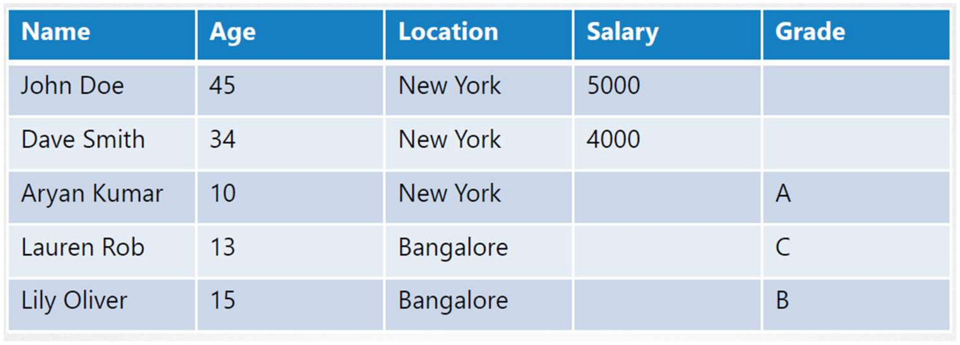
# ETCD

So what is ETCD? It is a distributed reliable key-value store that is simple, secure, and fast. So let's break it up.

So what is a key-value store?

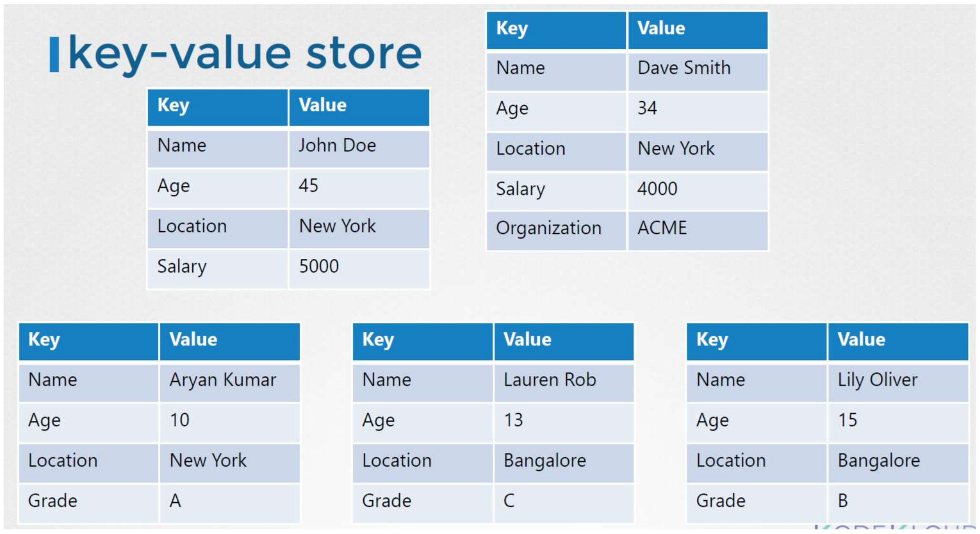
Traditionally, databases have been in a tabular format. You must have heard about SQL or relational databases that store data in the form of rows and columns. For example, here is a table that stores information regarding a few individuals. The row represents each person, and the column represents the type of information being stored. Now, say we want to add additional details about these individuals, for example, salary. So we add an additional column. Doing so impacts the entire table and all of the individuals in them. We then add salary details to this column. However, not all of them are working, so those cells remain empty. We're then required to store grade information. So we add a new column and update the students with their respective grades. Again, only students have grades, so the corresponding fields for adults are empty. Every time new information needs to be added, the entire table is affected and leads to a lot of empty cells.



| Name | Age | Location | Salary | Grade |
|---|---|---|---|---|
| John Doe | 45 | New York | 5000 | |
| Dave Smith | 34 | New York | 4000 | |
| Aryan Kumar | 10 | New York | | A |
| Lauren Rob | 13 | Bangalore | | C |
| Lily Oliver | 15 | Bangalore | | B |

A key-value store stores information in the form of documents or pages. So each individual gets a document and all information about that individual is stored within that file. These files can be in any format or structure, and changes to one file do not affect the others.

The working individuals can have their files with salary fields, and the students can have their pages with grades only. You can add additional details to any of these documents without having to update other similar documents.



## key-value store

| Key | Value |
|---|---|
| Name | John Doe |
| Age | 45 |
| Location | New York |
| Salary | 5000 |

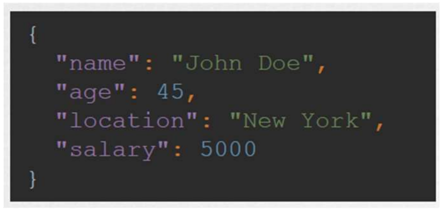
| Key | Value |
|---|---|
| Name | Dave Smith |
| Age | 34 |
| Location | New York |
| Salary | 4000 |
| Organization | ACME |

| Key | Value |
|---|---|
| Name | Aryan Kumar |
| Age | 10 |
| Location | New York |
| Grade | A |

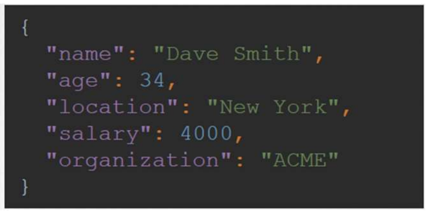
| Key | Value |
|---|---|
| Name | Lauren Rob |
| Age | 13 |
| Location | Bangalore |
| Grade | C |

| Key | Value |
|---|---|
| Name | Lily Oliver |
| Age | 15 |
| Location | Bangalore |
| Grade | B |

While you could store and retrieve simple key and values, when your data gets complex, you typically end up transacting in data formats like JSON or YAML.

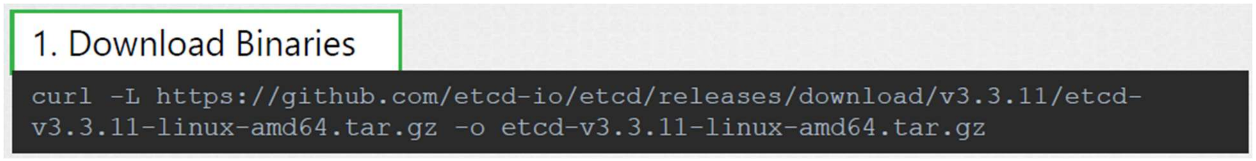


```
{
  "name": "John Doe",
  "age": 45,
  "location": "New York",
  "salary": 5000
}
```



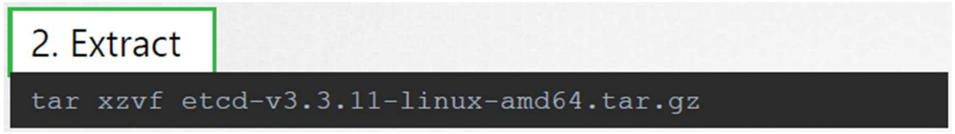
```
{
  "name": "Dave Smith",
  "age": 34,
  "location": "New York",
  "salary": 4000,
  "organization": "ACME"
}
```

It's easy to install and get started with ETCD. Download the binary, extract it, and run. Download the relevant binary for your operating system from the GitHub releases pages, extract it, and run the ETCD executable.



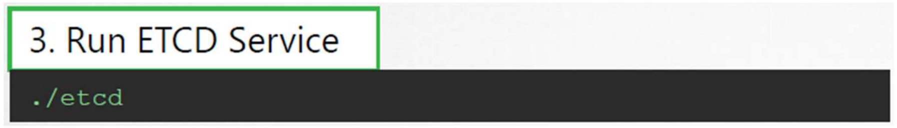
1. Download Binaries

```
curl -L https://github.com/etcd-io/etcd/releases/download/v3.3.11/etcd-
v3.3.11-linux-amd64.tar.gz -o etcd-v3.3.11-linux-amd64.tar.gz
```



2. Extract

```
tar xzvf etcd-v3.3.11-linux-amd64.tar.gz
```



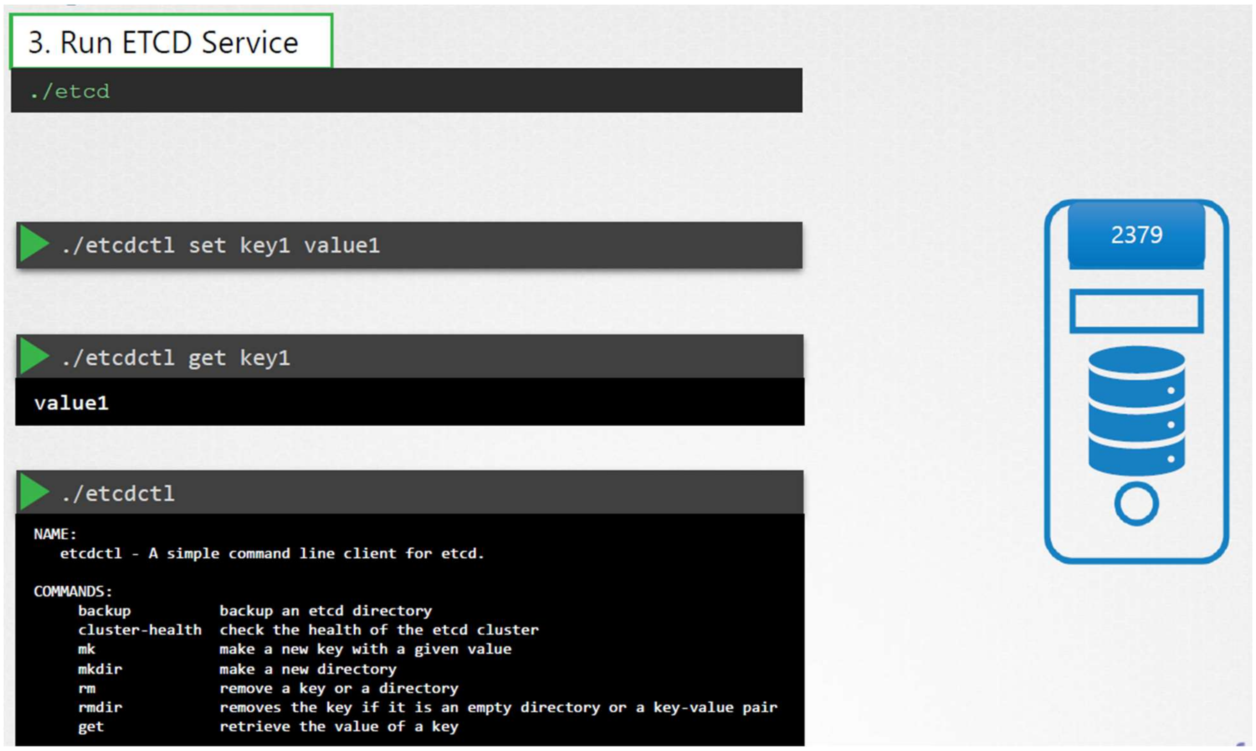
3. Run ETCD Service

```
./etcd
```

When you run ETCD, it starts a service that listens on port 2379 by default. You can then attach any clients to the ETCD service to store and retrieve information.

A default client that comes with ETCD is the ETCD control client. The ETCD control client is a command line client for ETCD. You can use it to store and retrieve key-value pairs.

To store a key-value pair, run the `etcdctl set key1` command followed by the value, value1. This creates an entry in the database with the information. To retrieve the stored data, run the `etcdctl get key1` command. To view more options, run the `etcdctl` command without any arguments.



```
./etcd
```

```
./etcdctl set key1 value1
```

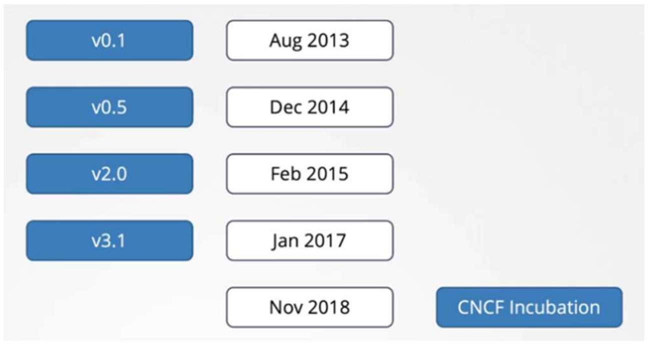
```
./etcdctl get key1
value1
```

```
./etcdctl
NAME:
   etcdctl - A simple command line client for etcd.

COMMANDS:
     backup          backup an etcd directory
     cluster-health  check the health of the etcd cluster
     mk              make a new key with a given value
     mkdir           make a new directory
     rm              remove a key or a directory
     rmdir           removes the key if it is an empty directory or a key-value pair
     get             retrieve the value of a key
```

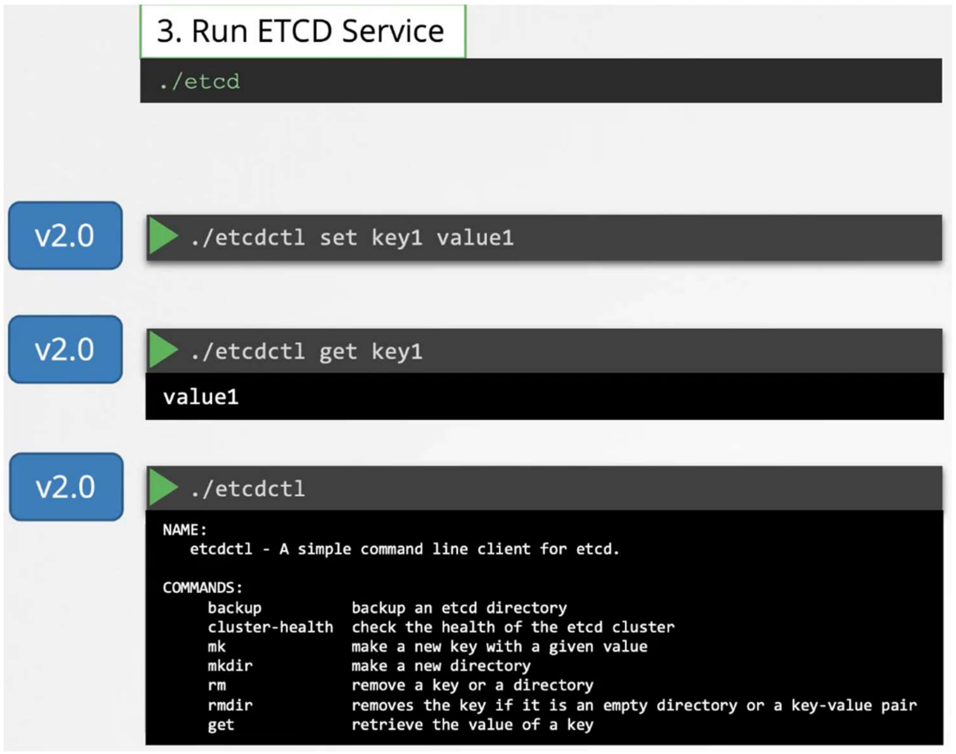
You might come across different versions of commands while exploring ETCD and related articles online. So it's important to understand the history of ETCD releases so you know what changed when.

So the first version of ETCD, version 0.1, was released in August 2013. The official stable version, version 2.0, was released in February 2015, and this is when the RAFT consensus algorithm was redesigned, and it supported more than 10,000 writes per second. And in January 2017, ETCD released version three with a lot more optimizations and performance improvements. And finally, in November 2018, that's when the ETCD project was incubated in CNCF.



So what's most important here is to note the change between version two and version three. So version two had been used a lot, and with version three there were a lot of changes made. So the API versions changed between version two and version three, so that means the etcdctl commands changed as well. So that's what I'm gonna talk about next.

So the set and get commands I showed earlier were for version two and one worked for version three. Your etcdctl command is configured to work with V2 and V3 at the same time.



3. Run ETCD Service

```
./etcd
```

```
./etcdctl set key1 value1
```

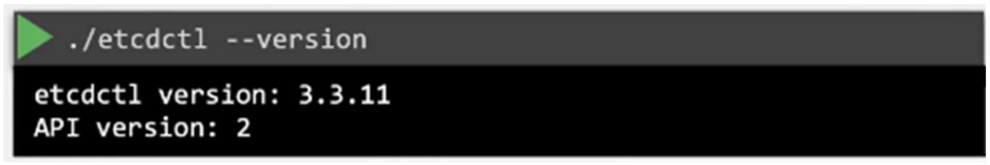
```
./etcdctl get key1
value1
```

```
./etcdctl
NAME:
   etcdctl - A simple command line client for etcd.

COMMANDS:
     backup          backup an etcd directory
     cluster-health  check the health of the etcd cluster
     mk              make a new key with a given value
     mkdir           make a new directory
     rm              remove a key or a directory
     rmdir           removes the key if it is an empty directory or a key-value pair
     get             retrieve the value of a key
```

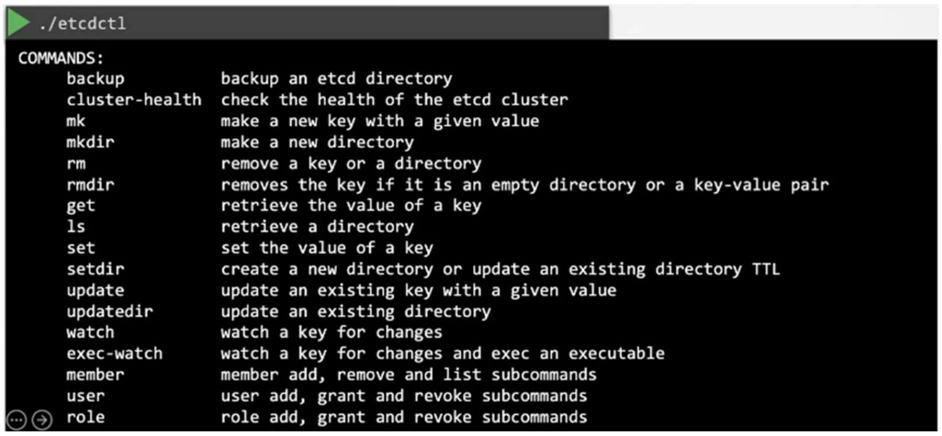
So how do you find out what version etcdctl is configured to work with? Run the etcdctl version command, and it should tell you if it's set to V2 or v3.

Now in this output you can see the etcdctl version itself, which is V3, so that's the first line. However, the API version is two. So remember that there are two types of versions, so one is the etcdctl utility itself, and the other is the API that it is configured to work with, which will either be two or three.



```
./etcdctl --version
etcdctl version: 3.3.11
API version: 2
```

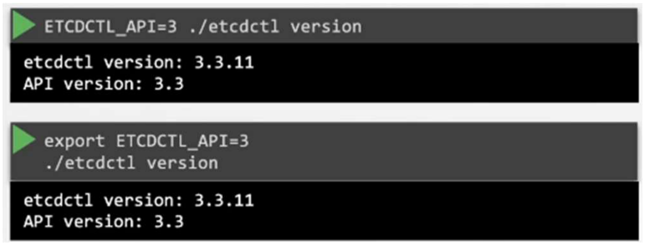
So this means, in the current state, it means that it's configured to work with V2. And so if you run the etcdctl command without any options, then it lists all the commands that it supports, and these are for version two.



```
./etcdctl
COMMANDS:
     backup          backup an etcd directory
     cluster-health  check the health of the etcd cluster
     mk              make a new key with a given value
     mkdir           make a new directory
     rm              remove a key or a directory
     rmdir           removes the key if it is an empty directory or a key-value pair
     get             retrieve the value of a key
     ls              retrieve a directory
     set             set the value of a key
     setdir          create a new directory or update an existing directory TTL
     update          update an existing key with a given value
     updatedir       update an existing directory
     watch           watch a key for changes
     exec-watch      watch a key for changes and exec an executable
     member          member add, remove and list subcommands
     user            user add, grant and revoke subcommands
     role            role add, grant and revoke subcommands
```

And note that with newer versions of ETCD, the default API version is set to V3, so you might want to check what version it is set to using this version command before starting to run the etcdctl commands.

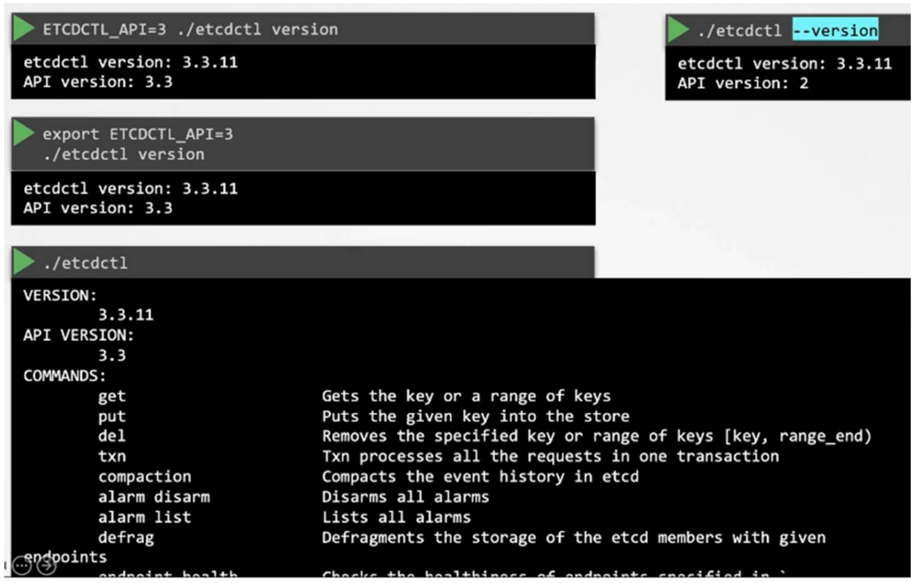
So to change ETCDCTL to work with the API version three, either set the environment variable called ETCDCTL_API to three for each command, so right before you run the command, or you can export it as an environment variable for the entire session using the export command. So that way you don't have to add it before each time the command is run.



```
ETCDCTL_API=3 ./etcdctl version
etcdctl version: 3.3.11
API version: 3.3
```

```
export ETCDCTL_API=3
./etcdctl version
etcdctl version: 3.3.11
API version: 3.3
```

Now also note that, with V3, version is now a command and not an option as it was with V2. So in V2, you had to run `--version`, so that was an option, and now it's a command, so you just run `version`. Now running `etcdctl` command lists the command in the API version three.



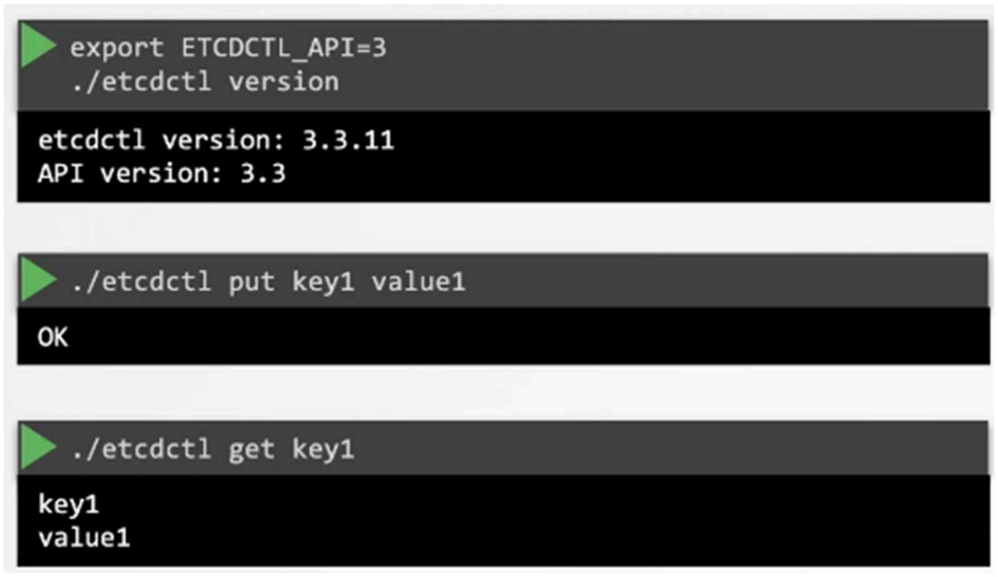
```
ETCDCTL_API=3 ./etcdctl version
etcdctl version: 3.3.11
API version: 3.3
```

```
./etcdctl --version
etcdctl version: 3.3.11
API version: 2
```

```
export ETCDCTL_API=3
./etcdctl version
etcdctl version: 3.3.11
API version: 3.3
```

```
./etcdctl
VERSION:
        3.3.11
API VERSION:
        3.3
COMMANDS:
        get                     Gets the key or a range of keys
        put                     Puts the given key into the store
        del                     Removes the specified key or range of keys [key, range_end)
        txn                     Txn processes all the requests in one transaction
        compaction              Compacts the event history in etcd
        alarm disarm            Disarms all alarms
        alarm list              Lists all alarms
        defrag                  Defragments the storage of the etcd members with given
endpoints
```

So with version three, now the command to set a value is `etcdctl put`, and the command to get a value is `etcdctl get`. And there are many other changes as well in a number of other commands, but that's kind of a high level on how ETCD works and ETCDCTL works.



```
export ETCDCTL_API=3
./etcdctl version

etcdctl version: 3.3.11
API version: 3.3
```

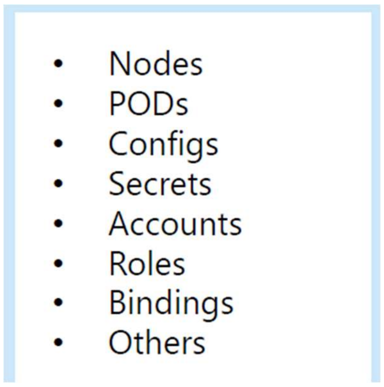
```
./etcdctl put key1 value1

OK
```

```
./etcdctl get key1

key1
value1
```

In the previous lecture, we discussed the basics of etcd. In this lecture, we will talk about etcd's role in Kubernetes. The etcd data store stores information regarding the cluster such as the nodes, pods, configmaps, secrets, accounts, roles, role bindings, and others. Every information you see when you run the `kubectl get` command is from the etcd server.



- Nodes
- PODs
- Configs
- Secrets
- Accounts
- Roles
- Bindings
- Others

Every change you make to your cluster, such as adding additional nodes, deploying pods, or replica sets, is updated in the etcd server. Only once it is updated in the etcd server is the change considered to be complete. Depending on how you set up your cluster, etcd is deployed differently.

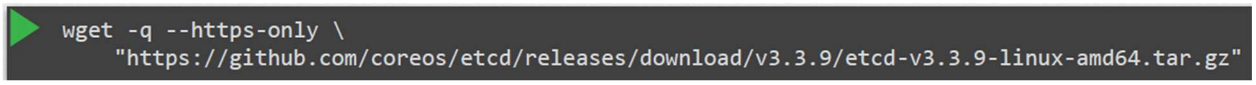
Throughout this section, we discuss two types of Kubernetes deployments: one deployed from scratch and the other using the kubeadm tool. The practice test environments are deployed using the kubeadm tool, and later in this course when we set up a cluster, we set it up from scratch.

If you set up your cluster from scratch, then you deploy etcd by downloading the etcd binaries yourself, installing the binaries, and configuring etcd as a service on your master node yourself.

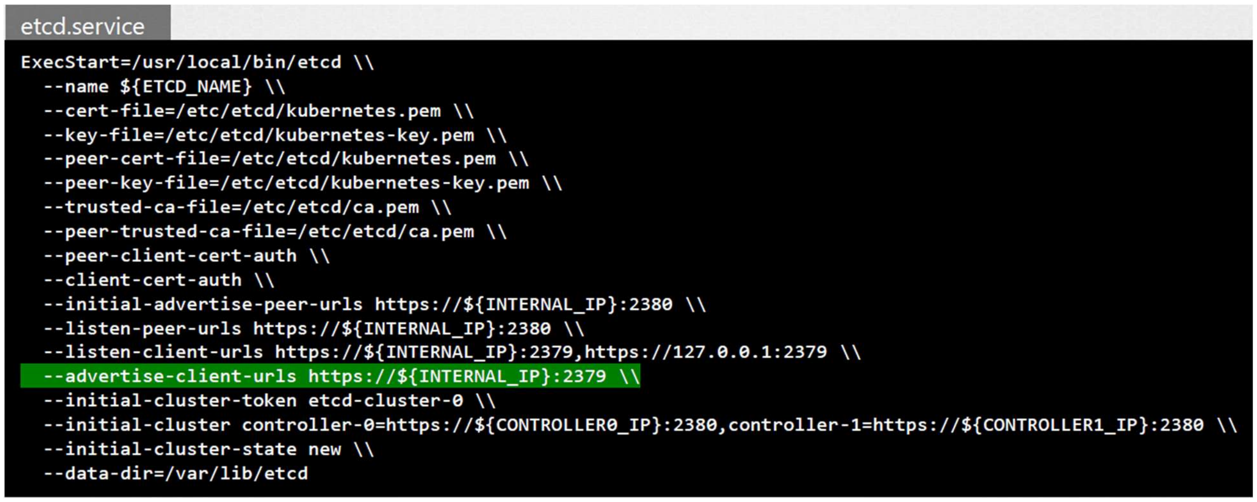
There are many options passed into the service. A number of them relate to certificates. We will learn more about these certificates, how to create them, and how to configure them later in this course. We have a whole section on TLS certificates.

The others are about configuring etcd as a cluster. We will look at those options when we set up high availability in Kubernetes.

The only option to note for now is the advertised client URL. This is the address on which etcd listens. It happens to be on the IP of the server and on port 2379, which is the default port on which etcd listens. This is the URL that should be configured on the kube API server when it tries to reach the etcd server.



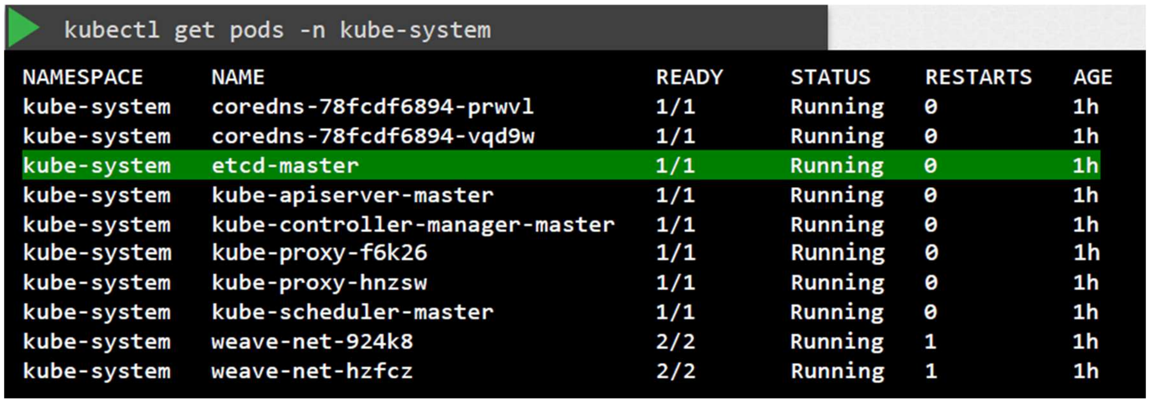
```
wget -q --https-only \
    "https://github.com/coreos/etcd/releases/download/v3.3.9/etcd-v3.3.9-linux-amd64.tar.gz"
```



etcd.service

```
ExecStart=/usr/local/bin/etcd \\
  --name ${ETCD_NAME} \\
  --cert-file=/etc/etcd/kubernetes.pem \\
  --key-file=/etc/etcd/kubernetes-key.pem \\
  --peer-cert-file=/etc/etcd/kubernetes.pem \\
  --peer-key-file=/etc/etcd/kubernetes-key.pem \\
  --trusted-ca-file=/etc/etcd/ca.pem \\
  --peer-trusted-ca-file=/etc/etcd/ca.pem \\
  --peer-client-cert-auth \\
  --client-cert-auth \\
  --initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-client-urls https://${INTERNAL_IP}:2379,https://127.0.0.1:2379 \\
  --advertise-client-urls https://${INTERNAL_IP}:2379 \\
  --initial-cluster-token etcd-cluster-0 \\
  --initial-cluster controller-0=https://${CONTROLLER0_IP}:2380,controller-1=https://${CONTROLLER1_IP}:2380 \\
  --initial-cluster-state new \\
  --data-dir=/var/lib/etcd
```

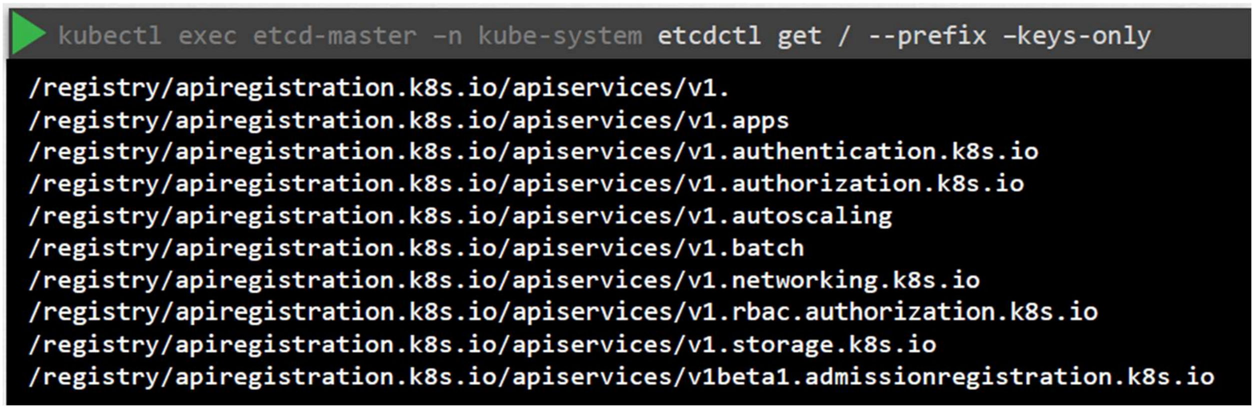
If you set up your cluster using kubeadm, then kubeadm deploys the etcd server for you as a pod in the kube-system namespace.



```
kubectl get pods -n kube-system

NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE
kube-system   coredns-78fcdf6894-prwvl           1/1     Running   0          1h
kube-system   coredns-78fcdf6894-vqd9w           1/1     Running   0          1h
kube-system   etcd-master                        1/1     Running   0          1h
kube-system   kube-apiserver-master              1/1     Running   0          1h
kube-system   kube-controller-manager-master     1/1     Running   0          1h
kube-system   kube-proxy-f6k26                   1/1     Running   0          1h
kube-system   kube-proxy-hnzsw                   1/1     Running   0          1h
kube-system   kube-scheduler-master              1/1     Running   0          1h
kube-system   weave-net-924k8                    2/2     Running   1          1h
kube-system   weave-net-hzfcz                    2/2     Running   1          1h
```

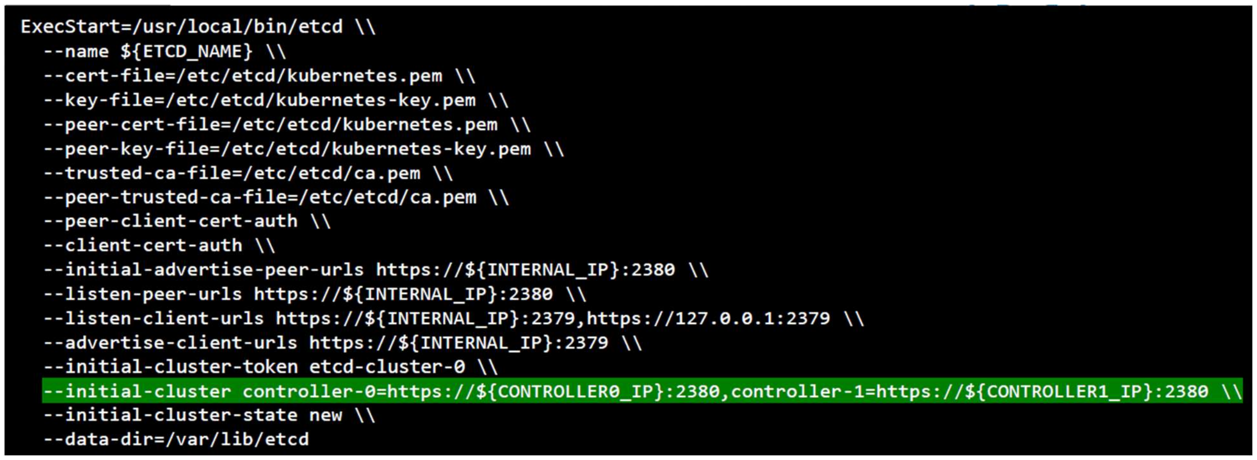
You can explore the etcd database using the etcd control utility within this pod. To list all keys stored by Kubernetes, run the etcd control get command like this.



```
kubectl exec etcd-master -n kube-system etcdctl get / --prefix -keys-only
/registry/apiregistration.k8s.io/apiservices/v1.
/registry/apiregistration.k8s.io/apiservices/v1.apps
/registry/apiregistration.k8s.io/apiservices/v1.authentication.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1.authorization.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1.autoscaling
/registry/apiregistration.k8s.io/apiservices/v1.batch
/registry/apiregistration.k8s.io/apiservices/v1.networking.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1.rbac.authorization.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1.storage.k8s.io
/registry/apiregistration.k8s.io/apiservices/v1beta1.admissionregistration.k8s.io
```

Kubernetes stores data in a specific directory structure. The root directory is a registry, and under that, you have various Kubernetes constructs such as minions or nodes, pods, replica sets, deployments, etc.

In a high availability environment, you will have multiple master nodes in your cluster. Then you will have multiple etcd instances spread across the master nodes. In that case, make sure that the etcd instances know about each other by setting the right parameter in the etcd service configuration. The initial cluster option is where you must specify the different instances of the etcd service. We talk about high availability in much more detail later in this course.



```
ExecStart=/usr/local/bin/etcd \\
  --name ${ETCD_NAME} \\
  --cert-file=/etc/etcd/kubernetes.pem \\
  --key-file=/etc/etcd/kubernetes-key.pem \\
  --peer-cert-file=/etc/etcd/kubernetes.pem \\
  --peer-key-file=/etc/etcd/kubernetes-key.pem \\
  --trusted-ca-file=/etc/etcd/ca.pem \\
  --peer-trusted-ca-file=/etc/etcd/ca.pem \\
  --peer-client-cert-auth \\
  --client-cert-auth \\
  --initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-client-urls https://${INTERNAL_IP}:2379,https://127.0.0.1:2379 \\
  --advertise-client-urls https://${INTERNAL_IP}:2379 \\
  --initial-cluster-token etcd-cluster-0 \\
  --initial-cluster controller-0=https://${CONTROLLER0_IP}:2380,controller-1=https://${CONTROLLER1_IP}:2380 \\
  --initial-cluster-state new \\
  --data-dir=/var/lib/etcd
```