

# Introduction to ZooKeeper

Omid Vahdaty, BigData ninja



# Sources

- <https://zookeeper.apache.org/doc/trunk/zookeeperOver.pdf>
- <http://www.tutorialspoint.com/zookeeper/index.htm>



# What is Zookeeper

- is a distributed coordination service to manage large set of hosts.
- allows R&D to focus on core application logic without worrying about the distributed nature of the application
- 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



# Features

**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

**Highly reliable data registry** – Availability of data even when one or a few nodes are down.



# Benefits

## **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.

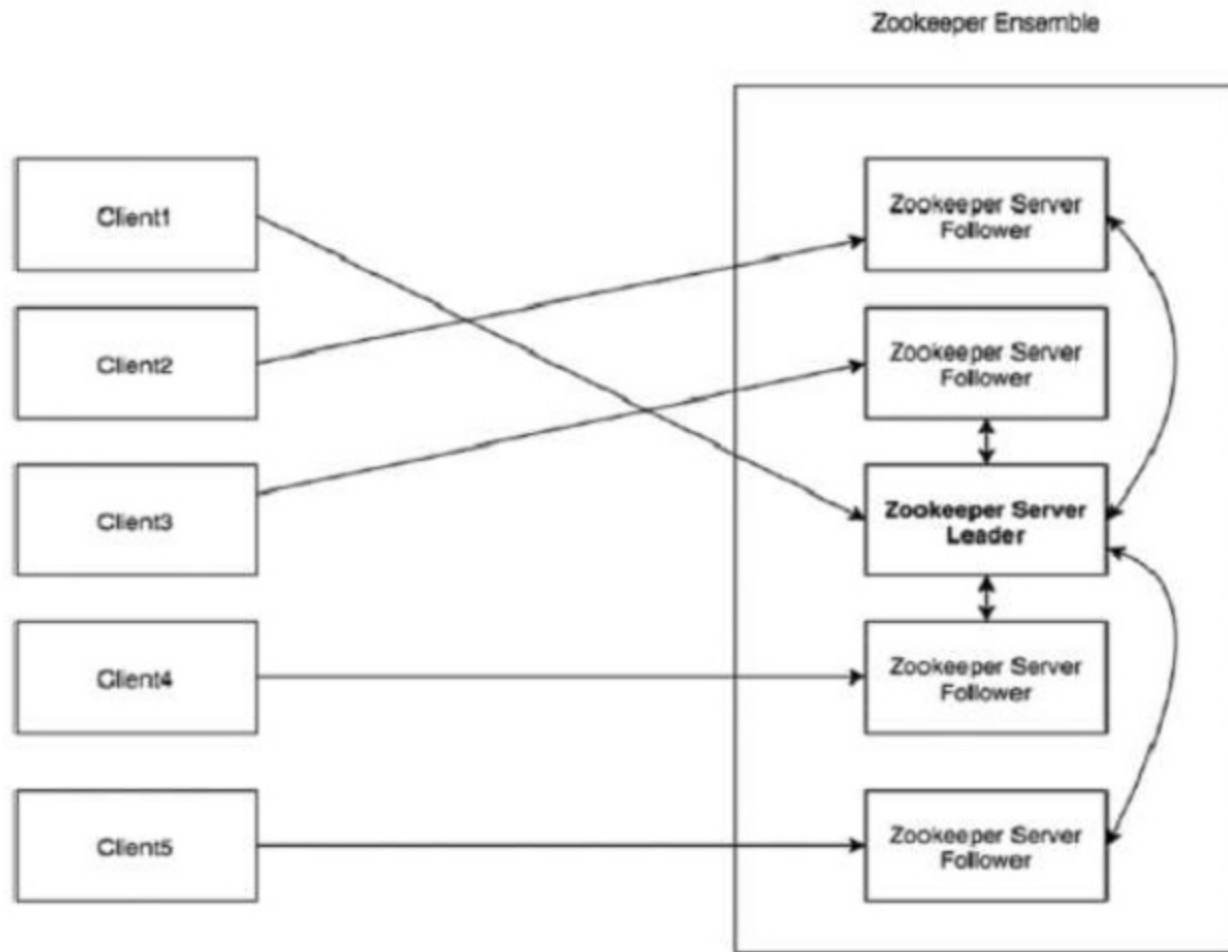


# Fundamental Concepts of Zookeeper

- Architecture
- Hierarchical namespace
- Session
- Watches
- ZooKeeper Failover Controller :



# Architecture



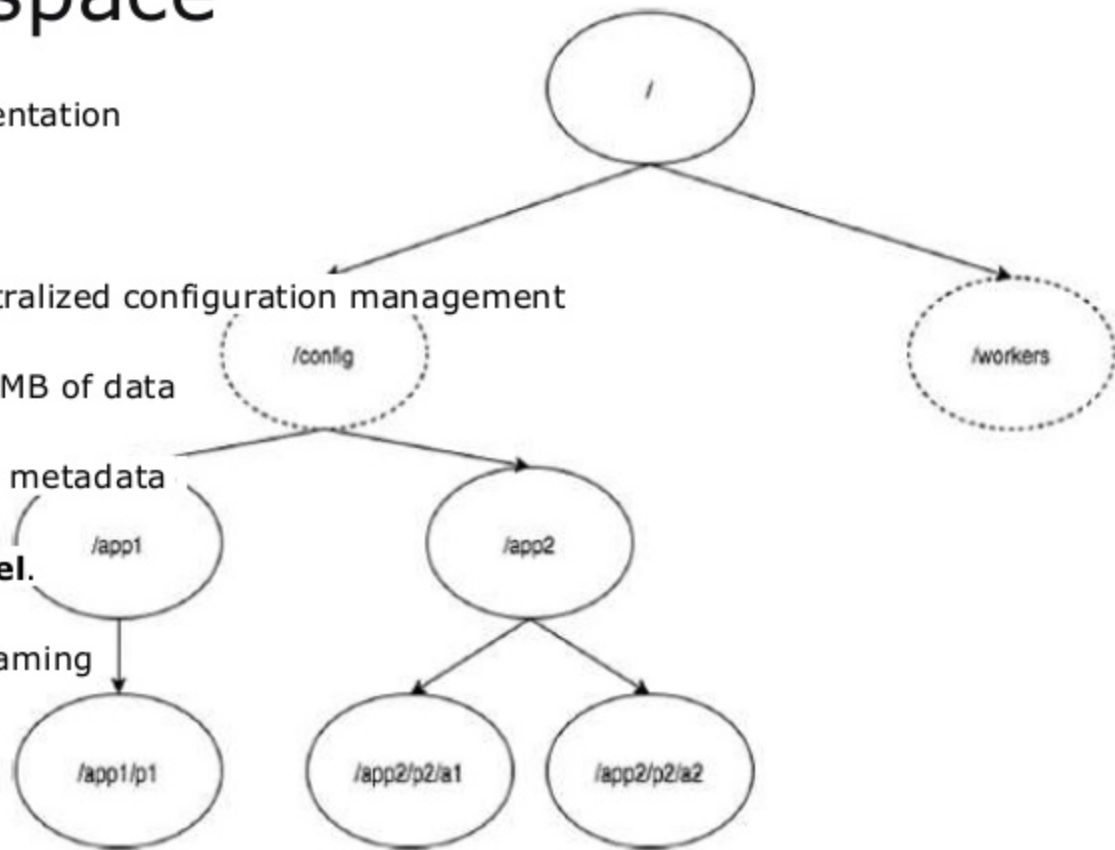
# Architecture Components:

- **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 server 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:** 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 :3.
- **Leader:** Server node which performs automatic recovery if any of the connected node failed. Leaders are elected on service startup.



# Hierarchical Namespace

- Is a file system used for memory representation
- root **znode** separated by "/"
  - **config** namespace is used for centralized configuration management
    - each znode can store upto 1MB of data
    - store synchronized data and metadata
    - AKA **ZooKeeper Data Model.**
  - **workers** namespace is used for naming



# Data Model

- for each znode → **stat** structure (**metadata** of a znode):
  - **Version number**
    - every time the data associated with the znode changes, its corresponding version number would also increased.
  - *Action control list* (**ACL**): authentication
  - **Timestamp**
    - time elapsed from znode creation and modification
    - ZooKeeper identifies every change to the znodes from "Transaction ID": **Zxid**
    - **Zxid**
      - unique

# Znode types

## Persistence znode (default)

- a. Persistence znode is alive even after the client is disconnected.

## Ephemeral znode


- a. Ephemeral znodes are active until the client is alive.
- b. if **client** gets **disconnected** → ephemeral znodes get **deleted** automatically.
- c. only ephemeral znodes are **not allowed** to have a **children** further.
- d. If an ephemeral znode is deleted, then the next suitable node will fill its position.

# Znode types

## Sequential znode

- a. Sequential znodes can be either **persistent** or **ephemeral**.
- b. if a znode with path/**myapp** is created as a sequential znode
  - i. ZooKeeper will change the path to **/myapp0000000001**
  - ii. set the next sequence number as **0000000002**.
  - iii. ZooKeeper never uses the same number for each znode.
- c. **Sequential** znodes user for **Locking** and **Synchronization**.

# Session

- Requests in a **session** are executed in **FIFO** order.
  - Once a client connects → **session id** is assigned to the client.
  - The client sends **heartbeats** at a particular time interval to keep the **session valid**.
  - If **not** received **heartbeats** from a client → it decides that the client **died**.
  - Session **timeouts** are usually represented in **milliseconds**.
  - **session ends** → **ephemeral** znodes get **deleted**.
- 

# Watches

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.

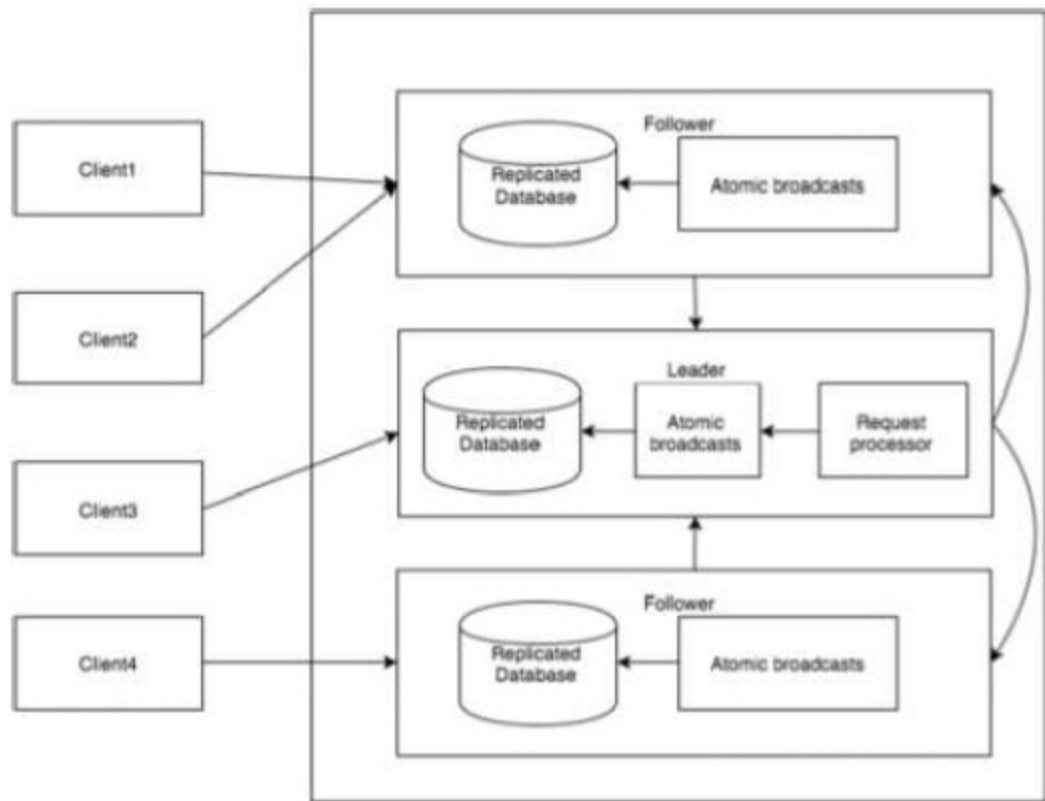
Znode changes are modification of data associated with the znode or changes in the znode's children. Watches are **triggered only once**. If a client wants a notification again, it must be done through another read operation. When a connection session is expired, the client will be disconnected from the server and the associated watches are also removed.



# Flow

1. Ensemble start → wait
2. Client connect → server (sessions ID)
3. Server → ACK to client
4. Not connected → No ACK? → repeat 2,3
5. Connected → heartbeat to server.
6. Possible to read from particular Znode. (read is from local DB on znode).
7. Write → client send server → server send Leader node → "request processor" to followers → Quorum? → success write: "atomic broadcast".

# Flow






# Install

- [http://www.tutorialspoint.com/zookeeper/zookeeper\\_installation.htm](http://www.tutorialspoint.com/zookeeper/zookeeper_installation.htm)
- [https://docs.midonet.org/docs/latest-en/quick-start-guide/ubuntu-1404\\_kilo/content/\\_zookeeper\\_installation.html](https://docs.midonet.org/docs/latest-en/quick-start-guide/ubuntu-1404_kilo/content/_zookeeper_installation.html)
- <http://myjeeva.com/zookeeper-cluster-setup.html> (install , logs, debug)
- [http://zookeeper.apache.org/doc/r3.3.4/zookeeperAdmin.html#sc\\_maintenance](http://zookeeper.apache.org/doc/r3.3.4/zookeeperAdmin.html#sc_maintenance)
- `zkServer.sh status`

# ZooKeeper CLI: zkCLI.sh

- 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.
  - CMD
    - 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
- 

# maintenance

[http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc\\_maintenance](http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance)




## Common errors:

- Opening socket connection to server localhost/127.0.0.1:2181. Will not attempt to authenticate using SASL
- Failed to resolve address - (forgot to handle DNS resolve)
- myid file is missing - inside the data folder of ZK , create a file , add an id of the server 1..N



# High level steps of install

1. **assuming 3 resolvable hostnames: master, master2, master 3**
  2. Download the package of zk
  3. Myid file needs to be created
  4. Update the zoo.cfg (sample config will follow)
  5. Notice the data dir, log dir, and server config
  6. Start each zkServer manually (not like hadoop - start-dfs.sh)
- 

# Example config : zoo.cfg

# The number of milliseconds of each tick

tickTime=2000

# The number of ticks that the initial

# synchronization phase can take

initLimit=10

# The number of ticks that can pass between

# sending a request and getting an acknowledgement

syncLimit=5



# Example config : zoo.cfg

# the directory where the snapshot is stored.

# do not use /tmp for storage, /tmp here is just

# example sakes.

**dataDir=/hadoop-data/hadoopuser/hdfs/zkdata**

# the port at which the clients will connect

**clientPort=2181**

# the directory where transaction log is stored.

# this parameter provides dedicated log device for ZooKeeper

**dataLogDir=/hadoop-data/hadoopuser/hdfs/zklog**



# Example config : zoo.cfg

# the maximum number of client connections.

# increase this if you need to handle more clients

**#maxClientCnxns=60**

# The number of snapshots to retain in dataDir

autopurge.snapRetainCount=3

# Purge task interval in hours

# Set to "0" to disable auto purge feature

**autopurge.purgeInterval=1**





# Example config : zoo.cfg

# configuration of Server ID

server.1=master:2888:3888

server.2=master2:2888:3888

server.3=master3:2888:3888



# Don't forget to configure the myid file

On master

```
echo 1 > /hadoop-data/hadoopuser/hdfs/zkdata/myid
```

On Master2

```
echo 1 > /hadoop-data/hadoopuser/hdfs/zkdata/myid
```

On master3

```
echo 1 > /hadoop-data/hadoopuser/hdfs/zkdata/myid
```



## Further read:

- Hadoop with QJM and ZKFC:
- <https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html>
- How to Install ZK:
- [http://www.tutorialspoint.com/zookeeper/zookeeper\\_installation.htm](http://www.tutorialspoint.com/zookeeper/zookeeper_installation.htm)



# ZK with HDFS HA = automatic failover

- <https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html>
- 2 new components to HDFS:
  - a ZooKeeper quorum
  - ZKFailoverController
    - **Health monitoring**
    - **ZooKeeper session management**
    - **ZooKeeper-based election**
- High level steps:

# Added config of ZK into hdfs

```
<property>
```

```
  <name>dfs.ha.automatic-failover.enabled</name>
```

```
  <value>true</value>
```

```
</property>
```

```
<property>
```

```
  <name>ha.zookeeper.quorum</name>
```

```
  <value>master:2181,master2:2181,master3:2181</value>
```

```
</property>
```



# Test your Failover

1. Reboot a machine with Active NN.
2. Test test the status of NN's and ZK's
3. Stabilize the cluster - start
  - a. ZK: ***zkServer start***
  - b. QJM: ***hadoop-daemon.sh start journalnode***
  - c. Zkfc: ***hadoop-daemon.sh start zkfc***
  - d. NN: ***hadoop-daemon.sh start namenode***
  - e. DN: ***hadoop-daemon.sh start datanode***

4. Confirm Via: