Hazelcast Certification – Code Challenge

Benchmarking & Reports

# Introduction

This document contains the description of the benchmarking scenarios and its results.

# Benchmarking restrictions

The benchmarking was performed with the following general configuration:

* **30 million** of credit cards
* **20** historical transactions per credit card
* **120** seconds of transaction generation process
* No backup

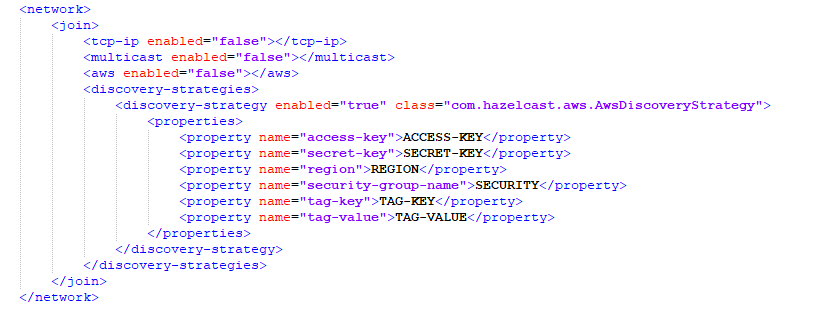
# Benchmarking setup

## Amazon EC2

The Amazon Elastic Compute Cloud (Amazon EC2) environment have been used for the benchmarking of the solution.

## Discovery configuration

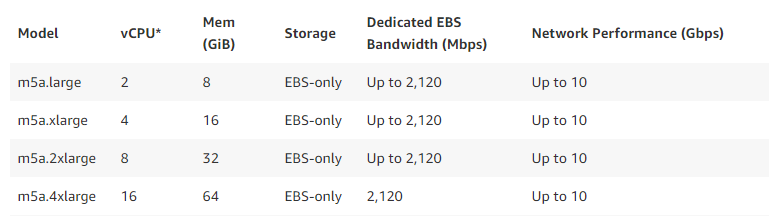
Members of the cluster and clients uses Discovery SPI as a discovery method.



1. Discovery configuration in ‘hazelcast.xml’ configuration file

## Instance types

The following instance types were used for the benchmarking of the system:



1. Technical specifications of the AWS EC2 Instance Types

# Benchmarking report

## Test set 1 - m5a.xlarge

Instance Type: m5a.xlarge

Transaction generator & fraud detection server: 1 client



1. Benchmarking results for test set 1

In this test set I tried to stablishing a relation between the number of the Executor Service’s threads and the TPS.

So the number of nodes were fixed to 8, enough to store the 30 million of credits cards and its transactions, and the number of the threads were increased gradually.

In test #010 only 1 thread were used, creating a huge bottleneck because, in practice, the detection process were executed sequentially.

As long as the number of threads were increased the TPS were increased as well, stablishing a direct relation between the number of concurrent threads submitting fraud detection operations and the TPS.

## Test set 2 - m5a.2xlarge

Instance Type: m5a.2xlarge

Transaction generator & fraud detection server: 1 client



1. Benchmarking results for test set 2

In this test set I tried to stablishing a relation between the number of nodes, the capacities of the servers and the TPS.

Analyzing tests #014 and #021 we can observe that the TPS increases a little bit (14%) when the clusters has better capacities.

However, when more members were added to the cluster (test #22) the performance went down, indicating that the power of fraud detection process overcome the capacity of the transaction generator (limited by the queue size).

## Test set 3 - m5a.4xlarge

Instance Type: m5a.4xlarge

Transaction generator & fraud detection server: 1 client



1. Benchmarking results for test set 3

In this last test set we can see again that, if the thread number is fixed, the user of the servers with more capabilities didn’t have a relevant impact in the TPS.

And, once again, we can stablish a relation between the number of threads and the TPS.

## Test set 4 - m5a.2xlarge & m5a.4xlarge

Instance Type: m5a.2xlarge & m5a.4xlarge

Transaction generator & fraud detection server: 1 client

### Result set 1 – Vertical scaling



1. Benchmarking results for test set 4 – result set 1
2. Threads and TPS correlation

Looking the results for the same AWS instance type, number of nodes, queue capacity and buffer size, we can observe that the number of threads used in the solution has a direct impact to the TPS.

It follows a logarithmic trend line because the test was limited to 32 virtual CPUs and, as soon as we overpass the number of CPUs, the performance increase slows down.

These results are very close to the test set 1 results.

### Result set 2



1. Benchmarking results for test set 4 – result set 2

Fixing the AWS instance type, number of nodes, queue capacity and number of threads, we can observe that the TPS doesn't have a significant variations when the buffer size of the futures is multiplied by 10. So we can conclude that there isn't a relation between them.

### Result set 3



1. Benchmarking results for test set 4 – result set 3

If we fixed the buffer size and multiplied the queue capacity by 10 the results are pretty much the same, because the transaction generator is way faster than de fraud detector process and doesn´t affect to the TPS.

### Result set 4 – Horizontal scaling



1. Benchmarking results for test set 4 – result set 4

If we scale out the system, adding more nodes, we can observe than the TPS has a very significant increase (29% between tests #033 & #030; and 41% between tests #037 & #036)

These results proves that the test set 2 was wrong (due to a misconfiguration of the test) and are more align with theorical capabilities of Hazelcast.