported.
Agenda

- Status quo - upgrade best practices
- Foreseeable problems & solutions
- Unforeseeable problems & suggestions
Uninterpretable data in etcd

CREATE `batch/v1/job`

encoded in `batch/v1/job`

master

deserialize as `batch/v1/job`
convert to `extensions/v1beta1/job`

store at `registry/jobs/<namespace>/<name>`

GET `batch/v1/job`

encoded in `batch/v1/job`

master

deserialize as `extensions/v1beta1/job`
convert to `batch/v1/job`

read from `registry/jobs/<namespace>/<name>`
Uninterpretable data in etcd

**Created today**

User creates a `batch/v1/job` and stores it in `registry/jobs/<namespace>/<name>`.

Deserialized as `batch/v1/job`.

Convert to `extensions/v1beta1/job`.

**Read 1 year later...**

User fetches `batch/v1/job`.

Encountered a 500 internal server error.

Failed to deserialize as `extensions/v1beta1/job`.

The API server doesn't have the schema.
To contributors:
  • Don’t remove API versions (#52185)
  • Storage migration system

To cluster admins:
  • Kubernetes v1.6+ is safe
  • Read release notes

1. KEPP
2. https://github.com/kubernetes-sigs/kube-storage-version-migrator
Clients are outdated

Is your custom automation depending on API in extensions/v1beta1?

API server will remove extensions/v1beta1 endpoints in 1.15
Clients are outdated

To cluster admins: clients need to be updated before cluster is updated.

To contributors: warn users if to-be-deleted API is used.
Policy breaks after upgrade

```yaml
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
webhooks:
  - name: enforce-image-policy.kubernetes.io
    rules:
      - apiGroups:
        - "batch"
        apiVersions:
        - "v1"
        resources:
        - jobs
```
Policy breaks after upgrade

```yaml
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
webhooks:
  - name: enforce-image-policy.kubernetes.io
    rules:
      - apiGroups:
          - "batch"
        apiVersions:
          - "v1"
        resources:
          - jobs

What if jobs have new versions?

- `batch/v2`
- `awesome_batch/v1`
```
Policy breaks after upgrade

To cluster admins:
- Upgrade webhooks before upgrading clusters

To contributors:
- Fix the API ([KEP](#))
- Are there similar issues in other APIs?
HA master upgrade: inconsistent schema

Code snippet:

```yaml
apiVersion: v1
kind: Pod
spec:
  ...
  ...
```

Diagram:

- Pod Schema:
  - `apiVersion: v1`
  - `kind: Pod`
  - `spec:
      ...
      ...
    `

- HA master:
  - Pod Schema
  - Pod Schema
  - Pod Schema
HA master upgrade: inconsistent schema

New Pod Schema

apiVersion: v1
kind: Pod
spec:
  ...
  tolerations: ...
  ...

Old Pod Schema

apiVersion: v1
kind: Pod
spec:
  ...
  No such field
  ...

Old Pod Schema

apiVersion: v1
kind: Pod
spec:
  ...

Rolling upgrading HA masters...

Reference: https://github.com/kubernetes/kubernetes/issues/46073
HA master upgrade: inconsistent schema

To cluster admins:
  • Don’t use the new APIs until rolling upgrade has completed

To contributors:
  • Consistent HA apiserver configurations
Agenda

- Status quo - upgrade best practices
- Foreseeable problems & solutions
- Unforeseeable problems & suggestions
I cannot foresee unforeseeable problems...
Testing

- OSS test coverage is limited
- Bugs only reveal in specific upgrade path
  - Worked for 1.7.13 -> 1.8.8
  - But failed for 1.7.12 -> 1.8.8
- Bugs only reveal in certain configurations

*Test in as real as possible environment before you upgrade!*
If upgrade goes really wrong...

ABORT
Downgrade blockers

- Addons crash without latest features
  - Addon operators
  - Better addon code review

- Data loss: missing new API or new API fields
  - Multi-phase upgrades
Thank you & a brief summary of best practices

Before upgrade:
- Back up etcd
- Read release notes
- Upgrade clients
- Upgrade configurations

During upgrade:
- Upgrade master before node
- Don’t use new APIs until HA master upgrade is done
- No in-place kubelet upgrade