uFlexi Technical Architecture Addendum:

AWS Platform Project

August 2018

Authors

Polymorphism: Tom Baldwin, Karen Miller, Vic Luscombe, Cari Morton, Tunde Adeola.

Polymorphism Limited

Electric Works, Sheffield S1 2BJ

+44 (0) 114 286 6356

www.polymorphism.co.uk

polymorphism
enterprise software development

Introduction

The *uFlexi* demo and test environments currently sit on a single *Bytemark* server. In order to support increased capacity and to provide a more reliable service in preparation for the US launch early in 2019, the environments were transferred to an *Amazon Web Services* (*AWS*) platform.

AWS provides a cost effective and reliable way to manage infrastructure; replacing the need for an in-house operations team and expensive hardware, and allowing users to pay only for the services they consume. Scaling up and down is easily achieved as demand dictates.

Project Summary

A development environment has been set up called `dev-uflexi.com` running on *AWS*, utilising: *CloudFront*, *Elastic Beanstalk (EB)*, *S3* (for file storage), *Route 53* and *RDS*. An additional worker environment runs cron jobs for `dev-uflexi`. Test-uflexi.com is set up in similar fashion to `dev-uflexi`. Demo-uflexi.com remains on the *Bytemark* servers ready to be moved to *AWS* before launch.

In addition to porting the *uFlexi* environments to *AWS*, changes to the *uFlexi* software were necessary; improvements were made to the security of user images and documents, and to provide enhanced password hashing. Furthermore, changes were required to the application's web context configuration to work on the *AWS EB* servers.

Performance testing was carried out using *BlazeMeter* to identify functionality that may struggle to perform under high load and to tune the *AWS* services so they react in a reliable way under variations in capacity.

Read-only replica database instances have been used to reduce load on the main database instances: Reports, 'Make a Booking' and 'Seller availability' now read from the replica rather than the main database. This has visibly improved the performance in these areas.

The key technical challenges prior to launch are the need to update legacy frameworks, a greater focus on testing and fixing defects found. A notable example of legacy framework is *Struts 1*, which has been out of support for several years. Although replacement, in the form of *Spring MVC*, is now in place, and is used for new areas of the application, investment is required to upgrade fully to *Spring MVC*. *Struts*' dependency on an early Java Servlet Specification now limits our ability to upgrade to the latest *Spring Framework*.

To increase confidence, more performance testing and additional optimising of the system will be required, including the following:

- Improved unit test coverage: There are currently a number of known failing tests; 100% pass rate will be required as we introduce an automated build and delivery system.
- A full suite of regression tests (manual or automated) is also necessary in all the browsers that *uFlexi* supports, and on both Mac and Windows. (A decision about what browsers and versions are to be supported will need to be made.)
- There are now a partial set of *Selenium* tests, but more work is required on these, and on the performance tests.
- Development work will be needed to fix any defects found during testing.
- User acceptance testing.
- Test users in different time zones (if this is a likely scenario).
- It would be prudent to trial run common disaster recovery scenarios in *AWS*, such as using a DB snapshot to replace a production database.

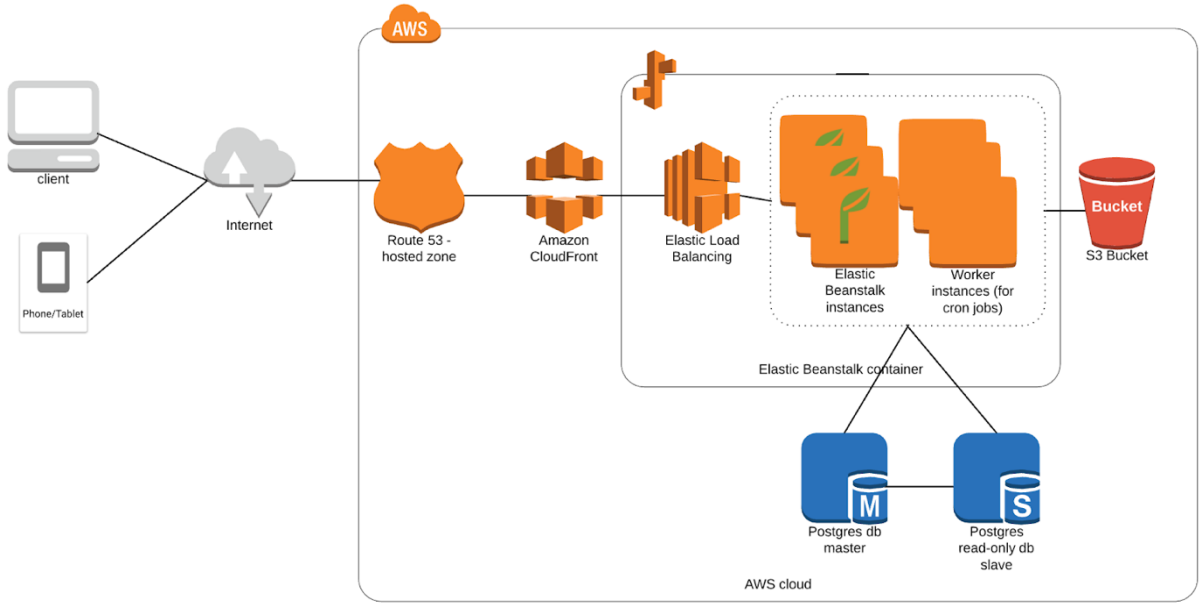
Further work is also required to create a pipeline to production. Code will be checked in at one end of the pipeline and after passing through a series of processes, deployments to production made in a managed way.

New processes are required to manage ‘business as usual’ production defects and maintenance work, alongside new development. An improved release process will be necessary, including a release document to schedule which functionality, bug fixes and other tasks are to be included in each release and build version.

AWS

AWS: Architecture Overview

The diagram below provides an overview of the *uFlexi* architecture for the production system. This set-up is duplicated for both test and development.



AWS: Identity and Access Management (IAM)

Access to the system is via IAM. Different users are configured to provide each with specific access to the system.

Root user has IAM management capability.

Dev users are set as administrators.

S3 Access users (see *AWS:S3* section) have programmatic access to their specific *S3 Bucket*.

BlazeMeter user (for Performance Testing) has access to the *CloudWatch* monitoring data.

AWS: EC2 (Elastic Compute Cloud)

EC2 provides the area (cloud) to store and create our computing environments (instances).

AWS: Elastic Beanstalk Environments

Elastic Beanstalk provides us with the ability to deploy, monitor, and scale an application quickly and easily.

Three environments have been created for production, development and test. Each environment is a copy of the same design, with configuration changes to use different databases, buckets etc.

A *uFlexi* environment consists of the following *AWS* services: *Elastic Beanstalk* / development *RDS* instance / *CloudFront* / *Route 53* / security groups / worker environment to run cron jobs /queues and dead letter queues.

The configuration of each environment has been saved; these can be used to quickly launch a new instance of the environment.

AWS: Relational Database Service (RDS)

Multiple database instances are required; one per production, development and test environment. All database instances uniquely identify its usage and each database instance has a database `nems` and also `nemslog` used for audit purposes.

Users set up on the system are `Postgres` and `nems`.

Database Instance	Usage
development	Development db
test	Test db
production	Live db

Daily snapshots are taken of the databases for backup. The scheduling of snapshots is configurable.

AWS provides security groups that control which services can interact with each other. Using these security groups, it's possible to control which application is allowed access to each database and which IP addresses have the right to connect to the databases.

The cost of the *RDS* instances is the highest portion of the *AWS* cost overall.

The option to allow automatic minor upgrades of the database was selected and upgrades are scheduled for early on Monday mornings (configurable).

Unfortunately, the first upgrade from *PostgreSQL* 10.3 to *PostgreSQL* 10.4 on the development application `dev-uflexi` failed and the application went down unexpectedly during performance testing. A subsequent manual upgrade was successful. The automatic upgrades have been left in place for monitoring purposes; however, until teething problems are resolved it is suggested we manually apply patches to production once testing has been carried out on the dev and test database instances.

Database configuration is achieved through the use of parameters in a DB parameter group. If a parameter group isn't specified during the database creation, then a default group is created.

Read-only replica databases were created from the development, test and production instances. Software changes were required to enable the applications to connect to the replicas. The replicas are to be used in areas such as reporting and the AAG/Seller Selection; anywhere where there is no data to be saved, hence there's an opportunity to reduce load on the main database instance.

AWS: Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. *Route 53* can effectively connect *uFlexi* user requests to infrastructure running in *AWS* – such as *uFlexi's* load balancers, or *Amazon S3* buckets.

Route 53 can handle domain name transfers; `test-uflexi.com` was transferred to AWS during the Platform Project. The `dev-uflexi.com` name was registered for use as the developers' application. The `demo-uflexi` domain is to be transferred in near future.

Route 53 is a pay as you go service like many *AWS* services. We pay for the number of hosted zones used and by the number of requests routed.

Domain names registered with Route 53
`dev-uflexi.com`
`test-uflexi.com`
`demo-uflexi.com` (tba)

AWS: S3 (Simple Storage Solution)

uFlexi resources are stored in an *AWS S3 Bucket*. A bucket is simply a folder (cloud storage area) where the *uFlexi* resources are held.

uFlexi Bucket Access

For security, only the application specific `uflexi-user` has read/write permissions on the folders/resources in the bucket. This ensures that images/documents cannot be accessed outside the application (i.e. there is no public access on the bucket).

In addition to the application specific user, security added in the *Java* code ensures users can only access the sellers/buyers associated with their own agency or partnered agencies. It also takes into consideration the directory access setting set on the agency, ensuring seller and buyer photos aren't shown in the directory area (where users can view the system anonymously).

Deployment	User
dev-uflexi	dev-uflexi-user
test-uflexi	test-uflexi-user
prod-uflexi	prod-uflexi-user

Resources stored in uFlexi S3

- Agency Logo
- Agency Photo
- Agency Banner
- Buyer Photo
- Buyer Banner
- Buyer Document
- Seller Photo
- Seller Document
- Agency Pool Logo
- Agency Pool Photo
- Agency Pool Banner
- Role Image
- Verification (Vetting) Certificate Image
- Verification (Vetting) Issuing Authority Image

Note. Additional images used in *uFlexi*, such as those in the colour picker (branding), seller availability (showing how to select availability) are still stored as part of the packaged WAR file. These images are not dynamic (i.e. cannot be altered by a user). If there is a need to move these to *S3*, this can be done at a later date.

AWS: Elastic Load Balancer

The *AWS Classic Load Balancer* is currently used to provide load balancing across multiple *Amazon EC2* instances. It operates at both the request level and connection level.

Some of the key features of the *Classic Load Balancer* include:

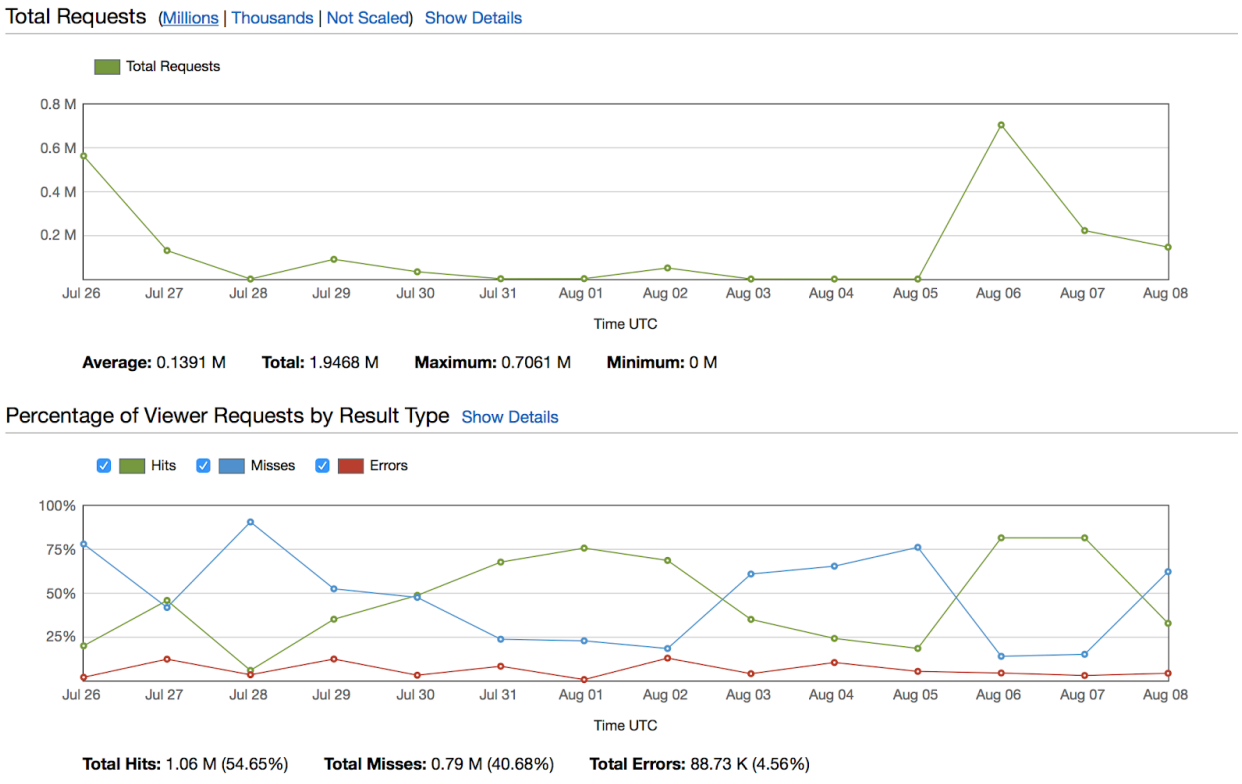
- Automatically scale request handling capacity in response to incoming application traffic.

- Detect the health of *Amazon EC2* instances. When it detects unhealthy *EC2* instances, it no longer routes traffic to those instances and spreads the load across the remaining healthy instances.

AWS: CloudFront

CloudFront is a Content Delivery Network (CDN) which caches content and speeds up delivery. *CloudFront* acts as a proxy for both *S3* and the *EC2* instance and adds an extra layer of security in front of our uFlexi applications.

Like many of the *AWS* services *CloudFront* has some useful monitoring tools, such as its 'Cache Statistic Reports'; example charts shown below.



SSL Security

Each of the three applications (dev, test and production) use SSL (Secure Sockets Layer) for added security, ensuring all data passed between the web server and browsers remain private and integral. This is easily recognisable through the https address the browser URL. Any http requests are automatically re-routed to https. The configuration for the re-route is set in *CloudFront*.

AWS Costs

AWS offers a free tier option for the first year. The free tier provides a small amount of resource and services for no charge; additional costs are accrued for any usage above free tier limits. Once the offer expires costs revert to standard pay-as-you-go rates.

For example, 750 hours per month for a small database instance `db.t2.micro` are available at no cost and 20 GB of database storage. 5GB of S3 storage is provided for free including 20,000 `GET` Requests and 2,000 `PUT` Requests. 2,000,000 https and http requests for *CloudFront* are also included.

AWS also offers a number of services at no charge and with no time period limitation. For example, 1,000,000 requests of *Simple Queue Service*.

The bulk of the *uFlexi* costs are due to database usage. As you'd expect, increasing compute and memory capacity of the database instance increases the cost. The capacity of database that uFlexi requires means that free tier usage isn't sufficient. Currently we're using `t2.medium` instances, however the *Technical Architecture* document recommends `t2.large`. Hence it's anticipated that, at some point after launch with increasing uFlexi users, the database instance may have to be upgraded to `t2.large`.

You pay for what you use, there is no minimum price. Price varies depending on the state in which your databases are hosted. `t2.large` database instances with multi Availability Zone (more-or-less a data-centre site) deployments cost $0.38 per hour in the state of California.

AWS costs aren't easy to estimate; costs depend on how the many *AWS* services are configured and how much capacity is expected on a monthly basis.

AWS provide a simple calculator to help users project their costs: http://calculator.s3.amazonaws.com/index.html.

Costs during the Platform Project have been averaging out at around $1000 per month. Costs are expected to rise as the application is scaled up.

Testing

Acceptance Testing

"*Selenium* automates web browsers".

Selenium can be used for automated testing in browsers and also automating repetitive administrative jobs. It was used for automated acceptance testing of different scenarios and user journeys in *uFlexi*, such as 'seller change terms'. It was also used for testing the most complex functionality where increased load may cause problems, for example the during the booking process and viewing seller availability in the Aggregated Availability Grid (AAG).

Performance Testing

Load testing was carried out on an earlier version of the *uFlexi* codebase in 2010. The results of this testing showed that the site could be knocked offline by small spikes of 10 simultaneous bookings. Optimisations such as disabling the AAG functionality were considered for times of high load. Since then some fine-tuning of the AAG code has been carried out and configuration changes to the *Tomcat* thread pool have enabled larger capacity of 50 simultaneous bookings.

Performance testing was carried out on *AWS* using *BlazeMeter* with *JMeter* scripts.

BlazeMeter performance testing involved running multiple *JMeter* scripts over a configurable period of time. The first scripts were run over a 15 or 20 minute period with a maximum of 20 concurrent users. Later, the same tests were re-run for a similar period, with increased numbers.

During the tests it was possible to see how *AWS* scaled up *Elastic Beanstalk* instances to cope with the increased load on the application. The scaling up/down is affected by configurable parameters such as 'Scaling cooldown' and 'Scale up increment' etc.

The *JMeter* scripts covered common usage scenarios in *uFlexi*. The tests ranged from very simple tests, for example, a user logging in and out of *uFlexi*, to more complex scenarios such as 'make a booking'.

The main statistics produced for each test are described below:

- **Maximum Users**: The Maximum number of Virtual Users that were tested. This is important for understanding load capabilities.

- **Average Throughput**: The average number of Hits per Second per user during the test. This is important for understanding the amount of traffic *uFlexi* can handle.

- **Errors**: The error percentage out of all requests. This is important for understanding if your *uFlexi* works properly, or not.

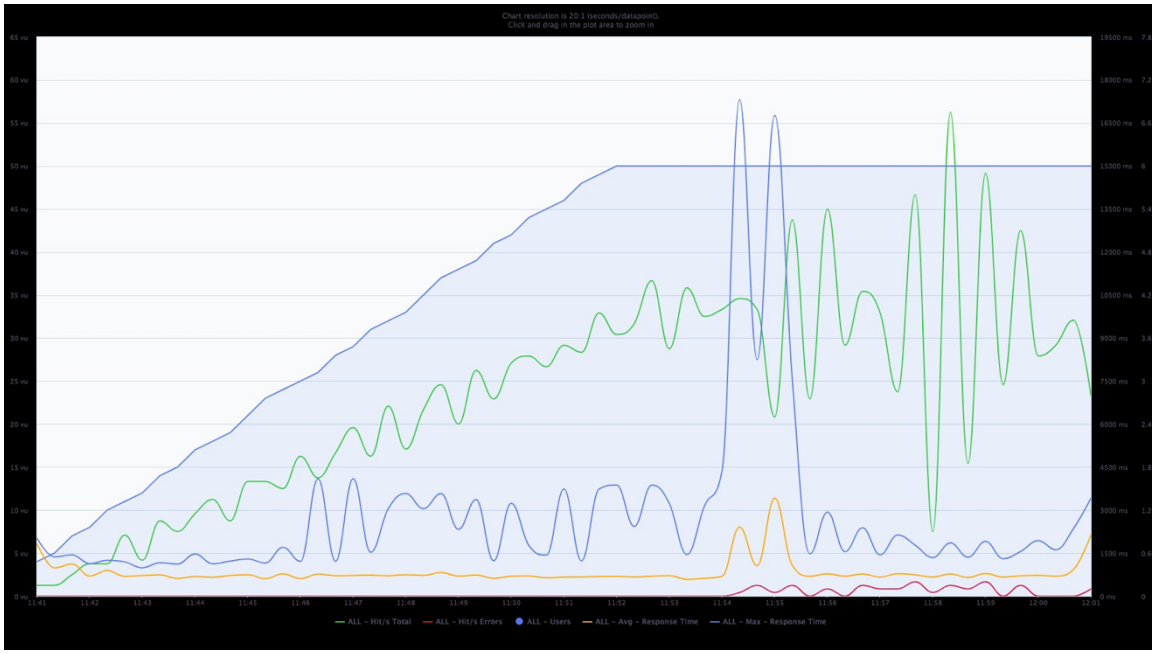
- **Average Response Time**: How long, in milliseconds, did it take the average user to receive a response for their request. This is important for measuring and understanding the users' experience when using *uFlexi*.

- **90% Response Time**: The slowest response time, in milliseconds, the 90th percentile receives. This is important for measuring and understanding the majority of the users' experience, when using *uFlexi*, excluding extreme incidents.

- **Average Bandwidth**: The average amount of data transferred in KIB. This helps to understand the typical network load on the servers during high traffic scenarios. Note: A KIB = Kibibyte (1 kibibyte is 1024 bytes).

Example Test 1: 50 concurrent users logging in and out over a 20-minute period

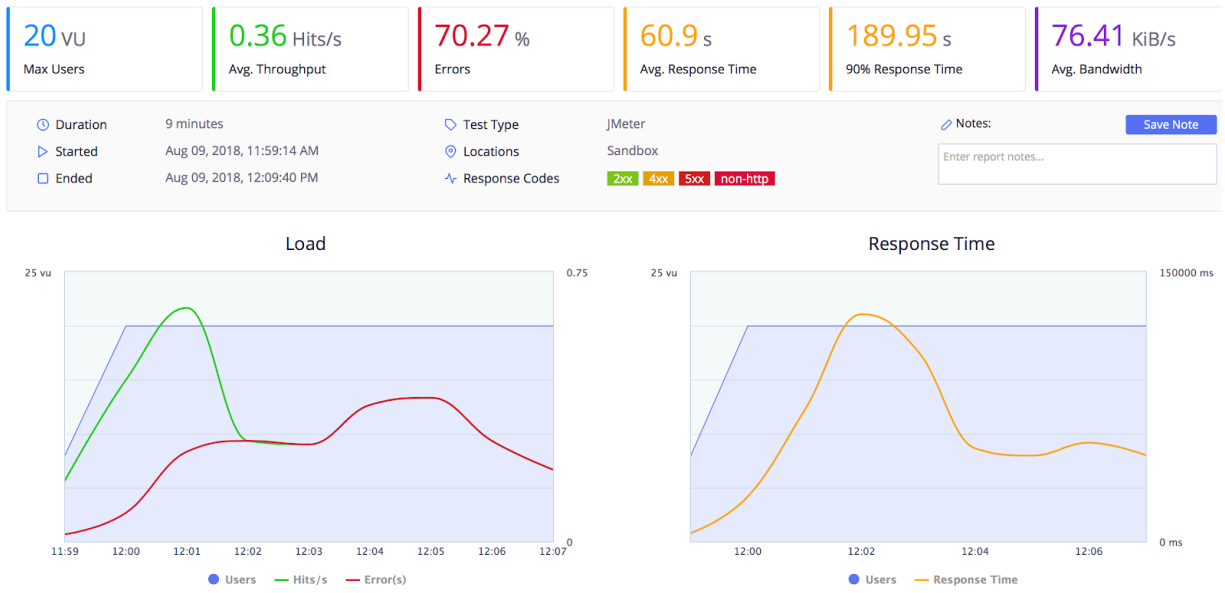
Load balancer set to min 1 max 4 , database instance `db.t2.medium`



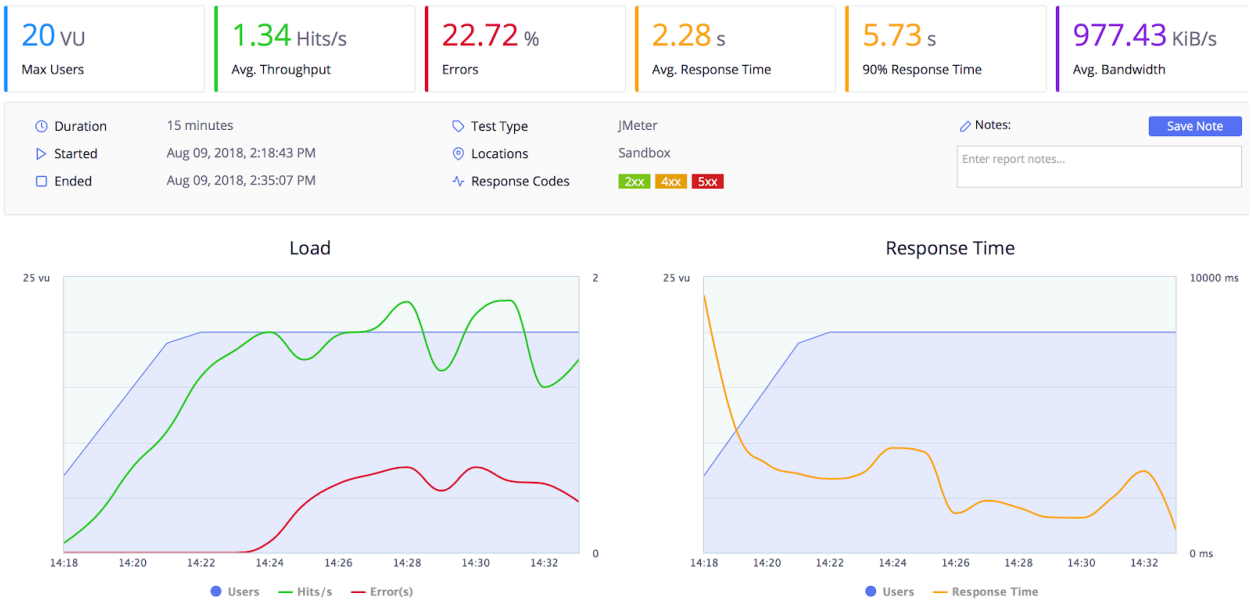
Specific areas of the system are more CPU- and database-intensive than others. Individual performance tests concentrated on these areas to filter out specific bottlenecks in the system.

Example Test 2 - Bookings Report Test

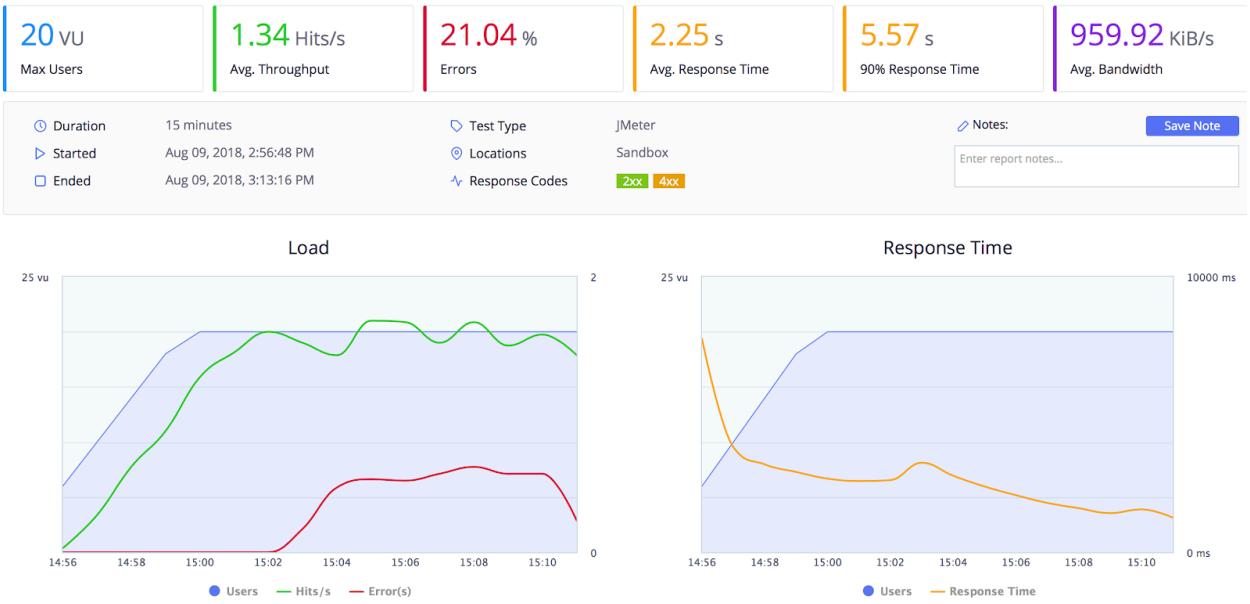
The bookings report test was run on the *BlazeMeter SandBox* which limits concurrent users to 20. The first run through resulted in very poor results; a significantly high number of errors and slow response time:



The max JVM heap size was increased from 256m to 512m and the test run again, yielding improved results:



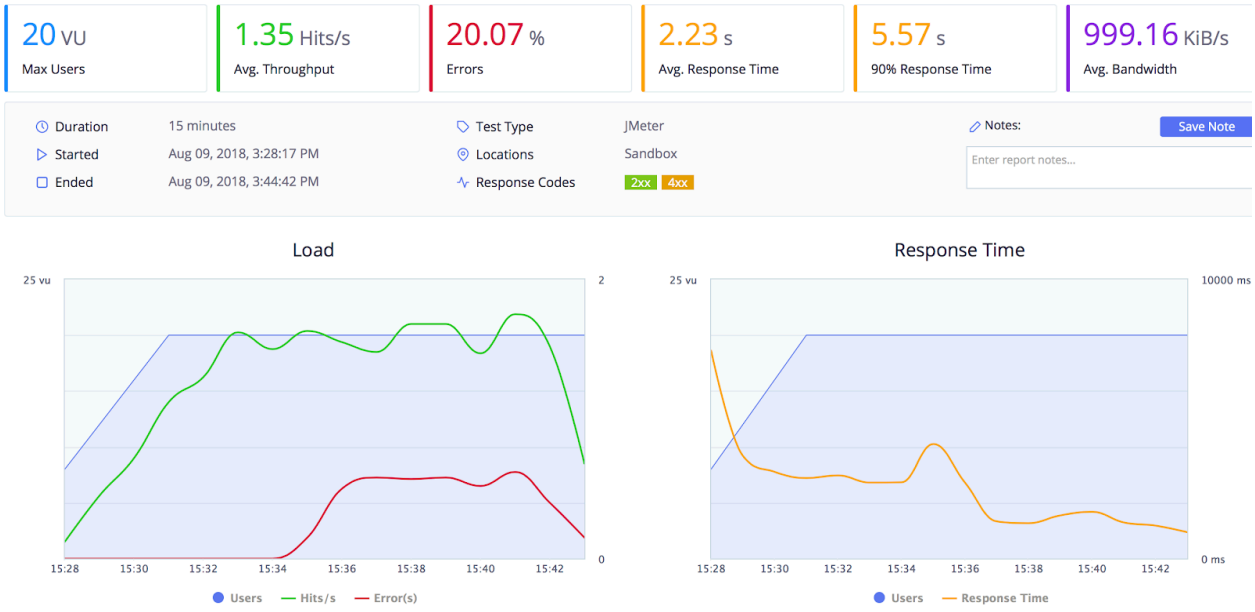
Finally, the max JVM heap size was increased from 512m to 1024m and the test run again.



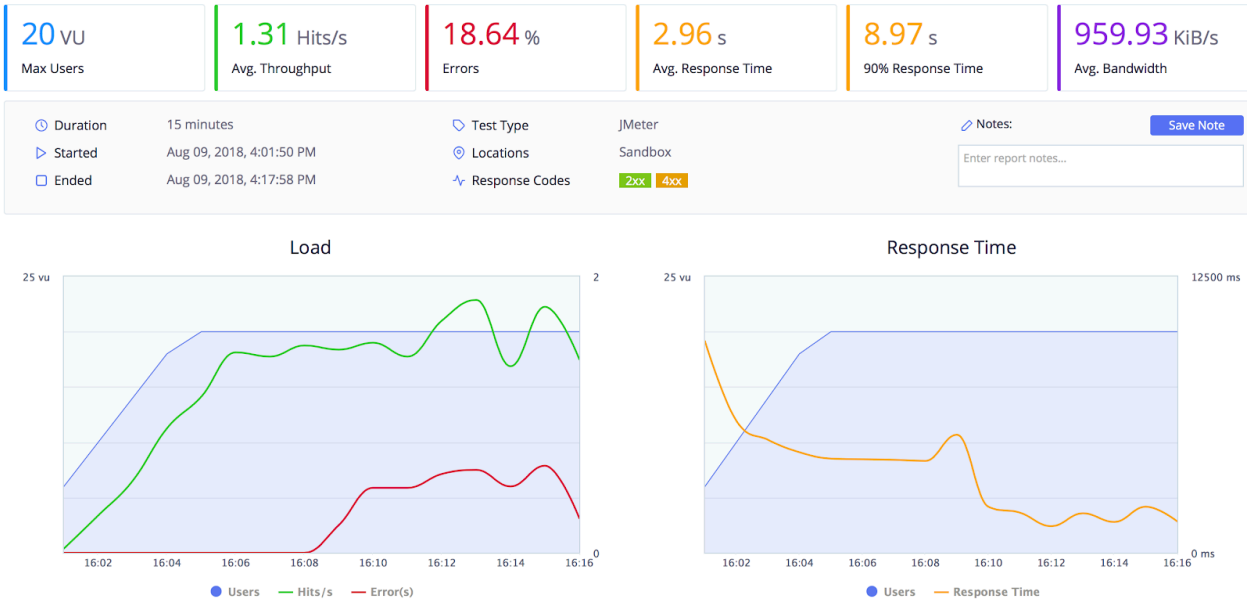
The results showed that increasing the memory to 1024m resulted in faster response times, fewer errors and fewer types of error. The 5xx errors were eradicated.

It was noted that the 4xx errors were seen mainly around the time that the *Elastic Beanstalk* servers scaled up so the next approach was to change the threshold at which the scaling up/down occurred.

The 'breach duration' was changed from 5 minutes to 3 minutes and the test run again. The results showed slightly fewer errors and marginally improved response times:

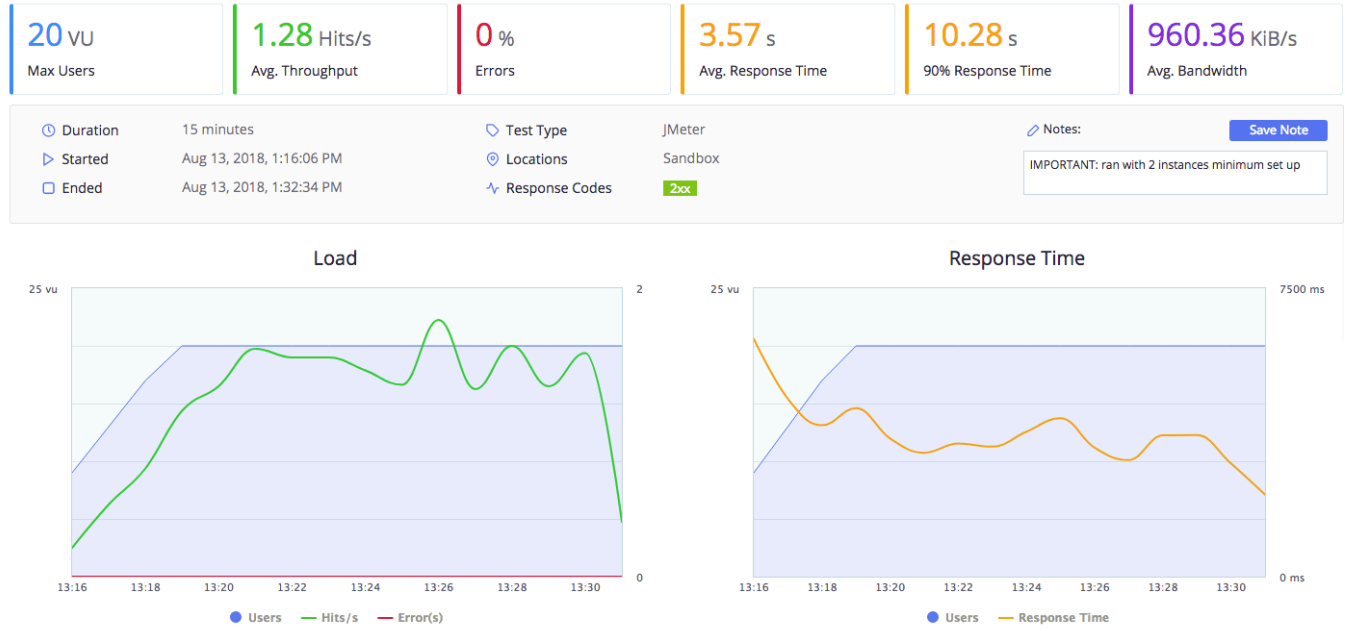


The upper threshold (at which scaling up occurs) was then amended from 6,000,000 bytes to 5,500,000 bytes and the test rerun:



Configuration changes were completed to produce an error free test. To examine performance changes due to instance creation, a minimum of two instances was configured to give the following results:

As can be seen, the response time is slightly higher than the previous test results. This is due to actual data being returned, rather than error responses (which were previously Unauthorised errors and would have been returned immediately from the server).



Example Test 3 - AAG & Seller Selection Accordion

The AAG and Seller Selection Accordions on the Purchase Build screen are extremely CPU- and DB-intensive. Being read-only requests from the database they provide the ideal place to re-route the DB requests through to the read-only slave database, to improve performance and take the "load" off the master.

LOGIN: Mary Jones

ACCOUNT: Client Accounts – no spending controls

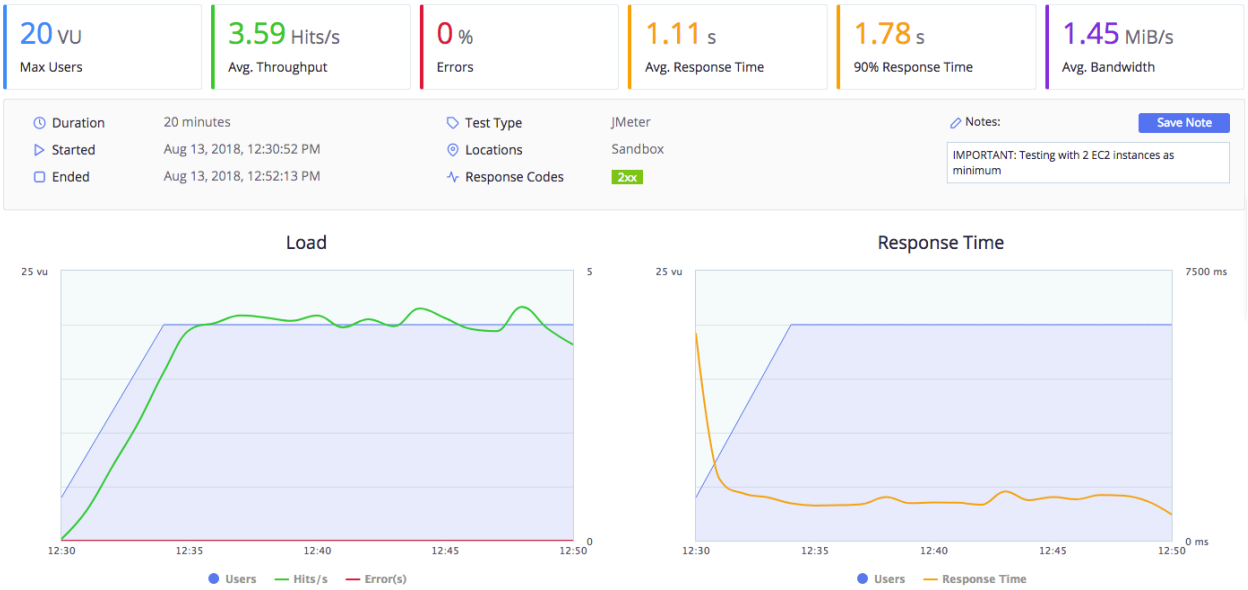
ROLE: Healthcare Support > St. John's Exercise Assistant

LOCATION: St. John Care Home 91790

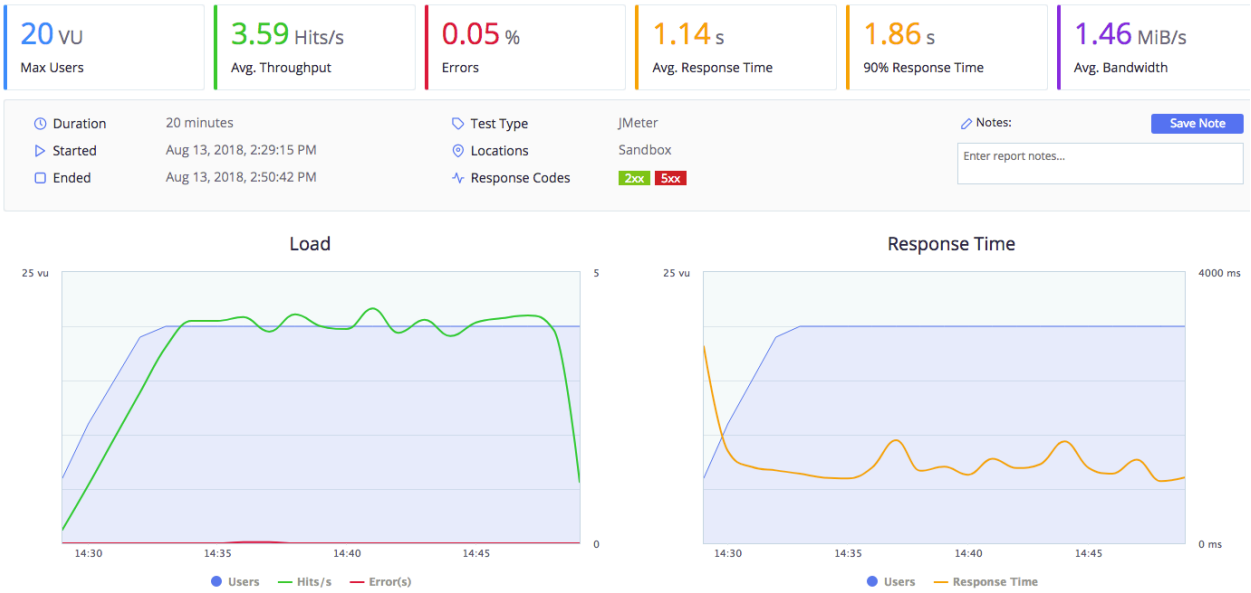
Results for reaching AAG only:

The number of errors increases dramatically just after 16:36. These are all 401 "Unauthorized" errors. This appeared to be happening when a new *EC2* instance was created.

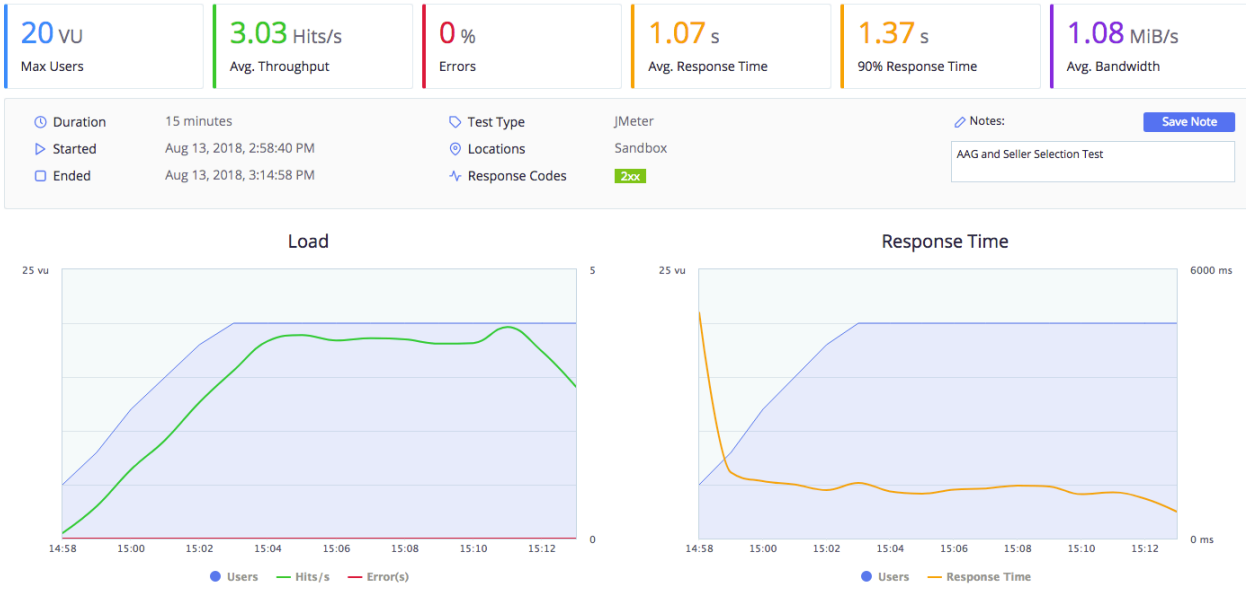
Adjusting the load balancing configuration gave the following result with a minimum of 2 instances for the test which reaches both the AAG accordion:



Reverting back to starting at a single *EC2* instance gave the following results. Whilst there are a small number of errors, less than 0.05%, this may have just been a glitch in the system. The error produced was a Gateway Timeout error when trying to connect to *Google*, this may have been caused by a slow local internet connection.

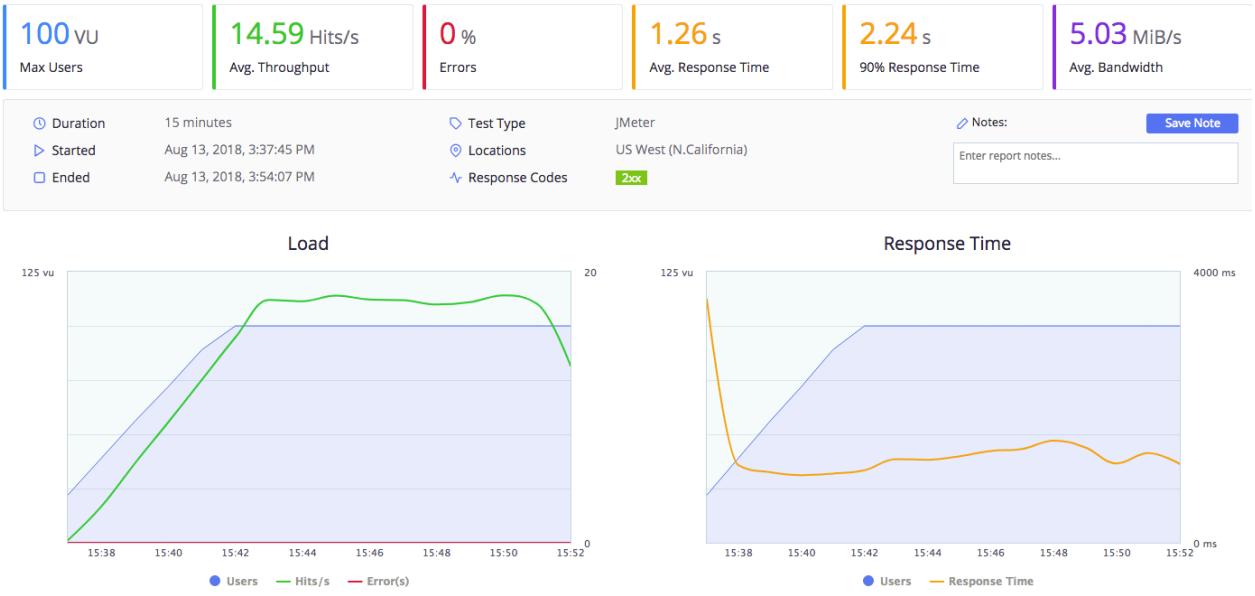


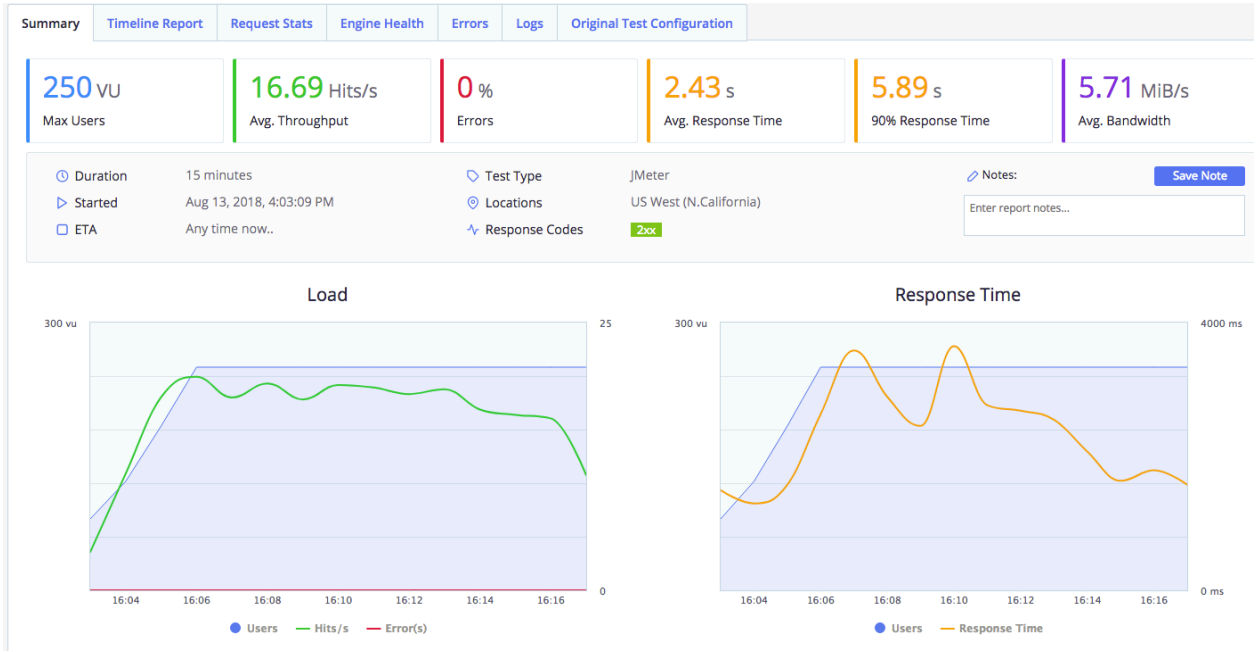
The same test was repeated with the addition of the seller selection accordion, as can be seen, it shows very similar results:

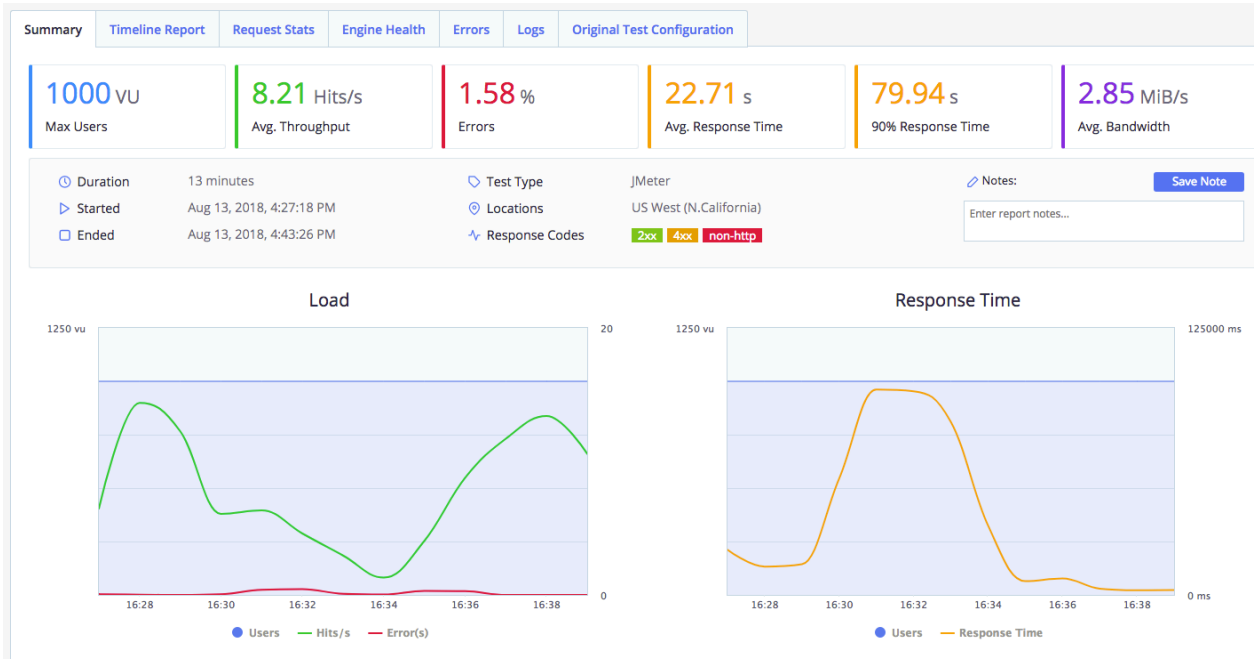


To test the number of concurrent users on the system. The number was increased to 100 and then to 250.

The following results show a slower response speed, obviously due to the increased volume.



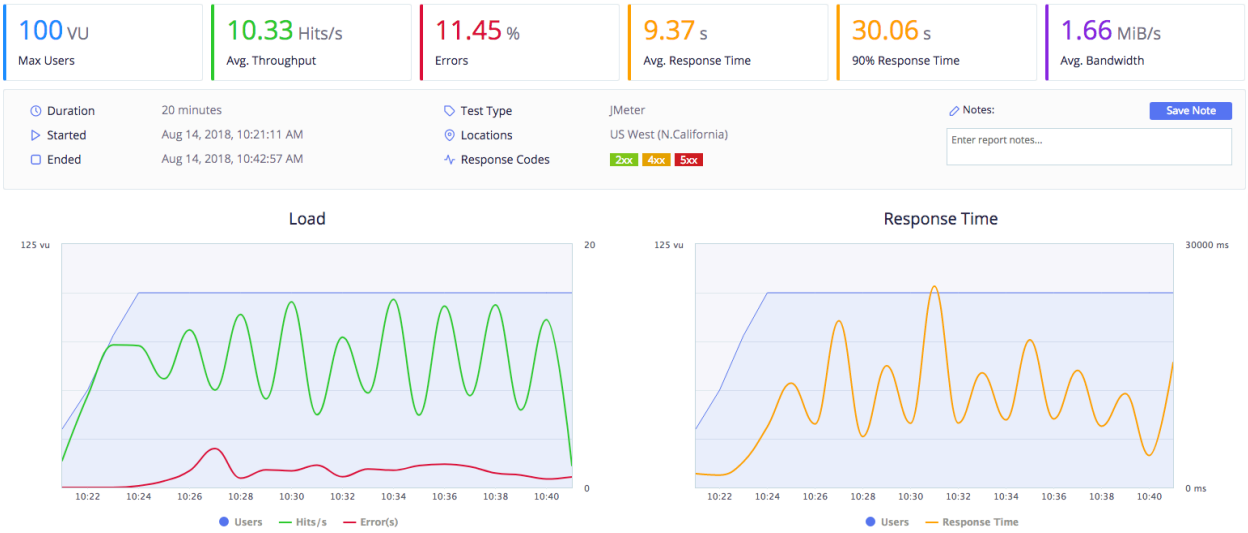

Summary
Timeline Report
Request Stats
Engine Health
Errors
Logs
Original Test Configuration
250 VU
Max Users
16.69 Hits/s
Avg. Throughput
0 %
Errors
2.43 s
Avg. Response Time
5.89 s
90% Response Time
5.71 MiB/s
Avg. Bandwidth
Duration
15 minutes
Started
Aug 13, 2018, 4:03:09 PM
ETA
Any time now..
Test Type
JMeter
Locations
US West (N.California)
Response Codes
2xx
Notes:
Save Note
Enter report notes...
Load
300 vu
25
0
16:04
16:06
16:08
16:10
16:12
16:14
16:16
Users
Hits/s
Error(s)
Response Time
300 vu
4000 ms
0 ms
16:04
16:06
16:08
16:10
16:12
16:14
16:16
Users
Response Time


Summary
Timeline Report
Request Stats
Engine Health
Errors
Logs
Original Test Configuration
1000 VU
Max Users
8.21 Hits/s
Avg. Throughput
1.58 %
Errors
22.71 s
Avg. Response Time
79.94 s
90% Response Time
2.85 MiB/s
Avg. Bandwidth
Duration
13 minutes
Started
Aug 13, 2018, 4:27:18 PM
Ended
Aug 13, 2018, 4:43:26 PM
Test Type
JMeter
Locations
US West (N.California)
Response Codes
2xx
4xx
non-http
Notes:
Save Note
Enter report notes...
Load
1250 vu
20
0
16:28
16:30
16:32
16:34
16:36
16:38
Users
Hits/s
Error(s)
Response Time
1250 vu
125000 ms
0 ms
16:28
16:30
16:32
16:34
16:36
16:38
Users
Response Time

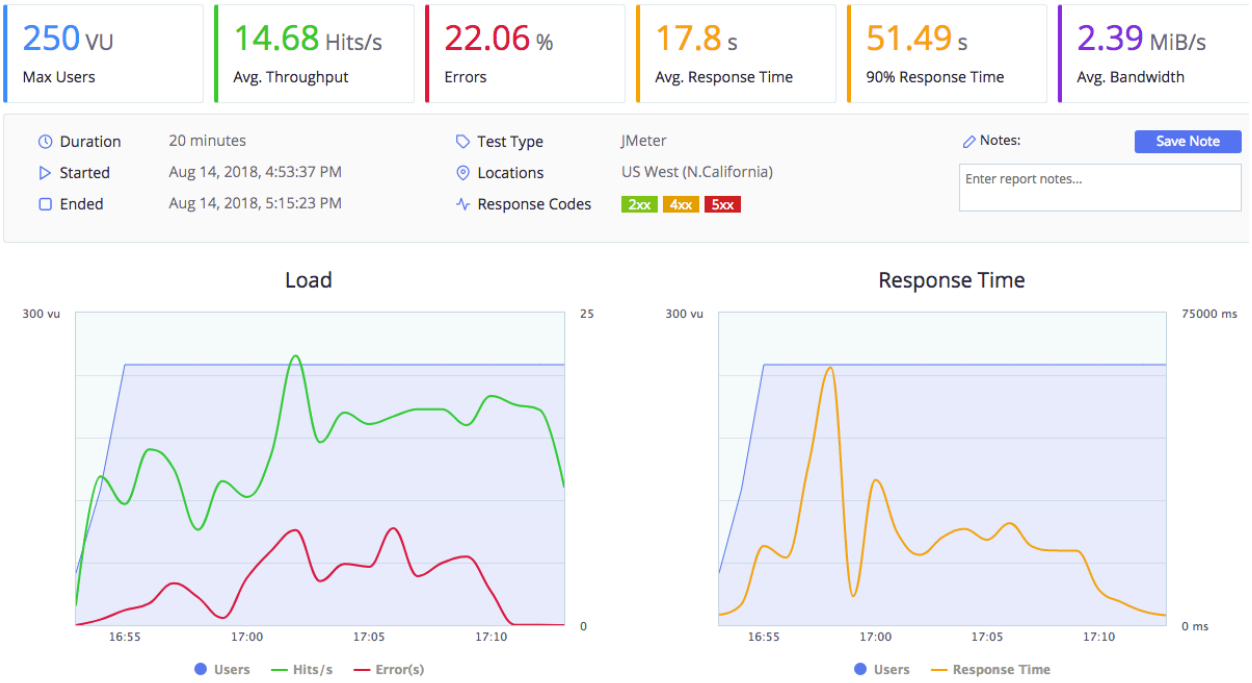
Ensuring the purchase build screen produced viable results, the tests were re-run with future dates. Producing a richer data set, the average response time is still only marginally slower. It must be noted however, that the 90th percentile result shows a response time of 5.61s. Investigating the data, it can be seen that the AAG has a 90th percentile response time of 5.503s (average response time 2.908s).

Increasing the number of users showed a significant change in the response time and errors. The slowest areas for the page (as expected) were the AAG, followed by the seller selection accordion. The majority of failures are from gateway timeout errors (504) when showing the AAG. Further investigation into the errors is required. It has been noted that database errors on both the development and its read-only replica were found. It is highly likely that the database errors could be the cause of the gateway timeouts, but further investigation is required. No errors were noted in the *Elastic Beanstalk* logs.

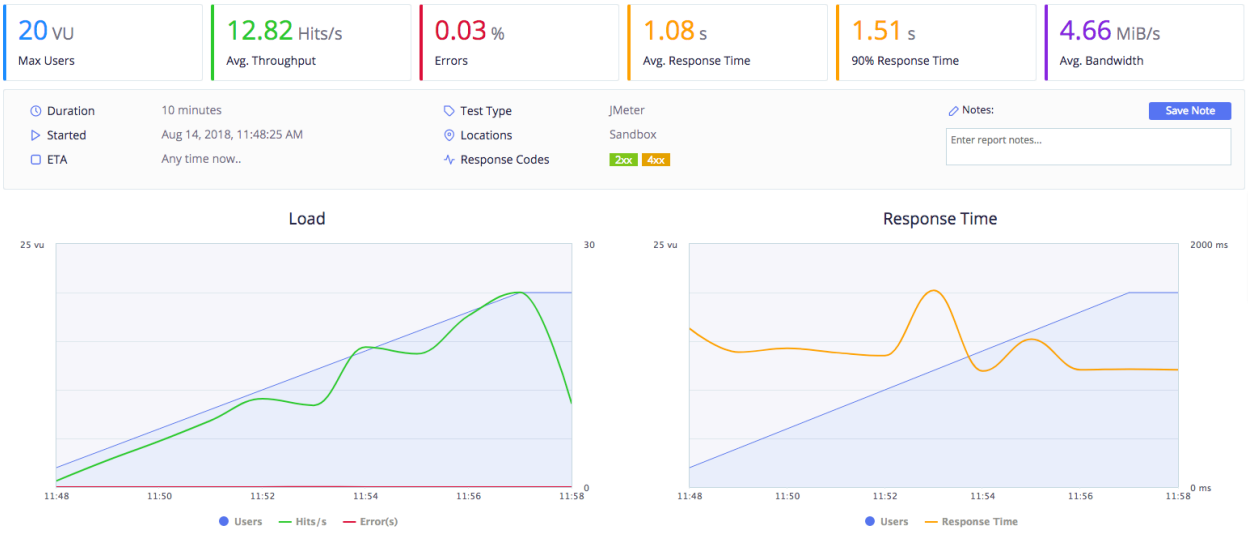
During the test, the number of *EC2* instances increased from 1 to 4 (4 is currently our configured maximum) at 6 minutes, 14 minutes and 22 minutes into the test (the last instance being created due to the test throughput, albeit after the test had finished). The system gradually removed these, at approximately 8 minute intervals after the test completed.



The next move up to 250 users produced a similar result, albeit slower response times. Again, gateway time-outs caused the majority of issues. It must be noted that this test is actually 250 users simultaneously accessing the most CPU/DB intensive areas.

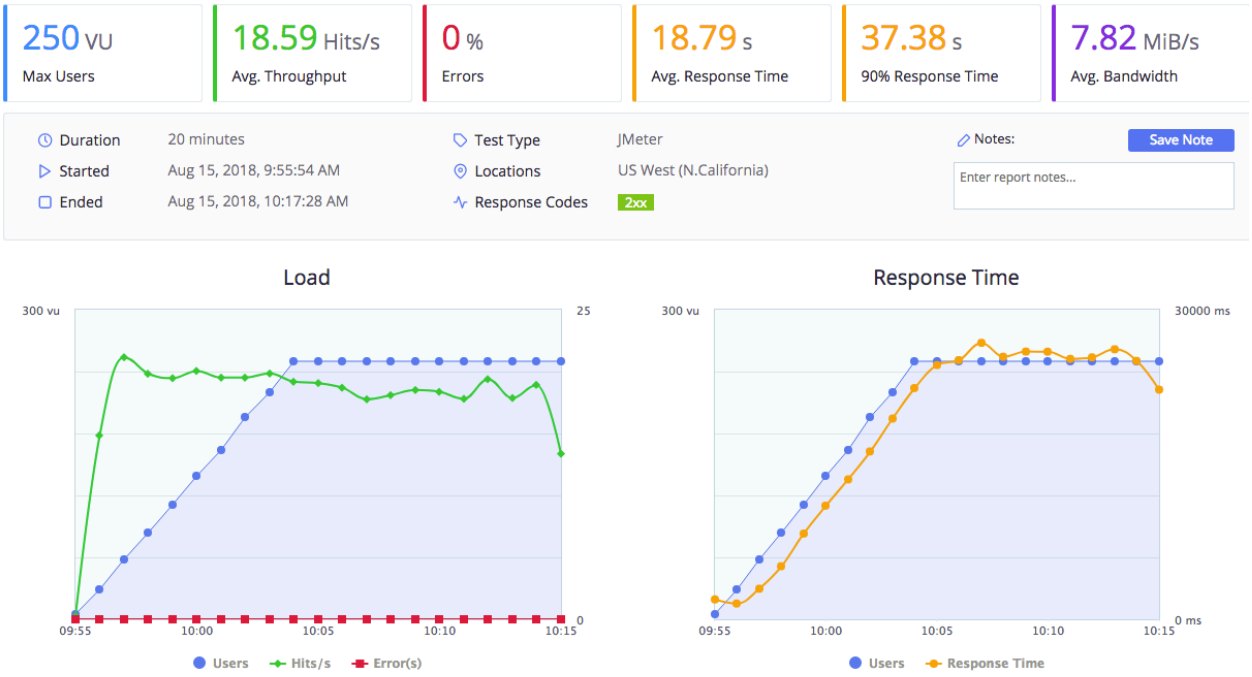


Although there were 2 (0.03%) errors in the test, the response time (including the 90th percentile response time) are low. The page has much less CPU/DB utilisation than the previous test.



Bumping up the users to 250 gives the following results. There are no errors but the response time slows down. Starting at 1 EC2 instance, 2 additional instances were created 10 minutes and 17 minutes into the test. It must be noted, that the master database encountered errors

(constraint missing/unable to create). Further investigation is required to determine if the errors have affected the test outcome.



Test: Seller - About Me: References

As a comparison, a basic Seller test was configured for a seller to login and view their references

Test: Log in as seller wr1@yahoo.com and view references.

Additional information

In preparation for US launch, the following front-end changes have been made:

- Americanised wording
- 12-hour clock
- Multi-language
- Accessibility using ‘userway’

And the following system changes:

- Remap of demo system to US addresses
- Use of *Google* API for enhanced address searching & validation
- SMS using redirected US phone number