Zookeeper

Definition

- ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
- All of these kinds of services are used in some form or another by distributed applications.
- Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them ,which make them brittle in the presence of change and difficult to manage.
- Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

Definition – Simple one

- ZooKeeper allows distributed processes to coordinate with each other through a shared hierarchical name space of data registers

In Action

- It is an independent framework, has no dependency on Hadoop
- Download zookeeper from Apache website
- Lets try with windows
  - Unzip Zookeeper binary to some directory, say E:\tmp
  - Open command prompt, cd to the directory where you have unzipped the tar to.
  - Set the env variable ZK_HOME and point it to the Directory where you have unzipped Zookeeper to
    - set ZK_HOME=E:\tmp
  - Create a new config file (from the same command prompt)
    - notepad %ZK_HOME%\conf\zoo.cfg
    - Save the file with below content (Replace E:/tmp with your Directory name)

```
tickTime=2000

dataDir=E:/tmp/zookeeper-3.4.6/data

clientPort=2181
```

In Action

- Start the server using the following command

```
start %ZK_HOME%\bin\zkServer
```

- Start a Client using the following command

```
start %ZK_HOME%\bin\zkCli
```

- Start one more Client

```
start %ZK_HOME%\bin\zkCli
```

We are simulating two Clients and One server from a windows box.

In Action (all commands in Client)

- Zookeeper stores all the data in hierarchial fashion
- To view the hierarchy, use the ls command
- Type `ls /`
- Create a node in one client using the below command

```
create /n1 MY_DATA_IN_N1
```

- View the Data in the same/another node using

```
get /n1
```

- Modify the data using Set and do a Get again

```
set /n1 NEW_DATA
```

- You can nest nodes

```
create /n1/n2 MYNEWNODE2
```

- You can set watches on nodes (from another client, try doing a set on n1 and observe the effect on the first client)

```
get /n1 watch
```

- You could also create ephemeral nodes using –e

```
create –e /e1 TESTDATA
```

- Observe the node in another client

```
ls /
```

- Close the client which has created the ephemeral node (do a force close and a graceful close using quit, and observe what happens in another client using ls)

```
ls /
```

- Finally, you can delete nodes using delete

```
delete /n1/n2
```

Terms

- Node = zNode (a node in ZK hierarchial data store with some data in it)
- A set of Server = quorum
- A Group of quorums = ensemble
- Ephemeral = short lived zNode (lifespan is for one session)
- Sequential = numbered node
- Persisted = long living node, usually persisted in a file in data dir

Setting up of a Quorum

- If you have one host at your disposal but want to run 3 ZK Servers do the below.
- Run the following sequence of commands

```
mkdir %ZK_HOME%\data\1

mkdir %ZK_HOME%\data\2

mkdir %ZK_HOME%\data\3


echo|set /p="1">%ZK_HOME%\data\1\myid

echo|set /p="2">%ZK_HOME%\data\2\myid

echo|set /p="3">%ZK_HOME%\data\3\myid
```

- Create three configuration files in %ZK_HOME%\conf as shown on right side (note the changes in red color)

zoo1.cfg

```
tickTime=2000
dataDir=E:/tmp/zookeeper-3.4.6/data/1
clientPort=2181
initLimit=5
syncLimit=2
server.1=localhost:2111:3111
server.2=localhost:2112:3112
server.3=localhost:2113:3113
```

zoo2.cfg

```
tickTime=2000
dataDir=E:/tmp/zookeeper-3.4.6/data/2
clientPort=2182
initLimit=5
syncLimit=2
server.1=localhost:2111:3111
server.2=localhost:2112:3112
server.3=localhost:2113:3113
```

```
tickTime=2000
dataDir=E:/tmp/zookeeper-3.4.6/data/3
clientPort=2183
initLimit=5
syncLimit=2
server.1=localhost:2111:3111
server.2=localhost:2112:3112
server.3=localhost:2113:3113
```

zoo3.cfg

Starting a quorum

- Start 3 ZK Servers (note the changes in red color)

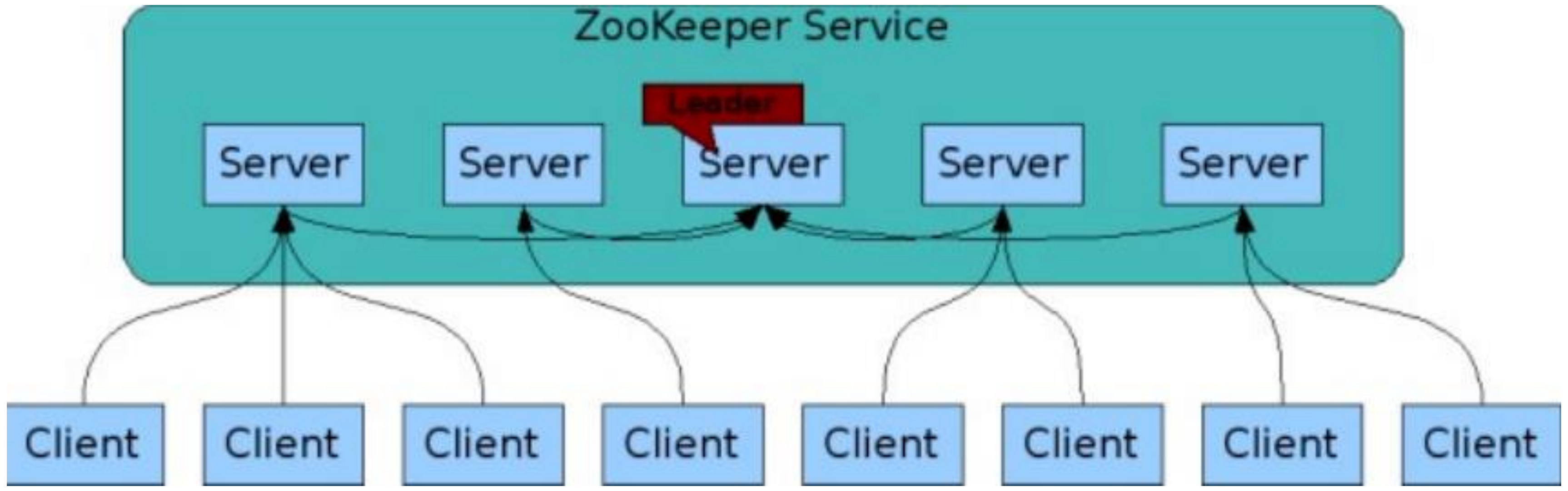
```
start java "-Dzookeeper.log.dir=%ZK_HOME%" "-Dzookeeper.root.logger=INFO,CONSOLE" -cp
"%ZK_HOME%\conf;%ZK_HOME%\*;%ZK_HOME%\lib\*" org.apache.zookeeper.server.quorum.QuorumPeerMain
"%ZK_HOME%\conf\zoo1.cfg"


start java "-Dzookeeper.log.dir=%ZK_HOME%" "-Dzookeeper.root.logger=INFO,CONSOLE" -cp
"%ZK_HOME%\conf;%ZK_HOME%\*;%ZK_HOME%\lib\*" org.apache.zookeeper.server.quorum.QuorumPeerMain
"%ZK_HOME%\conf\zoo2.cfg"


start java "-Dzookeeper.log.dir=%ZK_HOME%" "-Dzookeeper.root.logger=INFO,CONSOLE" -cp
"%ZK_HOME%\conf;%ZK_HOME%\*;%ZK_HOME%\lib\*" org.apache.zookeeper.server.quorum.QuorumPeerMain
"%ZK_HOME%\conf\zoo3.cfg"
```

- You may run the clients as usual

A Quorum


ZooKeeper Service
Leader
Server
Server
Server
Server
Server
Client
Client
Client
Client
Client
Client
Client
Client

ZK Guarentees

- Sequential Consistency - Updates from a client will be applied in the order that they were sent.
- Atomicity - Updates either succeed or fail. No partial results.
- Single System Image - A client will see the same view of the service regardless of the server that it connects to.
- Reliability - Once an update has been applied, it will persist from that time forward until a client overwrites the update.
- Timeliness - The clients view of the system is guaranteed to be up-to-date within a certain time bound.

Use Cases

- Configuration management - machines bootstrap config from a centralized source,
- facilitates simpler deployment/provisioning
- Naming service - like DNS, mappings of names to addresses
- Distributed synchronization - locks, barriers, queues
- Leader election - a common problem in distributed coordination
- Centralized and highly reliable (simple) data registry

API Access

- Include

```
<dependency>
      <groupId>org.apache.zookeeper</groupId>
      <artifactId>zookeeper</artifactId>
      <version>3.5.1-alpha</version>
</dependency>
```

- import org.apache.zookeeper.ZooKeeper;
- List Children of a Node

```
ZooKeeper zoo =  new ZooKeeper("localhost:2181",3000, null);

List<String> childs = zoo.getChildren("/",false);

for(String znode: childs)

  System.out.println(znode);

zoo.close();
```

Create, Set and Get Data from ZNode

```
ZooKeeper zk = new ZooKeeper("localhost:2181", 3000, null);

try{

      zk.create("/zktests","TESTDATA".getBytes(), Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);

}catch(Exception ex){

      //ex.printStackTrace();

      System.out.println("/ztests already exists...setting the data now...");

}

byte[] data = zk.getData("/zktests", null, null);

System.out.println(new String(data));

zk.setData("/zktests", ("This is test data set from java @ " +
                        System.currentTimeMillis()).getBytes(),-1);

zk.close();
```

Sample https://github.com/iXat-Training/Hadoop101/blob/master/23_ZooKeeperSamples/src/main/java/com/ixat/zk/_ZooKeeperSamples/SetData.java

Watcher

- implement Watcher
- Override process(WatchedEvent we) {}
- Set the watcher again in the callback

- Sample - https://github.com/iXat-Training/Hadoop101/blob/master/23_ZooKeeperSamples/src/main/java/com/ixat/zk/_ZooKeeperSamples/DataWatcher.java

Curator

- API Framework to work with ZK
- Provides utility API's for commonly used patterns in Distributed Programming
- A sample here https://github.com/iXat-Training/Hadoop101/blob/master/23_ZooKeeperSamples/src/main/java/com/ixat/zk/_ZooKeeperSamples/CuratorDataWatcher.java