

APACHE ZOOKEEPER

Overview – What is ZooKeeper?

- An open source, high-performance coordination service for distributed application.
- Exposes common services in simple interface:
 - Naming
 - Configuration management
 - Locks & synchronization
 - Groups services
- Build your own on it for specific needs



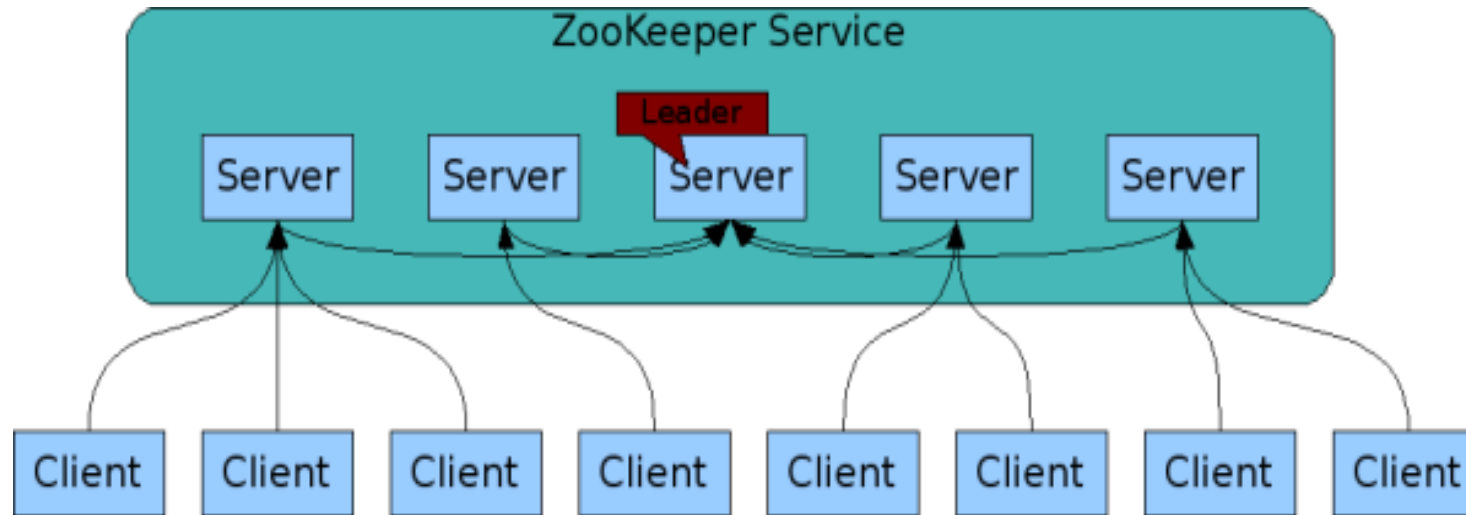
Overview – Who uses ZooKeeper?

- Companies:
 - Yahoo!
 - Zynga
 - Rackspace
 - LinkedIn
 - Netflix, and many more...
- Projects:
 - Apache Map/Reduce (Yarn)
 - Apache HBase
 - Apache Kafka
 - Apache Storm
 - Neo4j, and many more...

Overview – ZooKeeper Use Cases

- Configuration Management
 - Cluster member nodes bootstrapping configuration from a centralized source in unattended way
- Distributed Cluster Management
 - Node join / leave
 - Node statuses in real time
- Naming service – e.g. DNS
- Distributed synchronization – locks, barriers, queues
- Leader election in a distributed system

The ZooKeeper Service (ZKS)



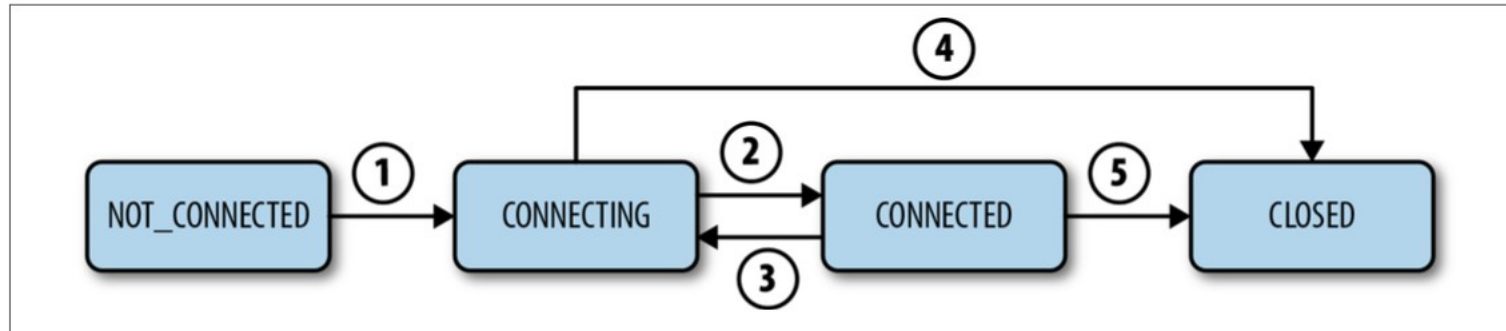
- ZooKeeper Service is replicated over a set of machines
- All machines store a copy of the data (in-memory)
- A leader is elected on service startup
- Clients only connect to a single ZooKeeper server and maintain a TCP connection

The ZKS - Sessions

- Before executing any request, client must establish a session with service
- All operations client submits to service are associated to a session
- Client initially connects to any server in ensemble, and only to single server.
- Session offer *order guarantees* – requests in session are executed in FIFO order

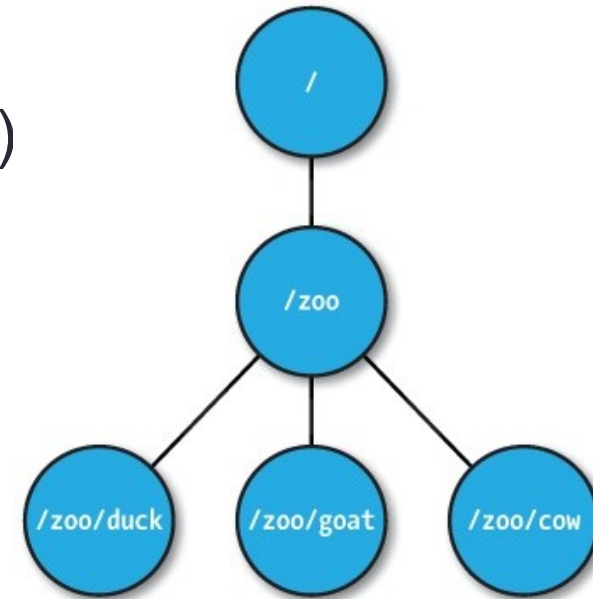
The ZKS – Session States and Lifetime

- Main possible states: CONNECTING, CONNECTED, CLOSED, NOT_CONNECTED



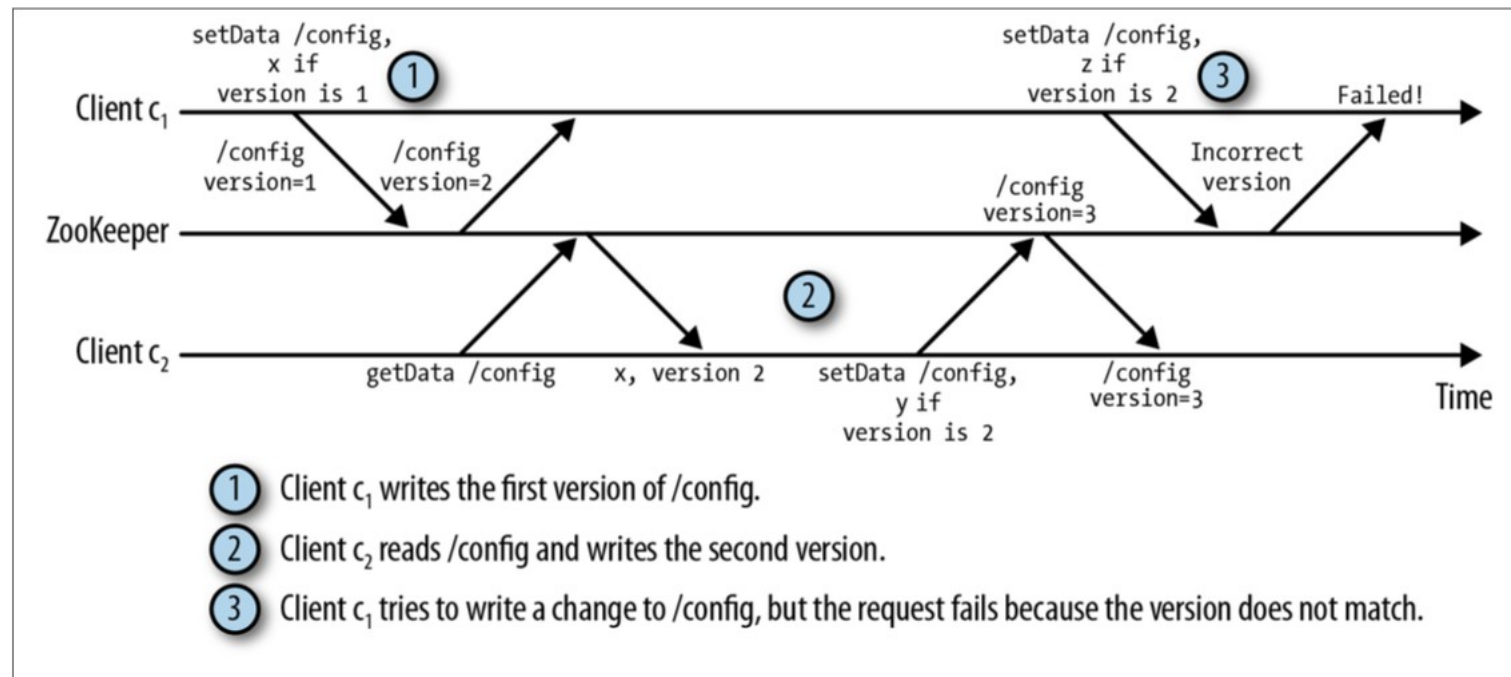
The ZooKeeper Data Model (ZDM)

- Hierarchical name space
- Each node is called as a ZNode
- Every ZNode has data (given as byte[]) and can optionally have children
- ZNode paths:
 - Canonical, absolute, slash-separated
 - No relative references
 - Names can have Unicode characters
- ZNode maintain **stat structure**



ZDM - Versions

- Each Znode has version number, is incremented every time its data changes
- *setData* and *delete* take version as input, operation succeeds only if client's version is equal to server's one



ZDM – ZNodes – Stat Structure

- The Stat structure for each znode in ZooKeeper is made up of the following fields:

• czxid	<code>[zk: localhost(CONNECTED) 0] stat /zookeeper</code>
• mzxid	<code>cZxid = 0x0</code>
• pzxid	<code>ctime = Thu Jan 01 05:30:00 IST 1970</code>
• ctime	<code>mZxid = 0x0</code>
• mtime	<code>mtime = Thu Jan 01 05:30:00 IST 1970</code>
• dataVersion	<code>pZxid = 0x0</code>
• cversion	<code>cversion = -1</code>
• aclVersion	<code>dataVersion = 0</code>
• ephemeralOwner	<code>aclVersion = 0</code>
• dataLength	<code>ephemeralOwner = 0x0</code>
• numChildren	<code>dataLength = 0</code>
	<code>numChildren = 1</code>

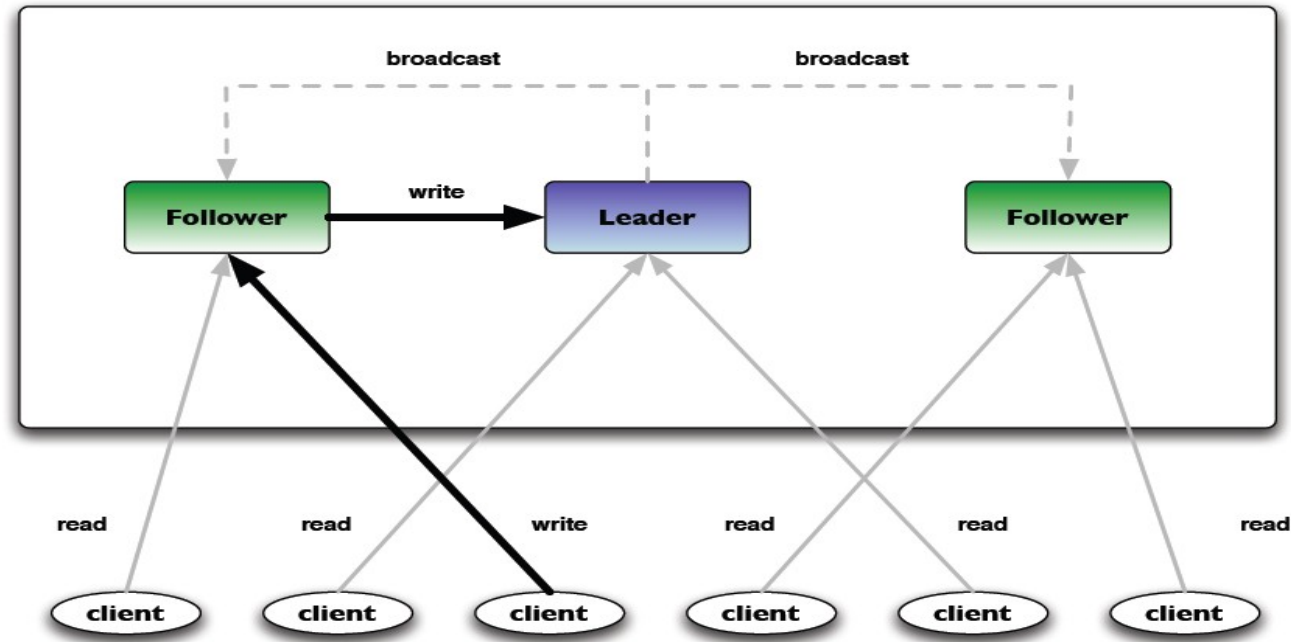
ZDM – Types of ZNode

- Persistent ZNode
 - Have lifetime in ZooKeeper's namespace until they're explicitly deleted (can be deleted by *delete* API call)
- Ephemeral ZNode
 - Is deleted by ZooKeeper service when the creating client's session ends
 - Can also be explicitly deleted
 - Are not allowed to have children
- Sequential Znode
 - Is assigned a sequence number by ZooKeeper as a part of name during creation
 - Sequence number is integer (4bytes) with format of 10 digits with 0 padding. E.g. /path/to/znode-0000000001

ZDM – Znode Operations

Operation	Description
create	Creates a znode in a specified path of the ZooKeeper namespace
delete	Deletes a znode from a specified path of the ZooKeeper namespace
exists	Checks if a znode exists in the path
getChildren	Gets a list of children of a znode
getData	Gets the data associated with a znode
setData	Sets/writes data into the data field of a znode
getACL	Gets the ACL of a znode
setACL	Sets the ACL in a znode
sync	Synchronizes a client's view of a znode with ZooKeeper

ZDM – Znode – Reads & Writes



- Read requests are processed locally at the ZooKeeper server to which client is currently connected
- Write requests are forwarded to leader and go through majority consensus before a response is generated

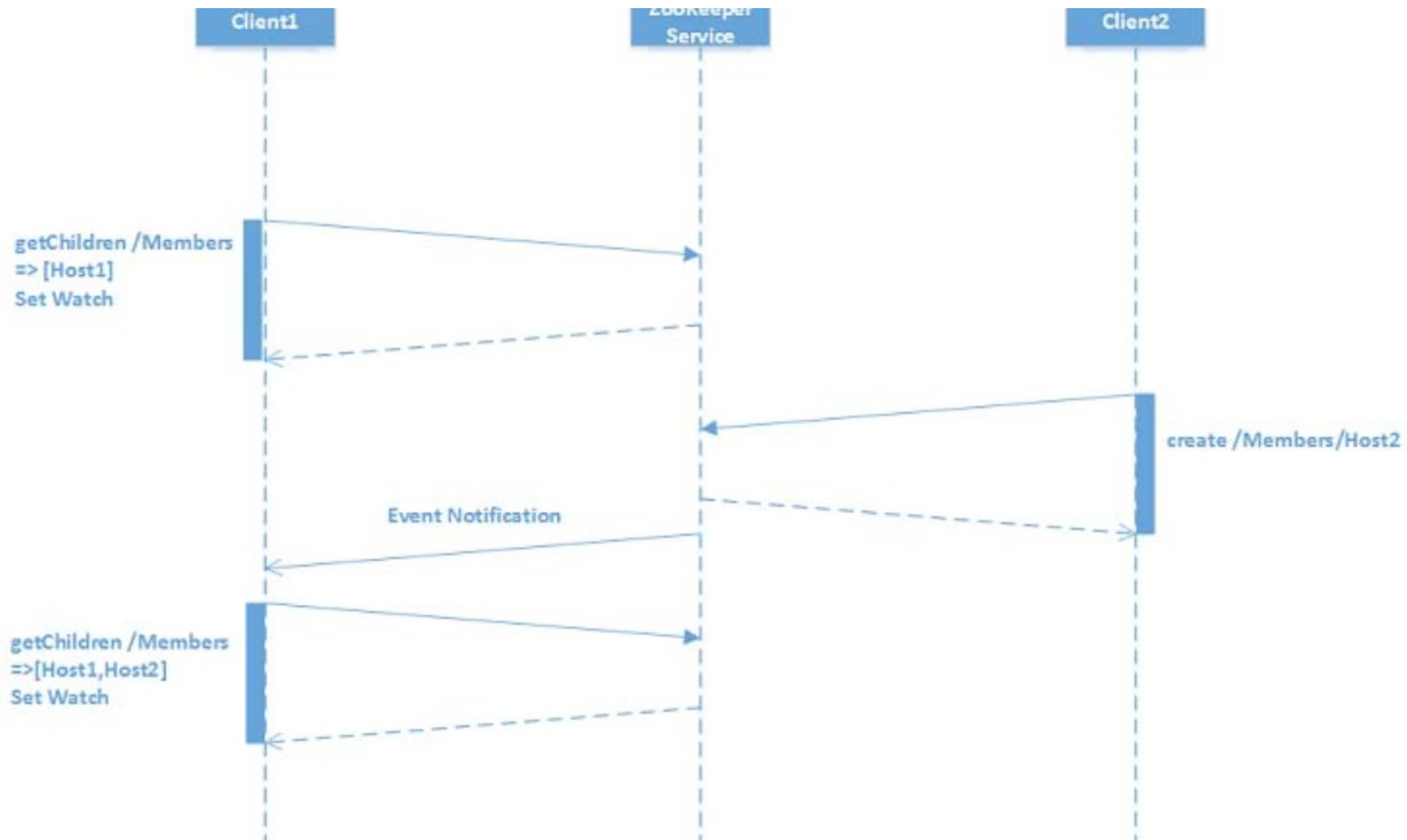
ZDM – Consistency Guarantees

- Sequential Consistency
- Atomicity
- Single System Image
- Reliability
- Timeliness (Eventual Consistency)

ZDM - Watches

- A watch event is one-time trigger, sent to client that set watch, which occurs when data for which watch was set changes.
- Watches allow clients to get notifications when a znode changes in any way (NodeChildrenChanged, NodeCreated, NodeDataChanged, NodeDeleted)
- All of read operations – **getData()**, **getChildren()**, **exists()** – have option of setting watch
- ZooKeeper Guarantees about Watches:
 - Watches are ordered, order of watch events corresponds to the order of the updates
 - A client will see a watch event for znode it is watching before seeing the new data that corresponds to that znode

ZDM – Watches (cont)



ZDM – Access Control List

- ZooKeeper uses ACLs to control access to its znodes
- ACLs are made up of pairs of (scheme:id, permission)
- Build-in ACL schemes
 - **world**: has single id, *anyone*
 - **auth**: doesn't use any id, represents any authenticated user
 - **digest**: use a *username:password*
 - **host**: use the client host name as ACL id identity
 - **ip**: use the client host IP as ACL id identity
- ACL Permissions:
 - **CREATE**
 - **READ**
 - **WRITE**
 - **DELETE**
 - **ADMIN**
- E.g. (ip:192.168.0.0/16, READ)

Recipe #1: Queue

- A distributed queue is very common data structure used in distributed systems.
- Producer: generate / create new items and put them into queue
- Consumer: remove items from queue and process them
- Addition and removal of items follow ordering of FIFO

Recipe #1: Queue (cont)

- A ZNode will be designated to hold a queue instance, queue-znode
- All queue items are stored as znodes under queue-znode
- Producers add an item to queue by creating znode under queue-znode
- Consumers retrieve items by getting and then deleting a child from queue-znode

QUEUE-ZNODE : “queue instance”

|-- QUEUE-00000000001 : “item1”

|-- QUEUE-00000000002 : “item2”

|-- QUEUE-00000000003 : “item3”

Recipe #1: Queue (cont)

- Let **/_QUEUE_** represent top-level znode, is called queue-znode
- Producer put something into queue by creating a **SEQUENCE_EPHEMERAL** znode with name “queue-N”, N is monotonically increasing number
create (“queue-”, SEQUENCE_EPHEMERAL)
- Consumer process **getChildren()** call on queue-znode with watch event set to true
M = getChildren(/_QUEUE_, true)
- Client picks up items from list and continues processing until reaching the end of the list, and then check again
- The algorithm continues until **get_children()** returns empty list

Recipe #2: Group Membership

- A persistent Znode */membership* represent the root of the group in ZooKeeper tree
- Any client that joins the cluster creates ephemeral znode under */membership* to locate memberships in tree and set a watch on */membership*
- When another node joins or leaves the cluster, this node gets a notification and becomes aware of the change in group membership

Recipe #2: Group Membership (cont)

- Let **/_MEMBERSHIP_** represent root of group membership
 - Client joining the group create ephemeral nodes under root
 - All members of group will register for watch events on **/_MEMBERSHIP**, thereby being aware of other members in group
- L = getChildren("/_MEMBERSHIP", true)**
- When new client joins group, all other members are notified
 - Similarly, a client leaves due to failure or otherwise, ZooKeeper automatically delete node, trigger event
 - Live members know which node joined or left by looking at the list of children **L**

Leader Election – SID-ZXID(epoch+count)

2020-08-31 21:24:35,972 [myid:2] - INFO

[QuorumPeer[myid=2](plain=[0:0:0:0:0:0:0:0]:2182)(secure=disabled):QuorumPeer@1371] – LOOKING

2020-08-31 21:24:35,975 [myid:2] - INFO

[QuorumPeer[myid=2](plain=[0:0:0:0:0:0:0:0]:2182)(secure=disabled):FastLeaderElection@944] - New election. **My id = 2, proposed zxid=0x0**

2020-08-31 21:24:35,999 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection\$Messenger\$WorkerReceiver@389] - Notification: **my state:LOOKING; n.sid:2, n.state:LOOKING, n.leader:2**, n.round:0x1, n.peerEpoch:0x0, n.zxid:0x0, message format version:0x2, n.config version:0x0

2020-08-31 21:24:36,004 [myid:2] - INFO [QuorumConnectionThread-[myid=2]-2:QuorumCnxManager@513] - Have smaller server identifier, so dropping the connection: **(myId:2 --> sid:3)**

2020-08-31 21:24:36,009 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection\$Messenger\$WorkerReceiver@389] - Notification: my state:LOOKING; n.sid:3, n.state:LOOKING, n.leader:3, n.round:0x1, n.peerEpoch:0x0, n.zxid:0x0, message format version:0x2, n.config version:0x0

2020-08-31 21:24:36,017 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection\$Messenger\$WorkerReceiver@389] - Notification: **my state:LOOKING; n.sid:1, n.state:FOLLOWING, n.leader:3**, n.round:0x1, n.peerEpoch:0x1, n.zxid:0x0, message format version:0x2, n.config version:0x0

2020-08-31 21:24:36,018 [myid:2] - INFO

[QuorumPeer[myid=2](plain=[0:0:0:0:0:0:0:0]:2182)(secure=disabled):QuorumPeer@857] - Peer state changed: following

2020-08-31 21:24:36,018 [myid:2] - INFO

[QuorumPeer[myid=2](plain=[0:0:0:0:0:0:0:0]:2182)(secure=disabled):QuorumPeer@1453] - **FOLLOWING**

Lab : Zookeeper