

**Collaboard Install**

Performance TESTS

|  |  |
| --- | --- |
| Title: | Collaboard performance tests |
| Abstract: | This document details performance tests for CollaBoard and specifies the sizing recommendation |
| Status\* | Draft |
| Document Owner: | IBV |
| Document Author: | Gian Paolo Santopaolo |
| \*Draft, In progress, Reviewed, Authorized, Approved, Completed | |

|  |  |  |
| --- | --- | --- |
| **Document Control** | | |
| Document Distribution: | Internal: | All dev Teams |
|  | External: |  |
| Document Authorization: | Authorizer(s): | Gian Paolo Santopaolo |

|  |  |  |  |
| --- | --- | --- | --- |
| **Change History** | | | |
| Date | Ver | Changed By | Details |
| 22/12/2020 | 1.0 | GP Santopaolo | Initial Draft |
| 23/12/2020 | 1.1 | GP Santopaolo | Added load test results |
| 04/0182021 | 1.2. | GP Santopaolo | Added real data usage |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Index

[Overview 3](#_Toc60669644)

[Introduction 4](#_Toc60669645)

[Performance Test 4](#_Toc60669646)

[Load Test 4](#_Toc60669647)

[Stress Testing 4](#_Toc60669648)

[Performance test 6](#_Toc60669649)

[Performance Test Case 6](#_Toc60669650)

[Performance Test Run 1 – Azure 125 Concurrent Users 9](#_Toc60669651)

[Environment 9](#_Toc60669652)

[Test results 9](#_Toc60669653)

[Conclusion 11](#_Toc60669654)

[Performance Test Run 2 – Azure 200 Concurrent Users 12](#_Toc60669655)

[Environment 12](#_Toc60669656)

[Test Results 12](#_Toc60669657)

[Conclusion 14](#_Toc60669658)

[Performance Test Run 3 – Azure 200 Concurrent Users 15](#_Toc60669659)

[Environment 15](#_Toc60669660)

[Test Results 15](#_Toc60669661)

[Conclusion 17](#_Toc60669662)

[Real-world data analysis 18](#_Toc60669663)

[Performance test vs. the real world 18](#_Toc60669664)

[Comparing performance test with real-world usage 18](#_Toc60669665)

[Total users vs. concurrent users 19](#_Toc60669666)

# Overview

This document details the performance test and sizes recommendation for customers who want to apply to their environment.

The document aims to compare performance test data against real user usages to determine the appropriate environment sizing based on the total number of users.

Our engineers can answer further questions.

# Introduction

This chapter aims to define the difference between load tests and other types of tests and which test is suitable for which situation.

## Performance Test

Performance testing is the general name for tests that check how the system behaves and performs. Performance testing examines responsiveness, stability, scalability, reliability, speed, and resource usage of the software and infrastructure. Different types of performance tests provide additional data.

Before performance testing, it's essential to determine the system's business goals, so we can tell if your system behaves satisfactorily or not according to your customers' needs.

After running performance tests, we can analyze different KPIs, such as the number of virtual users, hits per second, errors per second, response time, latency, and bytes per second (throughput) and the correlations between them. We can identify bottlenecks, bugs, and errors through the reports, then decide what needs to be done.

## Load Test

A load test is a type of performance test that checks how systems function under a hefty number of concurrent users performing transactions over a certain period of time. In other words, the test measures how systems handle heavy load volumes. There are a few types of open-source load testing tools, JMeter being the most popular.

Load tests are useful to determine how many users the system can handle. We can configure tests to simulate various user scenarios that can focus on different parts of your system (such as a checkout page, for example). We can determine how the load behaves when coming from different geo-locations or how the load might build-up, then level out to a sustained level. Load tests should be performed all the time to ensure your system is always on point, which is why it should be integrated into continuous integration cycles

## Stress Testing

A stress test is a type of performance test that checks the system's upper limits by testing it under extreme loads. Stress tests examine how the system behaves under intense loads and how it recovers when going back to normal usage. Are the KPIs like throughput and response time the same as before the spike in load? Stress tests also look for memory leaks, slowdowns, security issues, and data corruption.

Stress testing can be conducted through load testing tools by defining a test case with a very high number of concurrent virtual users.

Just as a stress test is a type of performance test, there are types of load tests. If the stress test includes a sudden, high ramp-up in the number of virtual users, it is called a Spike Test. If we stress test for an extended period to check the system's sustainability over time with a slow ramp-up, it's called a Soak Test.

Run stress tests against the website or app before major events, like Black Friday, ticket selling for a popular concert with high demand, or elections. We recommend stress testing every once in a while, so we know our system's endurance capabilities. This ensures you're always prepared for unexpected traffic spikes and gives you more time and resources to fix your bottlenecks.

# Performance test

## Performance Test Case

Every test we performed was supposing one user to perform the following actions

Each SignalR action was performed with a one-second delay.

Each Web API request was performed with a three seconds delay.

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Action** | **Action Type** |
| Login | GetAuthenticatedMode | Web API |
|  | Authenticate | Web API |
|  | AssertUser | Web API |
|  | GetAuthenticatedUser | Web API |
|  | PostLoginProject | SignalR |
| PostNew sticky | PostNew | SignalR |
|  | PostUnlock | SignalR |
| PostNew small ink | PostNew | SignalR |
| PostNew sticky2 | PostNew | SignalR |
|  | PostUnlock | SignalR |
| PostNew small ink 2 | PostNew | SignalR |
| PostNew shape | PostNew | SignalR |
|  | PostUnlock | SignalR |
| PostNew and Post Update | PostNew | SignalR |
|  | PostPropertyUpdated (fill) | SignalR |
|  | PostUnlock | SignalR |
| PostNew and Post Update | PostNew | SignalR |
|  | PostPropertyUpdated (content) | SignalR |
|  | PostPropertyUpdated (position) | SignalR |
|  | PostUnlock | SignalR |
| PostNew Connector | PostNew | SignalR |
|  | PostUnlock | SignalR |
|  | PostNew | SignalR |
|  | PostTileRelationChanged | SignalR |
|  | PostUnlock | SignalR |
| PostNew big ink | PostNewBigInk | Web API |
| PostNew shapes, multi-select, updates | PostNew shape 1 | SignalR |
|  | PostUnlock shape 1 | SignalR |
|  | PostNew shape 2 | SignalR |
|  | PostUnlock shape2 | SignalR |
|  | PostNew shape 3 | SignalR |
|  | PostUnlock shape 3 | SignalR |
|  | PostNew shape 4  PostUnlock shape 4 | SignalR |
|  | PostLock shape 1 | SignalR |
|  | PostLock shape 2 | SignalR |
|  | PostLock shape 3 | SignalR |
|  | PostLock shape 4 | SignalR |
|  | PostPropertyUpdated (position) | SignalR |
|  | PostUnlock multi selection | SignalR |
| Reset pwd, other actions | PostNew | SignalR |
|  | PostPropertyUpdated (border color) | SignalR |
|  | PostPropertyUpdated (fill color) | SignalR |
|  | PostPropertyUpdated (position) | SignalR |
|  | PostPropertyUpdated (undo update position) | SignalR |
|  | PostUnlock | SignalR |
| Add vote, remove vote | PostNew text | SignalR |
|  | PostTileVotingChanged (add vote) | SignalR |
|  | PostTileVotingChanged (remove votes) | SignalR |
|  | PostUnlock text | SignalR |
| PostNew shape, update, and undo | "PostNew shape | SignalR |
|  | PostPropertyUpdated (border color) | SignalR |
|  | PostPropertyUpdated (fill color) | SignalR |
|  | PostPropertyUpdated (position) | SignalR |
|  | PostPropertyUpdated (undo position) | SignalR |
|  | PostUnlock | SignalR |
| PostNew, multiple updates | PostNew text | SignalR |
|  | PostPropertyUpdated (content) | SignalR |
|  | PostPropertyUpdated (font) | SignalR |
|  | PostPropertyUpdated (scale) | SignalR |
|  | PostPropertyUpdated (rotation) | SignalR |
|  | PostUnlock | SignalR |
| PostNew sticky, multiple updates | PostNew sticky | SignalR |
|  | PostPropertyUpdated (content) | SignalR |
|  | PostPropertyUpdated (background color) | SignalR |
|  | PostUnlock | SignalR |
| PostNew 5 sticky notes, duplicate | PostNew 5 sticky notes | SignalR |
|  | CopyTiles | Web API |
|  | PostPropertyUpdated | SignalR |
|  | PostUnlock (multi-selection) | SignalR |
|  | GetCopiedTilesInfo | Web API |
|  | PostPropertyUpdated | SignalR |
|  | PostUnlock (copied tiles) | SignalR |
| PostNew 10 sticky notes, group | PostNew (5 sticky notes) | SignalR |
|  | PostUnlock (multi-selection) | SignalR |
|  | PostNew (5 sticky notes) | SignalR |
|  | PostUnlock (multi-selection) | SignalR |
|  | PostLock (all) | SignalR |
|  | PostUndropped | SignalR |
|  | PostNew (group) | SignalR |
|  | PostDropped | SignalR |
|  | PostUnlock (all tiles) | SignalR |
|  | PostUnlock (group) | SignalR |
| Upload pdf | PostNew (pdf) | SignalR |
|  | Upload file | Web API |
|  | PostUploaded | SignalR |
|  | PostUnlock | SignalR |
| Upload ppt | PostNew (ppt) | SignalR |
|  | Upload file | Web API |
|  | PostUploaded | SignalR |
|  | PostUnlock | SignalR |
| Upload video | PostNew (video) | SignalR |
|  | Upload file | Web API |
|  | PostUploaded | SignalR |
|  | PostUnlock | SignalR |
| Upload image | PostNew (image) | SignalR |
|  | Upload file | Web API |
|  | PostUploaded | SignalR |
|  | PostUnlock | SignalR |
| Upload image 2 | PostNew (image 2) | SignalR |
|  | Upload file | Web API |
|  | PostUploaded | SignalR |
|  | PostUnlock | SignalR |

## Performance Test Run 1 – Azure 125 Concurrent Users

We performed the test on an Azure environment defined as below.

The test was performed with 125 concurrent users, each performing the actions described in the paragraph "Performance Test Case."

### Environment

We performed the test against an azure environment.

Web App running on

* AppService P1v2 (210 total ACU – 3.5 GB memory) (West Europe)
* AppService P1v2 (210 total ACU – 3.5 GB memory) (West US)
* AppService S1 (100 total ACU – 1.75 GB memory) (Est Asia)

Web APIs running on

* AppService P3v1 (840 total ACU – 14 GB memory) (West Europe)
* AppService P3v1 (840 total ACU – 14 GB memory) (West US)
* AppService S1 (100 total ACU – 1.75 GB memory) (Est Asia)

State machines (all)

* AppService P3v1 (840 total ACU – 14 GB memory) (West Europe)

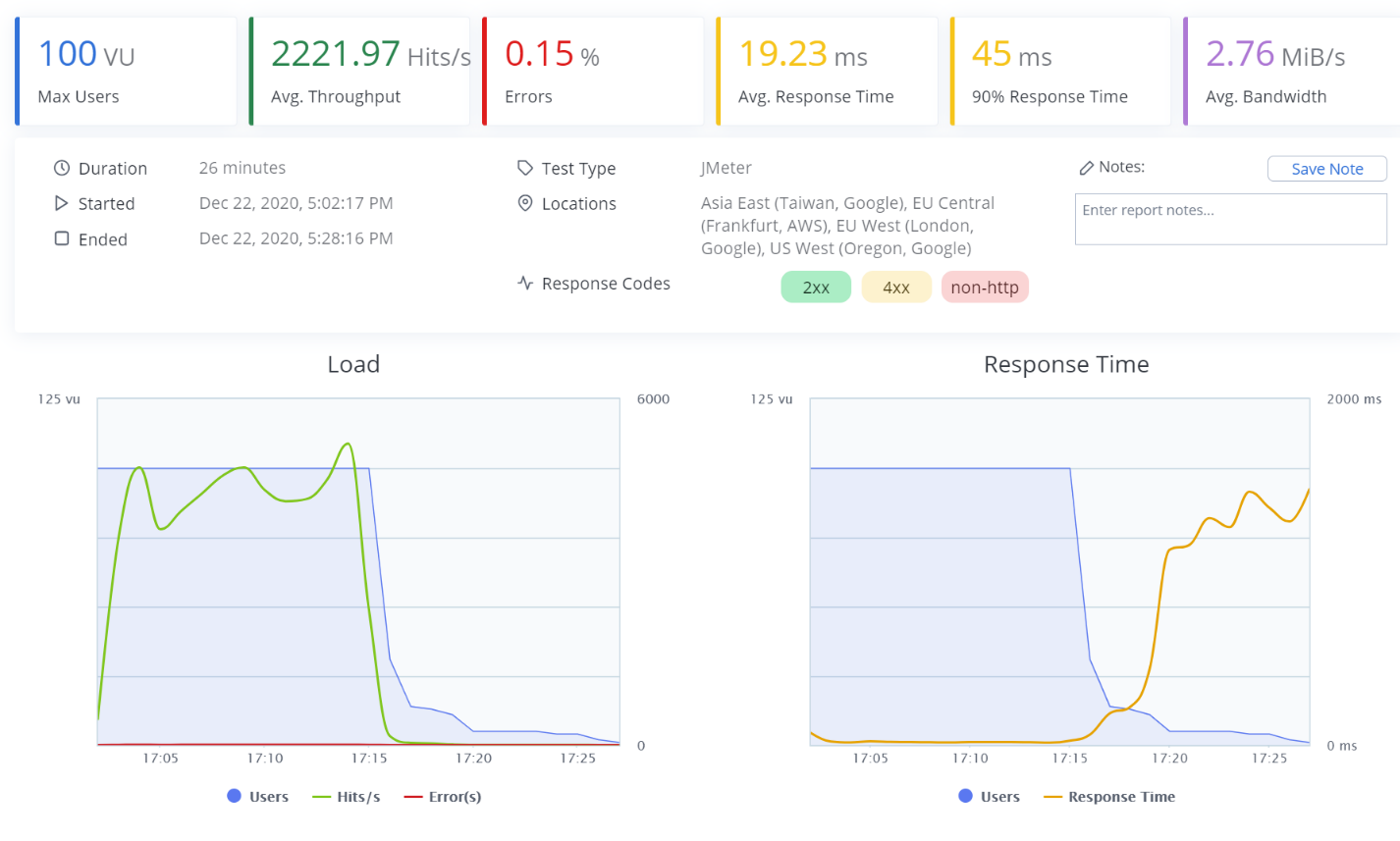
Redis Cache – Standard 2.5GB (West Europe)

SQL Server Premium P1 125 DTU (West Europe)

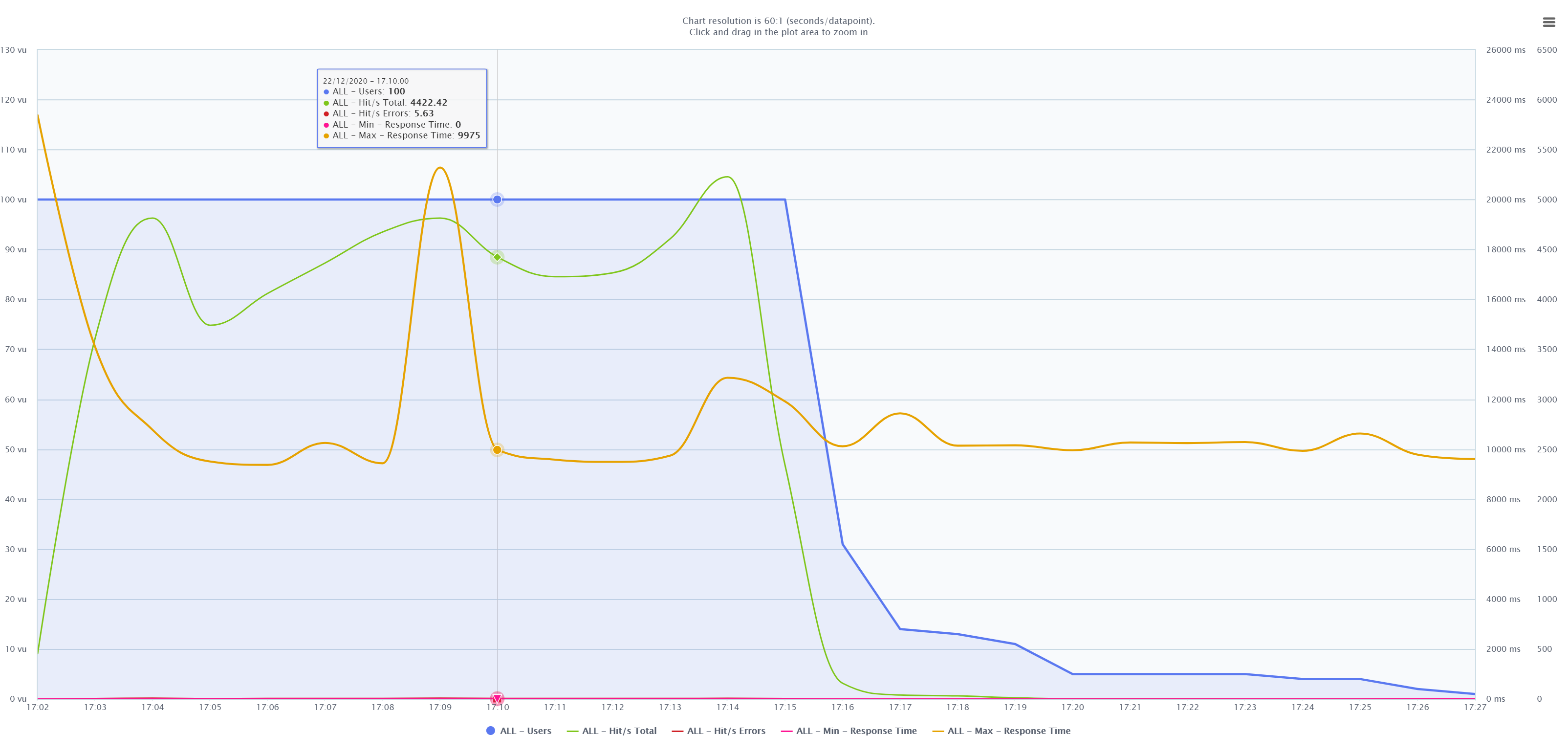
Storage Account StorageV2 (general purpose v2) Zone-redundant storage (ZRS) (West Europe)

### Test results

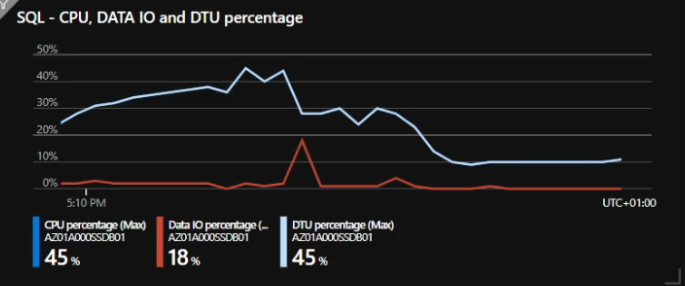
Summary



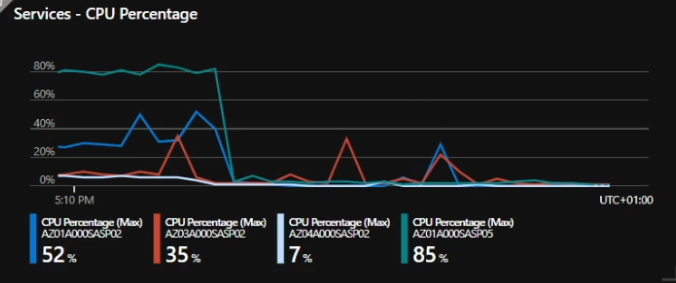
Timeline report



SQL Server usage over time



Services CPU usage over time



### Conclusion

With this test, none of the resources reached their limit

## Performance Test Run 2 – Azure 200 Concurrent Users

We performed the test on an Azure environment defined as below.

The test was performed with 200 concurrent users, each performing the actions described in the paragraph "Performance Test Case."

### Environment

We performed the test against an azure environment.

Web App running on

* AppService P1v2 (210 total ACU – 3.5 GB memory) (West Europe)
* AppService P1v2 (210 total ACU – 3.5 GB memory) (West US)
* AppService S1 (100 total ACU – 1.75 GB memory) (Est Asia)

Web APIs running on

* AppService P3v1 (840 total ACU – 14 GB memory) (West Europe)
* AppService P3v1 (840 total ACU – 14 GB memory) (West US)
* AppService S1 (100 total ACU – 1.75 GB memory) (Est Asia)

State machines (all)

* AppService P3v1 (840 total ACU – 14 GB memory) (West Europe)

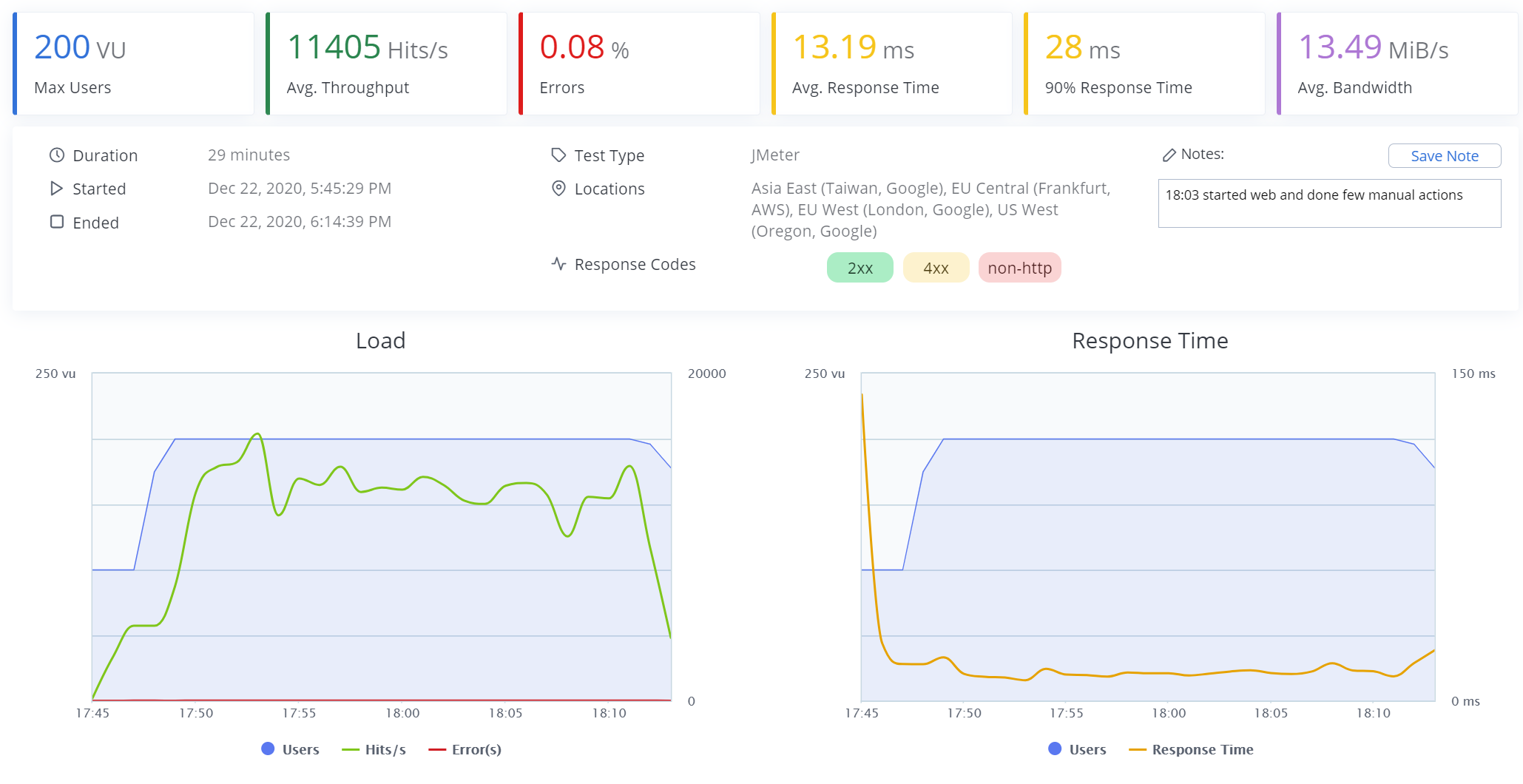
Redis Cache – Standard 2.5GB (West Europe)

SQL Server Premium P1 125 DTU (West Europe)

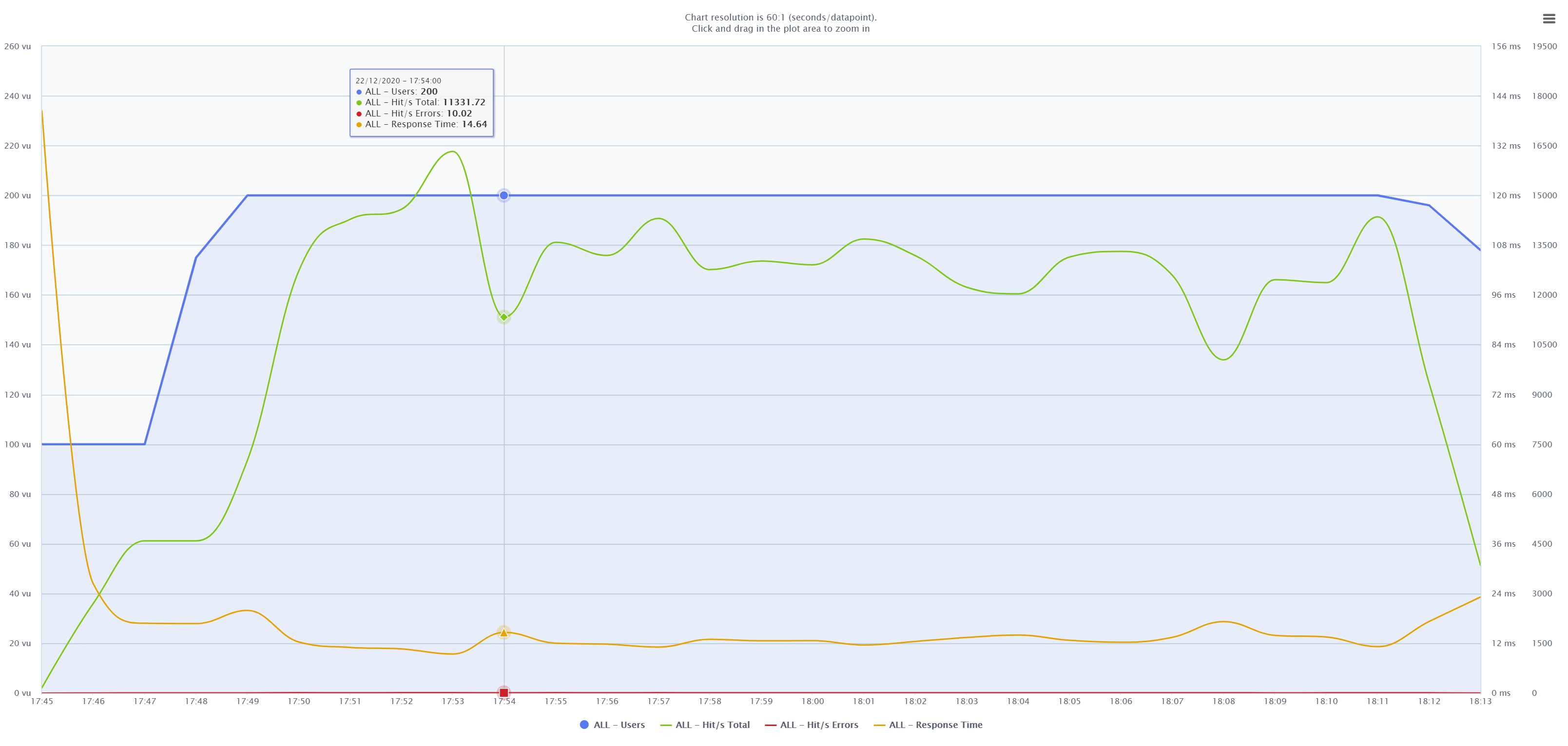
Storage Account StorageV2 (general purpose v2) Zone-redundant storage (ZRS) (West Europe)

### Test Results

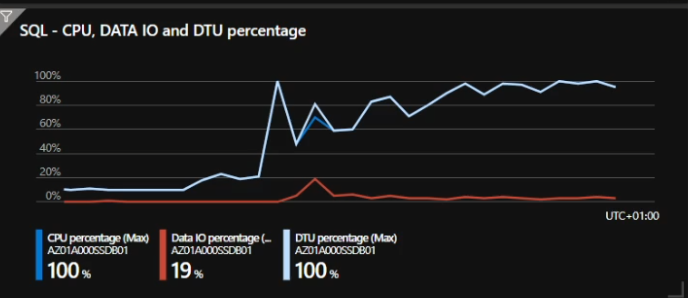
Summary



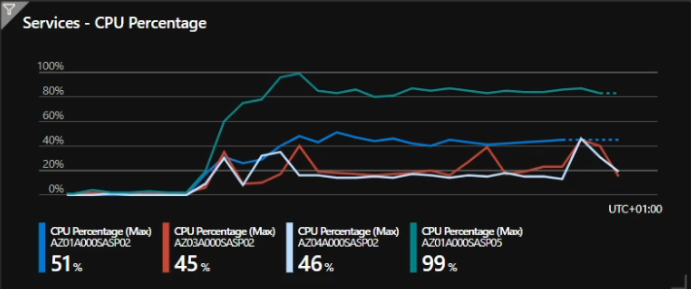
Timeline report



SQL Server usage over time



Services CPU usage over time



### Conclusion

With this test, we noticed a peak in the CPU usage for the state machine server. Even though a peek in this server does not mean a slow down of the UI, for such load, the recommendation is to split state machines into different servers.

The peak for the SQL server means that the machines reched its limint, there was no error during the peak time, thoug, so all requests were served correctly.

The recommendation here is to move to a plan a bit better in terms of CPU and IOPS.

## Performance Test Run 3 – Azure 200 Concurrent Users

We performed the test on an Azure environment defined as below.

The test was performed with 200 concurrent users, each performing the actions described in the paragraph "Performance Test Case."

### Environment

We performed the test against an azure environment.

Web App running on

* AppService P1v2 (210 total ACU – 3.5 GB memory) (West Europe)
* AppService P1v2 (210 total ACU – 3.5 GB memory) (West US)
* AppService S1 (100 total ACU – 1.75 GB memory) (Est Asia)

Web APIs running on

* AppService P3v1 (840 total ACU – 14 GB memory) (West Europe)
* AppService P3v1 (840 total ACU – 14 GB memory) (West US)
* AppService S1 (100 total ACU – 1.75 GB memory) (Est Asia)

State machines (all)

* AppService P3v1 (840 total ACU – 14 GB memory) (West Europe)

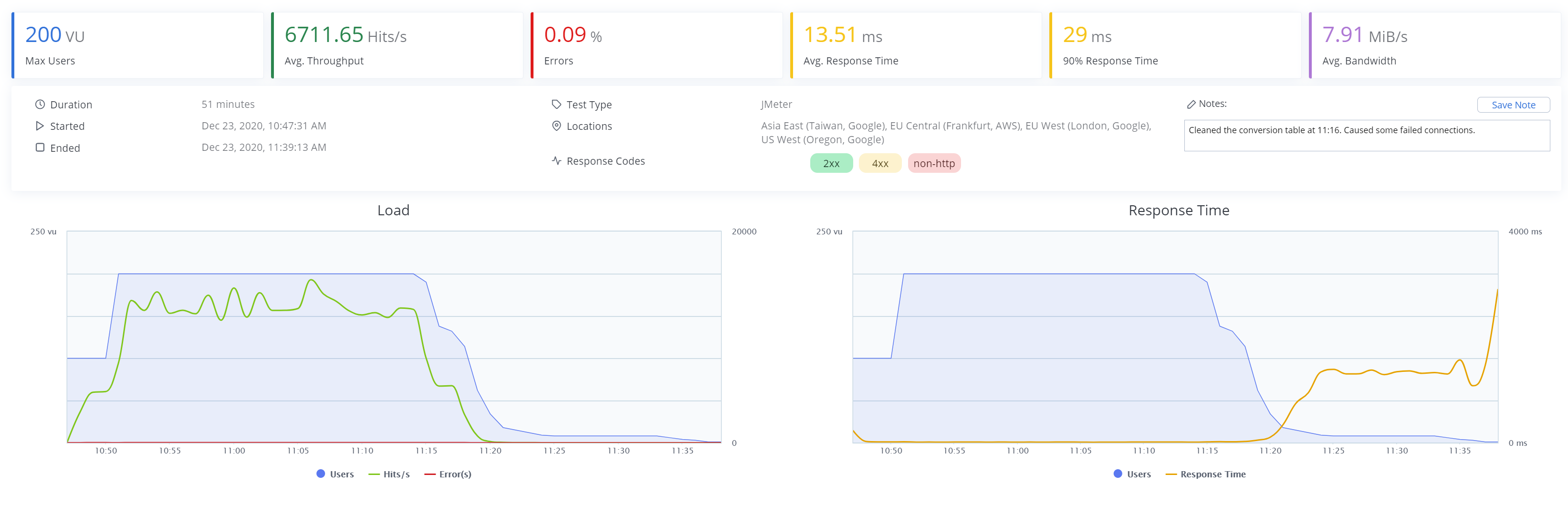
Redis Cache – Standard 2.5GB (West Europe)

SQL Server Gen5 Bunisness Critical 2 vCore (West Europe)

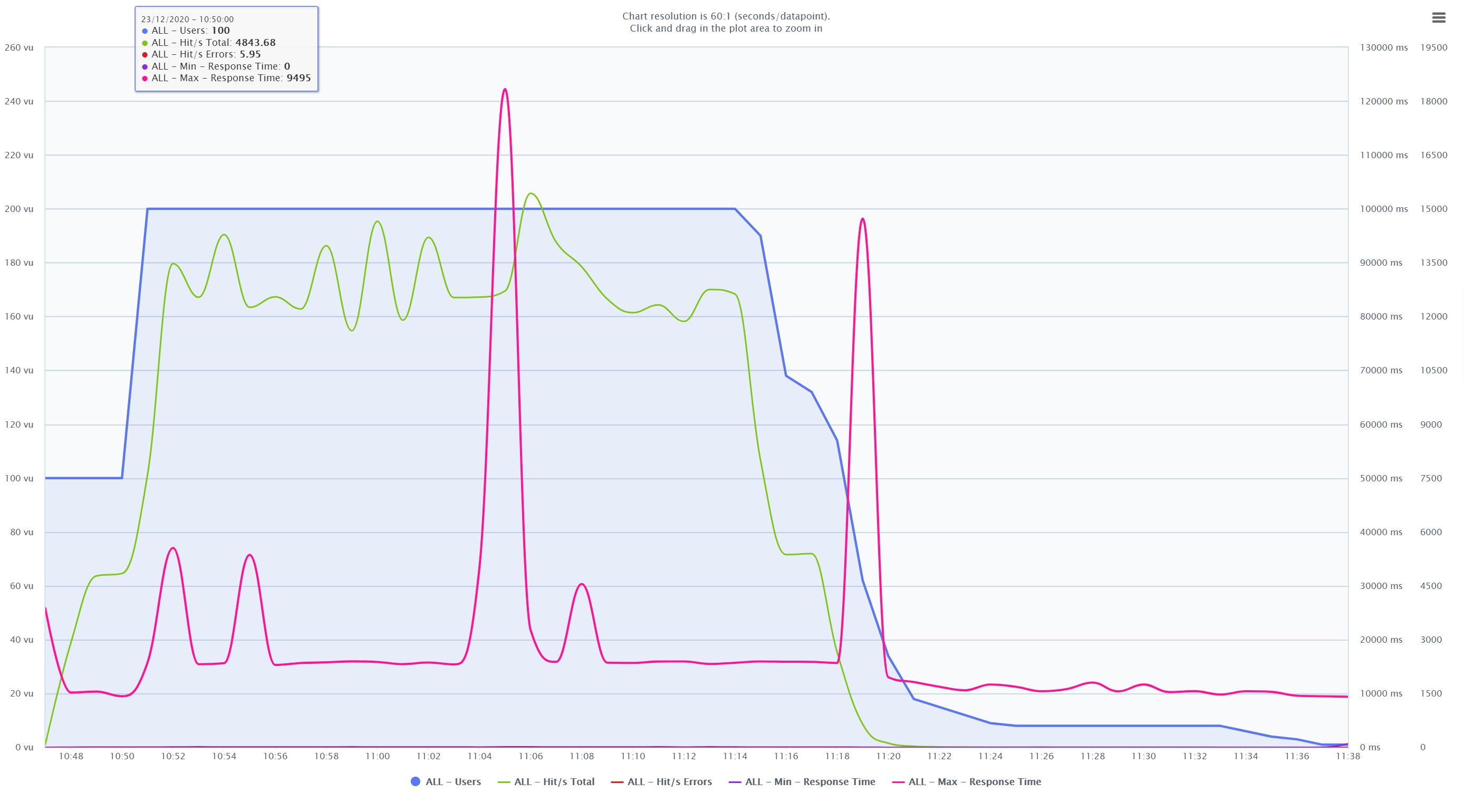
Storage Account StorageV2 (general purpose v2) Zone-redundant storage (ZRS) (West Europe)

### Test Results

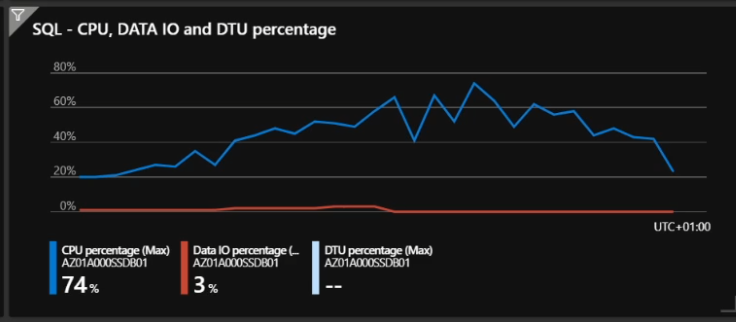
Summary



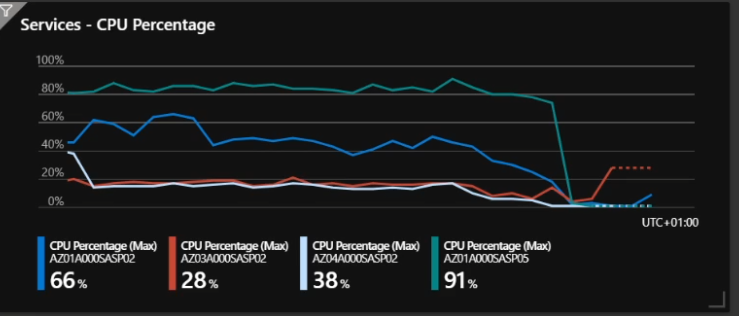
Timeline report



SQL Server usage over time



Services CPU usage over time



### Conclusion

The only difference between this run and Test Run 2 was the SQL server. We upgraded from Premium P1 125 DTU to Gen5 Bunisness Critical 2 vCore.

CPU performance is pretty comparable (1.75 to 2), but we noticed a noticeable increase in IOPS that leads the whole environment to work seamlessly.

With this test, none of the resources reached their limit

# Real-world data analysis

By analyzing the data from our production environment, we got the following data usage

|  |  |
| --- | --- |
| 100 logged in Users |  |
| Open Projects | 40 |
| Max Users per Project | 20 |
| Avg Users per Project | 3 |
| Max Tiles per Project | 1500 |
| Avg Tiles per Project | 200 |
| Adding Tiles in the same minute | 10 |
| Changing/Moving Tiles in the same minute | 10 |

# Performance test vs. the real world

We have done a series of performance tests against our application to determine the bottleneck and environment limits in terms of resources.

We also analyzed real-world scenarios and application usages based on our production environment.

## Comparing performance test with real-world usage

On the performance test (run 2 and 3), we got 200 users performing one SignalR requests every second, translating into 200 requests per second.

In the real world, we got an average of 20 actions per minute. Since one action is made of several requests, we can safely say that we got 200 requests per minute or 3.4 requests per second.

Based on these assumptions, we can say that an environment with the following sizing:

Web App running on

* AppService P1v2 (210 total ACU – 3.5 GB memory) (West Europe)
* AppService P1v2 (210 total ACU – 3.5 GB memory) (West US)
* AppService S1 (100 total ACU – 1.75 GB memory) (Est Asia)

Web APIs running on

* AppService P3v1 (840 total ACU – 14 GB memory) (West Europe)
* AppService P3v1 (840 total ACU – 14 GB memory) (West US)
* AppService S1 (100 total ACU – 1.75 GB memory) (Est Asia)

State machines (all)

* AppService P3v1 (840 total ACU – 14 GB memory) (West Europe)

Redis Cache – Standard 2.5GB (West Europe)

SQL Server Gen5 Bunisness Critical 2 vCore (West Europe)

Storage Account StorageV2 (general purpose v2) Zone-redundant storage (ZRS) (West Europe)

Will be capable of handling 5.880 concurrent users (100:3.4=x:200).

## Total users vs. concurrent users

TBD extract stats about concurrent users versus the total number of registered users