Scenario # 1: Perform a load test of 1000 visitors visiting the web app's home page in a 15 seconds period.

Test Setup

Application under Test (AUT): https://www.demoblaze.com/

Thread Group: Stepping Thread Group

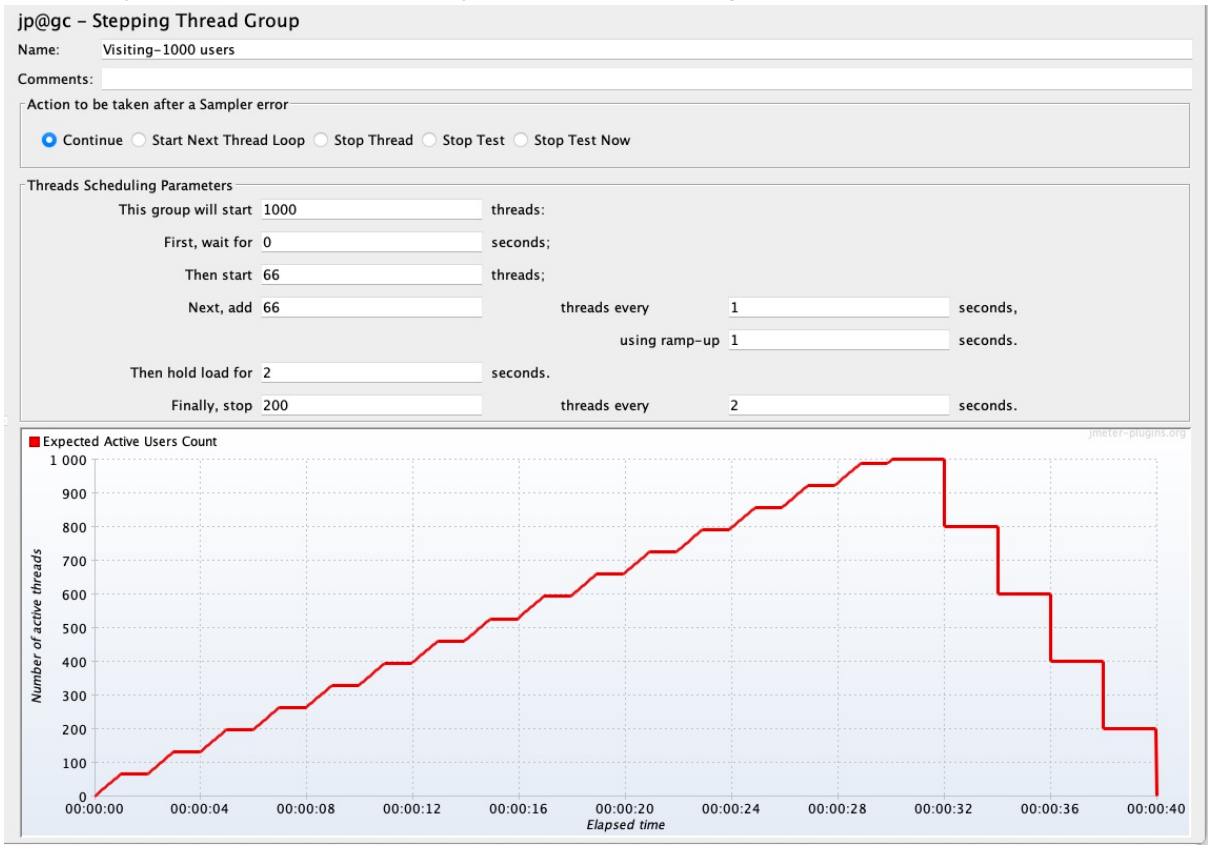
Http Request
GET Request to AUT home page
Mode of execution: Command Line
Results: Html Results

Test Scenarios

Script #	Scenario	Name
1	Visit the application with a 1000 user load in 15 seconds period	Visiting – 1000Users

Test Strategy

To establish a realistic load on the application, Stepping Thread Group plugin of JMeter was used. This thread group lets the user increase threads by portions, configure the hold target time if any and decrease the load by portions. The configuration is as follows:



At a time 66 users were stepped up and this continued for the next 15 seconds so as to reach the desired count of 1000 users. Once 100 users are up, the execution holds for 2 seconds and then 200 users are ramped-down every 2 seconds. This configuration was done, so as to reduce the number of failures due to Loop Detection by the application server.

Note: This application has redirects. JMeter considers each redirect as a new request sample. So, the total number of samples would be higher than 1000.

Results & Observations

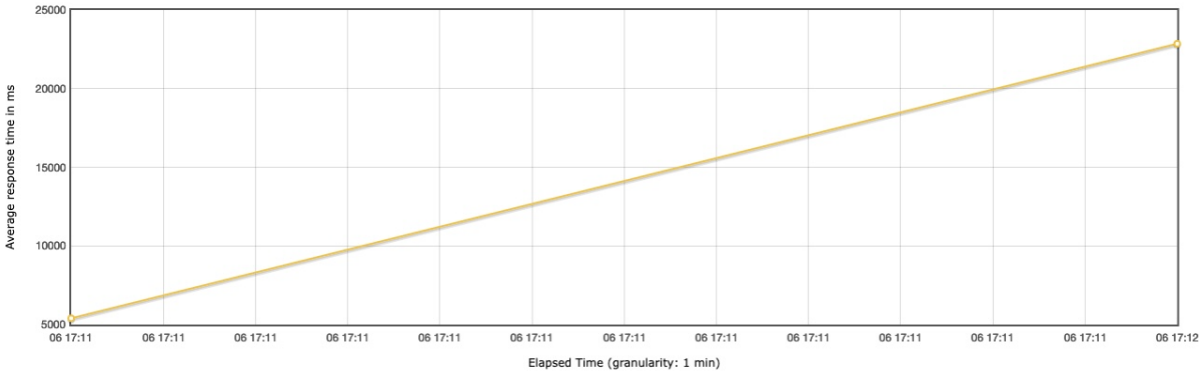
The tests were executed through the command line and the results were captured in Html format. Below are some of the observations from the results.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4445**	**23**	**0.52%**	**5863.67**	**738**	**31808**	**4537.00**	**11356.80**	**14289.60**	**22990.64**	**74.36**	**679.45**	**8.60**
HTTP Request	4445	23	0.52%	5863.67	738	31808	4537.00	11356.80	14289.60	22990.64	74.36	679.45	8.60

1. The application has quite a few redirects. So, the number of samples is higher then the thread count.
2. The error percentage is fairly low 0.52% and happens mainly due to the server detecting loop and abruptly stopping the handshake.

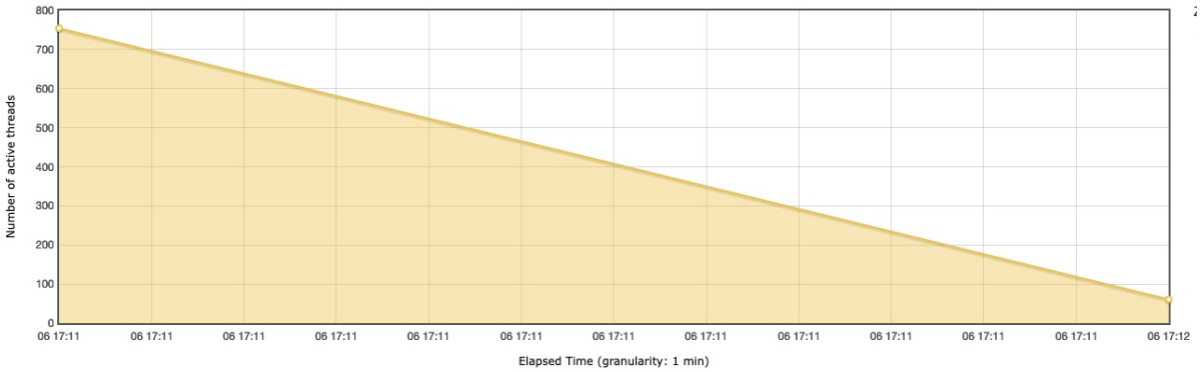
Type of error	Number of errors	% in errors	% in all samples
Non HTTP response code: org.apache.http.NoHttpResponseException/Non HTTP response message: www.demoblaze.com:443 failed to respond	16	69.57%	0.36%
Non HTTP response code: javax.net.ssl.SSLException/Non HTTP response message: Connection reset	3	13.04%	0.07%
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset	2	8.70%	0.04%
Non HTTP response code: javax.net.ssl.SSLHandshakeException/Non HTTP response message: Remote host terminated the handshake	2	8.70%	0.04%

3. The throughput of the application is fairly high and is clocked as 73.36 hits per second. This indicates the server is well equipped to handle a fair number of hits per second.
4. The average time is clocked at 5.8 seconds which means that the response time of all the samples average at 5.8 seconds.
5. The maximum and 99% time of the requests is on the higher side. This indicates that at a certain duration the response time for some requests were fairly high.
6. The Response Times Over Times graph is a linear graph and indicates that as the time elapsed and the load was increased, the response time of the application suffered.

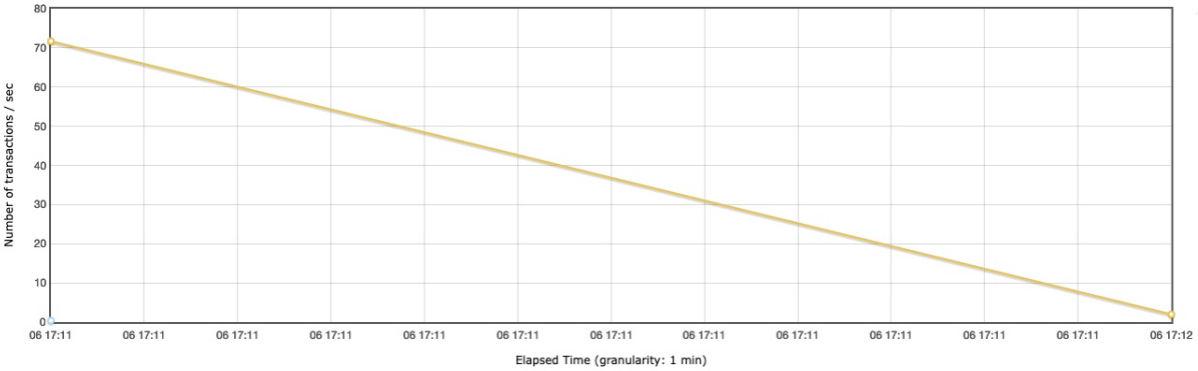


7. The Active threads over time graph shows the number of users active as the execution progressed. This is in accordance with the configuration of the Stepping Thread Group. At a certain point during the execution, the active threads were

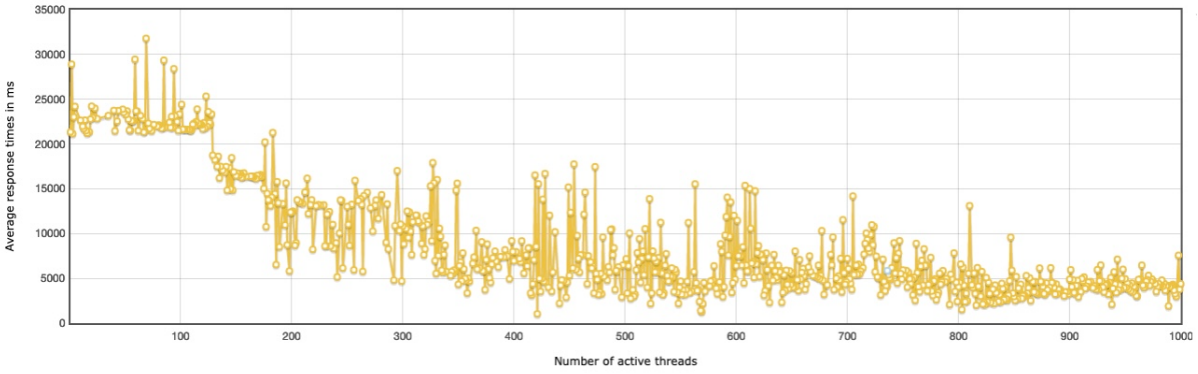
clocked as 1000 and after that the threads gradually decreased due to the ramp down configured.



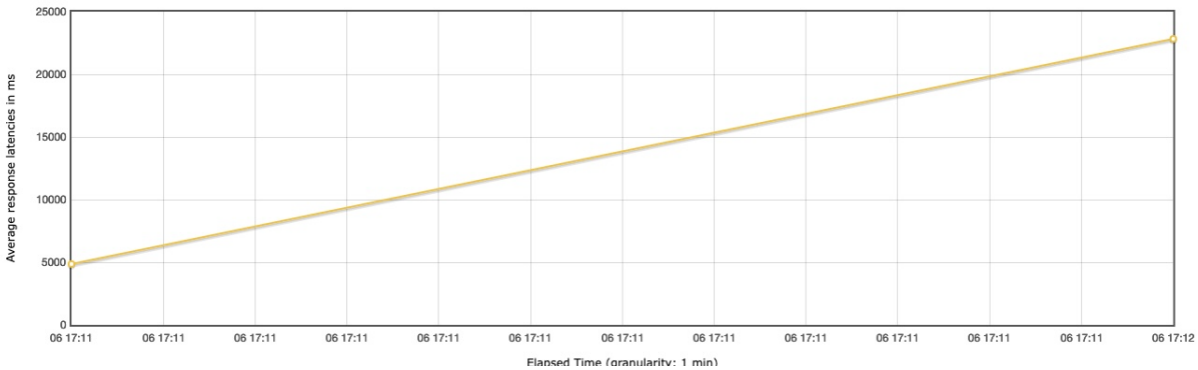
8. The transactions per second graph shows the transactions rate was high initially and linearly decreased when the ramp down occurred.



9. It was observed that response time of the application was very high when the test started and the number of users were low. The response time saw a sharp decline when the number of users were close to 200 and as the number of users increased the response time was steady and it reduced towards the end.



10. The latency time also increased gradually when the number of users increased as time passed in the test.



Scenario # 2: Perform an end-to-end performance test of a checkout workflow

Test Setup
Application under Test (AUT): https://www.demoblaze.com/

Controller: Critical Section Controller
Mode of execution: Command Line
Results: Html Results

Test Scenarios

Script #	ScriptName	Transactions	Total Transactions
1	Launch	1. Launch 2. Launch Entries	2
2	Login	1. Login 2. Login Check	2
3	SelectProduct	1. SelectProductCheck 2. SelectProductView	2
4	AddToCart	1. AddToCart	1
5	GoToCart	1. GoToCartCheck 2. GoToCartViewCart 3. GoToCartView	3
6	PlaceOrder	1. PlaceOrderDeleteCart 2. PlaceOrderCheck	2

Test Features

1. All the test scenarios were grouped under the Critical Section Control of JMeter tool. The Critical Section Controller provides a global lock so that the tests can execute in sequence. Since, the checkout flow happens in a sequence of steps, so this controller was used.
2. Inputs to the test such as username and password have been parameterized.
3. Auth token is extracted from the login response and fed to the subsequent requests by using Regular Expression Extractor.
4. Ultimate Thread Group is used.
5. Think times are added to each request to simulate a realistic load.

Assumption

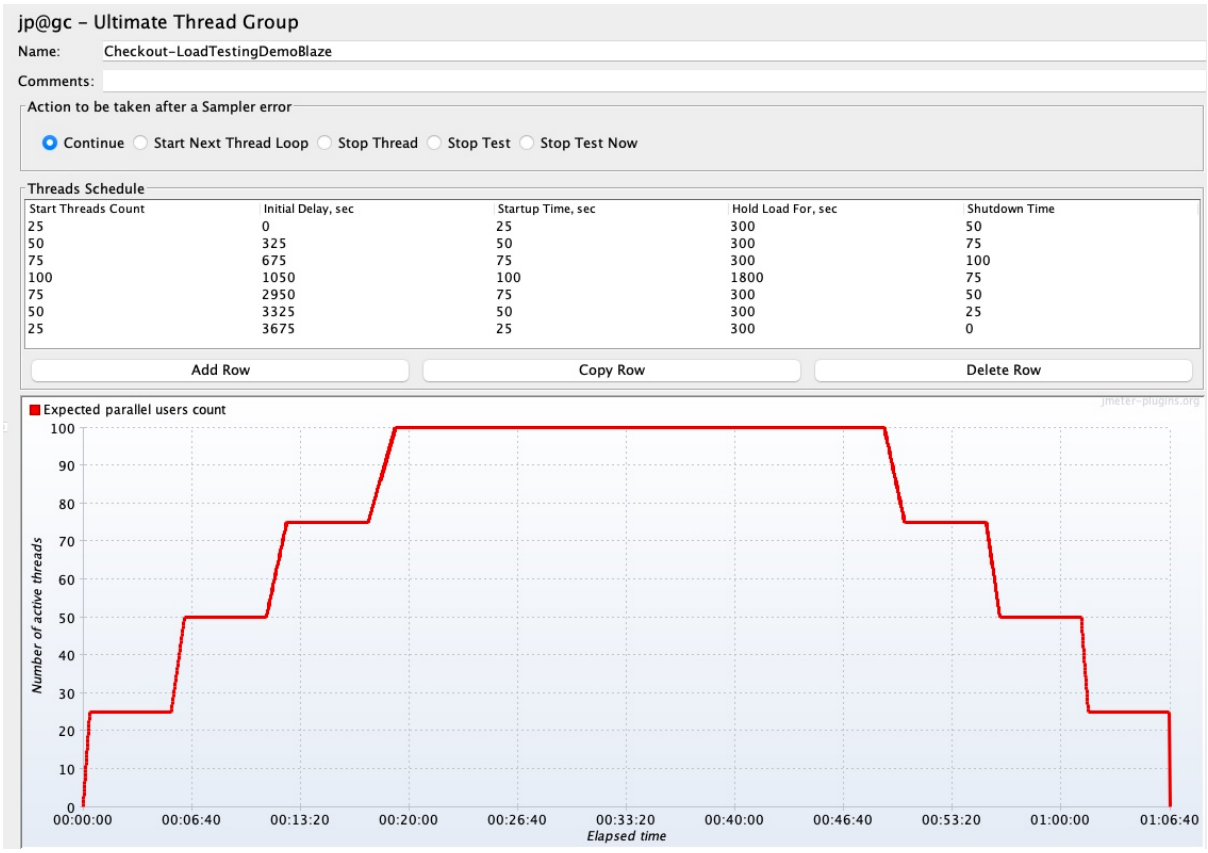
1. Expected virtual users is considered to be 100

Test Strategy

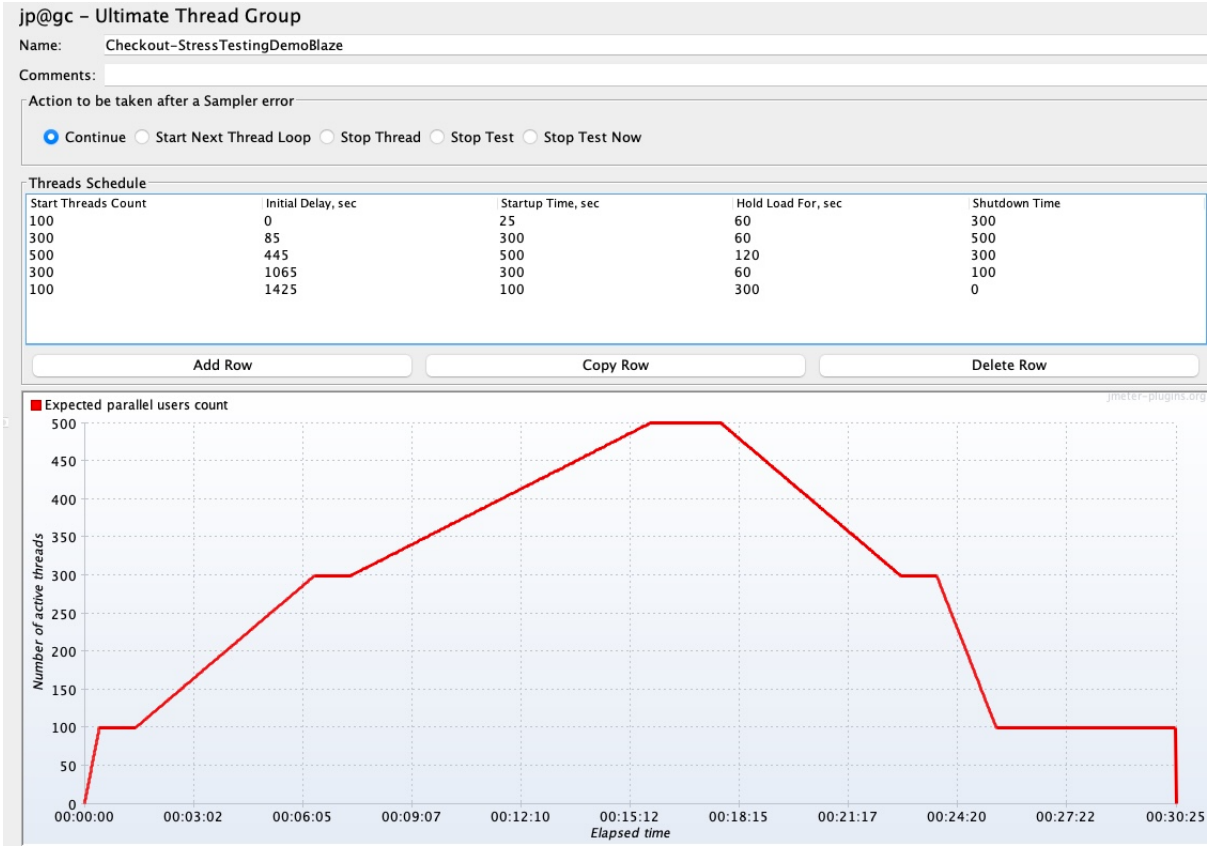
I have performed following types of testing on the application to understand the behavior of the Checkout flow under varying load and conditions:

1. Load Testing: Load Testing was conducted to test the application with the expected load. In the thread group the load has been configured to be increased gradually and in batches and also the ramp down has been configured. Load testing is done with 100 users by increasing the load with 25 users each. The timings are calculated with the below formula:

   Ramp up time of second record (Start-up time) = Shutdown Time of 1st record
   Initial Delay = (Initial delay + Startup time + Hold Time) of previous record.

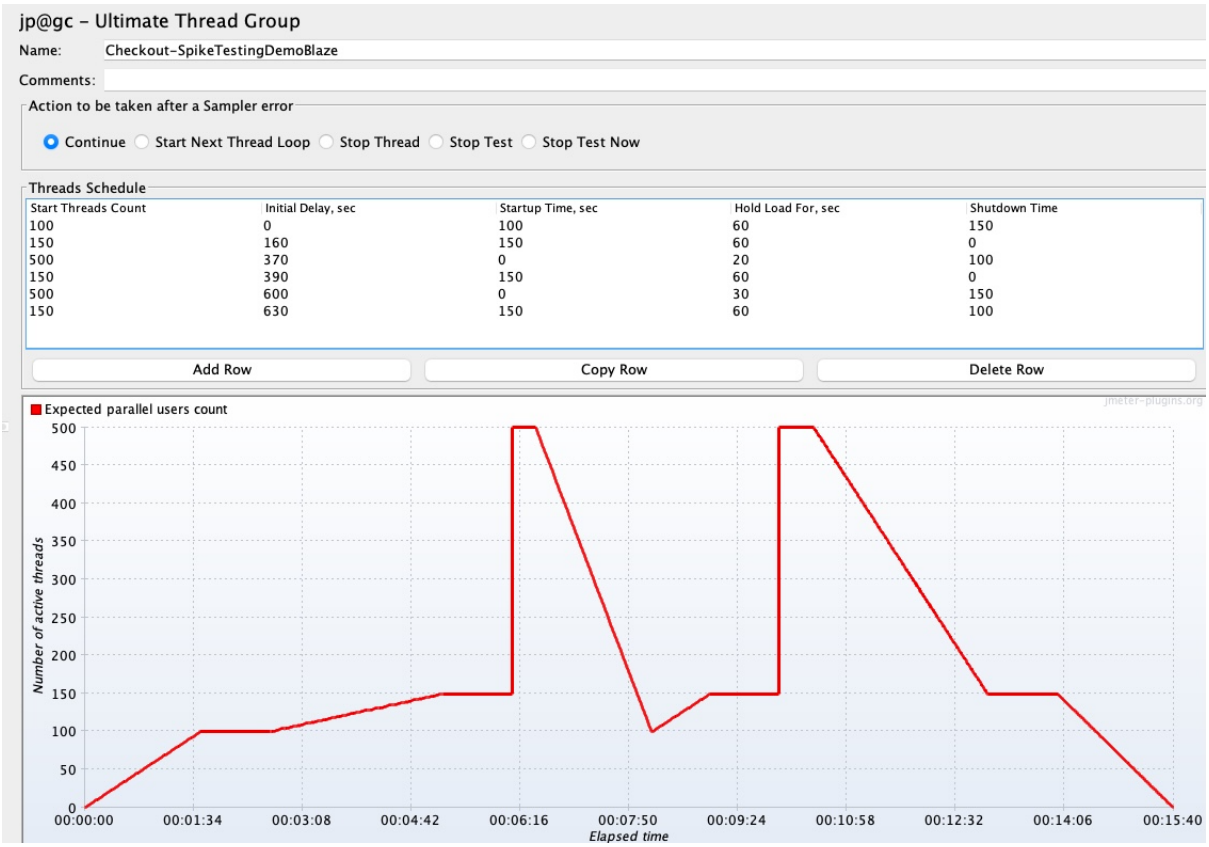


2. Stress Testing: If the application performs well without any errors during the load testing phase, I will be increasing the number of to understand the breaking point of the application. The test was performed repeatedly by increasing the number of users from 100 to 500. The load will be increased if the application does not break under the mentioned load.



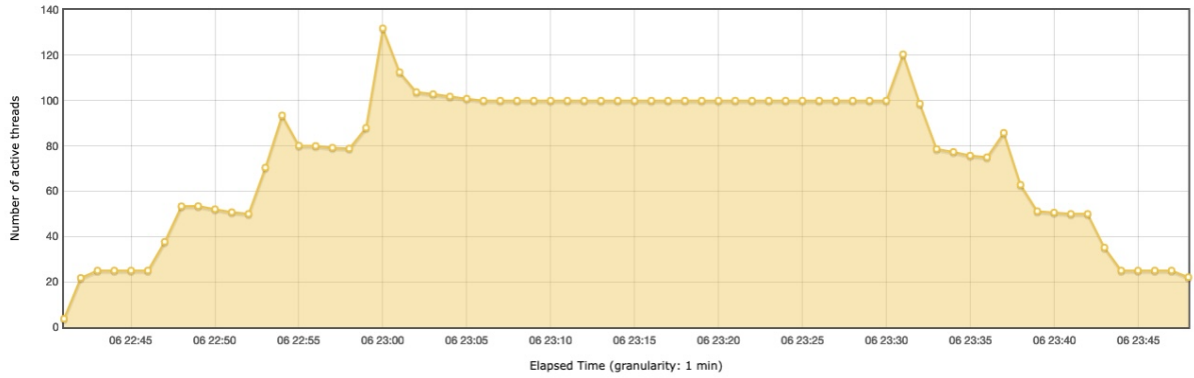
3. Spike Testing: Spike test is performed to check the behavior of the application during unexpected spikes. There are two spikes in the below configuration when the load of the application is suddenly increased to 500 at two instances. The timings are calculated with the below formula:

   Ramp up time of second record (Start-up time) = Shutdown Time of 1st record
   Initial Delay = (Initial delay + Startup time + Hold Time) of previous record.

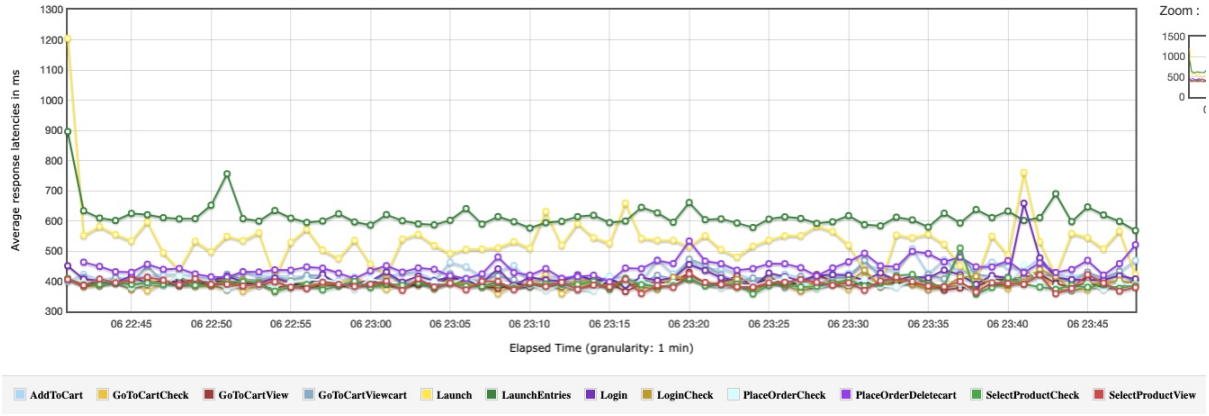


Results and Observations

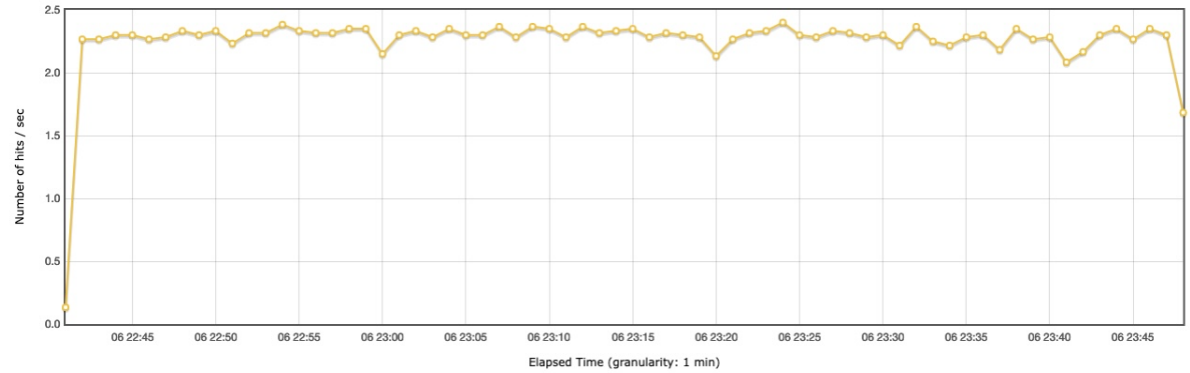
1. Load Testing
   a. All the requests passed without any errors
   b. Total throughput of the application was clocked at 138 requests/min which is considered to be a good server performance.
   c. The average response time of the requests was captured at 4.3 seconds which is a bit higher than the standard response time of 4 seconds. The reason for this response time going up is due to the larger response time in the requests LaunchEntries and Launch. This happened due to loading of the products in the home page.
   d. The Active Threads overtime were as per the thread group configuration.



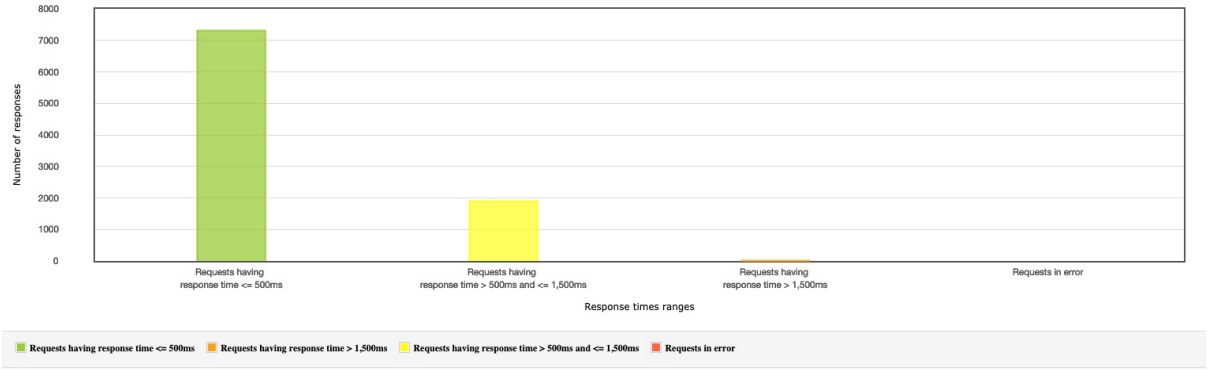
   e. From the Latency graph we can see that for the launch request the latency was on the higher side. This happened due to the delay in loading the products in the Home page.



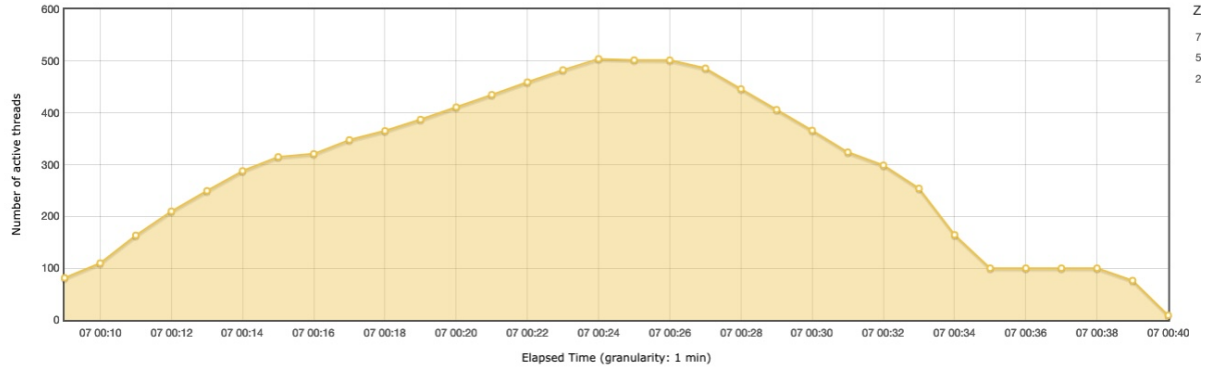
f. The hits per second is on the higher side and the throughput graph indicates that as the load increased the throughput of the application increased.



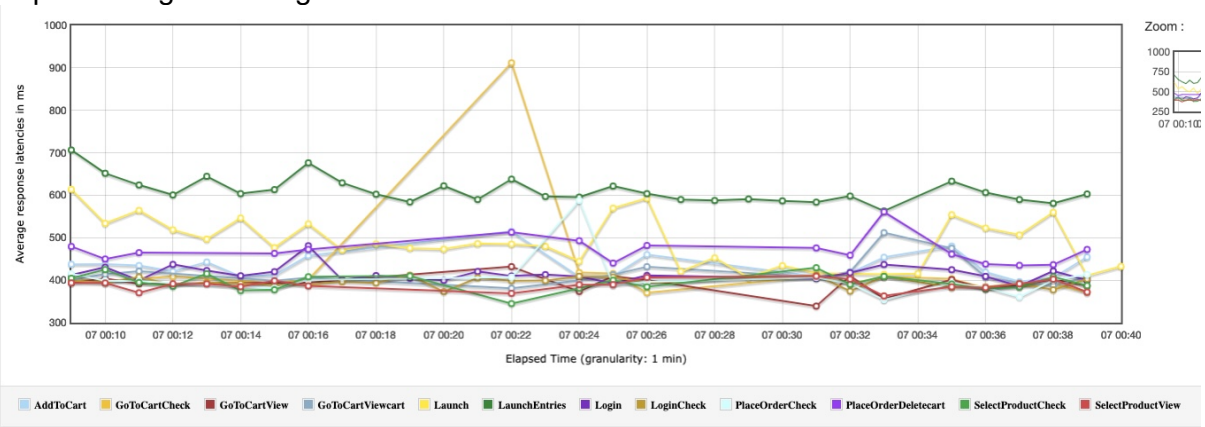
g. Maximum of the requests took less than 500ms to execute. This shows that the application behaved really well under the load of 100 virtual users.



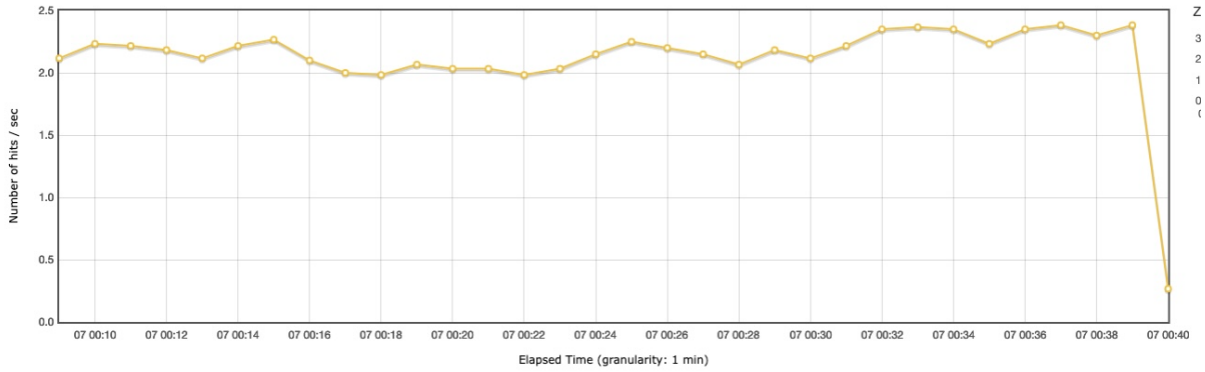
2. Stress Testing
   a. All requests passed without any errors.
   b. The throughput of the application was clocked at 130 requests/minute.
   c. The Average Response time was clocked at 4.5 seconds which is slightly on the higher side as compared to the standard response time of 4 seconds. This is due to the Launch entries and Launch requests that clocked higher response times due to the product loading in the product page.
   d. The Active Threads overtime shows the active threads overtime as configured in the thread group.



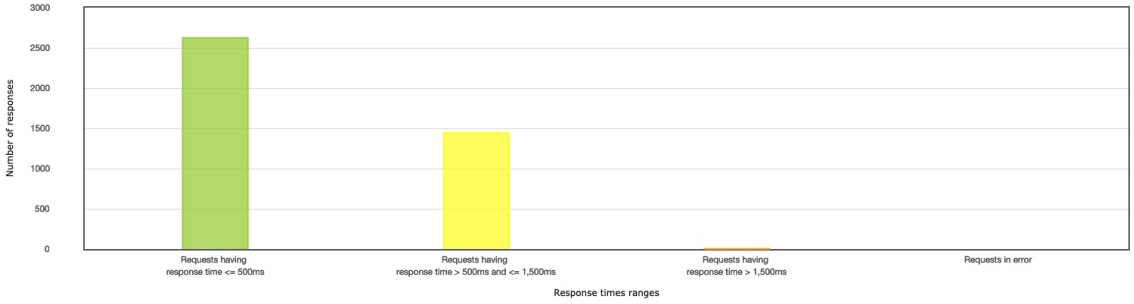
e. A spike in the latency for the Go to Cart request was clocked during the test. Latencies for other requests performed consistently with the latency for Launch request being on the higher side.



f. Throughput of the application decreased drastically towards the end of the test.

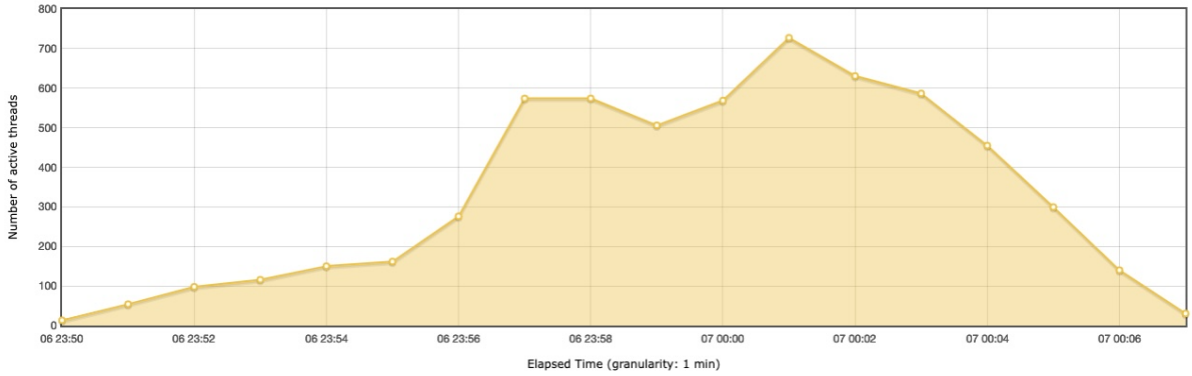


g. There were only very few requests that took more than 1500 ms response time. Most requests were clocked under 500ms

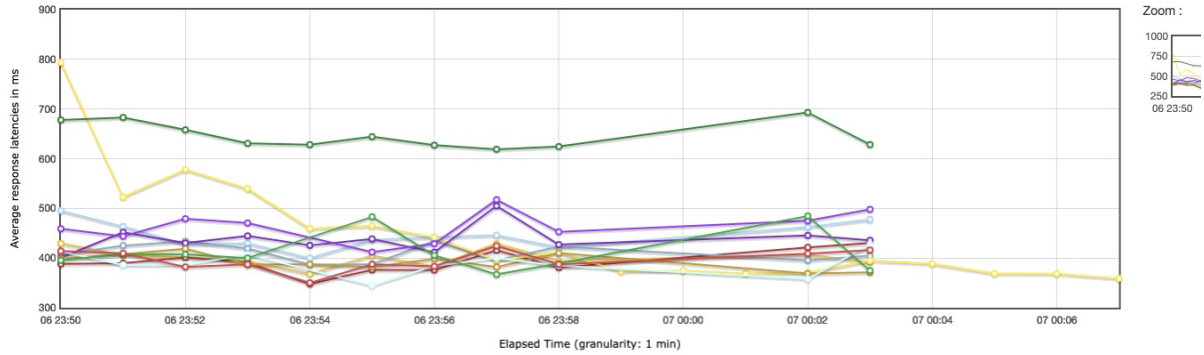


3. Spike Testing
   a. All the requests passed without any errors.

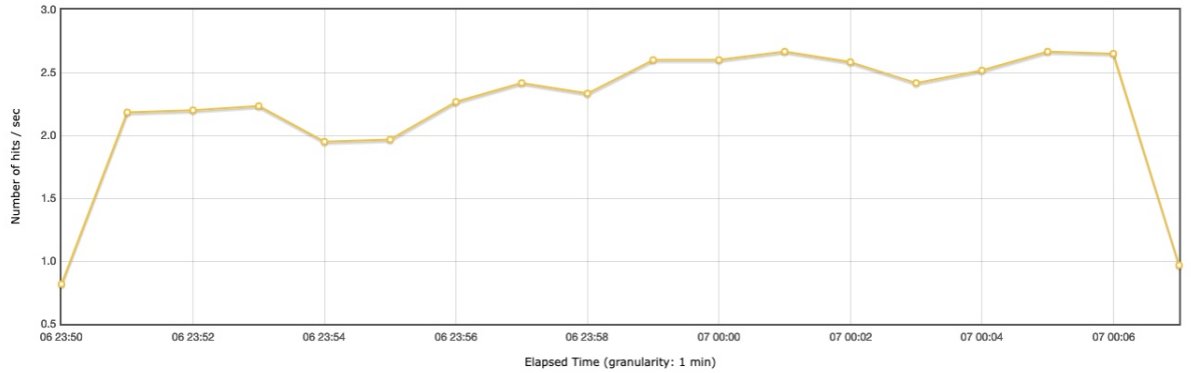
b. The throughput of the application was clocked at 142.8 requests/minute which is considered to be a real good throughput despite the unexpected spikes.
c. The Average response time was clocked at 4.4 seconds. The Launch Entries and the Launch method had response times on the higher side.
d. The Active Threads Overtime graph corresponds with the thread group configuration and also shows the spikes when the load was increased abruptly.



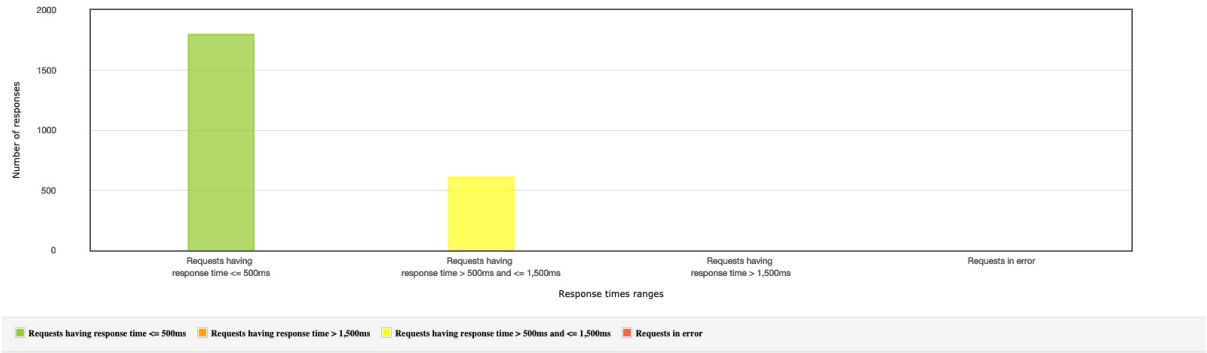
e. The Latency of the Launch and Launch Entries requests were clocked on the higher side and shows that the time taken to load the products was considerably higher.



f. The Hits per second also was clocked on the higher side for most of the requests. However, there was a linear increase when there was a spike.



g. Most requests were clocked below 500ms which is considered to be good despite the unexpected spike in load.



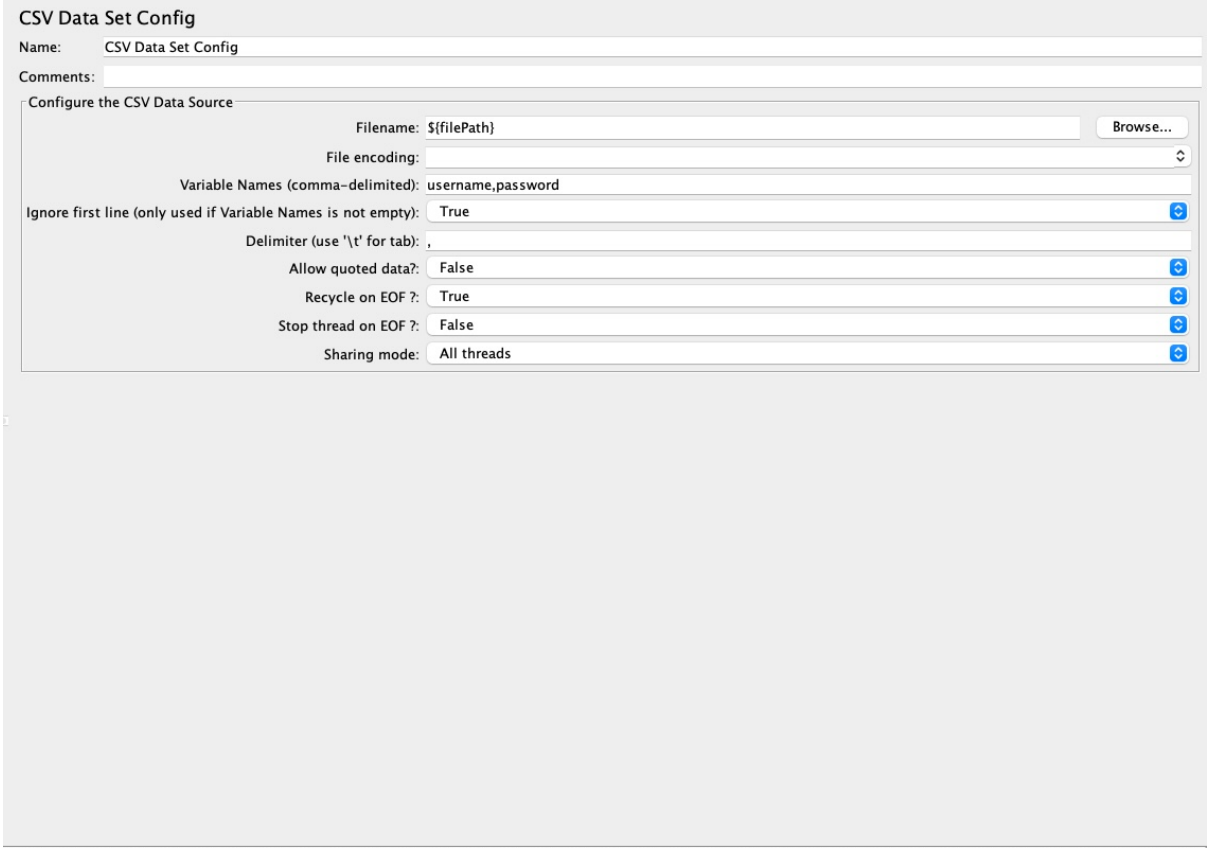
Scenario # 3: Define a way to pass parametrized login credentials for logging into an application with 10 different users.

A CSV was created and saved in the root folder where the test resides.

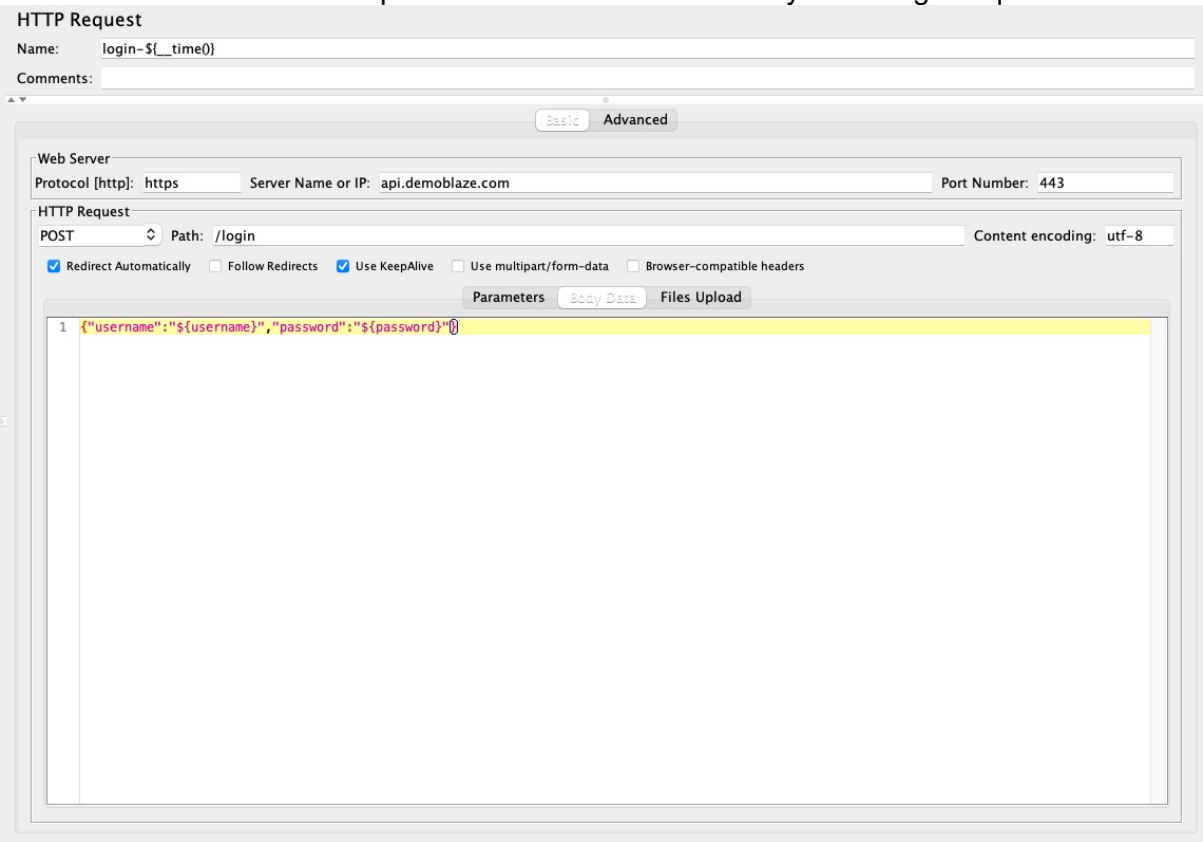
users

Username	Password
test_demo_user+1@gmail.com	s3cret
test_demo_user+2@gmail.com	s3cret
test_demo_user+3@gmail.com	s3cret
test_demo_user+4@gmail.com	s3cret
test_demo_user+5@gmail.com	s3cret
test_demo_user+6@gmail.com	s3cret
test_demo_user+7@gmail.com	s3cret
test_demo_user+8@gmail.com	s3cret
test_demo_user+9@gmail.com	s3cret
test_demo_user+10@gmail.com	s3cret

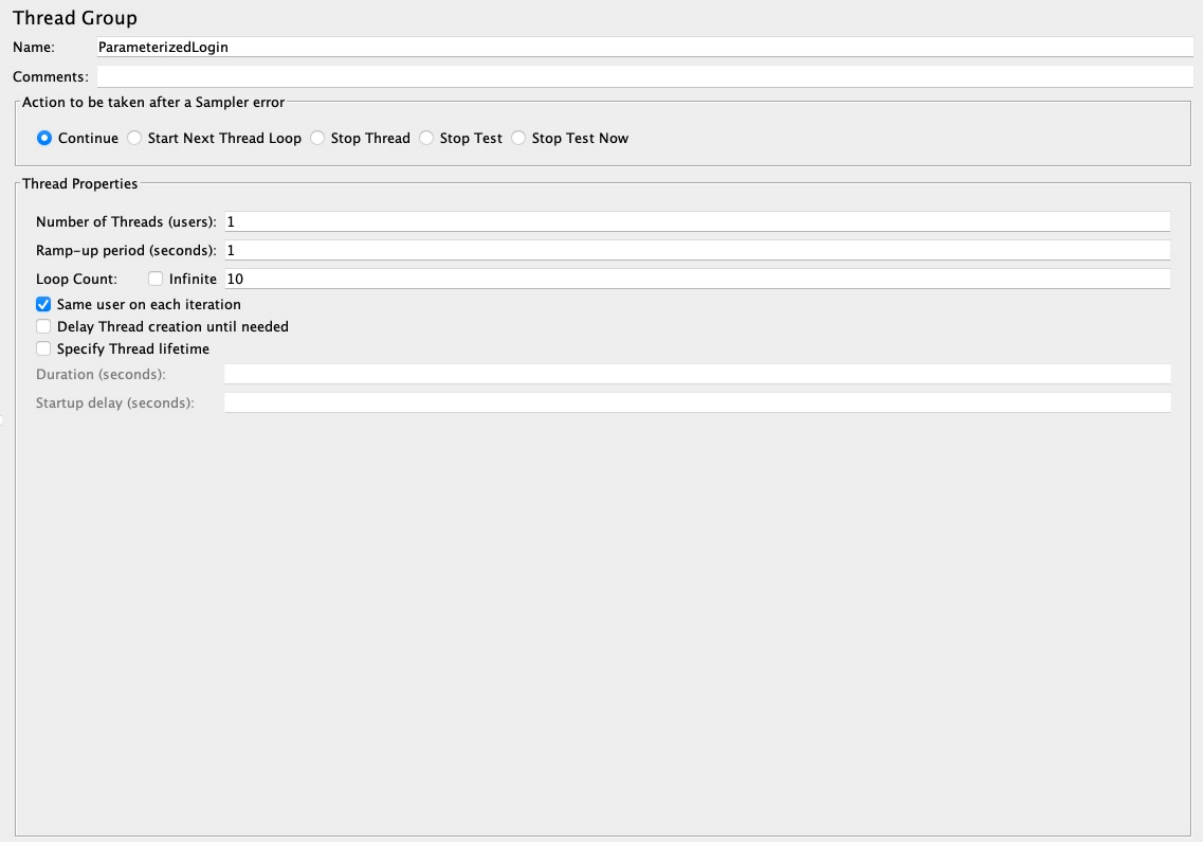
The JMeter component, CSV Data Set Config was added to fetch the data from the CSV.



The variables username and password were used in the body of the login request.



The thread group was configured to have with 1 thread to run with a loop of 10.



10 different usernames and passwords were picked up from the CSV and passed to the request body.

Scenario # 4: Define the pacing required for the application load test for the below specifications:

Total number of iterations in one hour - 15000

Total Number of users - 1000

Total flows – Search flow 50%, Place order - 35 % and Replace order - 15%

Test Setup

Application under Test (AUT): http://www.automationpractice.com

Note: This website was chosen for pacing calculation since, the previous website did not have Search and Reorder functionality.

Test Scenarios

Script #	ScriptName
1	Search
2	PlaceOrder
3	Reorder

Test Features

1. Each test has a number of transactions but each transaction should be executed in a sequence for 1 scenario. So, multiple transactions belonging to one scenario for example Place Order are put inside Critical Section Controller, so that the execution takes place in sequence.
2. I have used Throughput Controller to control the percentage load for each scenario.
3. Think time is added to each request with the help of a Constant timer

User Distribution

Script Name	Percentage Allocation	User Distribution
Search	50	500
PlaceOrder	35	350
Reorder	15	150

Think Time

Think time is added to each request to simulate a realistic load on the application.

Scenario	Transaction	Think Time (ms)
Search	Search_InSearchBar	2000
	Search_SelectSearchAutoComplete	2000
Total		4000
Place Order	PO_AddToCart	4000
	PO_Checkout_Proceed	2000
	PO_Checkout_Summary_Proceed	10000
	PO_Checkout_Address_Proceed	5000
	PO_Checkout_Shipping_Proceed	5000
	PO_Checkout_Select_Payment_Option	2000
	PO_Checkout_Confirm_Order	10000
	PO_Checkout_Back_To_Order	10000
Total		48000
ReOrder	RO_GotoMyAccount	2000
	RO_GotoOrderList	2000
	RO_Select_Order	10000
	PO_Summary_Proceed	5000
	PO_Address_Proceed	5000
	PO_Shipping_Proceed	5000
	RO_Select_Payment_Method	5000
	PO_Checkout_Confirm_Order	10000
Total		44000

Response Time

The test was executed for 1 user with 1 second ramp up time, no loop and without the think time to determine the response time of each transaction.

Scenario	Transaction	Response Time (ms)
Search	Search_InSearchBar	2752
	Search_SelectSearchAutoComplete	2814

Total		5566
Purchase Order	PO_AddToCart	4100
	PO_Checkout_Proceed	1468
	PO_Checkout_Summary_Proceed	1882
	PO_Checkout_Address_Proceed	1432
	PO_Checkout_Shipping_Proceed	1535
	PO_Checkout_Select_Payment_Option	1535
	PO_Checkout_Confirm_Order	3324
	PO_Checkout_Back_To_Order	1899
Total		17175
ReOrder	RO_GotoMyAccount	2117
	RO_GotoOrderList	2752
	RO_Select_Order	6488
	PO_Summary_Proceed	1417
	PO_Address_Proceed	1246
	PO_Shipping_Proceed	1689
	RO_Select_Payment_Method	1162
	PO_Checkout_Confirm_Order	2946
Total		19817

Iteration Per Second
Given total number of iterations = 15000 in 3600 seconds
Total number of iterations in 1 second (IPS) = 15000/3600 = 4.2

Script Name	User Distribution	Iterations Per Second (User distribution percentage of IPS)
Search	500	2.1
Place Order	350	1.5
Reorder	150	0.6

Pacing Calculation
Pacing is the time difference between each complete iteration of a business flow. The formula for calculating pacing is as below:

Pacing = (No. of Threads/Iterations Per Second) – (Response Time + Total Think Time)

Script Name	Response Time (s)	Think Time (s)	Iterations Per Second	Pacing (s)
Search	5.6	4	2.1	228
Place Order	17.2	48	1.5	168
Reorder	19.8	44	0.6	186
Total				582

Pacing Validation
The calculated value of pacing should always be greater than the sum of total response time and total think time.
Total Response Time (RT): 42.6 s
Total Think Time (TT): 96 s
Total time: RT + TT = 138.6 s

Hence, the pacing calculated value is more than the calculated sum of response and think time. So, this is a valid pacing value for the iterations.

Target Throughput Calculation

Throughout is defined as requests per unit of time. The time is calculated from the start of the first sample to the end of the last sample. This includes any intervals between samples, as it is supposed to represent the load on the server. The formula to calculate throughput is:

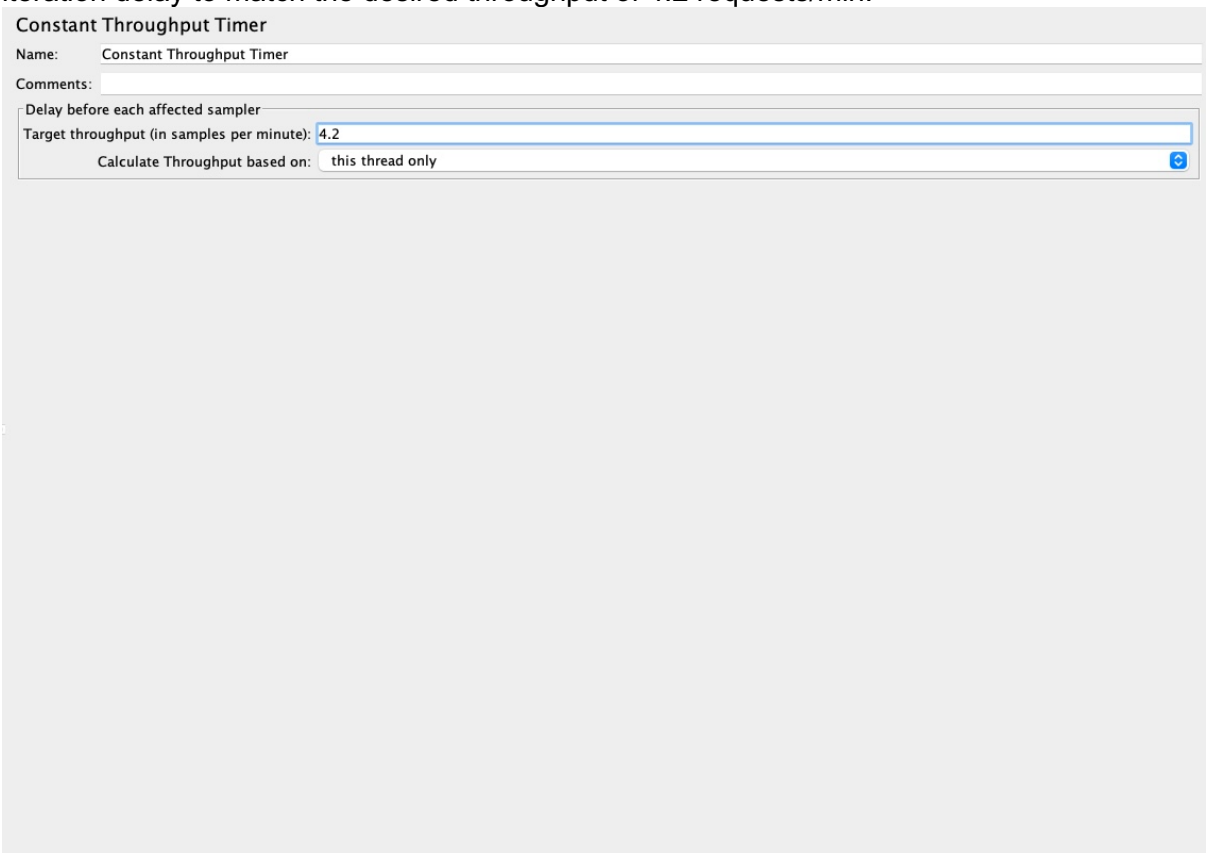
Throughput = (Number of requests)/(Response Time + Think Time + Pacing)

Script Name	Response Time(s)	Think Time (s)	Pacing (s)	Throughput (Requests/second)
Search	5.6	4	228	0.01
Place Order	17.2	48	168	0.03
Reorder	19.8	44	186	0.03
Total				0.07

So, the target throughput calculated is 0.07 requests/sec which is equal to 4.2 requests/min

Implementing Pacing in JMeter test

Pacing is implemented in the JMeter test with the help of Constant Throughput Timer. Target throughput value is specified in the test and upon execution of the test, JMeter adjusts the iteration delay to match the desired throughput of 4.2 requests/min.



Issues and Recommendations

Different type of performance tests was carried out on the application. Some issues captured during the performance test execution are as follows:

1. The response time of the application suffered while loading the products in the Launch page.
2. Request failures when applying a load of 1000 users for 15 seconds.

Some recommendations to improve application performance:

1. Implement caching in the product list page to render products faster.