

A Guide to Setting up Zookeeper

by Iwada Eja & Yashar Ahmadi

[Apache ZooKeeper](#) is an open-source software that allows for extremely resilient and dependable distributed coordination. It is often used to manage configuration information, naming services, distributed synchronization, quorum, and state in distributed systems. ZooKeeper is also used by distributed systems to implement consensus, leader election, and group management.

In this guide, we'll explore installing and configuring Apache Zookeeper 3.4.13 on Ubuntu 20.04. ZooKeeper is also designed to be replicated over a collection of hosts, known as an ensemble, in order to achieve high availability and resilience. we'll start by installing a single-node ZooKeeper server on its own, then explore building up a multi-node cluster. The solo installation is beneficial in development and testing environments, but for production environments, a cluster is the most practical choice.

Initial Setup

As Zookeeper relies on [Java](#), we'll also begin the guide by setting up Java and other prerequisite. we'll be also using a Ubuntu 18.04 server with 4GB RAM for the Standalone installation.

However, because ZooKeeper keeps data in memory to achieve high throughput and low latency, production systems work best with 8GB of RAM. Lower amounts of RAM may lead to JVM swapping, which could cause ZooKeeper server latency. High ZooKeeper server latency could result in issues like client session timeouts that would have an adverse impact on system functionality¹.

Step 1 — Creating a New User

we'll create a new user **zookeeper** to use throughout this guide.

```
sudo useradd zookeeper -m && usermod --shell /bin/bash zookeeper
sudo passwd zookeeper
usermod -aG sudo zookeeper
```

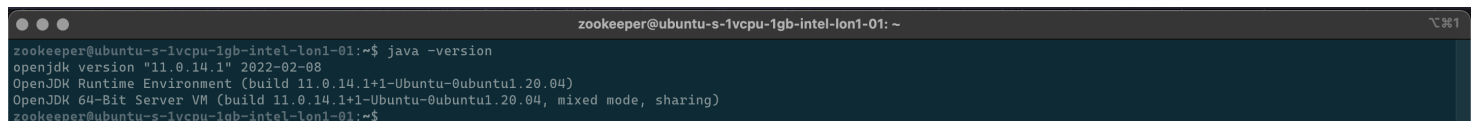
Step 2 — Installing Java

The following command installs the default Java Runtime Environment (JRE), from OpenJDK 11:

```
sudo apt install default-jre
```

To verify the installation with:

```
java -version
```

A terminal window with a dark background. The title bar shows 'zookeeper@ubuntu-s-1vcpu-1gb-intel-lon1-01: ~'. The terminal text shows the command 'java -version' and its output: 'openjdk version "11.0.14.1" 2022-02-08', 'OpenJDK Runtime Environment (build 11.0.14.1+1-Ubuntu-0ubuntu1.20.04)', and 'OpenJDK 64-Bit Server VM (build 11.0.14.1+1-Ubuntu-0ubuntu1.20.04, mixed mode, sharing)'.

```
zookeeper@ubuntu-s-1vcpu-1gb-intel-lon1-01:~$ java -version
openjdk version "11.0.14.1" 2022-02-08
OpenJDK Runtime Environment (build 11.0.14.1+1-Ubuntu-0ubuntu1.20.04)
OpenJDK 64-Bit Server VM (build 11.0.14.1+1-Ubuntu-0ubuntu1.20.04, mixed mode, sharing)
zookeeper@ubuntu-s-1vcpu-1gb-intel-lon1-01:~$
```

Zookeeper would need `JAVA_HOME` environment variable set.

Running `sudo update-alternatives --config java` would give us the path to our Java installation. We set this path by adding it to the relevant environment file

Step 3 — Creating a Data Directory for ZooKeeper

To survive a reboot, ZooKeeper saves all configuration and state data to disk. we'll build a data directory for ZooKeeper to read and write data in this phase. The data directory can be created on either the local filesystem or a remote storage disk. In this guide, the data directory will be created on my local filesystem.

Create a directory for ZooKeeper to use:

```
sudo mkdir -p /data/zookeeper
```

Grant the **zookeeper** user ownership to the directory:

```
sudo chown zookeeper:zookeeper /data/zookeeper
```

Step 4 — Downloading and Extracting the ZooKeeper Binaries

Here, we'll manually download and extract the ZooKeeper binaries to the 'opt' directory in this step. we'll install ZooKeeper manually because it will give me complete control over the version I use.

Use the `wget` command to get zookeeper binaries from [Zookeeper](https://dlcdn.apache.org/zookeeper/zookeeper-3.8.0/apache-zookeeper-3.8.0-bin.tar.gz) download page.

```
sudo wget https://dlcdn.apache.org/zookeeper/zookeeper-3.8.0/apache-zookeeper-3.8.0-bin.tar.gz
```

Extract the binaries from the compressed archive:

```
sudo tar -xvf apache-zookeeper-3.8.0-bin.tar.gz
```

Next, let's give the **zookeeper** user ownership of the extracted binaries so that it can run the

executables.

```
sudo chown zookeeper:zookeeper -R apache-zookeeper-3.8.0-bin
```

Step 5 — Configuring ZooKeeper

We're ready to configure ZooKeeper now that you've set up our environment.

The configuration file will be located in the `/opt/apache-zookeeper-3.8.0-bin/conf` folder. The ZooKeeper distribution includes a sample configuration file in this directory. The `zoo_sample.cfg` sample file contains the most commonly used configuration parameter definitions as well as sample values for these parameters. The following are some of the most popular parameters:-

- `tickTime` : Sets the length of a tick in milliseconds. A tick is a time unit used by ZooKeeper to measure the length between heartbeats. Minimum session timeouts are twice the `tickTime`.
- `dataDir` : Specifies the directory used to store snapshots of the in-memory database and the transaction log for updates. You could choose to specify a separate directory for transaction logs.
- `clientPort` : The port used to listen for client connections.
- `maxClientCnxns` : Limits the maximum number of client connections.

we'll create a configuration file named `zoo.cfg` at `apache-zookeeper-3.8.0-bin/conf`.

```
nano /opt/apache-zookeeper-3.8.0-bin/conf/zoo.cfg
```

We'll add the following properties and values to that file

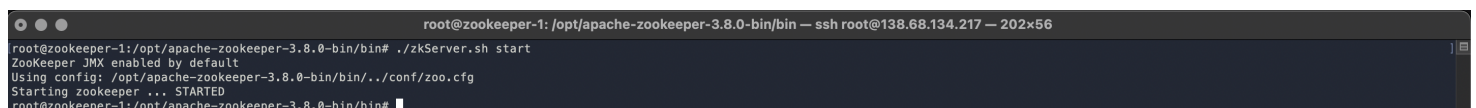
Step 6 — Starting ZooKeeper and Testing the Standalone Installation

Let's navigate back to the `/opt/apache-zookeeper-3.8.0-bin` directory.

```
cd /opt/apache-zookeeper-3.8.0-bin
```

Start ZooKeeper with the `zkServer.sh` command.

```
bin/zkServer.sh start
```



```
root@zookeeper-1: /opt/apache-zookeeper-3.8.0-bin/bin — ssh root@138.68.134.217 — 202x56
root@zookeeper-1:/opt/apache-zookeeper-3.8.0-bin/bin# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /opt/apache-zookeeper-3.8.0-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
root@zookeeper-1:/opt/apache-zookeeper-3.8.0-bin/bin#
```

Now we'll connect to the local ZooKeeper server with the following command:

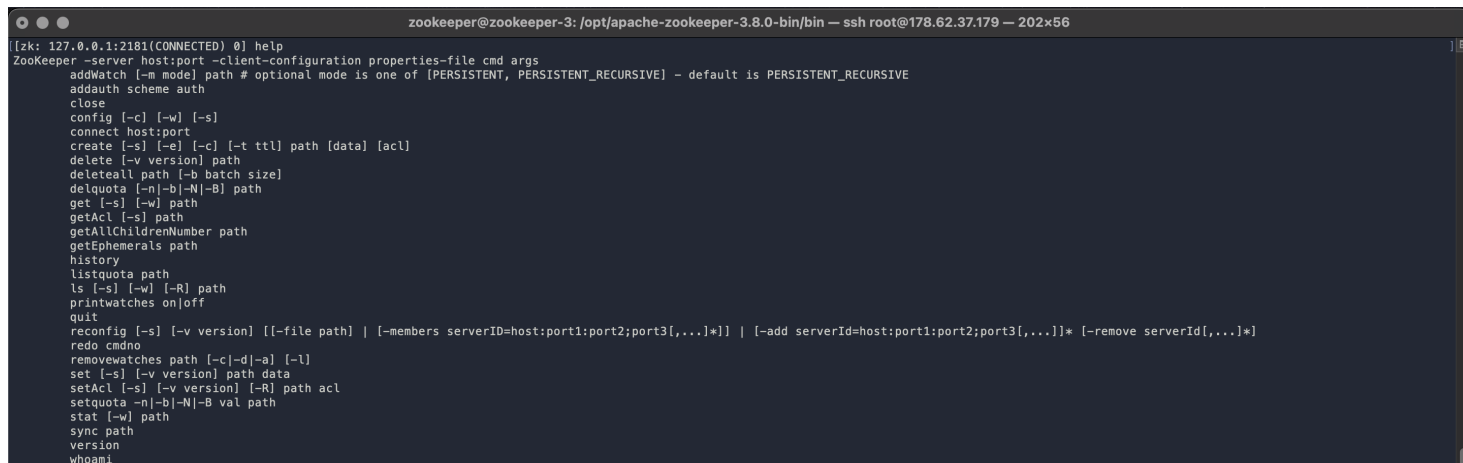
```
bin/zkCli.sh -server 127.0.0.1:2181
```

A prompt with the label 'CONNECTED' will appear. This confirms that we have a working ZooKeeper installation on our local machine. If you get any issues, double-check that your configuration is right.



```
WatchedEvent state:SyncConnected type:None path:null
[zk: 127.0.0.1:2181(CONNECTED) 0]
```

Typing `help` on this prompt will provide a list of commands that we can execute from the client.



Typing `quit` closes the client session and `bin/zkServer.sh stop` stops the ZooKeeper service. We could at this point also create a `Systemd` file to start and top ZooKeeper as a system service.

While the standalone ZooKeeper server is useful for development and testing, every production environment should have a replicated multi-node cluster.

Configuring a Multi-Node ZooKeeper Cluster

Step 7 - Configuring a Multi-Node ZooKeeper Cluster

Nodes in the ZooKeeper cluster that work together as an application form a *quorum*. Quorum refers to the minimum number of nodes that need to agree on a transaction before it's committed. A quorum needs an odd number of nodes so that it can establish a majority. An even number of nodes may result in a tie, which would mean the nodes would not reach a majority or consensus.

In a production environment, we should run each ZooKeeper node on a separate host. This prevents service disruption due to host hardware failure or reboots. This is an important and necessary architectural consideration for building a resilient and highly available distributed system.

In this part of the guide, I will install and configure two extra nodes in the quorum to demonstrate a multi-node setup.

I would set up two nodes[**zookeeper_2** and **zookeeper_3**] as i had already done earlier.

All nodes in a quorum will need the same configuration file. In the `zoo.cfg` file on each of the three nodes, add the additional configuration parameters and values for `initLimit` , `syncLimit` , and the

servers in the quorum, at the end of the file.

```
zookeeper@zookeeper-3: /opt/apache-zookeeper-3.8.0-bin/bin — ssh root@178.62.37.179 — 202x54
GNU nano 4.8 /opt/apache-zookeeper-3.8.0-bin/conf/zoo.cfg
tickTime=2000
dataDir=/data/zookeeper
clientPort=2181
maxClientCnxns=60
initLimit=10
syncLimit=5
server.1=138.68.134.217:2888:3888
server.2=209.97.176.31:2888:3888
server.3=0.0.0.0:2888:3888
```

- `initLimit` specifies the time that the initial synchronization phase can take. This is the time within which each of the nodes in the quorum needs to connect to the leader.
 - `syncLimit` specifies the time that can pass between sending a request and receiving an acknowledgment. This is the maximum time nodes can be out of sync from the leader.
- ZooKeeper nodes use a pair of ports, `:2888` and `:3888`, for follower nodes to connect to the leader node and for leader election, respectively.

To complete my multi-node configuration, we'll specify a node ID on each of the servers by creating a `myid` file on each node at the `/data/zookeeper` location. This file will contain a number that correlates to the server number assigned in the configuration file.

Step 8 — Running and Testing the Multi-Node Installation

With each node configured to work as a cluster, we are ready to start a quorum. In this step, we'll start the quorum on each node and then test my cluster by creating sample data in ZooKeeper.

To start a quorum node, we'll first change to the `/opt/apache-zookeeper-3.8.0-bin` directory on each node:

Start each node with the following command:

```
./zkCli.sh -server 127.0.0.1:2181
```

Then on my **zookeeper_3** node, we'll go to the `/opt/zookeeper` directory

```
bin/zkCli.sh -server zookeeper_1_ip:2181
```

Testing that I can successfully create, list, and then delete a znode is essential to establishing that your ZooKeeper cluster is installed and configured correctly.

Create a znode named `ds_ut_1` and associate the string `sample_data` with it:

```
[zkshell]: create /ds_ut_1 sample_data
```

```
zookeeper@zookeeper-3: /opt/apache-zookeeper-3.8.0-bin/bin — ssh root@178.62.37.179 — 202x56
[zk: 138.68.134.217:2181(CONNECTED) 0] create /ds_ut_1 sample_data
Created /ds_ut_1
[zk: 138.68.134.217:2181(CONNECTED) 1] █
```

Get the data associated with it:

```
get -s /ds_ut_1
```

A terminal window titled 'zookeeper@zookeeper-3: /opt/apache-zookeeper-3.8.0-bin/bin -- ssh root@178.62.37.179 -- 202x56'. The terminal shows the following commands and output:

```
[zk: 138.68.134.217:2181(CONNECTED) 0] create /ds_ut_1 sample_data
Created /ds_ut_1
[zk: 138.68.134.217:2181(CONNECTED) 1] get -s /ds_ut_1
sample_data
cZxid = 0x400000006
ctime = Fri Apr 01 19:39:26 UTC 2022
mZxid = 0x400000006
mtime = Fri Apr 01 19:39:26 UTC 2022
pZxid = 0x400000006
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 11
numChildren = 0
[zk: 138.68.134.217:2181(CONNECTED) 2]
```

The output confirms the value, `sample_data`, that you associated with `ds_ut_1`. ZooKeeper also provides additional information about creation time, `ctime`, and modification time, `mtime`. ZooKeeper is a versioned data store, so it also presents us with metadata about the data version. These data is replicated amongst the clusters. If we for instance check the each of the individual clusters, we would see the same data replicated there .

Delete the `ds_ut_1` znode:

```
delete /ds_ut_1
```

We can also check and see which of these nodes is the **leader** and which are **followers**.

```
./zkServer.sh status
```

If we stop the **leader** an election occurs and a new leader is immediately selected from the remaining nodes

```
./zkServer.sh status
```

References:

- https://zookeeper.apache.org/doc/r3.1.2/zookeeperStarted.html#sc_RunningReplicatedZooKeeper
- <https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-an-apache-zookeeper-cluster-on-ubuntu-18-04>