Apache Zookeeper

- The ZooKeeper framework was originally built at “Yahoo!” for accessing their applications in an easy and robust manner.
- Later, Apache ZooKeeper became a standard for organized service used by **Hadoop, HBase**, and other distributed frameworks.
- For example, Apache HBase uses ZooKeeper to **track the status** of distributed data.
- ZooKeeper is a distributed co-ordination service to manage large set of hosts.
- Co-ordinating and managing a service in a distributed environment is a **complicated process**.
- ZooKeeper solves this issue with its **simple architecture and API**.
- ZooKeeper allows developers to focus on **core application logic** without worrying about the distributed nature of the application.

- **Cluster synchronization**: Apache ZooKeeper is a service used by a **cluster** (group of nodes) to coordinate between themselves and maintain **shared data** with **robust synchronization** techniques.
- ZooKeeper is itself a distributed application providing services for writing a distributed application.

Common Services

- **Naming service** − Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.
- **Configuration management** − Latest and up-to-date configuration information of the system for a joining node.
- **Cluster management** − Joining / leaving of a node in a cluster and node status at real time.
- **Leader election** − Electing a node as leader for coordination purpose.
- **Locking and synchronization service** − Locking the data while modifying it. This mechanism helps you in automatic fail recovery while connecting other distributed applications like Apache HBase.
- **Highly reliable data registry** − Availability of data even when one or a few nodes are down.

Distributed applications vs zookeeper

- Distributed applications offer a lot of benefits, but they throw a few complex and hard-to-crack challenges as well.

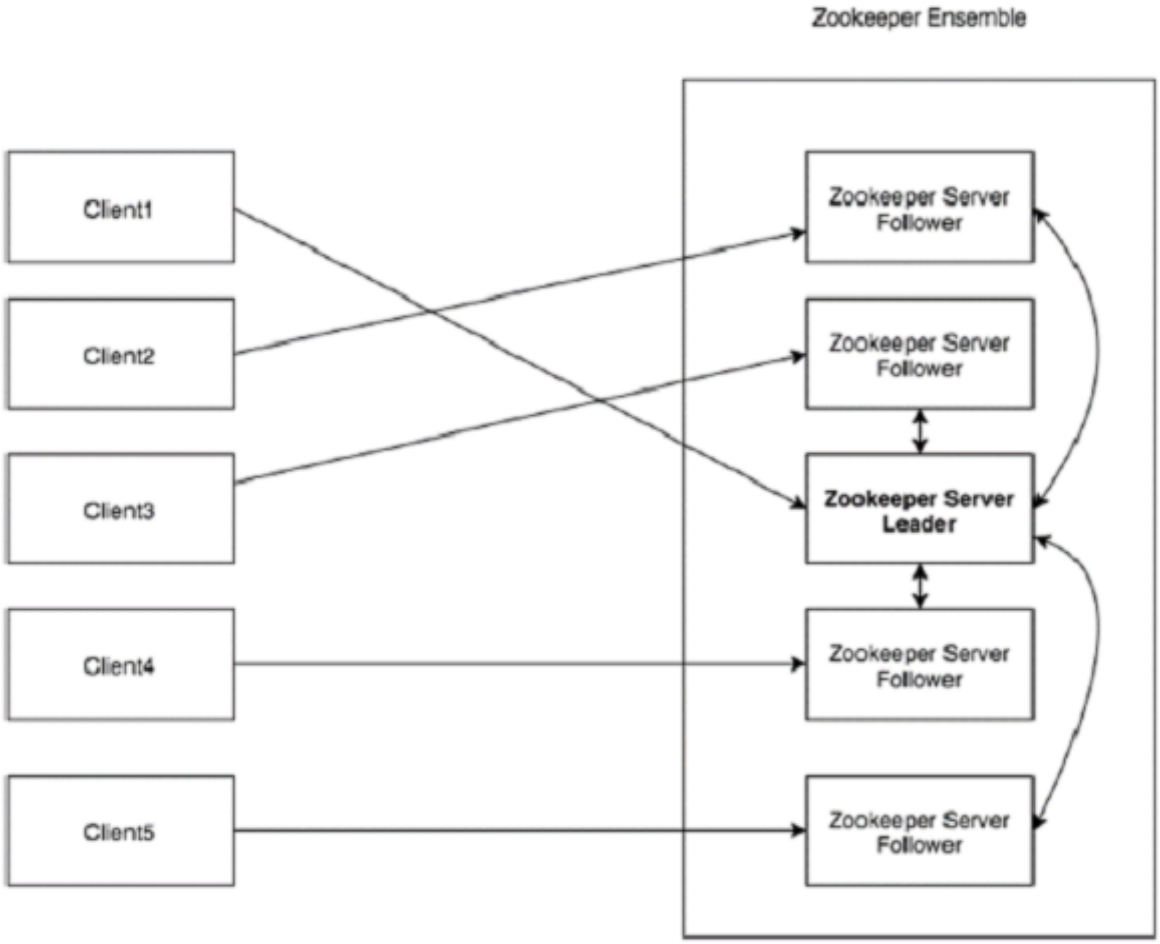
  (eg : distributed appliation problems: race condition, deadlock, inconsistency)
- ZooKeeper framework provides a complete mechanism to overcome all the challenges.
- Race condition and deadlock are handled using **fail-safe synchronization approach**.

- Another main drawback is inconsistency of data, which ZooKeeper resolves with **atomicity**.

BENEFITS OF ZOOKEEPER

- **Simple distributed coordination process**
- **Synchronization** − Mutual exclusion and co-operation between server processes. This process helps in Apache HBase for configuration management.
- **Ordered Messages**
- **Serialization** − Encode the data according to specific rules. Ensure your application runs consistently. This approach can be used in MapReduce to coordinate queue to execute running threads.
- **Reliability**
- **Atomicity** − Data transfer either succeed or fail completely, but no transaction is partial.

Zookeeper Architecture and Data Model



ach one of the components that is a part of the ZooKeeper architecture has been explained in the following table.

Part	Description
Client	Clients, one of the nodes in our distributed application cluster, access information from the server. For a particular time interval, every client sends a message to the server to let the sever know that the client is alive. Similarly, the server sends an acknowledgement when a client connects. If there is no response from the connected server, the client automatically redirects the message to another server.
Server	Server, one of the nodes in our ZooKeeper ensemble, provides all the services to clients. Gives acknowledgement to client to inform that the server is alive.
Ensemble	Group of ZooKeeper servers. The minimum number of nodes that is required to form an ensemble is 3.
Leader	Server node which performs automatic recovery if any of the connected node failed. Leaders are elected on service startup.
Follower	Server node which follows leader instruction.

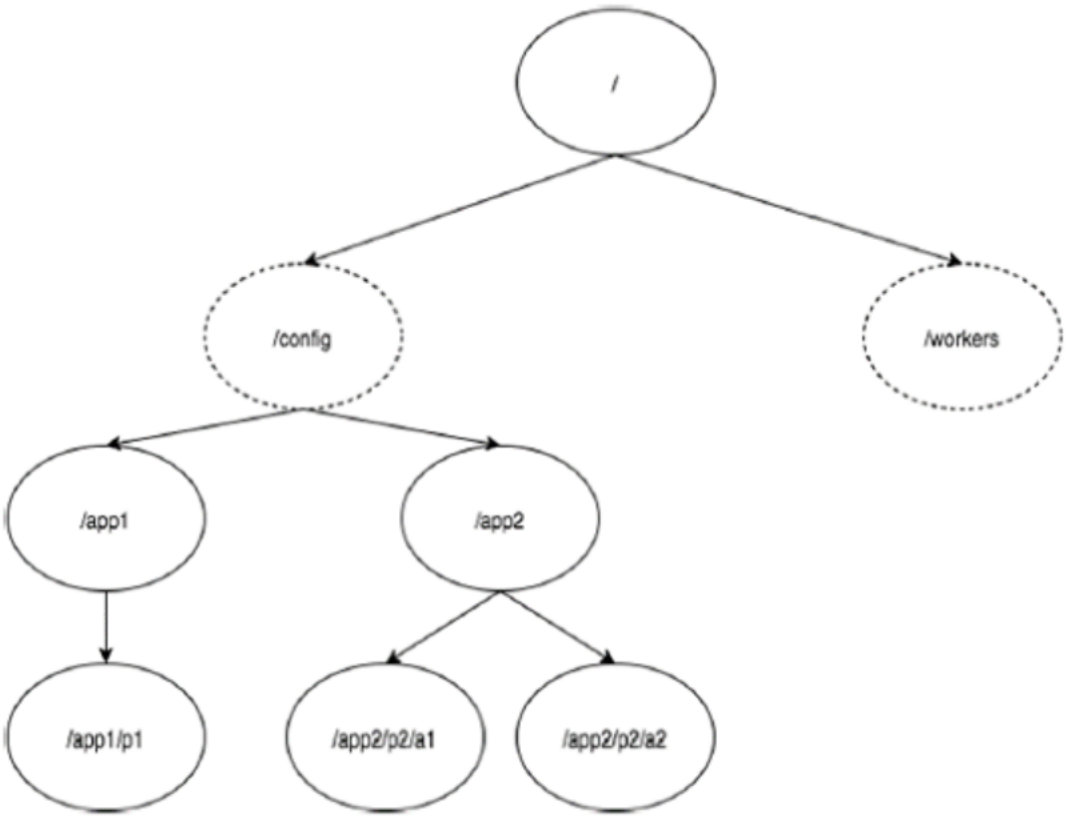
Hierarchical Namespace

The following diagram depicts the tree structure of ZooKeeper file system used for memory representation. ZooKeeper node is referred as **znode**. Every znode is identified by a name and separated by a sequence of path (/).

- In the diagram, first you have a root **znode** separated by "/". Under root, you have two logical namespaces **config** and **workers**.
- The **config** namespace is used for centralized configuration management and the **workers** namespace is used for naming.
- Under **config** namespace, each znode can store upto 1MB of data. This is similar to UNIX file system except that the parent znode can store data as well. The main purpose of this structure is to store synchronized data and describe the metadata of the znode. This structure is called as **ZooKeeper Data Model**.

Every znode in the ZooKeeper data model maintains a **stat** structure. A stat simply provides the **metadata** of a znode. It consists of *Version number, Action control list (ACL), Timestamp, and Data length.*

- **Version number** – Every znode has a version number, which means every time the data associated with the znode changes, its corresponding version number would also increased. The use of version number is important when multiple zookeeper clients are trying to perform operations over the same znode.
- **Action Control List (ACL)** – ACL is basically an authentication mechanism for accessing the znode. It governs all the znode read and write operations.
- **Timestamp** – Timestamp represents time elapsed from znode creation and modification. It is usually represented in milliseconds. ZooKeeper identifies every change to the znodes from "Transaction ID" (zxid). **Zxid** is unique and maintains time for each transaction so that you can easily identify the time elapsed from one request to another request.
- **Data length** – Total amount of the data stored in a znode is the data length. You can store a maximum of 1MB of data.



Types of znode

- Persistent
- Ephemeral
- Sequential

- **Persistence znode** – Persistence znode is alive even after the client, which created that particular znode, is disconnected. By default, all znodes are persistent unless otherwise specified.
- **Ephemeral znode** – Ephemeral znodes are active until the client is alive. When a client gets disconnected from the ZooKeeper ensemble, then the ephemeral znodes get deleted automatically. For this reason, only ephemeral znodes are not allowed to have a children further. If an ephemeral znode is deleted, then the next suitable node will fill its position. Ephemeral znodes play an important role in Leader election.
- **Sequential znode** – Sequential znodes can be either persistent or ephemeral. When a new znode is created as a sequential znode, then ZooKeeper sets the path of the znode by attaching a 10 digit sequence number to the original name. For example, if a znode with path **/myapp** is created as a sequential znode, ZooKeeper will change the path to **/myapp0000000001** and set the next sequence number as 0000000002. If two sequential znodes are created concurrently, then ZooKeeper never uses the same number for each znode. Sequential znodes play an important role in Locking and Synchronization.

Zookeeper Sessions

- Time unit of a series of operation
- Requests - FIFO
- Sessionid
- Heartbeats – client to ensemble
- Timeouts
- Sessions are very important for the operation of ZooKeeper. Requests in a session are executed in FIFO order. Once a client connects to a server, the session will be established and a **session id** is assigned to the client.
- The client sends **heartbeats** at a particular time interval to keep the session valid. If the ZooKeeper ensemble does not receive heartbeats from a client for more than the period (session timeout) specified at the starting of the service, it decides that the client died.
- Session timeouts are usually represented in milliseconds. When a session ends for any reason, the ephemeral znodes created during that session also get deleted.

Zookeeper Watches

- Simple mechanism for the client to get notifications about the changes in the ZooKeeper ensemble
- Clients can set watches while reading a particular znode
- Watches send a notification to the registered client for any of the znode (on which client registers) changes.
    - Data or children
- Triggered once
- Deleted at session termination
- Watches are a simple mechanism for the client to get notifications about the changes in the ZooKeeper ensemble. Clients can set watches while reading a particular znode. Watches send a notification to the registered client for any of the znode (on which client registers) changes.
- Znode changes are modification of data associated with the znode or changes in the znode's children. Watches are triggered only once. If a client wants a notification again, it must be done through another read operation. When a connection session is expired, the client will be disconnected from the server and the associated watches are also removed.

Zookeeper Command Line Interface Commands (CLI):

ZooKeeper Command Line Interface (CLI) is used to interact with the ZooKeeper ensemble for development purpose. It is useful for debugging and working around with different options.

To perform ZooKeeper CLI operations, first turn on your ZooKeeper server ("*bin/zkServer.sh start*") and then, ZooKeeper client ("*bin/zkCli.sh*"). Once the client starts, you can perform the following operation –

- Create znodes
- Get data
- Watch znode for changes
- Set data
- Create children of a znode
- List children of a znode
- Check Status
- Remove / Delete a znode

Now let us see above command one by one with an example.

Create Znodes

Create a znode with the given path. The **flag** argument specifies whether the created znode will be ephemeral, persistent, or sequential. By default, all znodes are persistent.

- **Ephemeral znodes** (flag: e) will be automatically deleted when a session expires or when the client disconnects.
- **Sequential znodes** guaranty that the znode path will be unique.
- ZooKeeper ensemble will add sequence number along with 10 digit padding to the znode path. For example, the znode path */myapp will be converted to /myapp0000000001* and the next sequence number will be */myapp0000000002*. If no flags are specified, then the znode is considered as **persistent**.

Syntax

```
create /path /data
```

Sample

```
create /FirstZnode "Myfirstzookeeper-app"
```

Output

```
[zk: localhost:2181(CONNECTED) 0] create /FirstZnode "Myfirstzookeeper-app"
Created /FirstZnode
```

To create a **Sequential znode**, add **-s flag** as shown below.

Syntax

```
create -s /path /data
```

Sample

```
create -s /FirstZnode second-data
```

Output

```
[zk: localhost:2181(CONNECTED) 2] create -s /FirstZnode "second-data"
Created /FirstZnode0000000023
```

To create an **Ephemeral Znode**, add **-e flag** as shown below.

Syntax

```
create -e /path /data
```

Sample

```
create -e /SecondZnode "Ephemeral-data"
```

Output

```
[zk: localhost:2181(CONNECTED) 2] create -e /SecondZnode "Ephemeral-data"
Created /SecondZnode
```

Remember when a client connection is lost, the ephemeral znode will be deleted. You can try it by quitting the ZooKeeper CLI and then re-opening the CLI.

Get Data

It returns the associated data of the znode and metadata of the specified znode. You will get information such as when the data was last modified, where it was modified, and information about the data. This CLI is also used to assign watches to show notification about the data.

Syntax

```
get /path
```

Sample

```
get /FirstZnode
```

Output

```
[zk: localhost:2181(CONNECTED) 1] get /FirstZnode
“Myfirstzookeeper-app”
cZxid = 0x7f
ctime = Tue Sep 29 16:15:47 IST 2015
mZxid = 0x7f
mtime = Tue Sep 29 16:15:47 IST 2015
pZxid = 0x7f
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 22
numChildren = 0
```

To access a sequential znode, you must enter the full path of the znode.

Sample

```
get /FirstZnode0000000023
```

Output

```
[zk: localhost:2181(CONNECTED) 1] get /FirstZnode0000000023
“Second-data”
cZxid = 0x80
ctime = Tue Sep 29 16:25:47 IST 2015
mZxid = 0x80
mtime = Tue Sep 29 16:25:47 IST 2015
pZxid = 0x80
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 13
numChildren = 0
```

Watch

Watches show a notification when the specified znode or znode's children data changes. You can set a **watch** only in **get** command.

Syntax

```
get /path [watch] 1
```

Sample

```
get /FirstZnode 1
```

Output

```
[zk: localhost:2181(CONNECTED) 1] get /FirstZnode 1
"Myfirstzookeeper-app"
cZxid = 0x7f
ctime = Tue Sep 29 16:15:47 IST 2015
mZxid = 0x7f
mtime = Tue Sep 29 16:15:47 IST 2015
pZxid = 0x7f
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 22
numChildren = 0
```

The output is similar to normal **get** command, but it will wait for znode changes in the background. <Start here>

Set Data

Set the data of the specified znode. Once you finish this set operation, you can check the data using the **get** CLI command.

Syntax

```
set /path /data
```

Sample

```
set /SecondZnode Data-updated
```

Output

```
[zk: localhost:2181(CONNECTED) 1] get /SecondZnode "Data-updated"
cZxid = 0x82
ctime = Tue Sep 29 16:29:50 IST 2015
mZxid = 0x83
mtime = Tue Sep 29 16:29:50 IST 2015
pZxid = 0x82
cversion = 0
```

```
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x15018b47db00000
dataLength = 14
numChildren = 0
```

If you assigned **watch** option in **get** command (as in previous command), then the output will be similar as shown below –

Output

```
[zk: localhost:2181(CONNECTED) 1] get /FirstZnode "Mysecondzookeeper-app"

WATCHER: :

WatchedEvent state:SyncConnected type:NodeDataChanged path:/FirstZnode
cZxid = 0x7f
ctime = Tue Sep 29 16:15:47 IST 2015
mZxid = 0x84
mtime = Tue Sep 29 17:14:47 IST 2015
pZxid = 0x7f
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 23
numChildren = 0
```

Create Children / Sub-znode

Creating children is similar to creating new znodes. The only difference is that the path of the child znode will have the parent path as well.

Syntax

```
create /parent/path/subnode/path /data
```

Sample

```
create /FirstZnode/Child1 firstchildren
```

Output

```
[zk: localhost:2181(CONNECTED) 16] create /FirstZnode/Child1 "firstchildren"
created /FirstZnode/Child1
[zk: localhost:2181(CONNECTED) 17] create /FirstZnode/Child2 "secondchildren"
created /FirstZnode/Child2
```

List Children

This command is used to list and display the **children** of a znode.

Syntax

```
ls /path
```

Sample

```
ls /MyFirstZnode
```

Output

```
[zk: localhost:2181(CONNECTED) 2] ls /MyFirstZnode
[mysecondsubnode, myfirstsubnode]
```

Check Status

Status describes the metadata of a specified znode. It contains details such as Timestamp, Version number, ACL, Data length, and Children znode.

Syntax

```
stat /path
```

Sample

```
stat /FirstZnode
```

Output

```
[zk: localhost:2181(CONNECTED) 1] stat /FirstZnode
cZxid = 0x7f
ctime = Tue Sep 29 16:15:47 IST 2015
mZxid = 0x7f
mtime = Tue Sep 29 17:14:24 IST 2015
pZxid = 0x7f
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 23
numChildren = 0
```

Remove a Znode

Removes a specified znode and recursively all its children. This would happen only if such a znode is available.

Syntax

```
rmr /path
```

Sample

```
rmr /FirstZnode
```

Output

```
[zk: localhost:2181(CONNECTED) 10] rmr /FirstZnode
[zk: localhost:2181(CONNECTED) 11] get /FirstZnode
Node does not exist: /FirstZnode
```

Delete **(delete /path)** command is similar to **remove** command, except the fact that it works only on znodes with no children.

Zookeeper four letter commands:

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.

```
$ 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                   - only exposed by the Leader
zk_synced_followers 4               - only exposed by the Leader
zk_pending_syncs    0               - only exposed by the Leader
zk_open_file_descriptor_count 23    - only available on Unix
platforms
zk_max_file_descriptor_count 1024   - only available on Unix
platforms
```

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
```