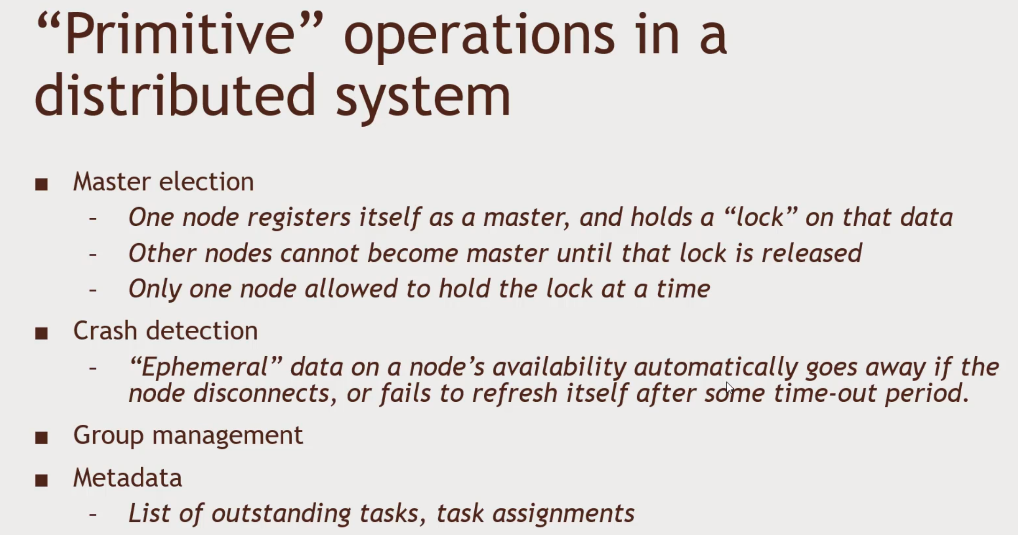
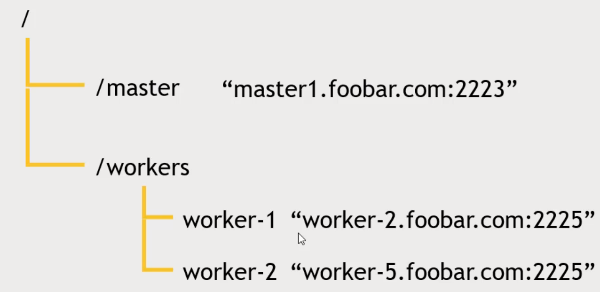
Building a system that is consistent.

Failure Mode

* Master crashes, needs to fail over to a backup
* Worker crashes – its work needs to be redistributed
* Network trouble – part of your cluster can’t see the rest of it.

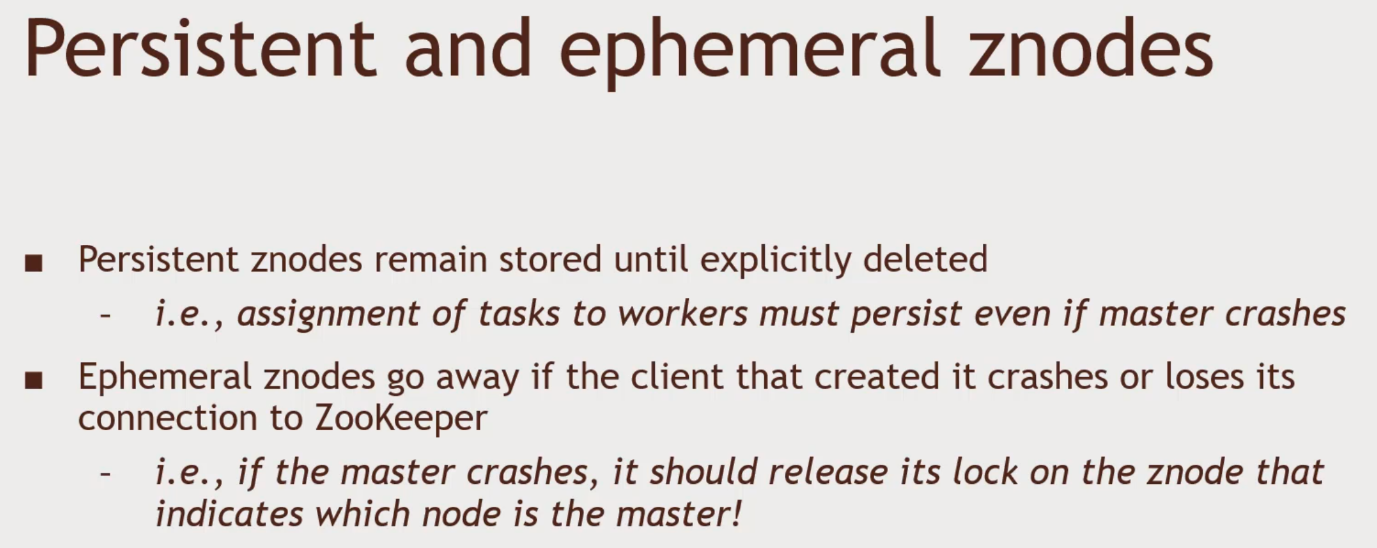
Zookeeper’s API

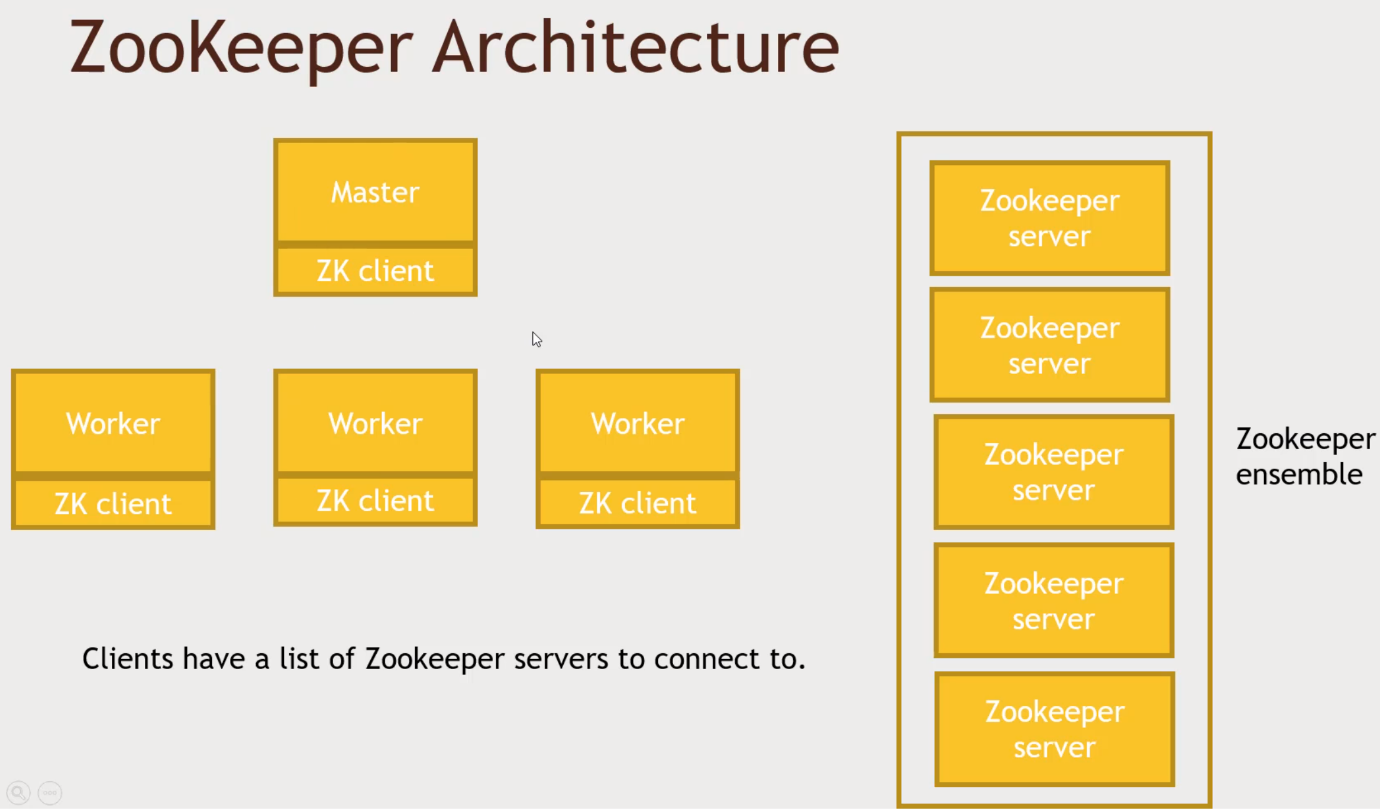
* Really a little distributed file system
  + With strong consistency guarantees
  + Replace the concept of ‘file’ with ‘’znode’ and you’ve pretty much got it
* Here’s the zookeeper API:
  + Create, delete, exits, setData, getData, getChildren

Think of it as a file system and see. Znodes as file.

Notification

* A client can register for notification on a znode
  + Avoids continuous pooling
  + Example: register for notification on/master – if it goes away, try to take over as the new master.

The assignment of tasks to workers should still persist on the workers.

ZK Client has a list of Zookeeper ensemble to keep track of multiple servers.

Ensemble is responsible for replication. Need a replication factor

Zookeeper Quorums – minimum number of servers that needs to agree on something to accept the answer. More than 1 zookeeper.

5 zookeeper server , 2 Quorums. Split brain scenario in Zookeeper. Majority of the servers. The amount of quorums needs More than half of total number of zookeeper servers.

Sounds like how MongoDB works

Zookeeper Practice. Simulating a failing master with Zookeeper

1. Hortonworks Docker Sandbox running
2. Login into Putty as Root
3. cd/usr/hdp/current/zookeeper-client/
4. cd bin
   1. you will need to code C++ and Java interfaces with Zookeeper in the real world
5. Connect to zookeeper
   1. ./zkCli.sh
   2. ‘create -e /testmaster ‘127.0.0.1:2223’ ephemeral ‘’testmaster’’ znode if we disconnect, this will go away
   3. ‘get / testmaster’
6. Quit out of zookeeper
   1. ‘quit’
   2. Zookeeper deleted ephemeral znode when you disconnected
   3. ‘get /testmaster’ and you will find that the testmaster doesn’t exist