09/07/24

Cluster Maintenance

OS Upgrades

There are multiple nodes in a cluster. If one node goes down, the users will be impacted according to the pod deployment. For example, since there is multiple replicas of a blue pod, the users accessing the blue application are not impacted, as they're being served through the other blue pod. However, users accessing the green pod are impacted as that was the only pod running the green application.

If the node came back only immediately, then the kubectl process starts and the pods come back online. However, if the node was down for more than five minutes, then the pods are terminated from that node. Kubernetes considers them as dead. If the pods were part of a replica set, then they are recreated on other nodes. When the node comes back online after the pod eviction

09/07/24

timeout, it comes up blanks without any pod scheduled on it. However, since the green pod was not part of a replicaset, it's just gone.

There is a safer way to ~~done~~ do upgrade or maintenance tasks - one can purposely drain the node of all the workloads so that the workloads are moved to other nodes in the cluster. When a node is drained, the pods are gracefully terminated from the node that they are on and recreated

`kubectl drain <node name>`

on another. The node is also cordoned or marked as unschedule, meaning no pods can be scheduled on this node until the restriction is removed. specifically Now that the pods are safe on the other nodes, the particular node can be rebooted. When it comes back online, it is still unschedulable. Then it is needed uncordon it, so that the pods can be scheduled on it again. Remember, the pods that were moved to the other nodes don't automatically fallback. If any of those pods were deleted or if new pods were created in the cluster,

09/07/24

then they would be created on this node. #

Apart from drain and uncordon, there is also another command called 'cordon'. Cordon simply marks a node unschedulable. Unlike drain, it doesn't terminate or move the pods on an existing node. It simply makes sure that new pods are not scheduled on that node.

Kubernetes Software Version

When Kubernetes is installed, a specific version of kubernetes is installed.

kubectl get nodes
output: Name . . . . . Version 1.30.2

The kubernetes version consists of three parts - major version, minor version and patch version. kubernetes releases is found on kubernetes github repository. The 'etcd' cluster and 'CoreDNS' servers have their own versions as they are separate projects.

09/07/24

The release notes of each release provides information about the supported versions of externally dependent apps like 'etcd' and 'coreDNS' etc.

cluster Upgrade Process

The Kubernetes components can have different release versions. Since the kube-api server is the primary component in the control plane and that is the component that all other components talks to, none of the components should ever be at a version higher than the kube-api server. The controller manager and scheduler can be at one version lower. If kube-api server was at X, the controller managers and kube-schedulers can be at X-1, and the kubelet and kube-proxy components can be at two versions lower, X-2.

This is not the case with the kubectl. The kubectl utility could be at 1.11, a version higher than the apiserver; 1.10, the same version as the api server, or 1.9, lower than the api server.

09/07/24

This permissible skew in versions allow to carry out live upgrades. Component by component upgrade is possible if required.

For example the installed version is 1.10 and Kubernetes releases version 1.11 and 1.12. (~~1 min 2 seconds~~).

10/07/24

At any time, kubernetes supports only up to the recent three minor versions. Suppose the current version ~~is~~ installed is 1.10 and the latest version is 1.13. Don't upgrade from 1.10 to 1.13 directly. The recommended approach is to upgrade one minor version at a time. The upgrade process depends on how the cluster is set up. For example, if the cluster is a managed kubernetes cluster deployed on cloud service providers like GCP, GKE lets the admin upgrade the cluster with just a few clicks.

If the cluster is deployed using tools like kubeadm, then the tool can help to plan

10/07/24

and upgrade the cluster. If the cluster is deployed from scratch, then manually upgrade the different components of the cluster.

Assume, there are master or worker nodes. The nodes and components are at version 1.10. Upgrading a cluster involves two major steps. First, upgrade the master node and then upgrade worker nodes.

While the master is being upgraded, the control plane components such as the apiserver, scheduler, and controller-manager go down briefly. The master going down doesn't mean worker nodes and applications on the cluster are impacted. All workloads hosted on the worker nodes continue to serve users as normal. Since the master is down all management functions are down, can't access the cluster using kubectl or other kubernetes api, can't deploy new app or delete or modify existing ones. The controller managers don't function either. If a pod was to fail and new pod won't be automatically created. As long as the nodes and the pods are up,

10/07/24

~~apps~~ apps should be up and users will not be impacted. Once the upgrade is complete and the cluster is back up, it should function normally. At this moment, the master and its components are at 1.11 and worker nodes are at 1.10.

There are different strategies available to upgrade the worker nodes. One is, to upgrade all of them at once, but then the pods are down and users are no longer able to access the apps. Once the upgrade is complete, the nodes are back up, new pods are scheduled, and users can resume access. This strategy requires downtime.

The second strategy is to upgrade one node at a time. First upgrade the first node where the workloads move to the second and third node and users are also from there. Once the first node is upgraded and back up, then update the second and third node one by one.

The third strategy would be to add new nodes to the cluster. Nodes with newer software

10/07/24

version. This is specially convenient in a cloud environment. Move the workload to the new node and remove the old one until all new nodes have the new software version.

Lets say, upgrade from 1.11 to 1.13. kubeadm has an upgrade command that helps in upgrading clusters.

kubeadm upgrade plan

It has a lot of information like the kubeadm version, latest kubeadm version, current cluster version, api server and controller version and their latest version etc. It also tells after upgrading the control plane components, must manually upgrade the kubelet versions on each node. Remember, 'kubeadm' doesn't install or upgrade kubelets. Finally, it gives the command to upgrade the cluster. Also, must upgrade the kubeadm tool itself before upgrading the cluster. The kubeadm also follows the same software version as kubernetes. So to go from 1.11 to 1.13, first upgrade to 1.12 then the

```
apt-get upgrade -y kubeadm=1.12.0-00
```

```
kubeadm upgrade apply v1.12.0
```

Output:

```
. . . .
[upgrade/successful] Success! . . . . .
```

The above command pulls the necessary images and upgrades the cluster components. Once complete, the control plane components are now at 1.12.

The next step is to upgrade the kubelet. Depending on the setup, kubelet may be or not running on master node. In this case, the cluster deployed with the kubeadm has kubelet on the master node, which are used to run the control plane components as part on the master nodes.

```
apt-get upgrade -y kubelet=1.12.0-00
```

```
systemctl restart kubelet
```

10/07/24

Next, the worker nodes. First move the workloads from the first worker node to other node.

master node

```
kubectl drain node-1
```

```
apt-get upgrade -y kubeadm=1.12.0-00
apt-get upgrade -y kubelet=1.12.0-00
kubeadm upgrade node config --kubelet-version v1.12.0
systemctl restart kubelet
```

Then in the master node

```
kubectl uncordon node-1
```

Do the same for node2 and node3.

Official Documentation

kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

10/07/24

Backup and Restore

A good practice is to store source code (here source code means all the code, configuration file related to one's kubernetes configuration) on github repository. That way, it can be maintained by a team. The source code repo should be configured with right backup solutions.

With managed or public source code repositories like github no need to worry about this. With that, even the entire cluster is destroyed, applications on the cluster can be redeployed by simply applying these configuration files on them.

While the declarative approach is the preferred approach, it is not necessary that all of the team members stick to those standards. A better way to backing up resource configuration is to query the kube-api server.

Query the kube-api server using the kubectl or by accessing the api server directly

10/07/24

and save all resource configurations for all objects created on the cluster as a copy.

```
kubectl get all --all-namespaces -o yaml > all-deploy-services.yaml
```

The above command is just for a few resource group. There are many other resource groups that must be considered. There are tools like Ark or now called Velero by HeptIO that can do this. It can help in taking backups of Kubernetes cluster using the Kubernetes api.

etcd The etcd cluster stores information about the state of a cluster. Instead of backing up resource as before, backing up etcd server can be an option. The etcd cluster is hosted on the master nodes, while configuring etcd, specify a location where all the data would be stored, the 'data' directory. The 'data' directory can be configured to be backed up by the backup tool. Etcd also comes with a built-in snapshot solution.

10/07/24

```
ETCDCTL_API=3 etcdctl \ snapshot save
snapshot.db
```

A snapshot file is created by the name in the current directory. If the ~~directory~~ ~~location~~ backup should be in a different location, specify the fullpath.

View the status of the backup

```
ETCDCTL_API=3 etcdctl \ snapshot status
snapshot.db
```

output:

HASH	Revision	Total KEYS	TOTAL SIZE

To restore the cluster from this backup, first -

```
systemctl stop kube-apiserver
```

Then -

10107124

```
ETCDCTL_API=3 etcdctl \
snapshot restore snapshot.db \
--data-dir /var/lib/etcd-from-backup
```

output:

```
I| mvcc: restore compact to 47....
```

When etcd restores from a backup, it initializes a new cluster configuration and configures the members of etcd as new members to a new cluster. This is to prevent a new member ~~from~~ from accidentally joining an existing cluster. On running this command, a new data directory is created. Then configure the etcd configuration file to use the new data directory.

Then →

```
systemctl daemon reload
```

```
systemctl restart etcd
```

Finally -

```
systemctl start kube-apiserver
```

10/07/24

With all the etcd commands -

```
ETCDCTL_API=3 etcdctl \
  snapshot save snapshot.db \
--endpoints=https://127.0.0.1:2379 \
--cacert=/etc/etcd/ca.crt \
--cert=/etc/etcd/etcd-server.crt \
  --key=/etc/etcd/etcd-server.key
```

In managed Kubernetes environment, at times etcd cluster may not be accessible. In that case backup by acquiring the kube-apiserver is probably the better way.

—o—

Logging and Monitoring

Monitor Cluster Components

Kubernetes doesn't come with a full featured built-in monitoring solution. Softwares like metrics server, prometheus, elastic stack, datadog etc do the job.

10/07/24

kubelet is responsible for receiving instructions from the kubernetes api master and running pods on the nodes. The kubelet also contains a sub-component known as cAdvisor or container advisor. cAdvisor is responsible for retrieving metrics from pods and exposing them through the kubelet api to make the metrics available for the metrics server.

```
git clone https://github.com/kubernetes-incubator/metrics-server
```

```
kubectl create -f deploy/1.8+/
```

```
kubectl top node
```

```
kubectl top pod
```

Managing Kubernetes logs

```
kubectl logs -f event-simulator-pod
```

streams the logs live

If the pod has multiple containers, must specify the container name in the command.

```
kubectl logs -f event-simulator-pod event-simulator
```

—o—