# Scaling Apache Zookeeper on Microsoft Azure

Jeffrey Gensler
College of Engineering
Univeristy of Illinois at Chicago
Chicago, Illinois 60189
Email: jgensl2@uic.edu

***Abstract*—Apache Zookeeper is a distributed key/value store commonly found in enterprise Hadoop deployments. I have researched optimal provisioning strategies for Apache Zookeeper on Microsoft Azure. The provisioning strategies here are targeted at companies that are considering using Azure as a platform for a Zookeeper cluster of 3 to 7 nodes in a single geographic region.**

***Keywords*—*Azure, Zookeeper, Ansible, Scale, Cloud.***

## I. Introduction

As part of my course project in CS 441: Cloud Computing, I have hypothesized and experimented with a set of provisioning rules for Apache Zookeeper. At the time of these experiments, Azure was going through quite a bit of change in its architecture. With the change in architecture, there has been a change in APIs and SDKs supporting these platforms. In my project, I have explored as much of the emerging architecture to test my experiments. Unfortunately, not all experiments here reflect the current technology available on Azure. I am using the REST client provided by Apache Zookeeper instead of letting clients use their respective SDKs.

The first step in developing a provisioning strategy was to experiment with IaaS and learn the optimal running configuration for Zookeeper. I have started with the smallest such virtual machines (Basic A0) and have left Zookeeper's JVM size at its default setting at half of the running machines RAM.

After figuring out near-optimal settings for a three node Zookeeper cluster using Azure's IaaS, I have tried deployed Zookeeper using Azure's scalable PaaS (via Azure Classic Cloud Services). Unfortunately, Azure does not offer a platform for running Java .jar files. The work that I have done gets close to running Java .jar files. Azure's Eclipse plugin also does a poor job at mocking the target environment making development painfully slow.

### A. Techniques

To begin experiments, I have built some automation using Azure's Python SDK because Ansible's Azure module was only compatible with the old version of Azure's platform. With this automation, rapidly provisioning clusters of nodes using shared resources was easy. The architecture is as follows: one node to launch JMeter from, one node to run ELK (to collect and visualize test runs), and three nodes for the Zookeeper cluster. I have written an Ansible script to template a JMeter JMX file and test against the Zookeeper cluster. After the script runs, Logstash is restarted and the results file is read from the beginning. Logstash transforms the csv fields from string to integers and uploads the data to Elasticsearch. From here on, a dashboard is manually created to analyze the results of the JMeter run.

For the technical resources, take a look at the repository's README.

### B. Experiments

As stated above, I started the Zookeeper cluster with Basic A0 VMs. I soon found that both the ELK and JMeter VMs needed upgrading to at least Basic A1 though both benefited from over provisioning.

Columns in order: Number Threads, Number Loops, Ramptime (seconds), Time to Complete (minutes), Exhibits Abnormal Behaviour, Percent Below (Meeting) 500 ms SLA, Average Elapsed

In this first table, the data is ordered by how it appears in Kibana.

| # Th | # Lo | RT | TTC | Ab? | % SLA | AVGE |
|---|---|---|---|---|---|---|
| 25 | 200 | 60 | 2 | No | 91 | 240 |
| 40 | 150 | 60 | 17 | Yes | 24 | 3487 |
| 9 | 200 | 60 | 5 | No | 63 | 800 |
| 9 | 20 | 120 | 2 | No | 99 | 54 |
| 10 | 10 | 120 | 2 | No | 99 | 102 |
| 10 | 10 | 120 | 6 | No | 45 | 908 |
| 5 | 5 | 120 | 1 | No | 100 | 124 |
| 14 | 200 | 60 | 3 | No | 90 | 200 |
| 100 | 400 | 10 | 12 | Yes | 1 | 5677 |
| 10 | 200 | 60 | 3 | No | 92 | 201 |
| 30 | 150 | 60 | 6 | No | 60 | 834 |
| 30 | 150 | 60 | 6 | No | 81 | 355 |
| 20 | 500 | 120 | 6 | Yes | 61 | 1198 |

This next table is the same data, sorted by Average Elapsed

| # Th | # Lo | RT | TTC | Ab? | % SLA | AVGE |
|---|---|---|---|---|---|---|
| 100 | 400 | 10 | 12 | Yes | 1 | 5677 |
| 40 | 150 | 60 | 17 | Yes | 24 | 3487 |
| 20 | 500 | 120 | 6 | Yes | 61 | 1198 |
| 10 | 10 | 120 | 6 | No | 45 | 908 |
| 30 | 150 | 60 | 6 | No | 60 | 834 |
| 9 | 200 | 60 | 5 | No | 63 | 800 |
| 30 | 150 | 60 | 6 | No | 81 | 355 |
| 25 | 200 | 60 | 2 | No | 91 | 240 |
| 10 | 200 | 60 | 3 | No | 92 | 201 |
| 14 | 200 | 60 | 3 | No | 90 | 200 |
| 5 | 5 | 120 | 1 | No | 100 | 124 |
| 10 | 10 | 120 | 2 | No | 99 | 102 |
| 9 | 20 | 120 | 2 | No | 99 | 54 |

All graphs will be available until the end of my Azure subscription. URL: http://myelk-1-ip.centralus.cloudapp.azure.com, login: kibana, password: kibana

The top three times also showed very abnormal behaviour (timeout) which is probably Java crashing from the load. However, the service restarts and keeps serving requests, which goes to show the importance of services.

### *C. Reflection*

Overall, JMeter is a pretty basic tool to test out applications. Overall, the experiments that I've run showed quite a bit of fluctuation from run to run. There are too few results to conclude if Azure or Zookeeper is to blame for the abnormalities.

## II. Conclusion

Azure is a young platform. Unlike Amazon, Azure has fewer options to deploy with and far fewer integrations. It is hard to conclude any serious results as the REST API is not production ready. Either way, this project has a been a huge learning experience.