

**Collaboard Load Test Analysis**

LOAD TEST FOR COLLABOARD on THE CONFIDENTIAL CLOUD

|  |  |
| --- | --- |
| Title: | Load Test Analysis |
| Abstract: | This document details load test done for Collaboard running on the confidential cloud with Constellation |
| Status\* | Completed |
| Document Owner: | IBV |
| Document Author: | Gian Paolo Santopaolo |
| \*Draft, In progress, Reviewed, Authorized, Approved, Completed | |

|  |  |  |
| --- | --- | --- |
| **Document Control** | | |
| Document Distribution: | Internal: | IBV |
|  | External: | Heiko Reiter, Martin Muhr, Thomas Strottner, Leonard Cohnen |
| Document Authorization: | Authorizer(s): | Gian Paolo Santopaolo |
| This document is to be considered confidential | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Change History** | | | |
| Date | Ver | Changed By | Details |
| 10/02/2023 | 1.0 | G.P. Santopaolo | Initial document |
| 13/02/2023 | 1.1 | G.P. Santopaolo | Added data about the two environments tested |
| 14/02/2023 | 1.2 | A. Gorog | Document revision |

Index

[Goal 3](#_Toc127279230)

[Load test for Collaboard 3](#_Toc127279231)

[Load tests at BWI 6](#_Toc127279232)

[Environments 6](#_Toc127279233)

[Constellation 6](#_Toc127279234)

[AKS 7](#_Toc127279235)

[Test result Constellation 9](#_Toc127279236)

[Test results AKS 11](#_Toc127279237)

# Goal

This document aims to explain the concept of Collaboard's load test and to analyze the load test data coming from a cluster running on Azure Confidential Computing with Constellation and another running on Azure Kubernetes Service (AKS).

# Load test for Collaboard

Collaboard's load tests are designed to understand the limit of an environment and how many concurrent users it can tolerate by providing each connected user with a seamless user experience.

Usually, an environment with resource usage around 85-95% (CPUs, Ram, SQL Server, etc.) can still satisfy the above requirement.

To reach this result, any autoscaling possibility given by any provider is not taken into consideration.

For our load test, there are two main factors to keep in consideration:

* Number of concurrent active users
* Total requests generated by the active user

The first is a known parameter; the second can only be measured and varies from environment to environment.

To measure the total requests generated by an active user, we developed a custom telemetry, collecting all the data needed anonymously.

With such an approach, we can understand how many requests a user can generate per second.

We noticed that the RPUPS (requests per user per second) might vary significantly by the user's actions. For example, imagine a user following a workshop, passively watching the presenter.

On the other side, a user prepares a new project by adding content (images, documents, videos, links) and using all the functionalities available on the platform, which will generate way more RPUPS that the first use case.

The platform's usage, as said before, may also significantly vary by environment: in our public production environment, we have users using the platform seven days a week, 24/h a day.

At the customer's site one, users use the application five days a week, mostly eight hours a day, using Microsoft Office documents.

At customer's site two, we have multiple locations all around the world, users use the application five days a week, mostly eight hours a day (but the usage, in this case, is spread in different time zones), and they use PDFs and videos.

The number of use cases is endless.

As you can imagine, each use case generates a different load for the platform.

## Comparing performance tests with real-world usage

Our test consists of two phases:

* In the first phase, we collect anonymous usage data at the specific site
* In the second phase, we run a series of standard load tests until we reach the environment's limit

Once we collect all the needed data, we can import everything into our business intelligence (BI) and extract the estimates.

The estimates compare the total number of requests per second (RPS) the cluster can handle versus the average request per second a real user can generate using the specific environment (RPUPS).

We are able to understand the total number of the active user by time interval (usually 15 minutes)

Thanks to our telemetry, we can also understand the total HTTP, Websocket, and file operations requests in the same timeframe.

We end up with a massive table like the one below where we are reporting only a few rows to give the idea (usually, our sample is about 3000 rows, about one month's worth of samples)

Table

Description automatically generated

Sample table of the real RPUPS

This way, we know precisely the real RPUPS the environment was dealing with in any given time frame.

We are then able to aggregate the above data by the RPUPS ranges

Graphical user interface, table

Description automatically generated

Aggregated real RPUPS

This way, we know that for 1690 time frames (1690 rows of the table above, Sample table for real RPUPS) the average RPUPS of the cluster was between 0.00 and 0.10 RPUPS, average 0,0663 RPUPS.

For 876 times, we had a real RPUPS of 0,1326, and so on.

We then use K6 to perform several load tests against the environment until we reach a total cluster capacity of around 85%-95%. At the same time, the application still provides a seamless user experience (with no lag) to the user.

Once we are happy with the result, we choose one of the load tests as a reference and import the data collected inside our BI.

A K6 test has a maximum length of 30 minutes, to which, for the purpose of our tests, we have to remove the ramp-up and cool-down time. So usually, we have about 20 minutes of usable data.

With all the data collected by K6, we are primarily interested in the Web APIs logs, web sockets, and file logs that we can resume after all the aggregation and manipulation in the table below.

Graphical user interface, text, application

Description automatically generated

Data collected from K6

From the table above, we understand that during the test, the cluster was tested with 472.136 API calls, 294.295 web sockets calls, and 386.611 file operations requests, which makes the cluster capable of 382 RPS (request per second).

Now for the last part, we have only to match the RPS (request per second) with the RPUPS (request per user per second). This way, we will be able to understand how many users the cluster will be able to handle.

As said before, the number of RPUPS (request per user per second) varies and, based on it, also changes the number of concurrent users a cluster can sustain before becoming less responsive, which for a real-time application like Collaboard is not good.

Based on the anonymous data sampled on the environment, we already know how the users of the cluster behave, so we can make some predictions that are resumed in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| RPUPS range | Samples count | RPUPS avg | Estimated max online users |
| 0.00 – 0.10 | 912 | 0.0663 | 5.759 |
| 0.10 – 0.20 | 135 | 0.1236 | 3.091 |
| 0.20 – 0.30 | 7 | 0.2318 | 1647 |
| 0.30 - 0.40 | 2 | 0.3140 | 1216 |
| 0.40 – 0.50 | 4 | 0.4405 | 867 |

Max online users' estimation

In most sample cases, the cluster's users produced between 0.00 and 0.10 requests per user per second.

In this scenario, the cluster will be able to handle, with no problem, up to 5.759 concurrent active users.

In the second case, observed in 135 samples, the cluster will be able to handle, with no lag, up to 3.091 users.

Between 0.30 and 0.40 RPUPS, the cluster will be able to handle 1216 users.

And as the last sample, between 0.40 and 0.50 RPUPS, the cluster will be able to handle 867 simultaneous users.

These are the facto data results made with math and statistic calculations.

The engineering team shall discuss them with the customer to understand its needs and how to make benefit from them.

# Load tests at BWI

Of course, we could not collect the anonymous usage data for the experiment with BWI.

For the purpose of the experiment, we used a set of anonymous data collected from one of our production environments between December 2022 and January 2023

# Environments

For the purpose of the test, we created, inside the BWI Azure subscription, two Kubernetes clusters, the more similar the possible.

Both clusters used SQL Server and Serids as a service and Azure Blob Storage.

The pods were also configured with the same Horizontal Pod Autoscaling (HPA) settings

## Constellation

This cluster was running with:

* Master node, virtual machine type: Standard\_DC8as\_v5 - 8vCPU - 32GB Ram
* 3 Worker Node Standard\_DC4as\_v5 - 4 vCPU - 16GB RAM.
* SQL Server Business Critical 10 vCPU
* Redis 2.5Gb
* Azure Blob Storage

HPA Configuration:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Targets | Min Pods | Max Pods |
| api | 70% | 2 | 20 |
| auth | 70% | 2 | 10 |
| copyporject | 70% | 1 | 3 |
| copytile | 70% | 1 | 3 |
| cprsanywere | 70% | 1 | 3 |
| fileconverter | 70% | 1 | 2 |
| imageresizer | 70% | 2 | 6 |
| licensing | 70% | 2 | 3 |
| onlineuserinspector | 70% | 1 | 3 |
| proxy | 70% | 2 | 10 |
| real-time | 75%/80% | 4 | 30 |

HPA configuration

The table above details the HPA configuration. The column “Name” indicates the pod.

The column “Targets” indicates the targets upon which the orchestrator will create a new instance of the pod. When it is a single value, it means only CPU. So in the case of the first row, the orchestrator will create a new instance of the pod once the pod running reaches 70% of the usage of the assigned resource.

In the last row, we have two percentages: CPU and memory.

The column “Min Pods” indicates the minimum number of pods the orchestrator will instantiate.

The column “Max Pods” indicates the maximum number of pods the orchestrator will instantiate.

The configuration above is a simplified version of the real HPA configuration running in Kubernetes to give some context to the reader.

If you need further information, our Kubernetes engineer will be happy to answer any of your questions.

## AKS

This cluster was running with:

* Master node on AKS
* 3 Worker Node, virtual machine type: Standard\_D4as\_v5 - 4 vCPU - 16GB RAM.
* SQL Server Business Critical 10 vCPU
* Redis 2.5Gb
* Azure Blob Storage

HPA Configuration:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Targets | Min Pods | Max Pods |
| Api | 70% | 2 | 20 |
| Auth | 70% | 2 | 10 |
| copyporject | 70% | 1 | 3 |
| Copytile | 70% | 1 | 3 |
| cprsanywere | 70% | 1 | 3 |
| fileconverter | 70% | 1 | 2 |
| imageresizer | 70% | 2 | 6 |
| licensing | 70% | 2 | 3 |
| onlineuserinspector | 70% | 1 | 3 |
| Proxy | 70% | 2 | 10 |
| real-time | 75%/80% | 4 | 30 |

HPA configuration

The table above details the HPA configuration. The column “Name” indicates the pod.

The column “Targets” indicates the targets upon which the orchestrator will create a new instance of the pod. When it is a single value, it means only CPU. So in the case of the first row, the orchestrator will create a new instance of the pod once the pod running reaches 70% of the usage of the assigned resource.

In the last row, we have two percentages: CPU and memory.

The column “Min Pods” indicates the minimum number of pods the orchestrator will instantiate.

The column “Max Pods” indicates the maximum number of pods the orchestrator will instantiate.

The configuration above is a simplified version of the real HPA configuration running in Kubernetes to give some context to the reader.

If you need further information, our Kubernetes engineer will be happy to answer any of your questions.

Note that the virtual machines between the two clusters are identical even though the VMs configured for Constellation were “Confidential” VMs versus the VMs for the AKS cluster that were standard Azure VMs.

The difference can also be found in the VM description Standard\_D***C***8as\_v5 vs. Standard\_D4as\_v5, where the “***C***” stands for “Confidential”.

# Test result Constellation

For the test on the Constellation cluster, we were able to perform a loaf test with K6 configured to run with 600 Virtual Users (VUs) max.

Below an overview of how the tets went from a K6 point of view

Graphical user interface, chart

Description automatically generated

Overview of the K6 test

Out of the 30-minute tests, we could use 23 minutes of data after removing the ramp-up and cool-down period.

The cluster during the 23 minutes of stress was capable of handling:

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP Req. total | WS Req. total | File Ops total | Total RPS |
| 11.333 | 652.563 | 64.738 | 527.00 |

Comparing the RPS data coming from the load test with the cluster usage from one of our production environments (as mentioned in the chapter above), we are now able to understand the maximum number of concurrent users the cluster will be able to serve by different scenarios:

Graphical user interface, table

Description automatically generated

An extended overview of the max number of users by RPUPS range

Below is a simplified overview of the table above

Table

Description automatically generated

A condensed overview of the max number of users by RPUPS range

# Test results AKS

Graphical user interface

Description automatically generated with medium confidence

Overview of the K6 test

For the test on the Constellation cluster, we were able to perform a loaf test with K6 configured to run with 600 Virtual Users (VUs) max.

Out of the 30-minute tests, we could use 21 minutes of data after removing the ramp-up and cool-down period.

The cluster during the 23 minutes of stress was capable os handling:

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP Req. total | WS Req. total | File Ops total | Total RPS |
| 18.667 | 1.083.442 | 99.184 | 953.00 |

For the cluster, we observed the following behavior regarding memory and CPU usage:

A picture containing table

Description automatically generated

Overview of cluster usage memory/CPU

We also observed the following behavior for the pod instances:

Table

Description automatically generated

Overview of cluster pod instances

Comparing the RPS data coming from the load test with the cluster usage from one of our production environments (as mentioned in the chapter above), we are now able to understand the maximum number of concurrent users the cluster will be able to serve by different scenarios:

Graphical user interface

Description automatically generated

An extended overview of the max number of users by RPUPS range

Below is a simplified overview of the table above

Table

Description automatically generated

A condensed overview of the max number of users by RPUPS range