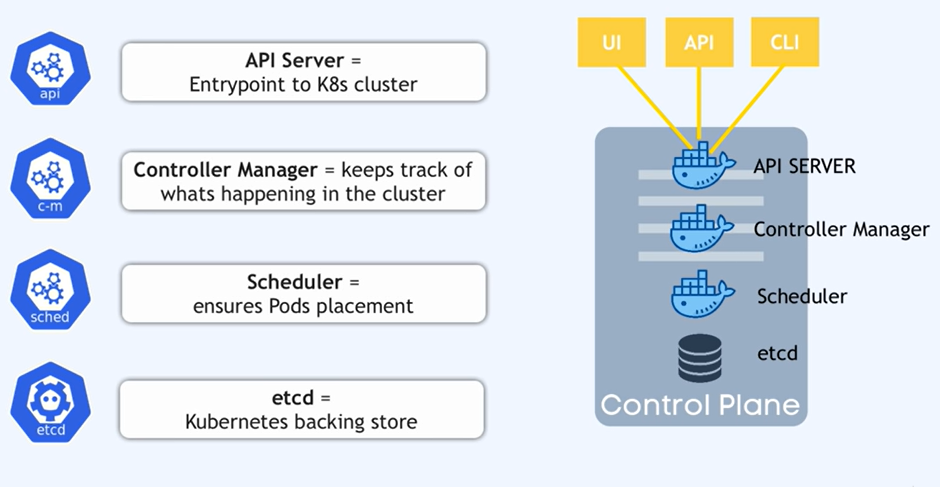
**ETCD is a memory component, and known as brain of Kubernetes**



**Operating etcd clusters for Kubernetes**

etcd is a consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

If your Kubernetes cluster uses etcd as its backing store, make sure you have a [back up](https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#backing-up-an-etcd-cluster) plan for those data.

You can find in-depth information about etcd in the official [documentation](https://etcd.io/docs/).

**Before you begin**

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using [minikube](https://minikube.sigs.k8s.io/docs/tutorials/multi_node/) or you can use one of these Kubernetes playgrounds:

* [Katacoda](https://www.katacoda.com/courses/kubernetes/playground)
* [Play with Kubernetes](http://labs.play-with-k8s.com/)

To check the version, enter kubectl version.

**Prerequisites**

* Run etcd as a cluster of odd members.
* etcd is a leader-based distributed system. Ensure that the leader periodically send heartbeats on time to all followers to keep the cluster stable.
* Ensure that no resource starvation occurs.

Performance and stability of the cluster is sensitive to network and disk I/O. Any resource starvation can lead to heartbeat timeout, causing instability of the cluster. An unstable etcd indicates that no leader is elected. Under such circumstances, a cluster cannot make any changes to its current state, which implies no new pods can be scheduled.

* Keeping etcd clusters stable is critical to the stability of Kubernetes clusters. Therefore, run etcd clusters on dedicated machines or isolated environments for [guaranteed resource requirements](https://etcd.io/docs/current/op-guide/hardware/).
* The minimum recommended version of etcd to run in production is 3.2.10+.

**Resource requirements**

Operating etcd with limited resources is suitable only for testing purposes. For deploying in production, advanced hardware configuration is required. Before deploying etcd in production, see [resource requirement reference](https://etcd.io/docs/current/op-guide/hardware/#example-hardware-configurations).

**Starting etcd clusters**

This section covers starting a single-node and multi-node etcd cluster.

**Single-node etcd cluster**

Use a single-node etcd cluster only for testing purpose.

1. Run the following:
2. etcd --listen-client-urls=http://$PRIVATE\_IP:2379 **\**
3. --advertise-client-urls=http://$PRIVATE\_IP:2379
4. Start the Kubernetes API server with the flag --etcd-servers=$PRIVATE\_IP:2379.

Make sure PRIVATE\_IP is set to your etcd client IP.

**Multi-node etcd cluster**

For durability and high availability, run etcd as a multi-node cluster in production and back it up periodically. A five-member cluster is recommended in production. For more information, see [FAQ documentation](https://etcd.io/docs/current/faq/#what-is-failure-tolerance).

Configure an etcd cluster either by static member information or by dynamic discovery. For more information on clustering, see [etcd clustering documentation](https://etcd.io/docs/current/op-guide/clustering/).

For an example, consider a five-member etcd cluster running with the following client URLs: http://$IP1:2379, http://$IP2:2379, http://$IP3:2379, http://$IP4:2379, and http://$IP5:2379. To start a Kubernetes API server:

1. Run the following:
2. etcd --listen-client-urls=http://$IP1:2379,http://$IP2:2379,http://$IP3:2379,http://$IP4:2379,http://$IP5:2379 --advertise-client-urls=http://$IP1:2379,http://$IP2:2379,http://$IP3:2379,http://$IP4:2379,http://$IP5:2379
3. Start the Kubernetes API servers with the flag --etcd-servers=$IP1:2379,$IP2:2379,$IP3:2379,$IP4:2379,$IP5:2379.

Make sure the IP<n> variables are set to your client IP addresses.

**Multi-node etcd cluster with load balancer**

To run a load balancing etcd cluster:

1. Set up an etcd cluster.
2. Configure a load balancer in front of the etcd cluster. For example, let the address of the load balancer be $LB.
3. Start Kubernetes API Servers with the flag --etcd-servers=$LB:2379.

**Securing etcd clusters**

Access to etcd is equivalent to root permission in the cluster so ideally only the API server should have access to it. Considering the sensitivity of the data, it is recommended to grant permission to only those nodes that require access to etcd clusters.

To secure etcd, either set up firewall rules or use the security features provided by etcd. etcd security features depend on x509 Public Key Infrastructure (PKI). To begin, establish secure communication channels by generating a key and certificate pair. For example, use key pairs peer.key and peer.cert for securing communication between etcd members, and client.key and client.cert for securing communication between etcd and its clients. See the [example scripts](https://github.com/coreos/etcd/tree/master/hack/tls-setup) provided by the etcd project to generate key pairs and CA files for client authentication.

**Securing communication**

To configure etcd with secure peer communication, specify flags --peer-key-file=peer.key and --peer-cert-file=peer.cert, and use HTTPS as the URL schema.

Similarly, to configure etcd with secure client communication, specify flags --key-file=k8sclient.key and --cert-file=k8sclient.cert, and use HTTPS as the URL schema. Here is an example on a client command that uses secure communication:

ETCDCTL\_API=3 etcdctl --endpoints 10.2.0.9:2379 \

--cert=/etc/kubernetes/pki/etcd/server.crt \

--key=/etc/kubernetes/pki/etcd/server.key \

--cacert=/etc/kubernetes/pki/etcd/ca.crt \

member list

**Limiting access of etcd clusters**

After configuring secure communication, restrict the access of etcd cluster to only the Kubernetes API servers. Use TLS authentication to do so.

For example, consider key pairs k8sclient.key and k8sclient.cert that are trusted by the CA etcd.ca. When etcd is configured with --client-cert-auth along with TLS, it verifies the certificates from clients by using system CAs or the CA passed in by --trusted-ca-file flag. Specifying flags --client-cert-auth=true and --trusted-ca-file=etcd.ca will restrict the access to clients with the certificate k8sclient.cert.

Once etcd is configured correctly, only clients with valid certificates can access it. To give Kubernetes API servers the access, configure them with the flags --etcd-certfile=k8sclient.cert, --etcd-keyfile=k8sclient.key and --etcd-cafile=ca.cert.

**Note:** etcd authentication is not currently supported by Kubernetes. For more information, see the related issue [Support Basic Auth for Etcd v2](https://github.com/kubernetes/kubernetes/issues/23398).

**Replacing a failed etcd member**

etcd cluster achieves high availability by tolerating minor member failures. However, to improve the overall health of the cluster, replace failed members immediately. When multiple members fail, replace them one by one. Replacing a failed member involves two steps: removing the failed member and adding a new member.

Though etcd keeps unique member IDs internally, it is recommended to use a unique name for each member to avoid human errors. For example, consider a three-member etcd cluster. Let the URLs be, member1=http://10.0.0.1, member2=http://10.0.0.2, and member3=http://10.0.0.3. When member1 fails, replace it with member4=http://10.0.0.4.

1. Get the member ID of the failed member1:
2. etcdctl --endpoints=http://10.0.0.2,http://10.0.0.3 member list

The following message is displayed:

8211f1d0f64f3269, started, member1, http://10.0.0.1:2380, http://10.0.0.1:2379

91bc3c398fb3c146, started, member2, http://10.0.0.2:2380, http://10.0.0.2:2379

fd422379fda50e48, started, member3, http://10.0.0.3:2380, http://10.0.0.3:2379

1. Remove the failed member:
2. etcdctl member remove 8211f1d0f64f3269

The following message is displayed:

Removed member 8211f1d0f64f3269 from cluster

1. Add the new member:
2. etcdctl member add member4 --peer-urls=http://10.0.0.4:2380

The following message is displayed:

Member 2be1eb8f84b7f63e added to cluster ef37ad9dc622a7c4

1. Start the newly added member on a machine with the IP 10.0.0.4:
2. export ETCD\_NAME="member4"
3. export ETCD\_INITIAL\_CLUSTER="member2=http://10.0.0.2:2380,member3=http://10.0.0.3:2380,member4=http://10.0.0.4:2380"
4. export ETCD\_INITIAL\_CLUSTER\_STATE=existing
5. etcd [flags]
6. Do either of the following:
   1. Update the --etcd-servers flag for the Kubernetes API servers to make Kubernetes aware of the configuration changes, then restart the Kubernetes API servers.
   2. Update the load balancer configuration if a load balancer is used in the deployment.

For more information on cluster reconfiguration, see [etcd reconfiguration documentation](https://etcd.io/docs/current/op-guide/runtime-configuration/#remove-a-member).

**Backing up an etcd cluster**

All Kubernetes objects are stored on etcd. Periodically backing up the etcd cluster data is important to recover Kubernetes clusters under disaster scenarios, such as losing all control plane nodes. The snapshot file contains all the Kubernetes states and critical information. In order to keep the sensitive Kubernetes data safe, encrypt the snapshot files.

Backing up an etcd cluster can be accomplished in two ways: etcd built-in snapshot and volume snapshot.

**Built-in snapshot**

etcd supports built-in snapshot. A snapshot may either be taken from a live member with the etcdctl snapshot save command or by copying the member/snap/db file from an etcd [data directory](https://etcd.io/docs/current/op-guide/configuration/#--data-dir) that is not currently used by an etcd process. Taking the snapshot will not affect the performance of the member.

Below is an example for taking a snapshot of the keyspace served by $ENDPOINT to the file snapshotdb:

ETCDCTL\_API=3 etcdctl --endpoints $ENDPOINT snapshot save snapshotdb

Verify the snapshot:

ETCDCTL\_API=3 etcdctl --write-out=table snapshot status snapshotdb

+----------+----------+------------+------------+

| HASH | REVISION | TOTAL KEYS | TOTAL SIZE |

+----------+----------+------------+------------+

| fe01cf57 | 10 | 7 | 2.1 MB |

+----------+----------+------------+------------+

**Volume snapshot**

If etcd is running on a storage volume that supports backup, such as Amazon Elastic Block Store, back up etcd data by taking a snapshot of the storage volume.

**Snapshot using etcdctl options**

We can also take the snapshot using various options given by etcdctl. For example

ETCDCTL\_API=3 etcdctl -h

will list various options available from etcdctl. For example, you can take a snapshot by specifying the endpoint, certificates etc as shown below:

ETCDCTL\_API=3 etcdctl --endpoints=https://127.0.0.1:2379 **\**

--cacert=<trusted-ca-file> --cert=<cert-file> --key=<key-file> **\**

snapshot save <backup-file-location>

where trusted-ca-file, cert-file and key-file can be obtained from the description of the etcd Pod.

**Scaling up etcd clusters**

Scaling up etcd clusters increases availability by trading off performance. Scaling does not increase cluster performance nor capability. A general rule is not to scale up or down etcd clusters. Do not configure any auto scaling groups for etcd clusters. It is highly recommended to always run a static five-member etcd cluster for production Kubernetes clusters at any officially supported scale.

A reasonable scaling is to upgrade a three-member cluster to a five-member one, when more reliability is desired. See [etcd reconfiguration documentation](https://etcd.io/docs/current/op-guide/runtime-configuration/#remove-a-member) for information on how to add members into an existing cluster.

**Restoring an etcd cluster**

etcd supports restoring from snapshots that are taken from an etcd process of the [major.minor](http://semver.org/) version. Restoring a version from a different patch version of etcd also is supported. A restore operation is employed to recover the data of a failed cluster.

Before starting the restore operation, a snapshot file must be present. It can either be a snapshot file from a previous backup operation, or from a remaining [data directory](https://etcd.io/docs/current/op-guide/configuration/#--data-dir). Here is an example:

ETCDCTL\_API=3 etcdctl --endpoints 10.2.0.9:2379 snapshot restore snapshotdb

Another example for restoring using etcdctl options:

ETCDCTL\_API=3 etcdctl --data-dir <data-dir-location> snapshot restore snapshotdb

For more information and examples on restoring a cluster from a snapshot file, see [etcd disaster recovery documentation](https://etcd.io/docs/current/op-guide/recovery/#restoring-a-cluster).

If the access URLs of the restored cluster is changed from the previous cluster, the Kubernetes API server must be reconfigured accordingly. In this case, restart Kubernetes API servers with the flag --etcd-servers=$NEW\_ETCD\_CLUSTER instead of the flag --etcd-servers=$OLD\_ETCD\_CLUSTER. Replace $NEW\_ETCD\_CLUSTER and $OLD\_ETCD\_CLUSTER with the respective IP addresses. If a load balancer is used in front of an etcd cluster, you might need to update the load balancer instead.

If the majority of etcd members have permanently failed, the etcd cluster is considered failed. In this scenario, Kubernetes cannot make any changes to its current state. Although the scheduled pods might continue to run, no new pods can be scheduled. In such cases, recover the etcd cluster and potentially reconfigure Kubernetes API servers to fix the issue.

**Note:**

If any API servers are running in your cluster, you should not attempt to restore instances of etcd. Instead, follow these steps to restore etcd:

* stop *all* API server instances
* restore state in all etcd instances
* restart all API server instances

We also recommend restarting any components (e.g. kube-scheduler, kube-controller-manager, kubelet) to ensure that they don't rely on some stale data. Note that in practice, the restore takes a bit of time. During the restoration, critical components will lose leader lock and restart themselves.