ZooKeeper Getting Started Guide

by

Table of contents

1. Getting Started: Coordinating Distributed Applications with ZooKeeper

This document contains information to get you started quickly with ZooKeeper. It is aimed primarily at developers hoping to try it out, and contains simple installation instructions for a single ZooKeeper server, a few commands to verify that it is running, and a simple programming example. Finally, as a convenience, there are a few sections regarding more complicated installations, for example running replicated deployments, and optimizing the transaction log. However for the complete instructions for commercial deployments, please refer to the ZooKeeper Administrator's Guide.

1.1. Pre-requisites

See System Requirements in the Admin guide.

1.2. Download

To get a ZooKeeper distribution, download a recent stable release from one of the Apache Download Mirrors.

1.3. Standalone Operation

Setting up a ZooKeeper server in standalone mode is straightforward. The server is contained in a single JAR file, so installation consists of creating a configuration.

Once you've downloaded a stable ZooKeeper release unpack it and cd to the root

To start ZooKeeper you need a configuration file. Here is a sample, create it in **conf/zoo.cfg**:

```
tickTime=2000
dataDir=/var/zookeeper
clientPort=2181
```

This file can be called anything, but for the sake of this discussion call it **conf/zoo.cfg**. Change the value of **dataDir** to specify an existing (empty to start with) directory. Here are the meanings for each of the fields:

tickTime

the basic time unit in milliseconds used by ZooKeeper. It is used to do heartbeats and the minimum session timeout will be twice the tickTime.

dataDir

the location to store the in-memory database snapshots and, unless specified otherwise, the transaction log of updates to the database.

clientPort

the port to listen for client connections

Now that you created the configuration file, you can start ZooKeeper:

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

ZooKeeper logs messages using log4j -- more detail available in the Logging section of the Programmer's Guide. You will see log messages coming to the console (default) and/or a log file depending on the log4j configuration.

The steps outlined here run ZooKeeper in standalone mode. There is no replication, so if ZooKeeper process fails, the service will go down. This is fine for most development situations, but to run ZooKeeper in replicated mode, please see Running Replicated ZooKeeper.

1.4. Managing ZooKeeper Storage

For long running production systems ZooKeeper storage must be managed externally (dataDir and logs). See the section on maintenance for more details.

1.5. Connecting to ZooKeeper

Once ZooKeeper is running, you have several options for connection to it:

- **Java**: Use

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

  This lets you perform simple, file-like operations.

- **C**: compile cli_mt (multi-threaded) or cli_st (single-threaded) by running `make cli_mt` or `make cli_st` in the **src/c** subdirectory in the ZooKeeper sources. See the README contained within **src/c** for full details.

  You can run the program from **src/c** using:

  ```
  LD_LIBRARY_PATH=. cli_mt 127.0.0.1:2181
  ```

  or

  ```
  LD_LIBRARY_PATH=. cli_st 127.0.0.1:2181
  ```

  This will give you a simple shell to execute file system like operations on ZooKeeper.

Once you have connected, you should see something like:

```
Connecting to localhost:2181
log4j:WARN No appenders could be found for logger
```

```
(org.apache.zookeeper.ZooKeeper).
log4j:WARN Please initialize the log4j system properly.
Welcome to ZooKeeper!
JLine support is enabled
[zkshell: 0]
```

From the shell, type `help` to get a listing of commands that can be executed from the client, as in:

```
[zkshell: 0] help
ZooKeeper host:port cmd args
        get path [watch]
        ls path [watch]
        set path data [version]
        delquota [-n|-b] path
        quit
        printwatches on|off
        createpath data acl
        stat path [watch]
        listquota path
        history
        setAcl path acl
        getAcl path
        sync path
        redo cmdno
        addauth scheme auth
        delete path [version]
        setquota -n|-b val path
```

From here, you can try a few simple commands to get a feel for this simple command line interface. First, start by issuing the list command, as in `ls`, yielding:

```
[zkshell: 8] ls /
[zookeeper]
```

Next, create a new znode by running `create /zk_test my_data`. This creates a new znode and associates the string "my_data" with the node. You should see:

```
[zkshell: 9] create /zk_test my_data
Created /zk_test
```

Issue another `ls /` command to see what the directory looks like:

```
```

```
[zkshell: 11] ls /
[zookeeper, zk_test]
```

Notice that the zk_test directory has now been created.

Next, verify that the data was associated with the znode by running the `get` command, as in:

```
[zkshell: 12] get /zk_test
my_data
cZxid = 5
ctime = Fri Jun 05 13:57:06 PDT 2009
mZxid = 5
mtime = Fri Jun 05 13:57:06 PDT 2009
pZxid = 5
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0
dataLength = 7
numChildren = 0
```

We can change the data associated with zk_test by issuing the `set` command, as in:

```
[zkshell: 14] set /zk_test junk
cZxid = 5
ctime = Fri Jun 05 13:57:06 PDT 2009
mZxid = 6
mtime = Fri Jun 05 14:01:52 PDT 2009
pZxid = 5
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0
dataLength = 4
numChildren = 0
[zkshell: 15] get /zk_test
junk
cZxid = 5
ctime = Fri Jun 05 13:57:06 PDT 2009
mZxid = 6
mtime = Fri Jun 05 14:01:52 PDT 2009
pZxid = 5
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0
dataLength = 4
numChildren = 0
```

(Notice we did a `get` after setting the data and it did, indeed, change.

Finally, let's `delete` the node by issuing:

```
[zkshell: 16] delete /zk_test
[zkshell: 17] ls /
[zookeeper]
[zkshell: 18]
```

That's it for now. To explore more, continue with the rest of this document and see the Programmer's Guide.

1.6. Programming to ZooKeeper

ZooKeeper has a Java bindings and C bindings. They are functionally equivalent. The C bindings exist in two variants: single threaded and multi-threaded. These differ only in how the messaging loop is done. For more information, see the Programming Examples in the ZooKeeper Programmer's Guide for sample code using of the different APIs.

1.7. Running Replicated ZooKeeper

Running ZooKeeper in standalone mode is convenient for evaluation, some development, and testing. But in production, you should run ZooKeeper in replicated mode. A replicated group of servers in the same application is called a *quorum*, and in replicated mode, all servers in the quorum have copies of the same configuration file. The file is similar to the one used in standalone mode, but with a few differences. Here is an example:

```
tickTime=2000
dataDir=/var/zookeeper
clientPort=2181
initLimit=5
syncLimit=2
server.1=zoo1:2888:3888
server.2=zoo2:2888:3888
server.3=zoo3:2888:3888
```

The new entry, **initLimit** is timeouts ZooKeeper uses to limit the length of time the ZooKeeper servers in quorum have to connect to a leader. The entry **syncLimit** limits how far out of date a server can be from a leader.

With both of these timeouts, you specify the unit of time using **tickTime**. In this example, the timeout for initLimit is 5 ticks at 2000 milleseconds a tick, or 10 seconds.

The entries of the form *server.X* list the servers that make up the ZooKeeper service. When the server starts up, it knows which server it is by looking for the file *myid* in the data

directory. That file has the contains the server number, in ASCII.

Finally, note the two port numbers after each server name: " 2888" and "3888". Peers use the former port to connect to other peers. Such a connection is necessary so that peers can communicate, for example, to agree upon the order of updates. More specifically, a ZooKeeper server uses this port to connect followers to the leader. When a new leader arises, a follower opens a TCP connection to the leader using this port. Because the default leader election also uses TCP, we currently require another port for leader election. This is the second port in the server entry.

Note:

If you want to test multiple servers on a single machine, specify the servername as *localhost* with unique quorum & leader election ports (i.e. 2888:3888, 2889:3889, 2890:3890 in the example above) for each server.X in that server's config file. Of course separate *dataDir*s and distinct *clientPort*s are also necessary (in the above replicated example, running on a single *localhost*, you would still have three config files).

1.8. Other Optimizations

There are a couple of other configuration parameters that can greatly increase performance:

- To get low latencies on updates it is important to have a dedicated transaction log directory. By default transaction logs are put in the same directory as the data snapshots and *myid* file. The dataLogDir parameters indicates a different directory to use for the transaction logs.
- *[tbd: what is the other config param?]*