TDDE41 Lab1, Tools for the topic

Daniel Wassing (danwa223)
Robin Andersson (roban591)

April 2019

1 Topic

We have chosen to work with the topic of Developability and Deployability (more specific research question to follow later). We believe that for a large scale architecture that these two -ilities are central to the effort of producing maintainable software and ensuring that it is capable of withstanding changes in a business environment.

2 ZooKeeper

ZooKeeper [2] is our chosen framework that we use to evaluate a research question on our selected topic. The purpose of ZooKeeper is to alleviate much developer effort from distributed services, providing a robust open-source way of handling a lot of crucial and non-trivial tasks. ZooKeeper exposes a simple set of primitives that distributed applications can build upon to achieve these non-trivial tasks. Thus, you do not have to develop your own service to handle them. In short, ZooKeeper is useful where you have multiple nodes in your cluster or infrastructure, such as a Hadoop Cluster [3].

In a distributed ZooKeeper implementation, multiple servers are present. This implementation is known as ZooKeeper's replicated mode. In this mode, one server is elected as a leader, and all additional servers are followers. If the leader for some reason fails, a new leader is elected. All servers know about each other, and every server maintains an image of server states, transaction logs and snapshots in persistent storage. Clients always connect to one specific server, but they maintain a list of possible servers to connect to, should the current connected server fail. [1]

Read operations can be performed from any ZooKeeper client to any server in the ensemble, but a write operation must go through the ZooKeeper leader and requires a majority consensus to succeed. ZooKeeper can also be run in standalone mode, with a single server and several clients, but loses the guarantee of high-availability and resilience. [1]

ZooKeeper also has a lot of consistency guarantees pertaining to sequential consistency, meaning that updates are applied in order and that they can only succeed or fail. When an updated succeed it persists until the point where a newer update is accepted. It is important to note, however, that ZooKeeper does not guarantee that every client will have identical views of ZooKeeper data at every instance in time. [1]

Distributed processes coordinated with each other using shared hierarchical namespaces, much like the UNIX and Linux file systems. Each namespace is a root node, and can have one or more child nodes. There are many co-notations to the term node, and ZooKeeper refers to them as Znodes. Data can be stored in a Znode. When data is written to or read from a Znode, all the data is written or read. Meaning that no partial read/writes can be made. Access Control Lists control who can create, read, update or delete a Znode. [1]

3 Our approach so far

We have opted to work with Java when experimenting with ZooKeeper. At this stage, we have managed to start a ZooKeeper instance, and through Java (IntelliJ IDEA) add new nodes with our custom changes to ZooKeeper. In this way we aim to undertake some tests and perform some experiments when apprehending our research question for Lab 2.

References

[1] AO DBA. Apache hadoop zookeeper - chapter 1 intro into zookeeper. `https://www.youtube.com/watch?v=gifeThkqHjg&feature=youtu.be&fbclid=IwAR2m7UK4ycE7ptnSR6cJatN5cxoPTWh4wsI1YNHQKdKOMjcYwAobmmwa0fg`, 2016. Accessed 2019-04-15.

[2] The Apache Software Foundation. Welcome to apache zookeeper. `https://zookeeper.apache.org/`, 2010. Accessed 2019-04-15.

[3] The Apache Software Foundation. Apache hadoop. `https://hadoop.apache.org/`, 2018. Accessed 2019-04-15.