**ZooKeeper Administrator's Guide**

**A Guide to Deployment and Administration**

**by**

# Table of contents

1. [Deployment](#_bookmark0) 2
   1. [System Requirements](#_bookmark1) 2
   2. [Clustered (Multi-Server) Setup](#_bookmark2) 3
   3. [Single Server and Developer Setup](#_bookmark3) 5
2. [Administration](#_bookmark4) 5
   1. [Designing a ZooKeeper Deployment](#_bookmark5) 5
   2. [Provisioning](#_bookmark6) 6
   3. [Things to Consider: ZooKeeper Strengths and Limitations](#_bookmark7) 6
   4. [Administering](#_bookmark8) 7
   5. [Maintenance](#_bookmark9) 7
   6. [Supervision](#_bookmark10) 8
   7. [Monitoring](#_bookmark11) 8
   8. [Logging](#_bookmark12) 8
   9. [Troubleshooting](#_bookmark13) 8
   10. [Configuration Parameters](#_bookmark14) 8
   11. [ZooKeeper Commands: The Four Letter Words](#_bookmark16) 17
   12. [Data File Management](#_bookmark17) 19
   13. [Things to Avoid](#_bookmark18) 20
   14. [Best Practices](#_bookmark19) 21

# Deployment

This section contains information about deploying Zookeeper and covers these topics:

* [System Requirements](#_bookmark1)
* [Clustered (Multi-Server) Setup](#_bookmark2)
* [Single Server and Developer Setup](#_bookmark3)

The first two sections assume you are interested in installing ZooKeeper in a production environment such as a datacenter. The final section covers situations in which you are setting up ZooKeeper on a limited basis - for evaluation, testing, or development - but not in a production environment.

* 1. **System Requirements**
     1. **Supported Platforms**

ZooKeeper consists of multiple components. Some components are supported broadly, and other components are supported only on a smaller set of platforms.

* **Client** is the Java client library, used by applications to connect to a ZooKeeper ensemble.
* **Server** is the Java server that runs on the ZooKeeper ensemble nodes.
* **Native Client** is a client implemented in C, similar to the Java client, used by applications to connect to a ZooKeeper ensemble.
* **Contrib** refers to multiple optional add-on components.

The following matrix describes the level of support committed for running each component on different operating system platforms.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operating System Client Server Native Client Contrib | | | | |
| GNU/Linux | Development and Production | Development and Production | Development and Production | Development and Production |
| Solaris | Development and Production | Development and Production | Not Supported | Not Supported |
| FreeBSD | Development and Production | Development and Production | Not Supported | Not Supported |
| Windows | Development and Production | Development and Production | Not Supported | Not Supported |
| Mac OS X | Development Only | Development Only | Not Supported | Not Supported |

Table 1: Support Matrix

For any operating system not explicitly mentioned as supported in the matrix, components may or may not work. The ZooKeeper community will fix obvious bugs that are reported for other platforms, but there is no full support.

* + 1. **Required Software**

ZooKeeper runs in Java, release 1.6 or greater (JDK 6 or greater). It runs as an *ensemble* of ZooKeeper servers. Three ZooKeeper servers is the minimum recommended size for an ensemble, and we also recommend that they run on separate machines. At Yahoo!,

ZooKeeper is usually deployed on dedicated RHEL boxes, with dual-core processors, 2GB of RAM, and 80GB IDE hard drives.

* 1. **Clustered (Multi-Server) Setup**

For reliable ZooKeeper service, you should deploy ZooKeeper in a cluster known as an

*ensemble*. As long as a majority of the ensemble are up, the service will be available. Because Zookeeper requires a majority, it is best to use an odd number of machines. For example, with four machines ZooKeeper can only handle the failure of a single machine; if two machines fail, the remaining two machines do not constitute a majority. However, with five machines ZooKeeper can handle the failure of two machines.

Note:

As mentioned in the ZooKeeper Getting Started Guide , a minimum of three servers are required for a fault tolerant clustered setup, and it is strongly recommended that you have an odd number of servers.

Usually three servers is more than enough for a production install, but for maximum reliability during maintenance, you may wish to install five servers. With three servers, if you perform maintenance on one of them, you are vulnerable to a failure on one of the other two servers during that maintenance. If you have five of them running, you can take one down for maintenance, and know that you're still OK if one of the other four suddenly fails.

Your redundancy considerations should include all aspects of your environment. If you have three ZooKeeper servers, but their network cables are all plugged into the same network switch, then the failure of that switch will take down your entire ensemble.

Here are the steps to setting a server that will be part of an ensemble. These steps should be performed on every host in the ensemble:

1. Install the Java JDK. You can use the native packaging system for your system, or download the JDK from:

<http://java.sun.com/javase/downloads/index.jsp>

1. Set the Java heap size. This is very important to avoid swapping, which will seriously degrade ZooKeeper performance. To determine the correct value, use load tests,

and make sure you are well below the usage limit that would cause you to swap. Be conservative - use a maximum heap size of 3GB for a 4GB machine.

1. Install the ZooKeeper Server Package. It can be downloaded from:

<http://zookeeper.apache.org/releases.html>

1. Create a configuration file. This file can be called anything. Use the following settings as a starting point:

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

initLimit=5 syncLimit=2

server.1=zoo1:2888:3888 server.2=zoo2:2888:3888 server.3=zoo3:2888:3888

You can find the meanings of these and other configuration settings in the section [Configuration Parameters](#_bookmark14). A word though about a few here:

Every machine that is part of the ZooKeeper ensemble should know about every other machine in the ensemble. You accomplish this with the series of lines of the form **server.id=host:port:port**. The parameters **host** and **port** are straightforward. You attribute the server id to each machine by creating a file named myid, one for each server, which resides in that server's data directory, as specified by the configuration file parameter **dataDir**.

1. The myid file consists of a single line containing only the text of that machine's id. So myid of server 1 would contain the text "1" and nothing else. The id must be unique within the ensemble and should have a value between 1 and 255.
2. If your configuration file is set up, you can start a ZooKeeper server:

$ java -cp zookeeper.jar:lib/slf4j-api-1.6.1.jar:lib/ slf4j-log4j12-1.6.1.jar:lib/log4j-1.2.15.jar:conf \ org.apache.zookeeper.server.quorum.QuorumPeerMain zoo.cfg

QuorumPeerMain starts a ZooKeeper server, [JMX](http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/) management beans are also registered which allows management through a JMX management console. The ZooKeeper JMX document contains details on managing ZooKeeper with JMX.

See the script *bin/zkServer.sh*, which is included in the release, for an example of starting server instances.

1. Test your deployment by connecting to the hosts:

In Java, you can run the following command to execute simple operations:

$ bin/zkCli.sh -server 127.0.0.1:2181

* 1. **Single Server and Developer Setup**

If you want to setup ZooKeeper for development purposes, you will probably want to setup a single server instance of ZooKeeper, and then install either the Java or C client-side libraries and bindings on your development machine.

The steps to setting up a single server instance are the similar to the above, except the configuration file is simpler. You can find the complete instructions in the Installing and Running ZooKeeper in Single Server Mode section of the ZooKeeper Getting Started Guide.

For information on installing the client side libraries, refer to the Bindings section of the ZooKeeper Programmer's Guide.

# Administration

This section contains information about running and maintaining ZooKeeper and covers these topics:

* [Designing a ZooKeeper Deployment](#_bookmark5)
* [Provisioning](#_bookmark6)
* [Things to Consider: ZooKeeper Strengths and Limitations](#_bookmark7)
* [Administering](#_bookmark8)
* [Maintenance](#_bookmark9)
* [Supervision](#_bookmark10)
* [Monitoring](#_bookmark11)
* [Logging](#_bookmark12)
* [Troubleshooting](#_bookmark13)
* [Configuration Parameters](#_bookmark14)
* [ZooKeeper Commands: The Four Letter Words](#_bookmark16)
* [Data File Management](#_bookmark17)
* [Things to Avoid](#_bookmark18)
* [Best Practices](#_bookmark19)

**2.1 Designing a ZooKeeper Deployment**

The reliablity of ZooKeeper rests on two basic assumptions.

1. Only a minority of servers in a deployment will fail. *Failure* in this context means a machine crash, or some error in the network that partitions a server off from the majority.
2. Deployed machines operate correctly. To operate correctly means to execute code correctly, to have clocks that work properly, and to have storage and network components that perform consistently.

The sections below contain considerations for ZooKeeper administrators to maximize the probability for these assumptions to hold true. Some of these are cross-machines

considerations, and others are things you should consider for each and every machine in your deployment.

* + 1. **Cross Machine Requirements**

For the ZooKeeper service to be active, there must be a majority of non-failing machines that can communicate with each other. To create a deployment that can tolerate the failure of F machines, you should count on deploying 2xF+1 machines. Thus, a deployment that consists of three machines can handle one failure, and a deployment of five machines can handle two failures. Note that a deployment of six machines can only handle two failures since three machines is not a majority. For this reason, ZooKeeper deployments are usually made up of an odd number of machines.

To achieve the highest probability of tolerating a failure you should try to make machine failures independent. For example, if most of the machines share the same switch, failure of that switch could cause a correlated failure and bring down the service. The same holds true of shared power circuits, cooling systems, etc.

* + 1. **Single Machine Requirements**

If ZooKeeper has to contend with other applications for access to resourses like storage media, CPU, network, or memory, its performance will suffer markedly. ZooKeeper has strong durability guarantees, which means it uses storage media to log changes before the operation responsible for the change is allowed to complete. You should be aware of this dependency then, and take great care if you want to ensure that ZooKeeper operations aren’t held up by your media. Here are some things you can do to minimize that sort of degradation:

* ZooKeeper's transaction log must be on a dedicated device. (A dedicated partition is not enough.) ZooKeeper writes the log sequentially, without seeking Sharing your log device with other processes can cause seeks and contention, which in turn can cause multi- second delays.
* Do not put ZooKeeper in a situation that can cause a swap. In order for ZooKeeper to function with any sort of timeliness, it simply cannot be allowed to swap. Therefore, make certain that the maximum heap size given to ZooKeeper is not bigger than the amount of real memory available to ZooKeeper. For more on this, see [Things to Avoid](#_bookmark18) below.
  1. **Provisioning**
  2. **Things to Consider: ZooKeeper Strengths and Limitations**
  3. **Administering**
  4. **Maintenance**

Little long term maintenance is required for a ZooKeeper cluster however you must be aware of the following:

* + 1. **Ongoing Data Directory Cleanup**

The ZooKeeper Data Directory contains files which are a persistent copy of the znodes stored by a particular serving ensemble. These are the snapshot and transactional log files. As changes are made to the znodes these changes are appended to a transaction log, occasionally, when a log grows large, a snapshot of the current state of all znodes will be written to the filesystem. This snapshot supercedes all previous logs.

A ZooKeeper server **will not remove old snapshots and log files** when using the default configuration (see autopurge below), this is the responsibility of the operator. Every serving environment is different and therefore the requirements of managing these files may differ from install to install (backup for example).

The PurgeTxnLog utility implements a simple retention policy that administrators can use. The API docs contains details on calling conventions (arguments, etc...).

In the following example the last count snapshots and their corresponding logs are retained and the others are deleted. The value of <count> should typically be greater than 3 (although not required, this provides 3 backups in the unlikely event a recent log has become corrupted). This can be run as a cron job on the ZooKeeper server machines to clean up the logs daily.

java -cp zookeeper.jar:lib/slf4j-api-1.6.1.jar:lib/slf4j-log4j12-1.6.1.jar:lib/ log4j-1.2.15.jar:conf org.apache.zookeeper.server.PurgeTxnLog <dataDir> <snapDir> -n

<count>

Automatic purging of the snapshots and corresponding transaction logs was introduced in version 3.4.0 and can be enabled via the following configuration parameters **autopurge.snapRetainCount** and **autopurge.purgeInterval**. For more on this, see [Advanced Configuration](#_bookmark15) below.

* + 1. **Debug Log Cleanup (log4j)**

See the section on [logging](#_bookmark12) in this document. It is expected that you will setup a rolling file appender using the in-built log4j feature. The sample configuration file in the release tar's conf/log4j.properties provides an example of this.

* 1. **Supervision**

You will want to have a supervisory process that manages each of your ZooKeeper server processes (JVM). The ZK server is designed to be "fail fast" meaning that it will shutdown (process exit) if an error occurs that it cannot recover from. As a ZooKeeper serving cluster is highly reliable, this means that while the server may go down the cluster as a whole is still active and serving requests. Additionally, as the cluster is "self healing" the failed server once restarted will automatically rejoin the ensemble w/o any manual interaction.

Having a supervisory process such as [daemontools](http://cr.yp.to/daemontools.html) or [SMF](http://en.wikipedia.org/wiki/Service_Management_Facility) (other options for supervisory process are also available, it's up to you which one you would like to use, these are just two examples) managing your ZooKeeper server ensures that if the process does exit abnormally it will automatically be restarted and will quickly rejoin the cluster.

* 1. **Monitoring**

The ZooKeeper service can be monitored in one of two primary ways; 1) the command port through the use of [4 letter words](#_bookmark16) and 2) JMX. See the appropriate section for your environment/requirements.

* 1. **Logging**

ZooKeeper uses **log4j** version 1.2 as its logging infrastructure. The ZooKeeper default log4j.properties file resides in the conf directory. Log4j requires that log4j.properties either be in the working directory (the directory from which ZooKeeper is run) or be accessible from the classpath.

For more information, see [Log4j Default Initialization Procedure](http://logging.apache.org/log4j/1.2/manual.html#defaultInit) of the log4j manual.

* 1. **Troubleshooting**

# Server not coming up because of file corruption

A server might not be able to read its database and fail to come up because of some file corruption in the transaction logs of the ZooKeeper server. You will see some IOException on loading ZooKeeper database. In such a case, make sure all the other

servers in your ensemble are up and working. Use "stat" command on the command port to see if they are in good health. After you have verified that all the other servers of the ensemble are up, you can go ahead and clean the database of the corrupt server. Delete all the files in datadir/version-2 and datalogdir/version-2/. Restart the server.

* 1. **Configuration Parameters**

ZooKeeper's behavior is governed by the ZooKeeper configuration file. This file is designed so that the exact same file can be used by all the servers that make up a ZooKeeper server

assuming the disk layouts are the same. If servers use different configuration files, care must be taken to ensure that the list of servers in all of the different configuration files match.

* + 1. **Minimum Configuration**

Here are the minimum configuration keywords that must be defined in the configuration file:

# clientPort

the port to listen for client connections; that is, the port that clients attempt to connect to.

# dataDir

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

Note:

Be careful where you put the transaction log. A dedicated transaction log device is key to consistent good performance. Putting the log on a busy device will adversely effect performance.

# tickTime

the length of a single tick, which is the basic time unit used by ZooKeeper, as measured in milliseconds. It is used to regulate heartbeats, and timeouts. For example, the minimum session timeout will be two ticks.

* + 1. **Advanced Configuration**

The configuration settings in the section are optional. You can use them to further fine tune the behaviour of your ZooKeeper servers. Some can also be set using Java system properties, generally of the form *zookeeper.keyword*. The exact system property, when available, is noted below.

# dataLogDir

(No Java system property)

This option will direct the machine to write the transaction log to the **dataLogDir** rather than the **dataDir**. This allows a dedicated log device to be used, and helps avoid competition between logging and snaphots.

Note:

Having a dedicated log device has a large impact on throughput and stable latencies. It is highly recommened to dedicate a log device and set **dataLogDir** to point to a directory on that device, and then make sure to point **dataDir** to a directory *not* residing on that device.

# globalOutstandingLimit

(Java system property: **zookeeper.globalOutstandingLimit.**)

Clients can submit requests faster than ZooKeeper can process them, especially if there are a lot of clients. To prevent ZooKeeper from running out of memory due to queued requests, ZooKeeper will throttle clients so that there is no more than

globalOutstandingLimit outstanding requests in the system. The default limit is 1,000.

# preAllocSize

(Java system property: **zookeeper.preAllocSize**)

To avoid seeks ZooKeeper allocates space in the transaction log file in blocks of preAllocSize kilobytes. The default block size is 64M. One reason for changing the size of the blocks is to reduce the block size if snapshots are taken more often. (Also, see **snapCount**).

# snapCount

(Java system property: **zookeeper.snapCount**)

ZooKeeper records its transactions using snapshots and a transaction log (think write- ahead log).The number of transactions recorded in the transaction log before a snapshot can be taken (and the transaction log rolled) is determined by snapCount. In order to prevent all of the machines in the quorum from taking a snapshot at the same time, each ZooKeeper server will take a snapshot when the number of transactions in the transaction log reaches a runtime generated random value in the [snapCount/2+1, snapCount] range.The default snapCount is 100,000.

# maxClientCnxns

(No Java system property)

Limits the number of concurrent connections (at the socket level) that a single client, identified by IP address, may make to a single member of the ZooKeeper ensemble. This is used to prevent certain classes of DoS attacks, including file descriptor exhaustion. The default is 60. Setting this to 0 entirely removes the limit on concurrent connections. **clientPortAddress**

**New in 3.3.0:** the address (ipv4, ipv6 or hostname) to listen for client connections; that is, the address that clients attempt to connect to. This is optional, by default we bind in such a way that any connection to the **clientPort** for any address/interface/nic on the server will be accepted.

# minSessionTimeout

(No Java system property)

**New in 3.3.0:** the minimum session timeout in milliseconds that the server will allow the client to negotiate. Defaults to 2 times the **tickTime**.

# maxSessionTimeout

(No Java system property)

**New in 3.3.0:** the maximum session timeout in milliseconds that the server will allow the client to negotiate. Defaults to 20 times the **tickTime**.

# fsync.warningthresholdms

(Java system property: **zookeeper.fsync.warningthresholdms**)

**New in 3.3.4:** A warning message will be output to the log whenever an fsync in the Transactional Log (WAL) takes longer than this value. The values is specified in milliseconds and defaults to 1000. This value can only be set as a system property. **autopurge.snapRetainCount**

(No Java system property)

**New in 3.4.0:** When enabled, ZooKeeper auto purge feature retains the **autopurge.snapRetainCount** most recent snapshots and the corresponding transaction logs in the **dataDir** and **dataLogDir** respectively and deletes the rest. Defaults to 3.

Minimum value is 3. **autopurge.purgeInterval** (No Java system property)

**New in 3.4.0:** The time interval in hours for which the purge task has to be triggered. Set to a positive integer (1 and above) to enable the auto purging. Defaults to 0. **syncEnabled**

(Java system property: **zookeeper.observer.syncEnabled**)

**New in 3.4.6, 3.5.0:** The observers now log transaction and write snapshot to disk by default like the participants. This reduces the recovery time of the observers on restart. Set to "false" to disable this feature. Default is "true"

* + 1. **Cluster Options**

The options in this section are designed for use with an ensemble of servers -- that is, when deploying clusters of servers.

# electionAlg

(No Java system property)

Election implementation to use. A value of "0" corresponds to the original UDP-based version, "1" corresponds to the non-authenticated UDP-based version of fast leader election, "2" corresponds to the authenticated UDP-based version of fast leader election, and "3" corresponds to TCP-based version of fast leader election. Currently, algorithm 3 is the default

Note:

The implementations of leader election 0, 1, and 2 are now **deprecated** . We have the intention of removing them in the next release, at which point only the FastLeaderElection will be available.

# initLimit

(No Java system property)

Amount of time, in ticks (see tickTime), to allow followers to connect and sync to a leader. Increased this value as needed, if the amount of data managed by ZooKeeper is large.

# leaderServes

(Java system property: zookeeper.**leaderServes**)

Leader accepts client connections. Default value is "yes". The leader machine coordinates updates. For higher update throughput at thes slight expense of read throughput the leader can be configured to not accept clients and focus on coordination. The default to this option is yes, which means that a leader will accept client connections.

Note:

Turning on leader selection is highly recommended when you have more than three ZooKeeper servers in an ensemble.

# server.x=[hostname]:nnnnn[:nnnnn], etc

(No Java system property)

servers making up the ZooKeeper ensemble. When the server starts up, it determines which server it is by looking for the file myid in the data directory. That file contains the server number, in ASCII, and it should match **x** in **server.x** in the left hand side of this setting.

The list of servers that make up ZooKeeper servers that is used by the clients must match the list of ZooKeeper servers that each ZooKeeper server has.

There are two port numbers **nnnnn**. The first followers use to connect to the leader, and the second is for leader election. The leader election port is only necessary if electionAlg is 1, 2, or 3 (default). If electionAlg is 0, then the second port is not necessary. If you want to test multiple servers on a single machine, then different ports can be used for each server.

# syncLimit

(No Java system property)

Amount of time, in ticks (see tickTime), to allow followers to sync with ZooKeeper. If followers fall too far behind a leader, they will be dropped.

# group.x=nnnnn[:nnnnn]

(No Java system property)

Enables a hierarchical quorum construction."x" is a group identifier and the numbers following the "=" sign correspond to server identifiers. The left-hand side of the assignment is a colon-separated list of server identifiers. Note that groups must be disjoint and the union of all groups must be the ZooKeeper ensemble.

You will find an example here

# weight.x=nnnnn

(No Java system property)

Used along with "group", it assigns a weight to a server when forming quorums. Such a value corresponds to the weight of a server when voting. There are a few parts of

ZooKeeper that require voting such as leader election and the atomic broadcast protocol. By default the weight of server is 1. If the configuration defines groups, but not weights, then a value of 1 will be assigned to all servers.

You will find an example here

# cnxTimeout

(Java system property: zookeeper.**cnxTimeout**)

Sets the timeout value for opening connections for leader election notifications. Only applicable if you are using electionAlg 3.

Note:

Default value is 5 seconds.

# 4lw.commands.whitelist

(Java system property: **zookeeper.4lw.commands.whitelist**)

**New in 3.4.10:** This property contains a list of comma separated [Four Letter Words](#_bookmark16) commands. It is introduced to provide fine grained control over the set of commands ZooKeeper can execute, so users can turn off certain commands if necessary. By default it contains all supported four letter word commands except "wchp" and "wchc", if the property is not specified. If the property is specified, then only commands listed in the whitelist are enabled.

Here's an example of the configuration that enables stat, ruok, conf, and isro command while disabling the rest of Four Letter Words command:

4lw.commands.whitelist=stat, ruok, conf, isro

Users can also use asterisk option so they don't have to include every command one by one in the list. As an example, this will enable all four letter word commands:

4lw.commands.whitelist=\*

# ipReachableTimeout

(Java system property: **zookeeper.ipReachableTimeout**)

**New in 3.4.11:** Set this timeout value for IP addresses reachable checking when hostname is resolved, as mesured in milliseconds. By default, ZooKeeper will use the first IP address of the hostname(without any reachable checking). When zookeeper.ipReachableTimeout is set(larger than 0), ZooKeeper will will try to pick up the first IP address which is reachable. This is done by calling Java API

InetAddress.isReachable(long timeout) function, in which this timeout value is used. If none of such reachable IP address can be found, the first IP address of the hostname will be used anyway.

# tcpKeepAlive

(Java system property: **zookeeper.tcpKeepAlive**)

**New in 3.4.11:** Setting this to true sets the TCP keepAlive flag on the sockets used by quorum members to perform elections. This will allow for connections between quorum members to remain up when there is network infrastructure that may otherwise break them. Some NATs and firewalls may terminate or lose state for long running or idle connections.

Enabling this option relies on OS level settings to work properly, check your operating system's options regarding TCP keepalive for more information. Defaults to **false**.

* + 1. **Authentication & Authorization Options**

The options in this section allow control over authentication/authorization performed by the service.

# zookeeper.DigestAuthenticationProvider.superDigest

(Java system property only: **zookeeper.DigestAuthenticationProvider.superDigest**) By default this feature is **disabled**

**New in 3.2:** Enables a ZooKeeper ensemble administrator to access the znode hierarchy as a "super" user. In particular no ACL checking occurs for a user authenticated as super.

org.apache.zookeeper.server.auth.DigestAuthenticationProvider can be used to generate the superDigest, call it with one parameter of "super:<password>". Provide the generated "super:<data>" as the system property value when starting each server of the ensemble.

When authenticating to a ZooKeeper server (from a ZooKeeper client) pass a scheme of "digest" and authdata of "super:<password>". Note that digest auth passes the authdata in plaintext to the server, it would be prudent to use this authentication method only on localhost (not over the network) or over an encrypted connection.

# isro

**New in 3.4.0:** Tests if server is running in read-only mode. The server will respond with "ro" if in read-only mode or "rw" if not in read-only mode.

# gtmk

Gets the current trace mask as a 64-bit signed long value in decimal format. See stmk

for an explanation of the possible values.

# stmk

Sets the current trace mask. The trace mask is 64 bits, where each bit enables or disables a specific category of trace logging on the server. Log4J must be configured to enable TRACE level first in order to see trace logging messages. The bits of the trace mask correspond to the following trace logging categories.

|  |  |
| --- | --- |
| 0b0000000000 | Unused, reserved for future use. |
| 0b0000000010 | Logs client requests, excluding ping requests. |
| 0b0000000100 | Unused, reserved for future use. |
| 0b0000001000 | Logs client ping requests. |
| 0b0000010000 | Logs packets received from the quorum peer that is the current leader, excluding ping requests. |
| 0b0000100000 | Logs addition, removal and validation of client sessions. |
| 0b0001000000 | Logs delivery of watch events to client sessions. |
| 0b0010000000 | Logs ping packets received from the quorum peer that is the current leader. |
| 0b0100000000 | Unused, reserved for future use. |
| 0b1000000000 | Unused, reserved for future use. |

Table 1: Trace Mask Bit Values

All remaining bits in the 64-bit value are unused and reserved for future use. Multiple trace logging categories are specified by calculating the bitwise OR of the documented values. The default trace mask is 0b0100110010. Thus, by default, trace logging includes client requests, packets received from the leader and sessions.

To set a different trace mask, send a request containing the stmk four-letter word followed by the trace mask represented as a 64-bit signed long value. This example uses the Perl pack function to construct a trace mask that enables all trace logging categories described above and convert it to a 64-bit signed long value with big-endian byte order. The result is appended to stmk and sent to the server using netcat. The server responds with the new trace mask in decimal format.

$ perl -e "print 'stmk', pack('q>', 0b0011111010)" | nc localhost 2181 250

* + 1. **Experimental Options/Features**

New features that are currently considered experimental.

# Read Only Mode Server

(Java system property: **readonlymode.enabled**)

**New in 3.4.0:** Setting this value to true enables Read Only Mode server support (disabled by default). ROM allows clients sessions which requested ROM support to connect to the server even when the server might be partitioned from the quorum. In this mode ROM clients can still read values from the ZK service, but will be unable to write values and see changes from other clients. See ZOOKEEPER-784 for more details.

* + 1. **Unsafe Options**

The following options can be useful, but be careful when you use them. The risk of each is explained along with the explanation of what the variable does.

# forceSync

(Java system property: **zookeeper.forceSync**)

Requires updates to be synced to media of the transaction log before finishing processing the update. If this option is set to no, ZooKeeper will not require updates to be synced to the media.

# jute.maxbuffer:

(Java system property: **jute.maxbuffer**)

This option can only be set as a Java system property. There is no zookeeper prefix on it. It specifies the maximum size of the data that can be stored in a znode. The default is 0xfffff, or just under 1M. If this option is changed, the system property must be set on all servers and clients otherwise problems will arise. This is really a sanity check. ZooKeeper is designed to store data on the order of kilobytes in size.

# skipACL

(Java system property: **zookeeper.skipACL**)

Skips ACL checks. This results in a boost in throughput, but opens up full access to the data tree to everyone.

# quorumListenOnAllIPs

When set to true the ZooKeeper server will listen for connections from its peers on all available IP addresses, and not only the address configured in the server list of the configuration file. It affects the connections handling the ZAB protocol and the Fast Leader Election protocol. Default value is **false**.

* + 1. **Communication using the Netty framework**

**New in 3.4:** [Netty](http://jboss.org/netty) is an NIO based client/server communication framework, it simplifies (over NIO being used directly) many of the complexities of network level communication for java applications. Additionally the Netty framework has built in support for encryption (SSL) and authentication (certificates). These are optional features and can be turned on or off individually.

Prior to version 3.4 ZooKeeper has always used NIO directly, however in versions

3.4 and later Netty is supported as an option to NIO (replaces). NIO continues to be the default, however Netty based communication can be used in place of NIO by setting the environment variable "zookeeper.serverCnxnFactory" to

"org.apache.zookeeper.server.NettyServerCnxnFactory". You have the option of setting this on either the client(s) or server(s), typically you would want to set this on both, however that is at your discretion.

TBD - tuning options for netty - currently there are none that are netty specific but we should add some. Esp around max bound on the number of reader worker threads netty creates.

TBD - how to manage encryption TBD - how to manage certificates

* 1. **ZooKeeper Commands: The Four Letter Words**

ZooKeeper responds to a small set of commands. Each command is composed of four letters. You issue the commands to ZooKeeper via telnet or nc, at the client port.

Three of the more interesting commands: "stat" gives some general information about the server and connected clients, while "srvr" and "cons" give extended details on server and connections respectively.

# conf

**New in 3.3.0:** Print details about serving configuration.

# cons

**New in 3.3.0:** List full connection/session details for all clients connected to this server. Includes information on numbers of packets received/sent, session id, operation latencies, last operation performed, etc...

# crst

**New in 3.3.0:** Reset connection/session statistics for all connections.

# dump

Lists the outstanding sessions and ephemeral nodes. This only works on the leader.

# envi

Print details about serving environment

# ruok

Tests if server is running in a non-error state. The server will respond with imok if it is running. Otherwise it will not respond at all.

A response of "imok" does not necessarily indicate that the server has joined the quorum, just that the server process is active and bound to the specified client port. Use "stat" for details on state wrt quorum and client connection information.

# srst

Reset server statistics.

# srvr

**New in 3.3.0:** Lists full details for the server.

# stat

Lists brief details for the server and connected clients.

# wchs

**New in 3.3.0:** Lists brief information on watches for the server.

# wchc

**New in 3.3.0:** Lists detailed information on watches for the server, by session. This outputs a list of sessions(connections) with associated watches (paths). Note,

depending on the number of watches this operation may be expensive (ie impact server performance), use it carefully.

# wchp

**New in 3.3.0:** Lists detailed information on watches for the server, by path. This outputs a list of paths (znodes) with associated sessions. Note, depending on the number of watches this operation may be expensive (ie impact server performance), use it carefully. **mntr**

**New in 3.4.0:** Outputs a list of variables that could be used for monitoring the health of the cluster.

* only exposed by the Leader
* only exposed by the Leader
* only exposed by the Leader
* only available on Unix platforms
* only available on Unix platforms

zk\_open\_file\_descriptor\_count 23

zk\_max\_file\_descriptor\_count 1024

$ echo mntr | nc localhost 2185

zk\_version 3.4.0

zk\_avg\_latency 0

zk\_max\_latency 0

zk\_min\_latency 0

zk\_packets\_received 70

zk\_packets\_sent 69

zk\_outstanding\_requests 0 zk\_server\_state leader zk\_znode\_count 4

zk\_watch\_count 0

zk\_ephemerals\_count 0

zk\_approximate\_data\_size 27

zk\_followers 4

zk\_synced\_followers 4

zk\_pending\_syncs 0

The output is compatible with java properties format and the content may change over time (new keys added). Your scripts should expect changes.

ATTENTION: Some of the keys are platform specific and some of the keys are only exported by the Leader.

The output contains multiple lines with the following format:

key \t value

Here's an example of the **ruok** command:

$ echo ruok | nc 127.0.0.1 5111 imok

* 1. **Data File Management**

ZooKeeper stores its data in a data directory and its transaction log in a transaction log directory. By default these two directories are the same. The server can (and should) be configured to store the transaction log files in a separate directory than the data files.

Throughput increases and latency decreases when transaction logs reside on a dedicated log devices.

* + 1. **The Data Directory**

This directory has two files in it:

* myid - contains a single integer in human readable ASCII text that represents the server id.
* snapshot.<zxid> - holds the fuzzy snapshot of a data tree.

Each ZooKeeper server has a unique id. This id is used in two places: the myid file and the configuration file. The myid file identifies the server that corresponds to the given data directory. The configuration file lists the contact information for each server identified by

its server id. When a ZooKeeper server instance starts, it reads its id from the myid file and then, using that id, reads from the configuration file, looking up the port on which it should listen.

The snapshot files stored in the data directory are fuzzy snapshots in the sense that during the time the ZooKeeper server is taking the snapshot, updates are occurring to the data tree. The suffix of the snapshot file names is the *zxid*, the ZooKeeper transaction id, of the last committed transaction at the start of the snapshot. Thus, the snapshot includes a subset of the updates to the data tree that occurred while the snapshot was in process. The snapshot, then, may not correspond to any data tree that actually existed, and for this reason we refer to it as a fuzzy snapshot. Still, ZooKeeper can recover using this snapshot because it takes

advantage of the idempotent nature of its updates. By replaying the transaction log against fuzzy snapshots ZooKeeper gets the state of the system at the end of the log.

* + 1. **The Log Directory**

The Log Directory contains the ZooKeeper transaction logs. Before any update takes place, ZooKeeper ensures that the transaction that represents the update is written to non-volatile storage. A new log file is started each time a snapshot is begun. The log file's suffix is the first zxid written to that log.

* + 1. **File Management**

The format of snapshot and log files does not change between standalone ZooKeeper servers and different configurations of replicated ZooKeeper servers. Therefore, you can pull these files from a running replicated ZooKeeper server to a development machine with a stand- alone ZooKeeper server for trouble shooting.

Using older log and snapshot files, you can look at the previous state of ZooKeeper servers and even restore that state. The LogFormatter class allows an administrator to look at the transactions in a log.

The ZooKeeper server creates snapshot and log files, but never deletes them. The retention policy of the data and log files is implemented outside of the ZooKeeper server. The server itself only needs the latest complete fuzzy snapshot and the log files from the start of that snapshot. See the [maintenance](#_bookmark9) section in this document for more details on setting a retention policy and maintenance of ZooKeeper storage.

Note:

The data stored in these files is not encrypted. In the case of storing sensitive data in ZooKeeper, necessary measures need to be taken to prevent unauthorized access. Such measures are external to ZooKeeper (e.g., control access to the files) and depend on the individual settings in which it is being deployed.

* 1. **Things to Avoid**

Here are some common problems you can avoid by configuring ZooKeeper correctly:

# inconsistent lists of servers

The list of ZooKeeper servers used by the clients must match the list of ZooKeeper servers that each ZooKeeper server has. Things work okay if the client list is a subset of the real list, but things will really act strange if clients have a list of ZooKeeper servers that are in different ZooKeeper clusters. Also, the server lists in each Zookeeper server configuration file should be consistent with one another.

# incorrect placement of transaction log

The most performance critical part of ZooKeeper is the transaction log. ZooKeeper syncs transactions to media before it returns a response. A dedicated transaction log device

is key to consistent good performance. Putting the log on a busy device will adversely effect performance. If you only have one storage device, put trace files on NFS and increase the snapshotCount; it doesn't eliminate the problem, but it should mitigate it. **incorrect Java heap size**

You should take special care to set your Java max heap size correctly. In particular, you should not create a situation in which ZooKeeper swaps to disk. The disk is death to ZooKeeper. Everything is ordered, so if processing one request swaps the disk, all other queued requests will probably do the same. the disk. DON'T SWAP.

Be conservative in your estimates: if you have 4G of RAM, do not set the Java max heap size to 6G or even 4G. For example, it is more likely you would use a 3G heap for a 4G machine, as the operating system and the cache also need memory. The best and only recommend practice for estimating the heap size your system needs is to run load tests, and then make sure you are well below the usage limit that would cause the system to swap.

# Publicly accessible deployment

A ZooKeeper ensemble is expected to operate in a trusted computing environment. It is thus recommended to deploy ZooKeeper behind a firewall.

* 1. **Best Practices**

For best results, take note of the following list of good Zookeeper practices:

For multi-tennant installations see the section detailing ZooKeeper "chroot" support, this can be very useful when deploying many applications/services interfacing to a single ZooKeeper cluster.