CS 441: Distributed Objects for Cloud Computing Project

Title: JPetStore Application

Team Members:

1. Shanmathi Mayuram Krithivasan smayur3@uic.edu - 660768712
2. Heeba Mohammedali hmoham23@uic.edu653864689
3. Kuttabadkar Yogeeta Monica ykutta2@uic.edu - 661868770
4. Nandhinee Neelakandan nneela2@uic.edu - 677734738

TABLE OF CONTENTS

Application Overview: JPetStore

JPetStore6 is an online website that allows its registered users to purchase animals. It allows for various operations such as user sign-up\login, browse catalogue of animals, selecting items to cart, checkout and complete purchase order.

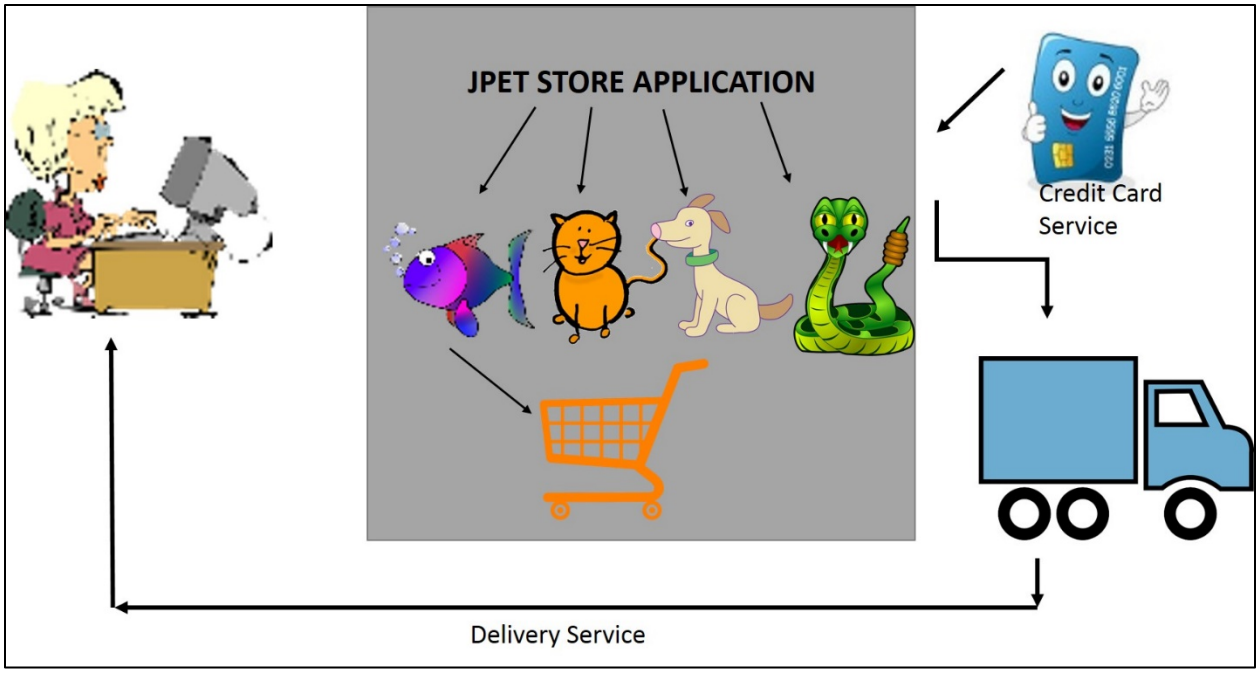


Fig: Overview of JPetStore application

Any user can search through a catalog of pets that are of various types and traits. When the customer likes an item, they select is and add to their shopping cart. Once they have completed their shopping, they proceed to checkout page (before which the user must sign in\register as new user, if not already done) and submit an order which contains details like their shipping & billing address, payment type and card information. Each customer's account is unique by their userid and stores details like name, address, contacts, etc.

MyBatis JPetStore6 is a Java EE web application based on Spring and Stripes, which is implemented using open-source freeware. It consists of around 28 classes, several JSP files and XML configuration files. To host this application we have chosen the Tomcat container. The below figure shows a simple sitemap of the application.

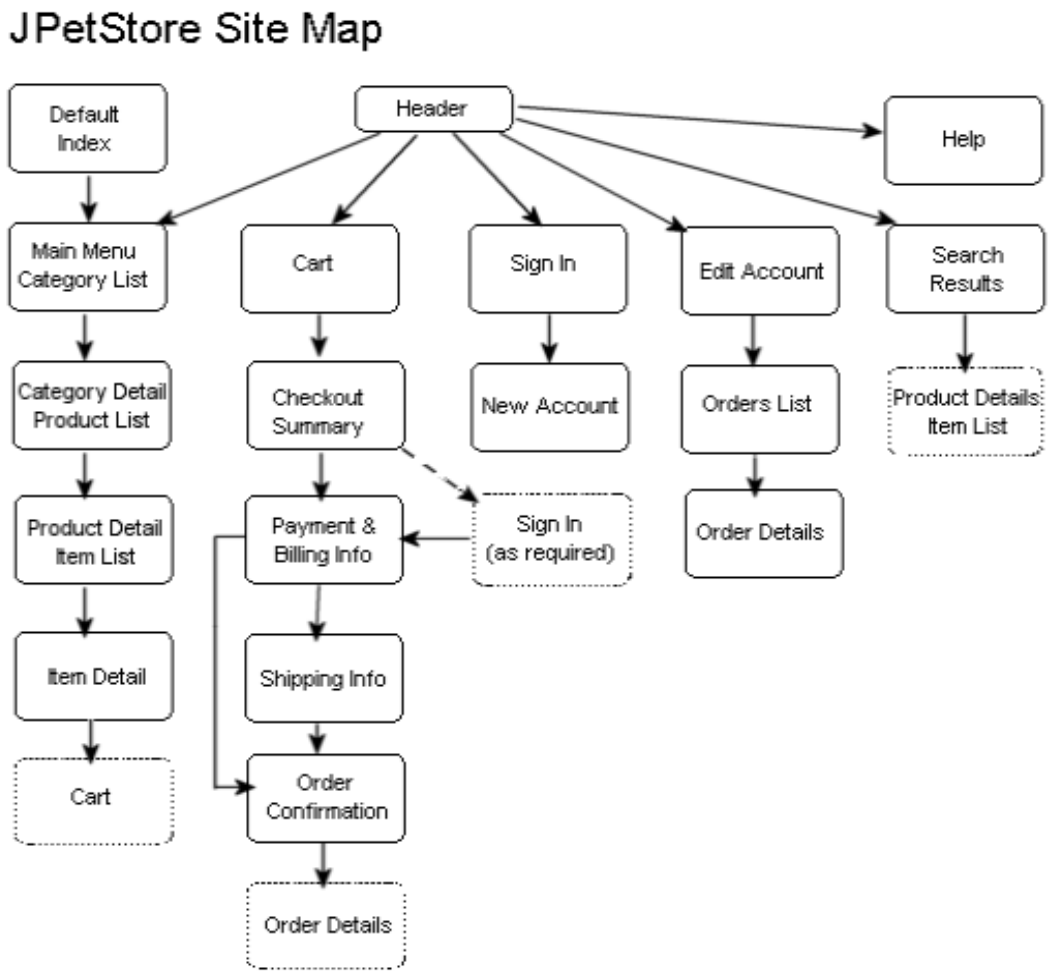


Fig: Site Map of JPetStore Application

Technical Characteristics of Application:

Development IDE: Eclipse or NetBeans
Supported Web Server: Tomcat or Jetty
Programming Language: Java
Framework: Java Structs and Spring (gives a Model-View-Controller framework which allows for easy separation of business concerns)
Database\ Object Mapping: Framework called iBATIS Database (uses simple XML descriptor file to describe the inputs and outputs of an SQL statement. It allows the programmer to simply pass a JavaBean into a Mapped Statement as a parameter (input) and receive a JavaBean as a result (output)
Complier\Software Development Kit: Sun JAVA SDK

Goal of the Report:

The goal of this report is to explain efforts made to host the JPetStore6 application onto the cloud using provider MS Azure, with best possible provisioning\de-provisioning strategies. In order to

perform this we required the application to be under some form of load, which is achieved using JMeter.

For this purpose, the steps performed under this report can be summarized as:

- JPetStore application is divided into various business transactions, each depicting a sequence of steps followed to achieve a single task
- JMeter scripts are written for each transaction, keeping validations and checks as necessary
- Application is hosted on the cloud by choosing one among the various deployment techniques
- Load is managed onto the application in the cloud using JMeter provided parameters
- Different strategies are tested for scaling up\down in MS Azure and results are documented
- Conclusion presents the best possible strategies for given load amounts

Local Deployment

For deployment, JPetStore needs to be built with Apache Maven. The only requirements are JDK >=1.5 and Maven >= 2.0.8

Running JpetStore with Tomcat:

1. Download and Unzip Tomcat
2. Download JpetStore source code
3. From Eclipse, Import -> Maven -> Existing Maven Projects -> unzipped jpetstore source
4. Delete parent pom references from pom.xml
5. Change the stripe dependency
6. Run as maven project to build war files. The war file gets created in the “target” directory
7. JpetStore is now accessible from the browser

Cloud Deployment of the application:

The JPetStore application is not a heavy weight application which is deployed as a Web App on Azure.

Why web app?

It supports Java using which JPetStore was implemented. Since we did not need more control into the server environment , hosting as web app was considered over Hosting as cloud service. The web app provides high availablility which is enabled through built in load balancing and traffic manager to manage heavy traffic loads. Also, JPetStore did not require change in the service on which it is running. Hence, the JPetStore app was hosted as a Webapp on Azure.

To create a WebApp:

1. We implemented the application in MS Azure as a webapp
2. It was deployed on Tomcat server8.0
3. A new webapp was created from the marketplace
4. A new Resource Group was created as "webappmysql-group" and the corresponding "App Service plan/Location" was selected
5. A unique URL for the webapp was also given

To deploy the application:

1. Go to the URL https://jpetstore123.scm.azurewebsites.net/ to find the Kudu Services
2. Go to Powershell under Debug Control tab and go to the following path site/wwwroot/bin/apache-tomcat8..0.27/webapps (D:\home\site\wwwroot\bin\apache-tomcat-8.0.27\webapps)

3. Import jpetstore.war file inside this directory.

4. The entire application is now imported under the directory jpetstore and the homepage can now be accessed from the link http://jpetstore123.azurewebsites.net/jpetstore/

JpetStore Program Structure:

```
/jpetstore                    <-- Maven pom.xml present here.
 /src
  /main/
   /java                      <-- Java code present here.
    /org/
     /mybatis
      /jpetstore
       /domain          <-- Business domain objects present here.
       /persistence     <-- Mapper interfaces present here.
       /service               <-- Application logic present here.
       /web
        /actions              <-- Presentation logic (actions) present here.
   /resources           <-- Non java files present here.
    /org
     /mybatis
  /jpetstore
       /persistence     <-- Mapper XML files present here
    /database
   /webapp
    /css
    /images
    /WEB-INF                  <-- web.xml and applicationContext.xml present here.
     /jsp                     <-- JSP files present here.
```

Fig: Structure of JPet Project

Business Transactions of JPetStore6:

For this project we have divided the application into 6 major transactions. Each transactions has a specific set of steps to be completed in order to achieve an end result. Further, each step is documents as a HTTP request in the corresponding JMeter Script. A Brief description of all transactions are given below:

1. Register New Users
2. Browsing Items, Adding to Cart and completing Checkout
3. Cart Operations - Updating\Removing items from cart
4. Manage Account - Editing user account information
5. View Orders - For users to view all orders purchased

Description of Business Transactions:
Transaction# 1

Transaction Name: Register New Users:

ID	Name	STEPS	VALIDATIONS
1	Enter the Store	1. Open the JPetStore application 2. Select "Enter the Store" link	
2	Catalog Home Page	1. User is redirected to view the catalog home page	
3	Sign In	1. Select "Sign-In" link on the top of the page	
4	New User Sign Up	1. Select "New User Registration" present on the bottom	
5	Register Now	2. Enter Username 3. Enter Password 4. Enter Password again for confirmation 5. Click on "Submit"	Userid should be unique for first time users to sign up
6	Catalog Page	1. On Successful account creation the user is redirected to catalog home page	If user sign up is unsuccessful, Exception is encountered. If user sign up is successful, user is redirected to catalog page
7	Sign Up	1. Select the "Sign-In" option at the top of the page	

8	User Sign- In	2. On catalog page, select "User Sign-In" option 3. Enter Username 4. Enter Password 5. Click on "Submit"	User and Password must be authenticated. If authentication fails, Invalid User login message is displayed.
9	User Home Page	1. After sign in user is shown their home page	Sign in is successful when user name is displayed in the home page as "Welcome <User_Name>"
10	Sign-Out	1. Select Sign-out option present on the top of the page	

Fig: Table showing steps of New User Registration

Transaction# 2

Transaction Name: Browsing Items, Adding to Cart and completing Checkout

ID	Name	STEPS	VALIDATIONS
1	Enter the Store	1. Open the JPetStore application 2. Select "Enter the Store" link	
2	Catalog Home Page	1. User is redirected to view the catalog home page	
3	Sign In	1. Select "Sign-In" link on the top of the page	
4	Default User Sign-In	1. Enter Username 2. Enter Password 3. Click on "Submit"	User and Password must be authenticated. If authentication fails, Invalid User login message is displayed. If successful, Welcome "<Username" is displayed on the home page
5	Browsing through Multiple Items and adding to cart (Loop through multiple times)	1. Select Type of Animal (Fish, Dog, Cat, Reptile, Bird) 2. Select from the sub items of main category 3. Select "Add to cart" (or) 4. Select Item name to view details of the item 5. Select "Add to Cart"	

6	Checkout Cart	1. On completion of shopping, go to Cart by selecting the symbol on top of the page 2. Select Proceed to Checkout 3. Validate all information (billing details, address), select "Continue"	
7	Confirm Order	1. After validating all information, select "Confirm"	Unique Order ID is generated for each order. If successful, Thank you message is displayed. If unsuccessful , Exception is returned
8	Sign-Out	1. Select Sign-out option present on the top of the page	

Fig: Table showing steps of Browsing Catalog, Adding to cart and checking out

Transaction# 3

Transaction Name: Cart Operations (Update\Remove items from Cart)

ID	Name	STEPS	VALIDATIONS
1	Enter the Store	1. Open the JPetStore application 2. Select "Enter the Store" link	
2	Catalog Home Page	1. User is redirected to view the catalog home page	
3	Sign In	1. Select "Sign-In" link on the top of the page	
4	Default User Sign-In	1. Enter Username 2. Enter Password 3. Click on "Submit"	User and Password must be authenticated. If authentication fails, Invalid User login message is displayed. If successful, Welcome "<Username" is displayed on the home page
5	Browsing through Multiple Items and adding to cart (Adding 10 items to cart –repeat steps 10	1. Select Type of as Fish/Dog/Cat/Bird 2. Select from the sub items of main category 3. Select "Add to cart" option	

	times)	against the item desired	
6	Update Cart (Multiple times change quantity of cart items)	1. On completion of shopping, go to Cart by selecting the symbol on top of the page 2. Update the quantity of the items in the cart 3. Select "Update" option	Check if cart is empty before updating. If empty, "Your cart is empty" message is displayed to the user
7	Remove Items from Cart	1. On completion of shopping, go to Cart by selecting the symbol on top of the page 2. Select the product to remove 3. Select "Remove from Cart" optiom	Check if cart is empty before removing item. If empty, "Your cart is empty" message is displayed to the user
8	Sign-Out	1. Select Sign-out option present on the top of the page	

Fig: Table showing steps of Cart Operations (Update and Remove)

Transaction# 4

Transaction Name: Manage Account

ID	Name	STEPS	VALIDATIONS
1	Enter the Store	1. Open the JPetStore application 2. Select "Enter the Store" link	
2	Catalog Home Page	1. User is redirected to view the catalog home page	
3	Sign In	1. Select "Sign-In" link on the top of the page	
4	Default User Sign-In	1. Enter Username 2. Enter Password 3. Click on "Submit"	User and Password must be authenticated. If authentication fails, Invalid User login message is displayed. If successful, Welcome "<Username" is displayed on the home page
5	My Account	1. Select the "My Account" option from the top of the page	User information, Account information and Profile information is displayed on the page

6	Update User Info	1. Update values in various fields of user information 2. Select "Save" button	Information should be updated on the page
7	Sign-Out	1. Select Sign-out option present on the top of the page	

Fig: Table showing steps of Manage Account

Transaction# 5

Transaction Name: View Orders

ID	Name	STEPS	VALIDATIONS
1	Enter the Store	3. Open the JPetStore application 4. Select "Enter the Store" link	
2	Catalog Home Page	2. User is redirected to view the catalog home page	
3	Sign In	2. Select "Sign-In" link on the top of the page	
4	Default User Sign-In	4. Enter Username 5. Enter Password 6. Click on "Submit"	User and Password must be authenticated. If authentication fails, Invalid User login message is displayed. If successful, Welcome "<Username" is displayed on the home page
5	Browsing through Multiple Items and adding to cart (Loop through multiple times)	1. Select Type of Animal (Fish/ Bird) 2. Select from the sub items of main category 3. Select "Add to cart" option against the item desired	
6	Checkout Cart	1. On completion of shopping, go to Cart by selecting the symbol on top of the page 2. Select Proceed to Checkout 3. Validate all information (billing details, address), select "Continue"	
7	Confirm Order	1. After validating all information, select "Confirm"	Unique Order ID is generated for each order. If successful, Thank you message

			is displayed. If unsuccessful , Exception is returned Note ORDERID generated, for viewing purpose
8	List Orders	2. Select the "My Account" option from the top of the page 3. Select "My Orders" option present on bottom of the page	Verify that "My orders" are displayed for the user
9	View Order	3. Select confirmed order ID 4. View the order details	User should see the completed order details
10	View Item	1. Select any of the listed items from the list within the order	Check if correct item details are displayed
10	Sign-Out	2. Select Sign-out option present on the top of the page	

Fig: Table showing steps of View orders

JMeter Scripts:

JMeter is used to load the application with user requests and monitor results. Each business transaction described above is made into a set of HTTP Requests and grouped into various Thread Groups. The technique followed is explained below.

1. Each business transaction is made as an Individual THREAD GROUP
2. Each step in the transaction is a HTTP REQUEST
3. Validations of the transactions are achieved using ASSERTIONS for each of the HTTP requests
4. CONTROLLERS are used to group number of steps for looping\various patterns of URL hits
5. Load is managed using thread group parameters of No. of users, No. of Iterations and Ramp up period
6. Different LISTERNERS are used for monitoring results or test runs

General JMeter Settings:

a) **HTTP Cookie Manager:** For each iteration we have choose to clear the cookies. This is reasoned as, cookies improves the speed of a request since it involves client side caching. Removing the cookies will ensure the application is tested for worst case scenarios and in the presence of cookies the performance metrics will improve.

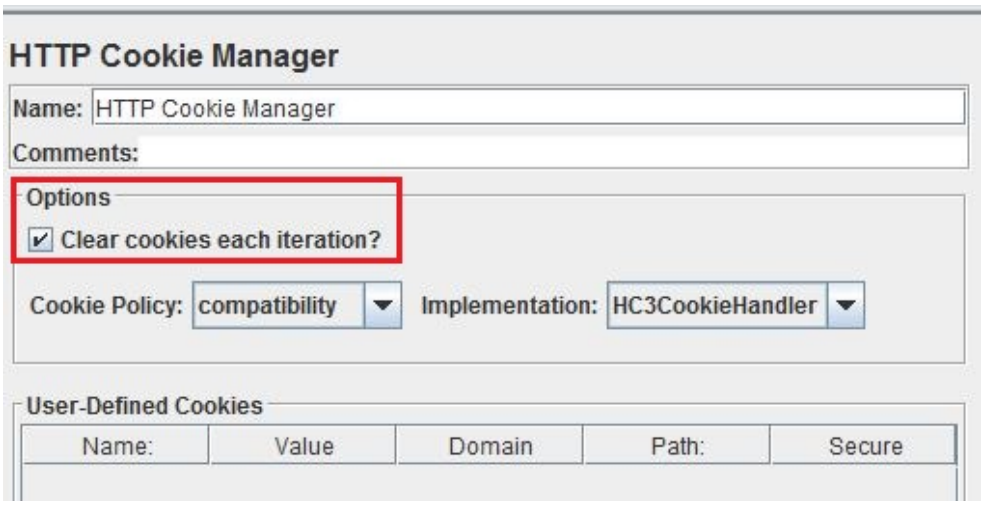


Fig: Cookie Manager Setting

b) **HTTP Cache Manager:** For each iteration the cache is also cleared. Since, caching speeds up response rates, removing this ensures we test the application for peak loads. In the presence of caching the performance metrics would improve.

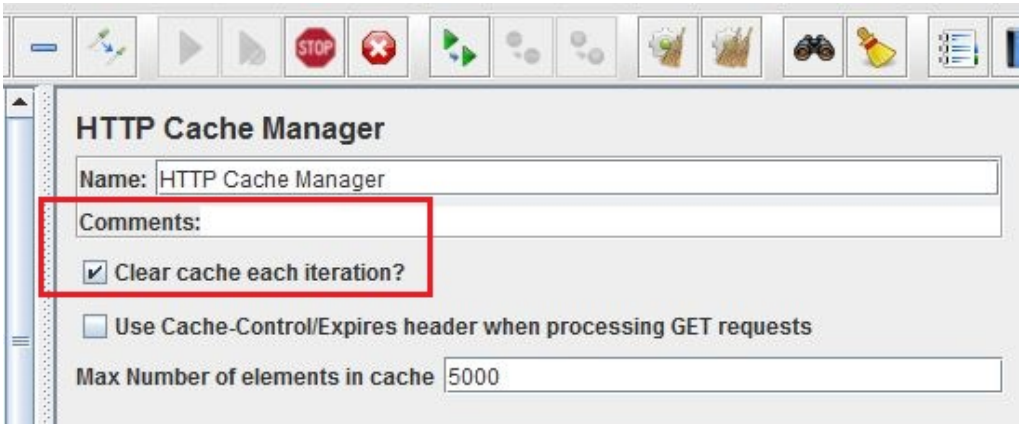


Fig: Cache Manager Setting

c) **Thread Groups:** Each business transaction is implemented as a thread group. Hence, the entire JMeter script consists of 5 major thread groups run individually as well as consecutively to document performance results.

d) **Thread Group Settings:** For locally deployed testing, the metrics were maintained to 100 users, 10 iterations with a ramp up of 5 seconds. For testing of the application deployed on the cloud, various loads were used which are documented in each test.
The entire load is calculated as SAMPLES, which was derived with a combination of

SAMPLES = NO OF USERS * NO. OF ITERATIONS * NO. OF URL HITS\ITERATION

Eg: 10,000 hits\samples = 100 users*10 iterations * 10 URL hits\ iteration

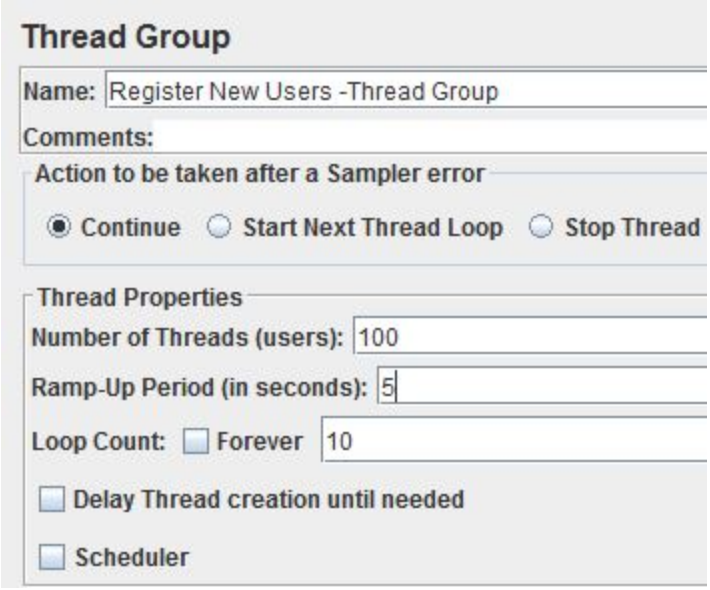


Fig: Thread Group Settings

e) **User Defined variables:** User defined variable "PATH" is used in all iterations in order to refer to the URL path (local\cloud) commonly. Hence, the home path for testing can be easily changed in one place, instead of changing multiple URL definitions. The Path variable is declared and used as,

Path http://localhost:8080/jpetstore
Usage: **${PATH}/catalog.action**

Fig: User Defined "PATH" variable

f) **Counter Variables:** Counter variables are used to increment values sequentially in order to satisfy unique key constraints within the application.

E.g.: USERID, QUANTITYNO
Usage: ${USERID}

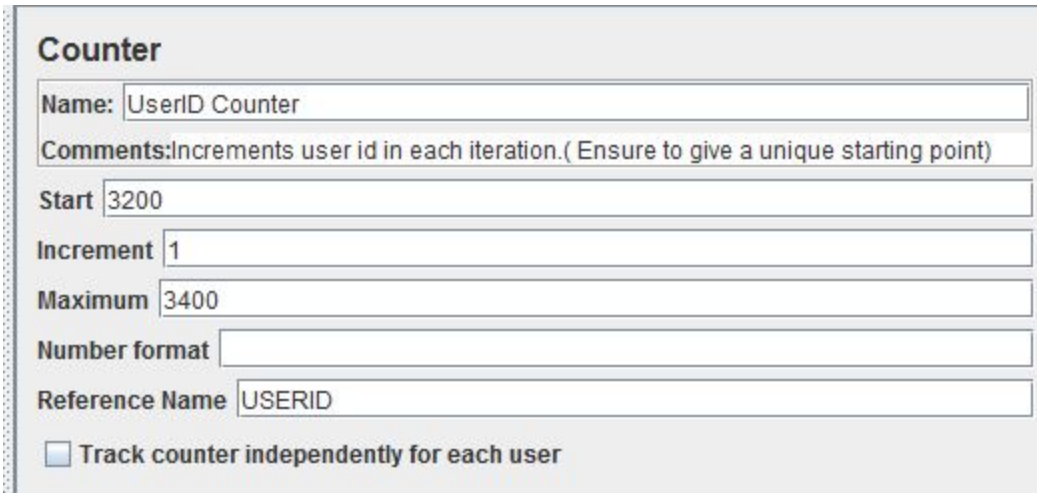


Fig: Counter for "USERID" variable

g) **Assertions**: Assertions are used to fulfil all validations in each step. Mostly Response Assertions verifying the Text Response to desired values.

h) **Controller**: Simple, Loop and Interleave controllers are used to obtain desired URL hits in specific patterns.

JMeter Scripts Detailed Descriptions:

Transaction# 1 **Transaction Name: Register New Users**

The JMeter script contains all steps in the above described transaction. Assertions are added for validations of successful new user sign up and log in. Some brief points of the JMeter script are explained below with necessary screenshots.

All JMeter general settings described in the above section are used for this script. The entire transaction is written within a "Simple Controller". This controller was chosen as it was desired to have step up step execution of each http request.

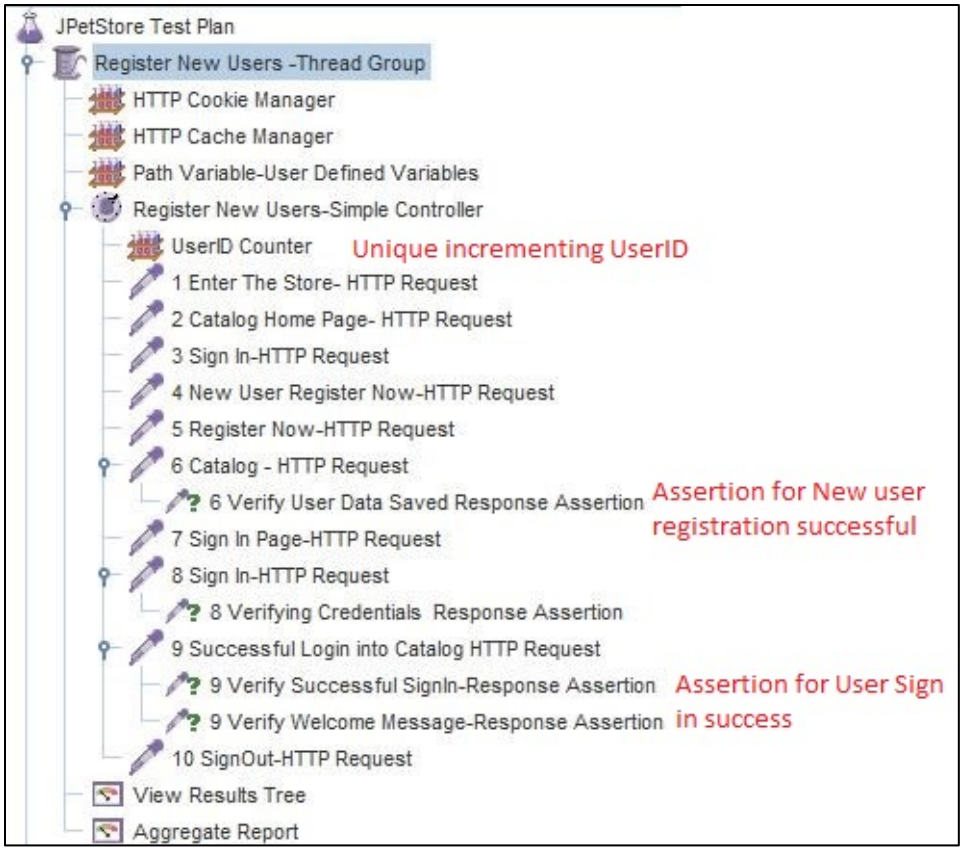


Fig: Screen shot showing steps of New User Registration

1. **User ID Counter**: This counter is set to increment by 1 for every user. It is reset to start from a new value on every test run to ensure that unique users are added to the application. This fulfills the primary key constraint of the Users table. If is used during the POST operation of the HTTP request as a paramenter.

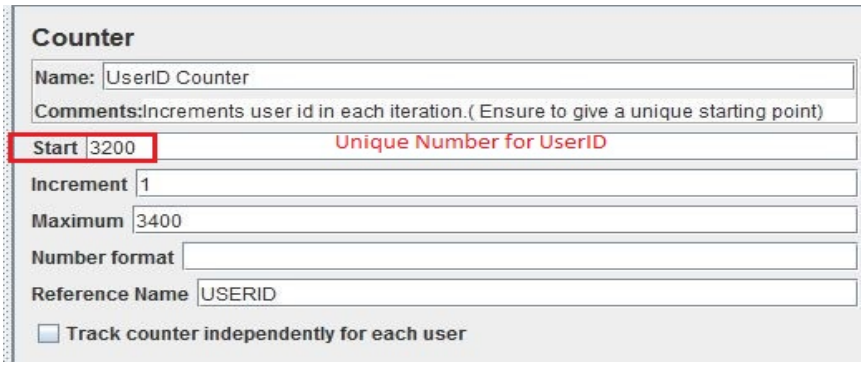


Fig: Screen Shot showing UserId counter

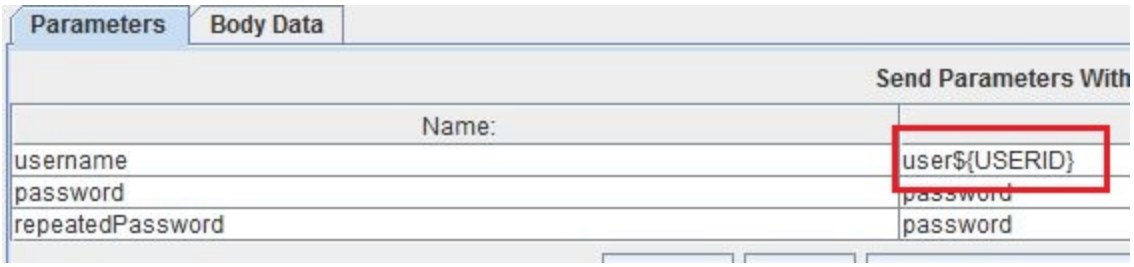


Fig: Screen Shot for Usage of UserID counter

2. **Register Now- HTTP Request:** This is a POST http request which sends user entered data to be saved. The response is a page redirection if successful, else an exception occurs

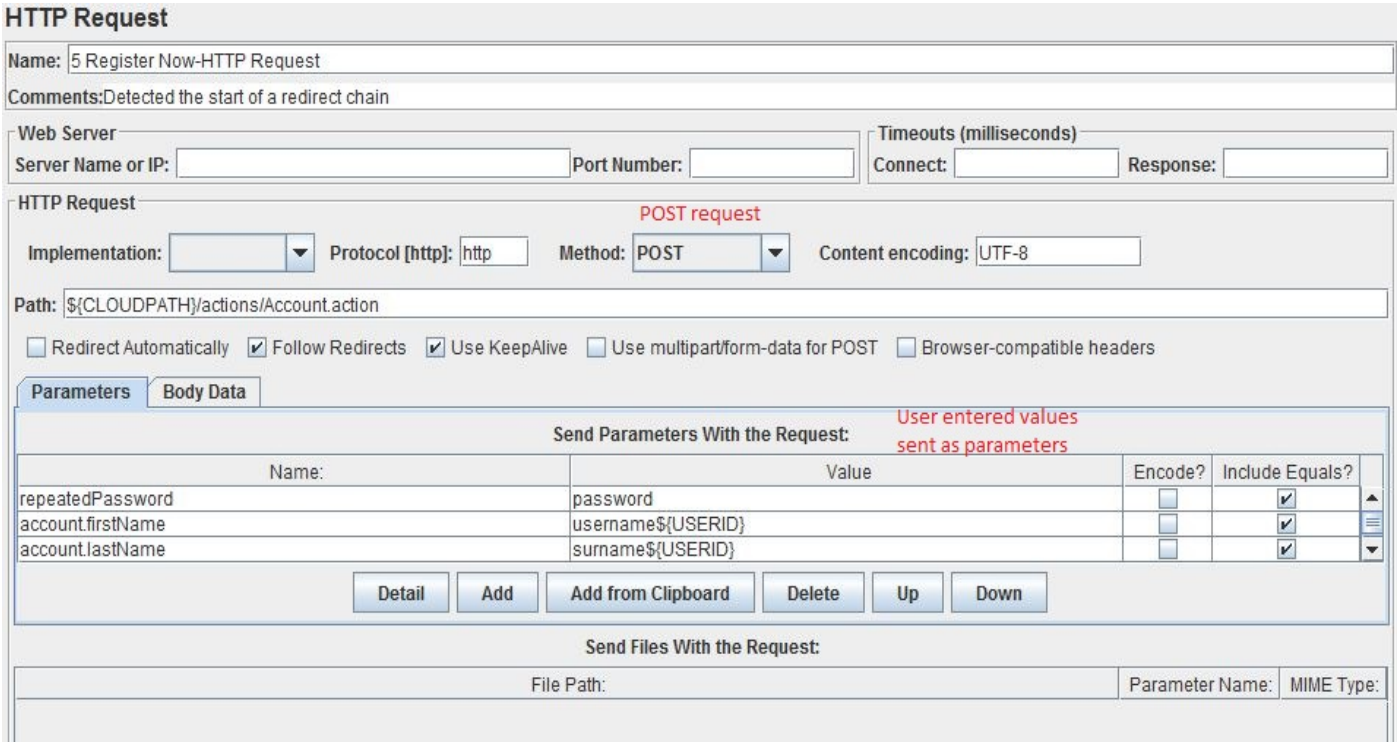


Fig: Screenshot of HTTP Request-Register Now

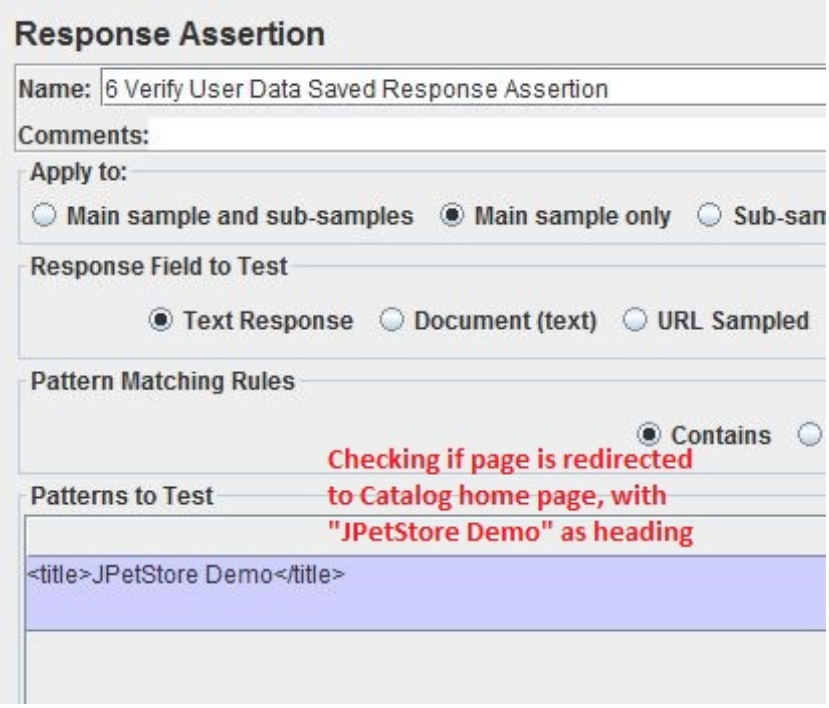


Fig: Screenshot of HTTP Request-Register Now Assertion

3. **Sign in- HTTP Request:** This is a POST http request which sends user entered credentials for validation. The response is a page redirection to home page, else an exception occurs. In the assertion we validate the User welcome message and invalid message error.

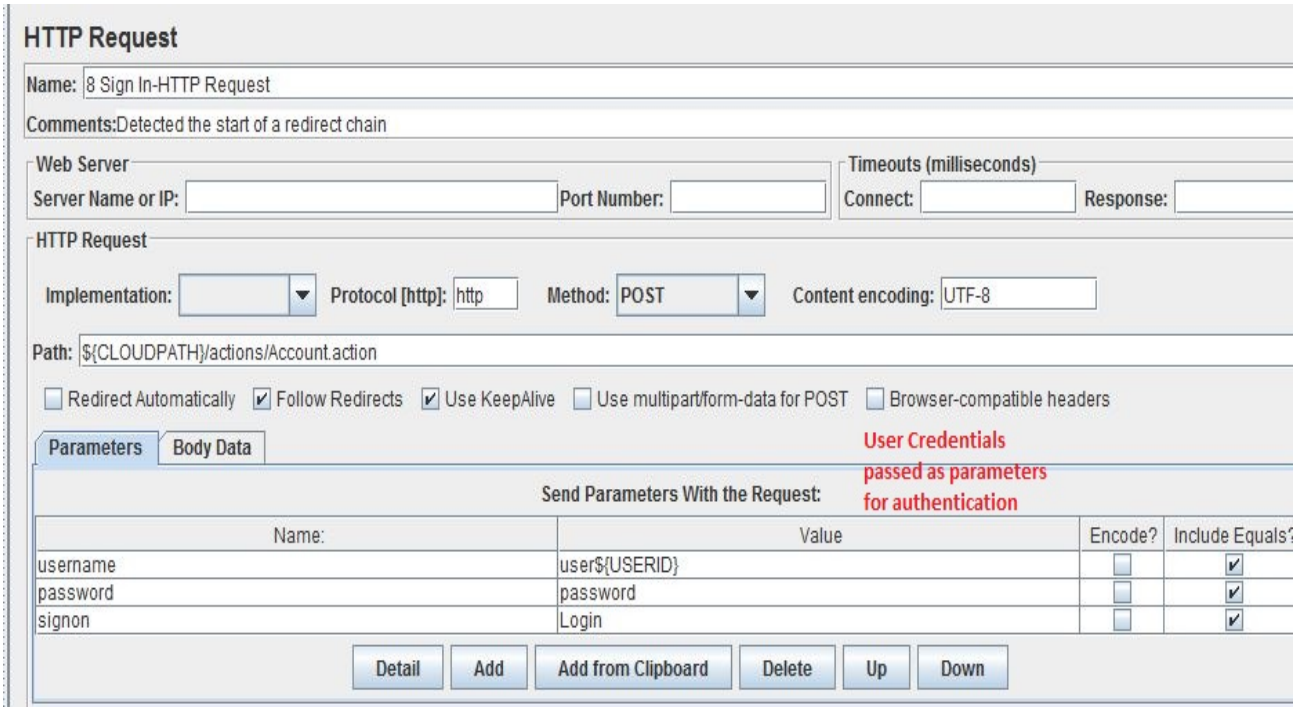


Fig: Screenshot of HTTP Request-Sign In of new user

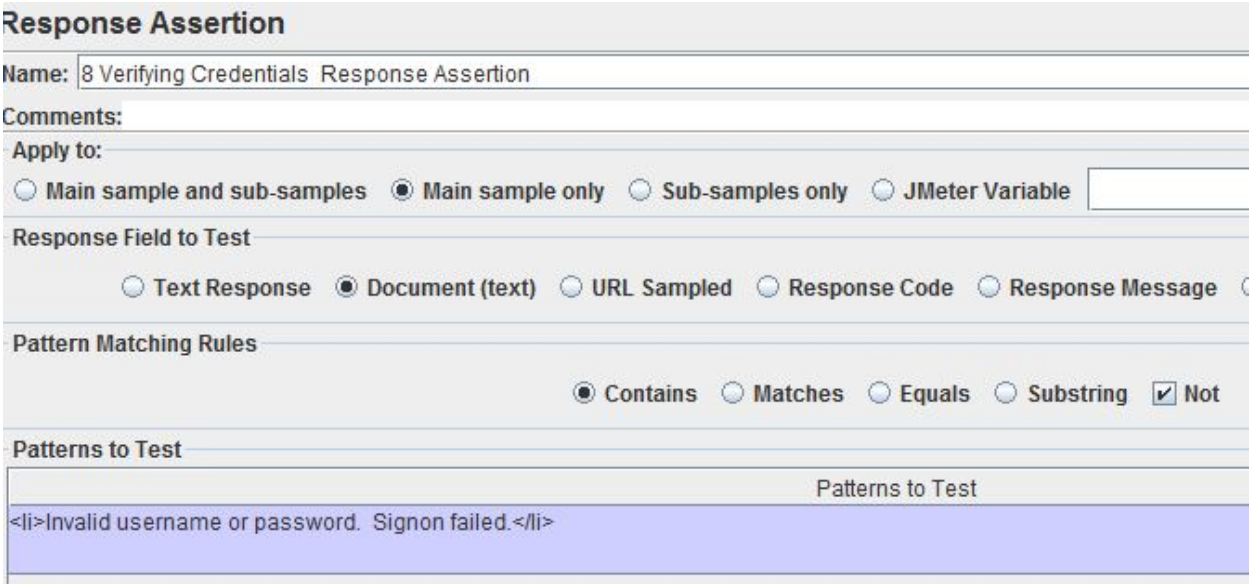


Fig: Screenshot of HTTP Response Assertion-Checking that Invalid username message is NOT present in response

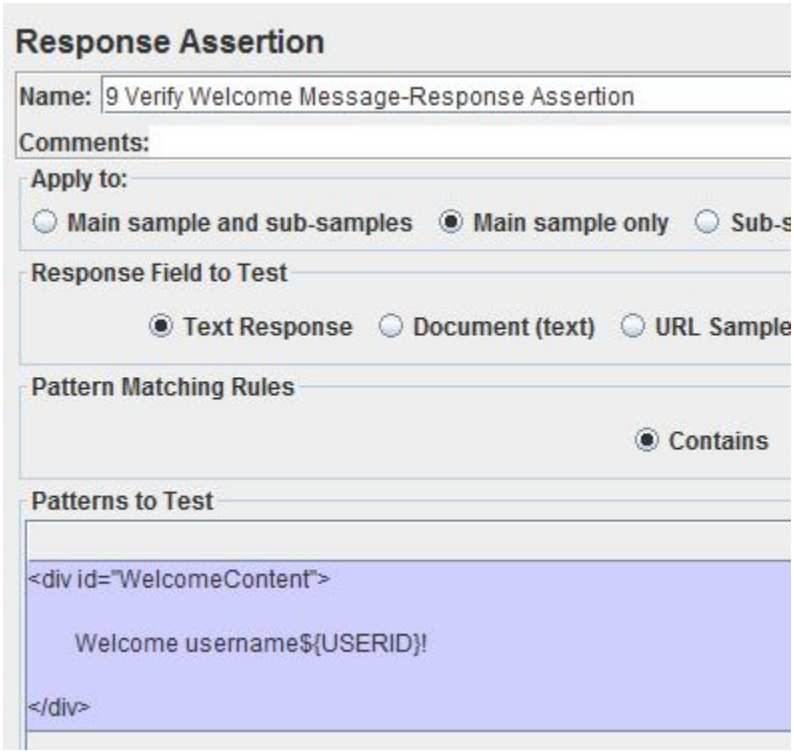


Fig: Screenshot of HTTP Response Assertion-Checking Welcome <Username> is present in response

When we run this thread group, we see multiple URL's generated. For this thread group the rough URL hits per iteration is 10. All assertions can be checked for pass\fail criteria by observing the "View results as a Tree" listener. If all assertions are passed, the results are all shown in green with 0% error.

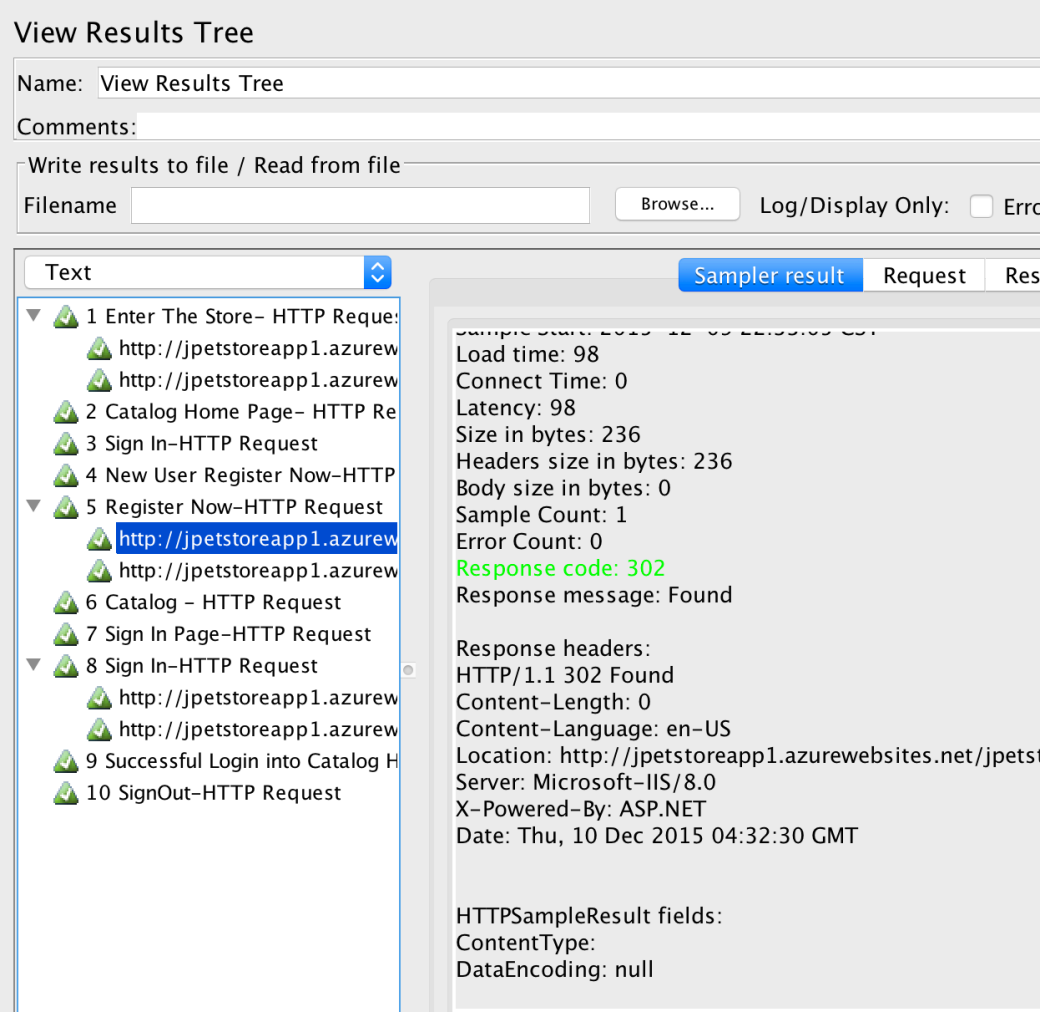


Fig: Screenshot of Thread Group Run- displaying all URL and their assertion values

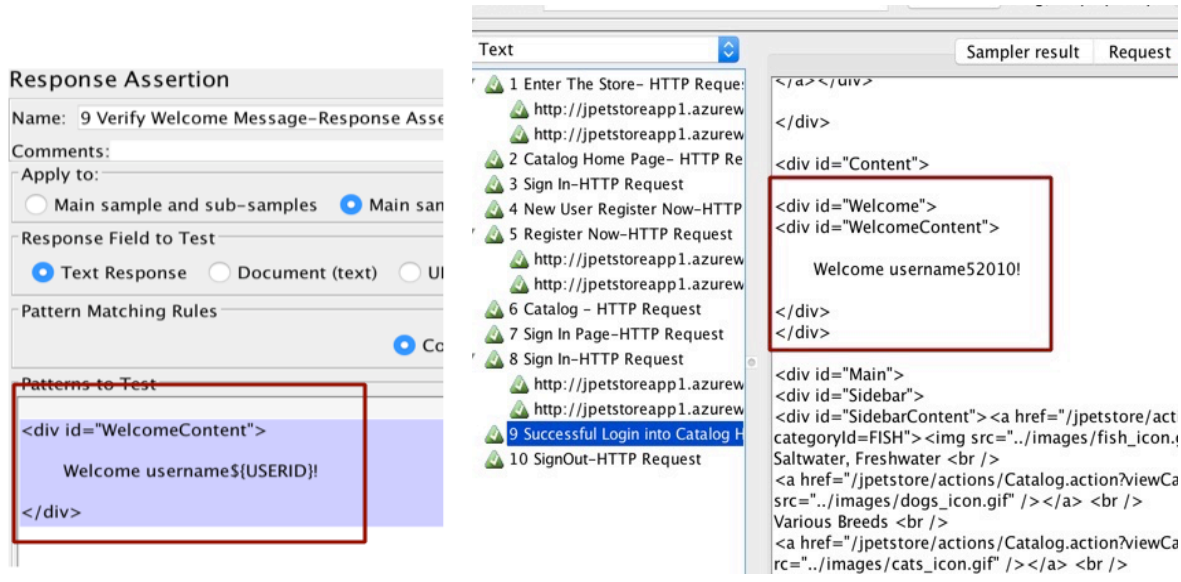


Fig: Screenshot of Thread Group Run- displaying Assertion in rule and the assertion value

Transaction# 2 **Transaction Name: Browsing, Adding to Cart and Checkout**

This JMeter script contains all steps in business transaction 2 described above. All JMeter general settings are used as well. Multiple simple controllers are used to group together common operations. Brief points are described below:

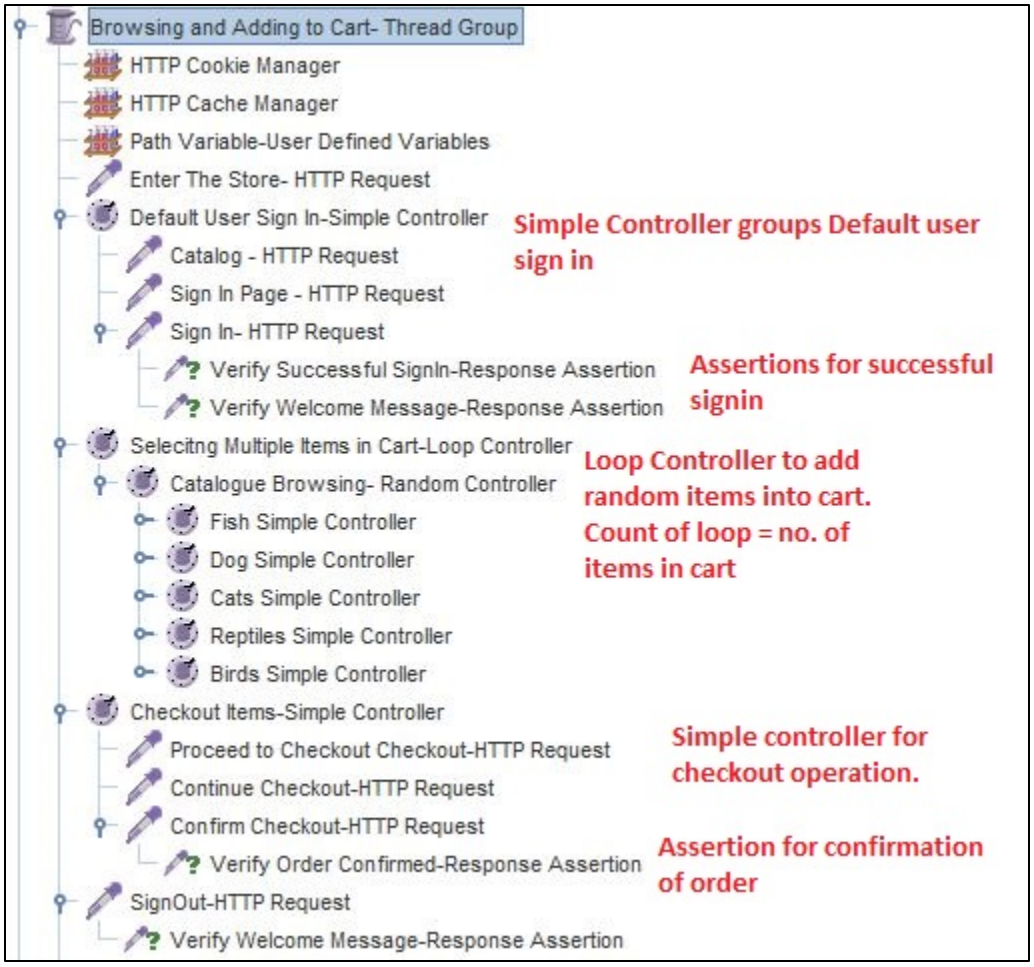


Fig: Screenshot of complete Transaction#2

1. **Default User Sign in- Simple Controller**: This controller groups the operations of the default user sign in ("j2ee" user). And confirms the successful sign in by validating the "Welcome ABC" message.
2. **Selecting Multiple Items in Cart- Loop Controller**: This loop controller contains nested random controller which gives us the flexibility to generate random items In the cart from the available list of choices. The value of the loop is the number of items that will be added to the cart.

   The user can add an item to the cart in 2 ways:

   a) By selecting the "Add to Cart" next to the listed item
   b) By selecting the "Add to Cart", present at the bottom of the page in the details of the item

Both ways are generated by the written script. This ensures maximum realistic hits on the application, as different users can use the application differently.

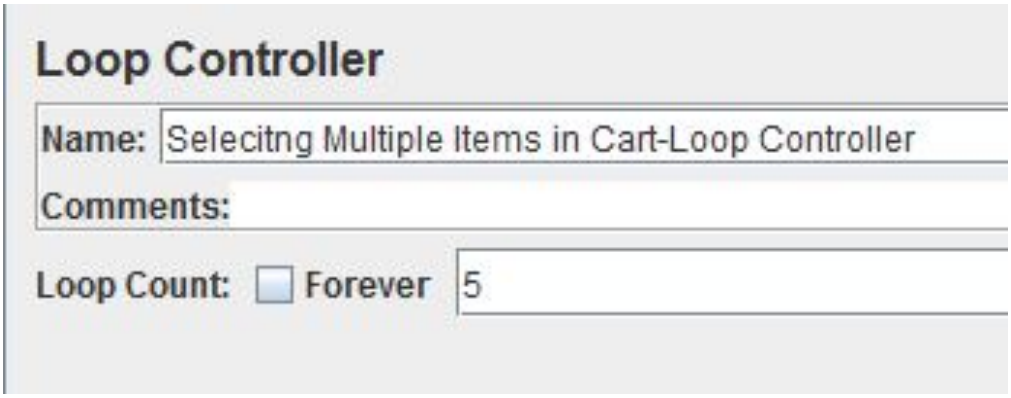


Fig: Loop Controller Count

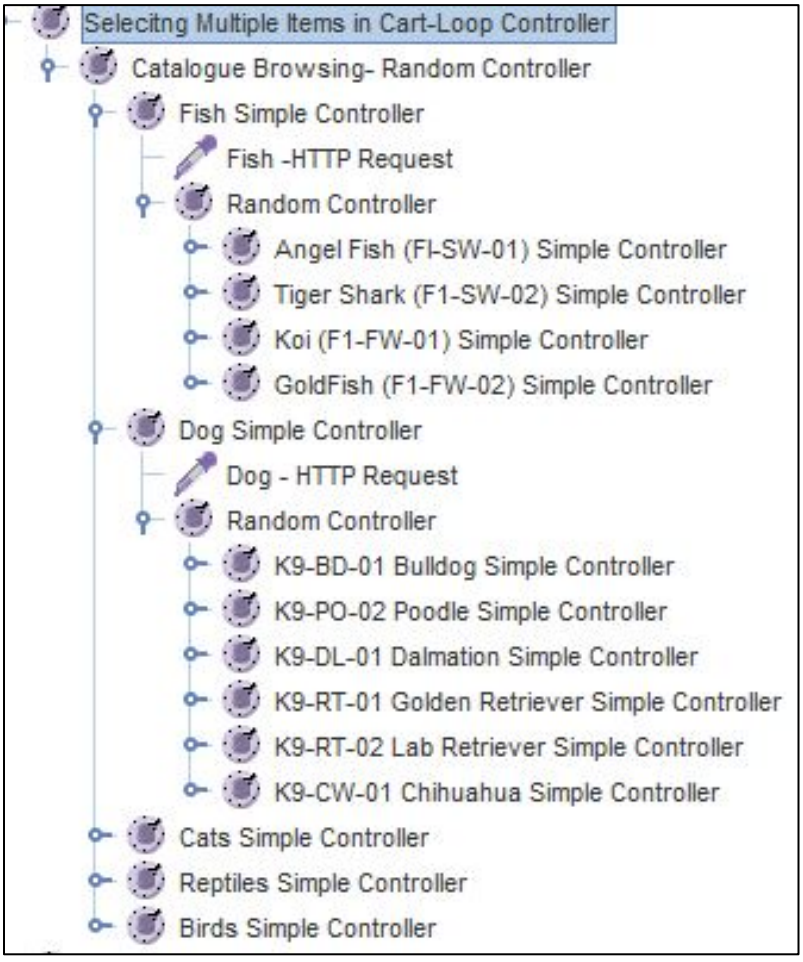


Fig: Screen shot – Selecting random items into cart

3. Checkout Items- Simple Controller: This controller groups the operations of checking out cart items and confirming purchase order. It validates the confirmation message “Thank you for your order” in the response text.

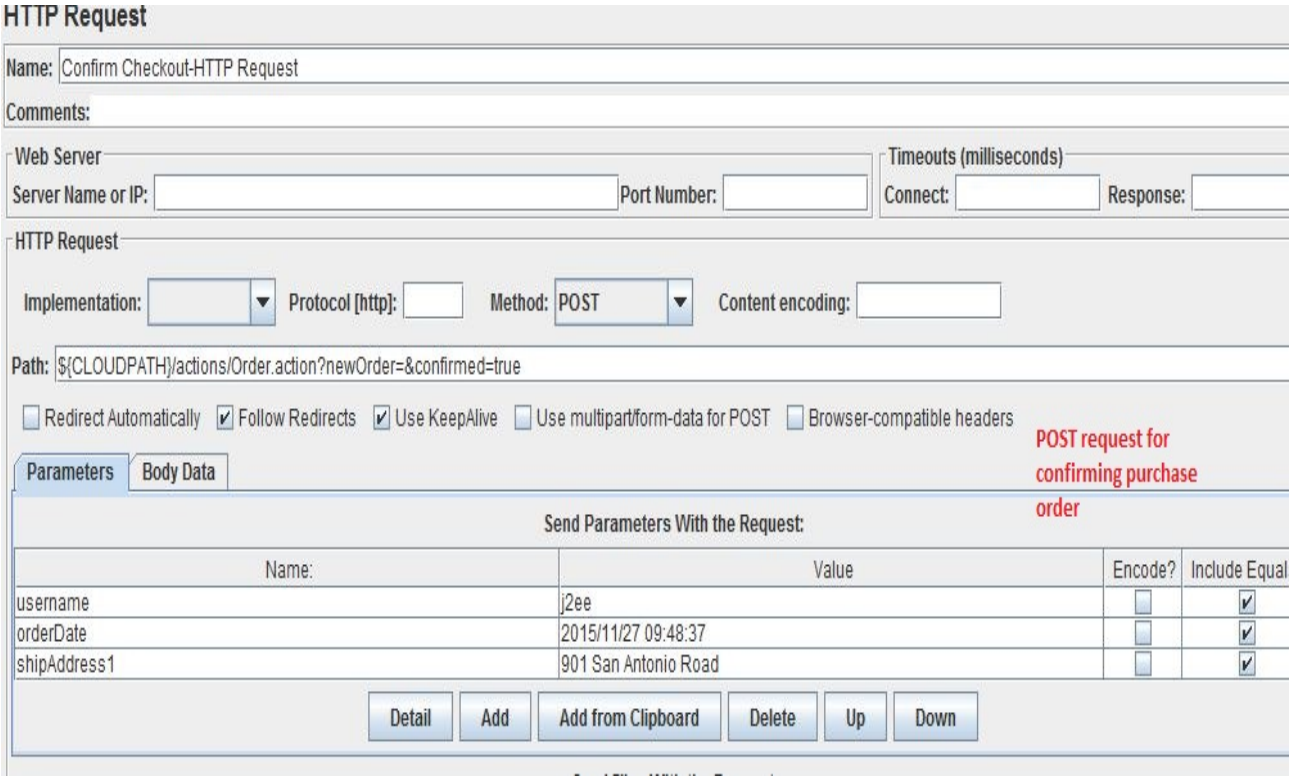


Fig: Screen shot – HTTP Request to confirm checkout

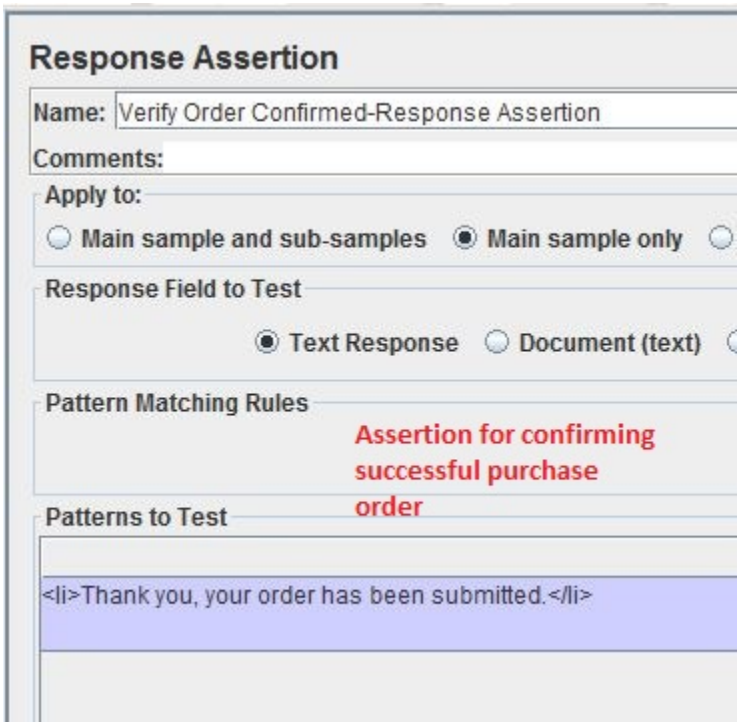


Fig: Screen shot – Assertion to check validation of confirmation message

When we run this thread group, we see multiple URL's generated. For this thread group the rough URL hits per iteration is 20. All assertions can be checked for pass\fail criteria

by observing the "View results as a Tree" listener. If all assertions are passed, the results are all shown in green with 0% error.

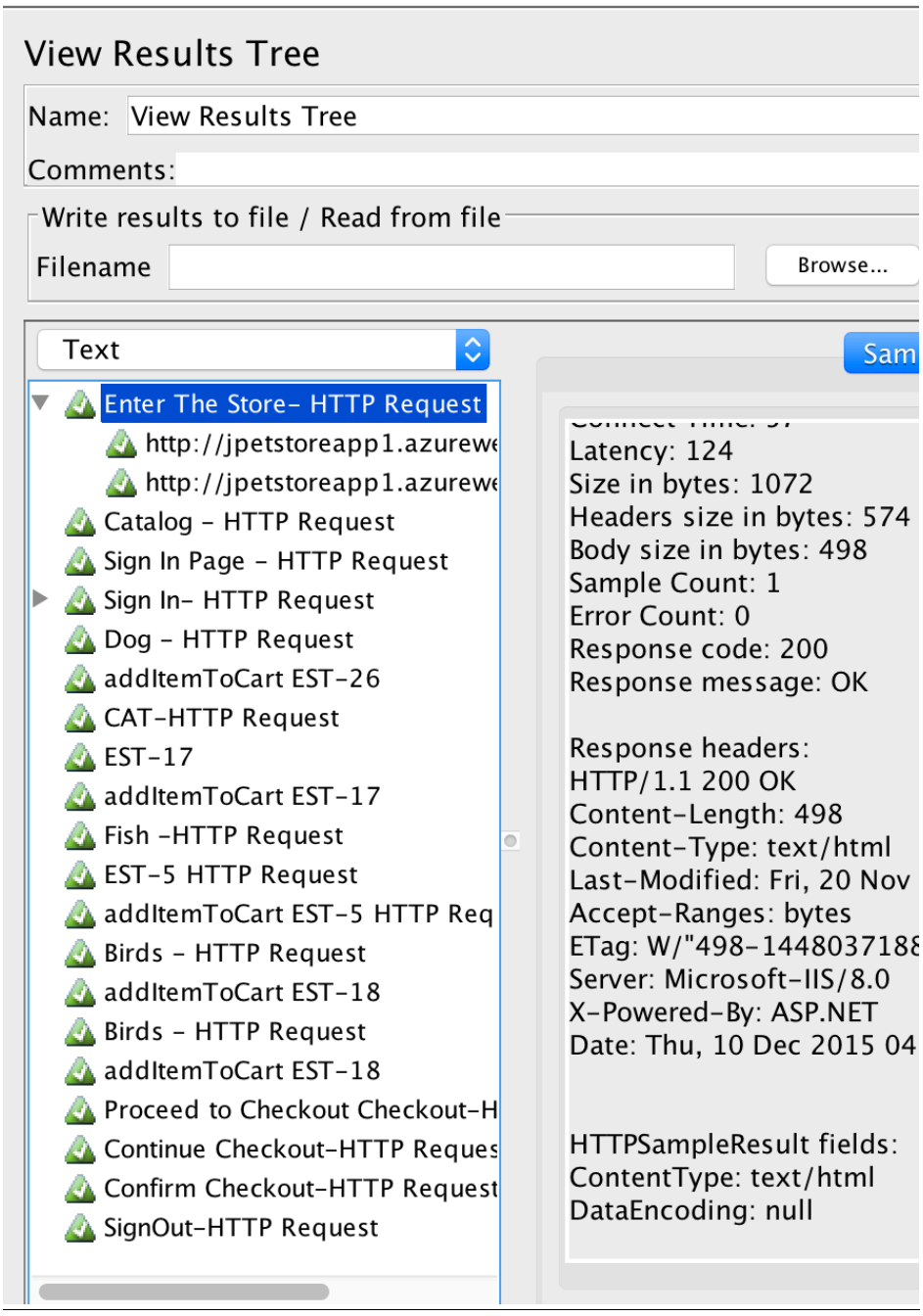


Fig: Screenshot of Thread Group Run- displaying all URL's and their assertion values

Transaction# 3 **Transaction Name: Cart Operations (Update\Remove) items from cart**

This thread group contains all requests to sign in, add items to cart and update the cart by performing update of quantities and remove item from cart operations. It contains several controllers are validations as described below. Items are added one by one and update is

performed on the added items followed up delete, and the next item is added ad process continues.

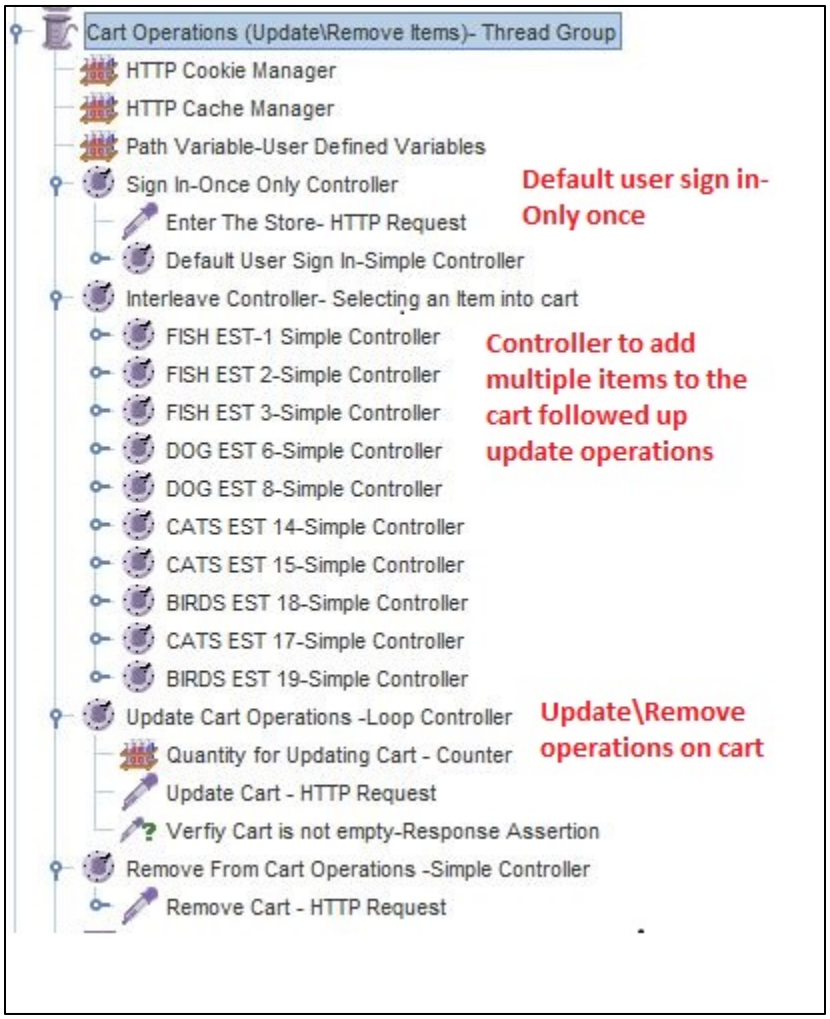


Fig: Screen shot- Thread group for Update Cart operations

1) **Sign in- Only once controller-** This controller executes only once for each user, this is added to ensure that the user can perform update operations specifically and no extra burden is added to sign in each time. The validations are checked (same as above scripts) for successful user login.
2) **Interleave Controller-Selecting an item to cart:** This is an interleave controller which allows us to select one random item (among Fish, Dog, Bird, Cat) add to the bag and proceed to perform update on this item, followed by removing the item and adding the next one. The count of iterations on the thread group decides the number of items added to cart.
3) **Update Cart Operations- Loop Controller:** This is a loop controller that indicates the number of items the item will be updated and saved. We increase the quantity of each item in the cart by 1 for the number of times the loop runs. And with each update we

check the validation that the cart is not empty. A counter is used to update the quantity of items in the cart.

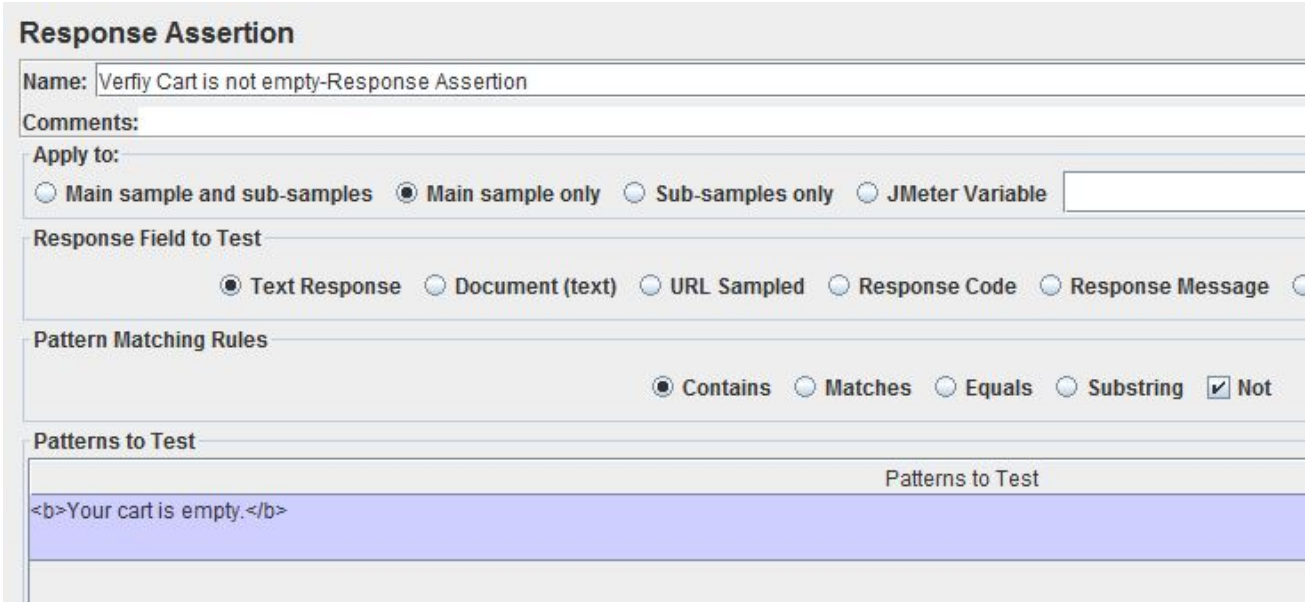


Fig: Assertion checking that cart is NOT empty

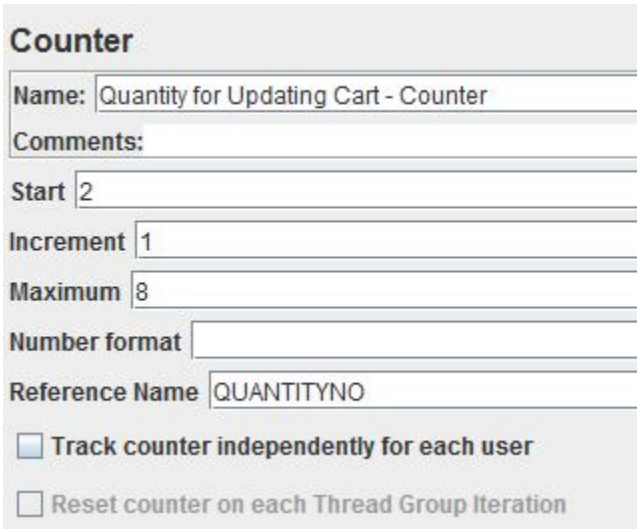


Fig: Counter to increment the number of items in the cart

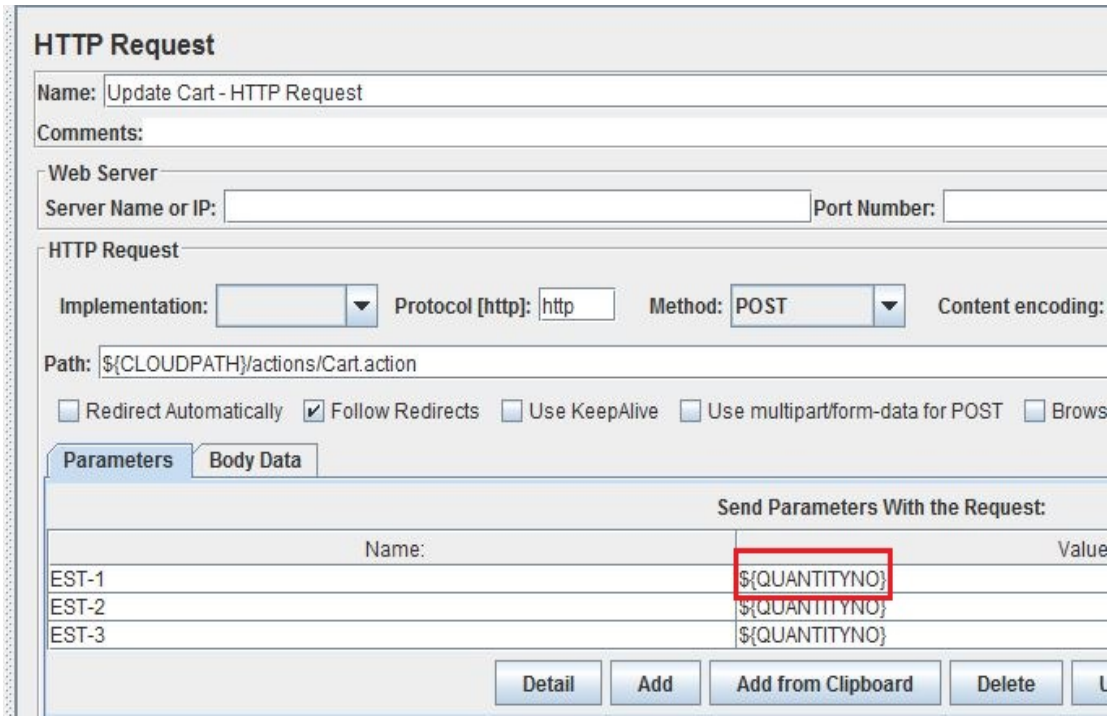


Fig: Using the counter value to update items in the cart

4) **Remove Item From Cart- Simple Controller:** This controller removed the added item from the cart. And check the assertion if cart is empty.

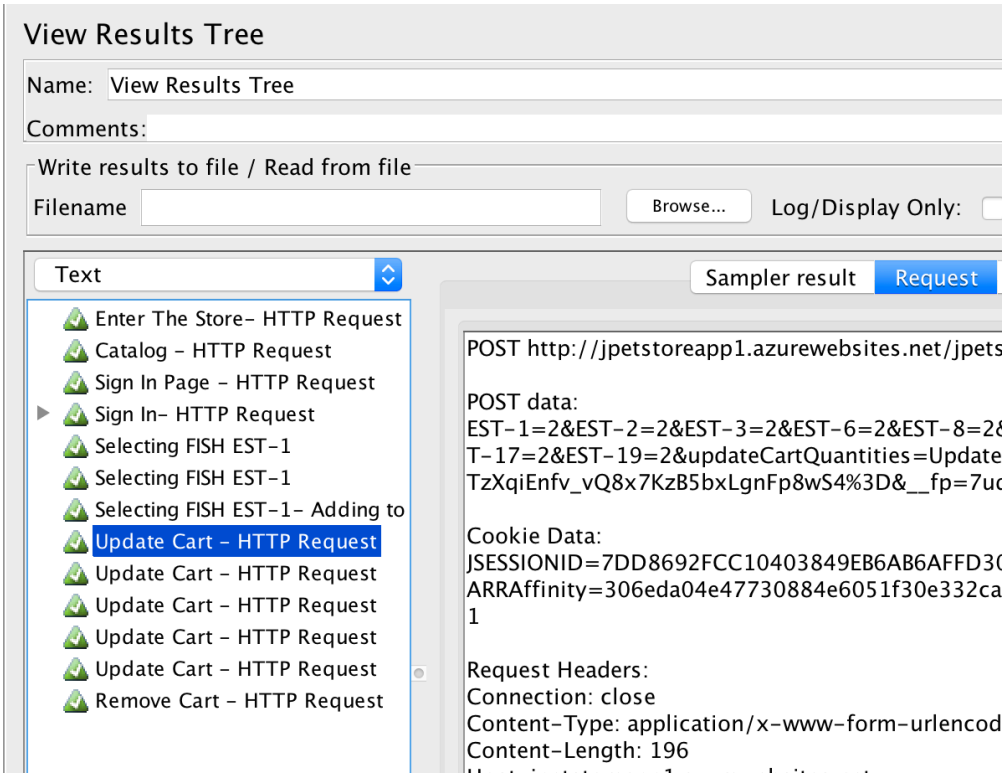


Fig: Screenshot of Thread Group Run- displaying all URL and their assertion values

Transaction# 4 **Transaction Name: Manage Account**

This thread group contains all steps explained in transaction 4. It consists of user sign in, updating details of "My Account" and saving multiple times, and resetting all values to prior values for consistency sake.

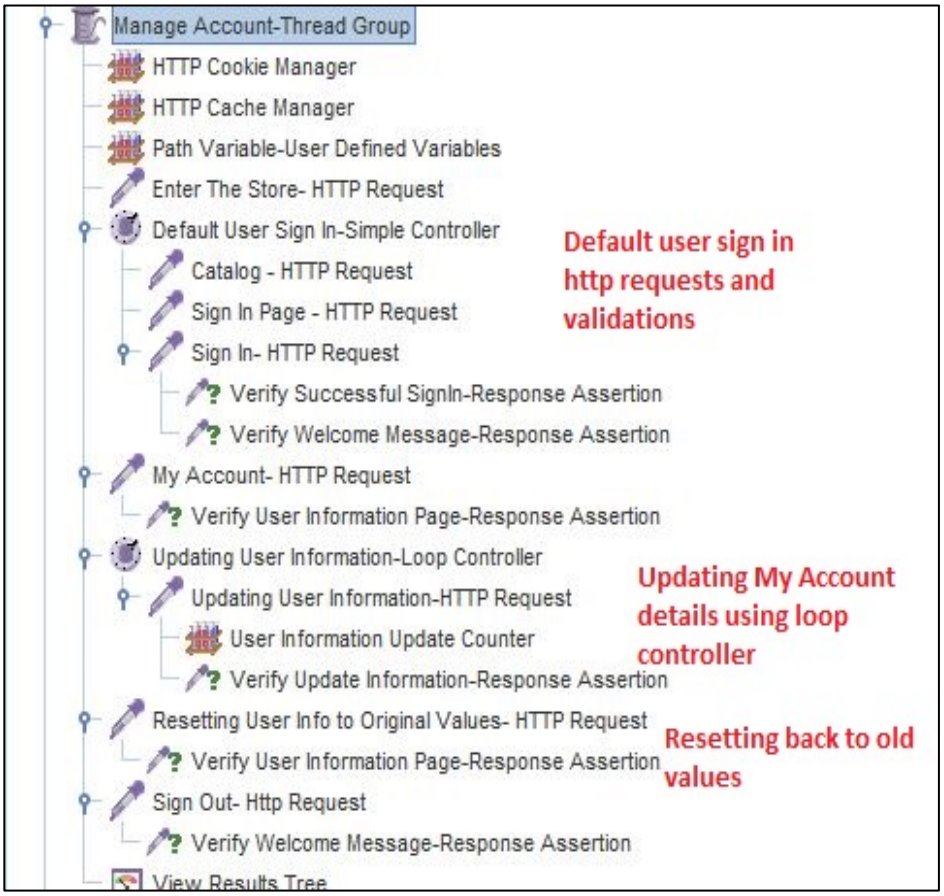


Fig: Screen shot – Thread Group for Updating My Account details

1) **Updating User Information- Loop Controller:** This controller updates the user information present on "My Account" page by using a counter variable that appends values to the existing name, address, card details and saves the details. The counter is incremented in steps of 1 to produce a unique value each time

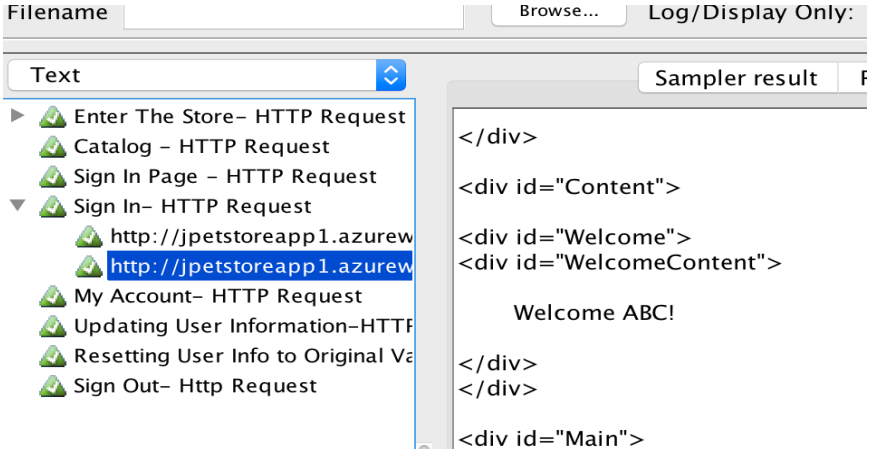


Fig: Screenshot of Thread Group Run- displaying all URL and their assertion values

Transaction# 5 **Transaction Name: View orders**

This thread group contains steps for sign in, add items to cart, confirm order and view the order details in the "My Orders" section.

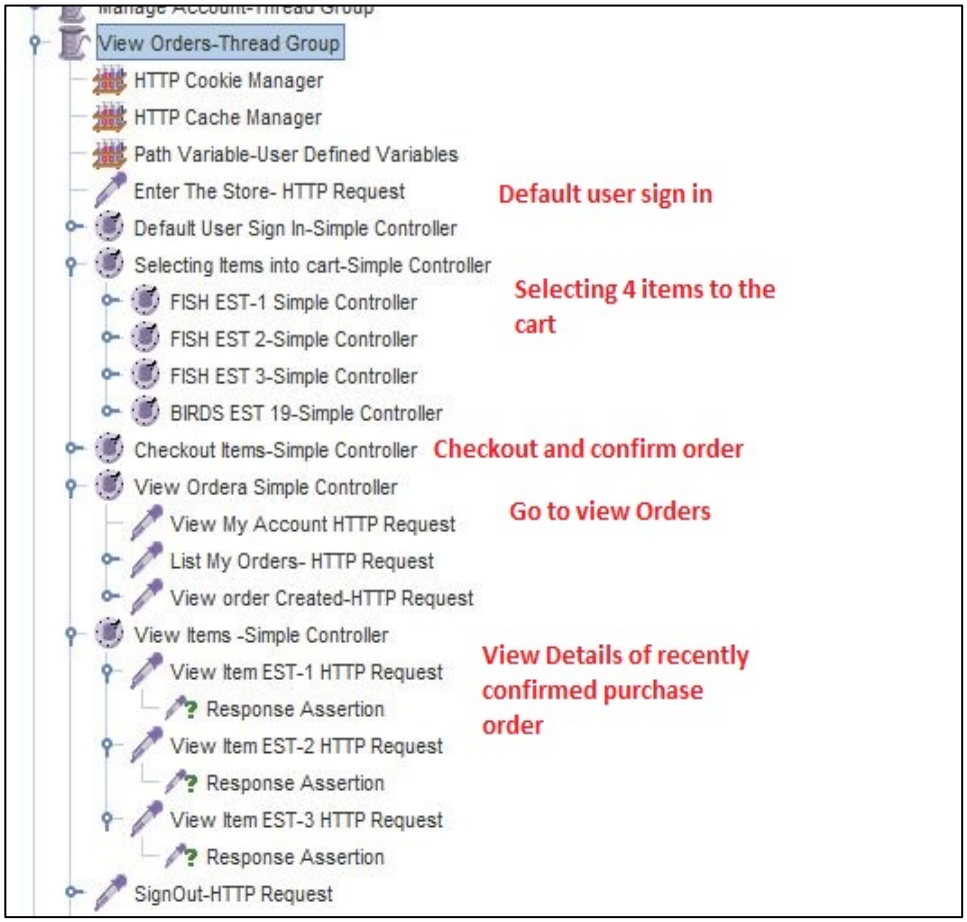


Fig: Screen shot- Thread group for transaction 5

1) **Checkout Items-Simple Controller:** After confirming all user details, and proceeding with the purchase order a unique ORDERID is generated by the application. We need to store this order id, so that we can view it in my orders. This order ID is extracted from the response of confirm orders using a Regular Expression Extractor.

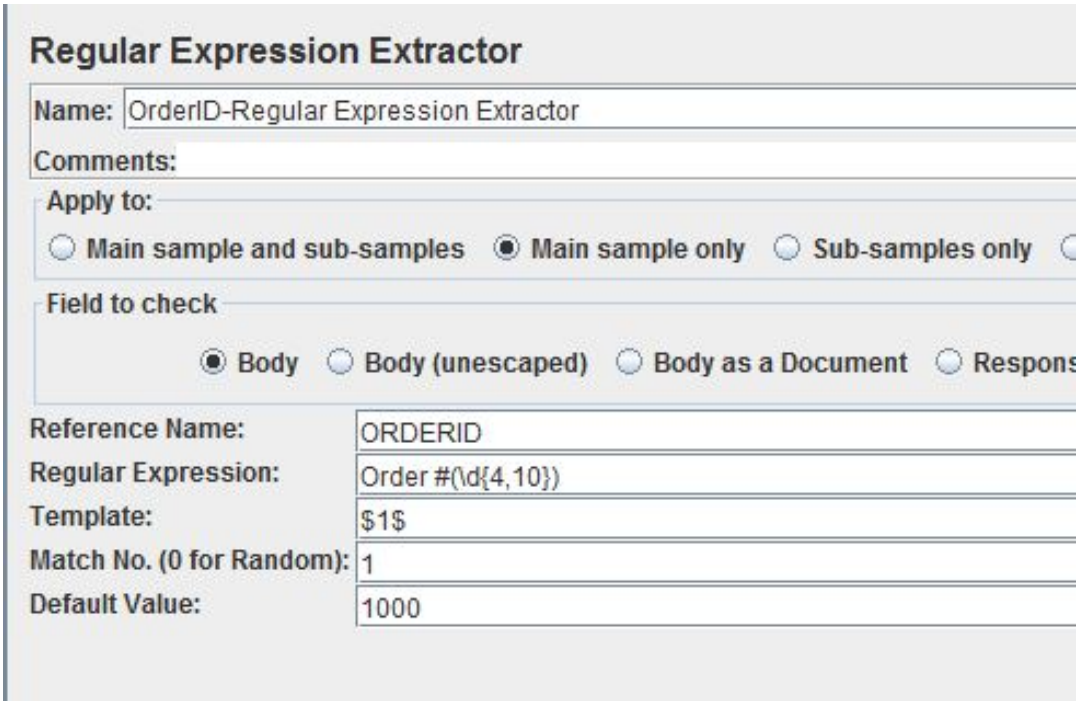


Fig: Regular Expression to extract the ordered generated by the application

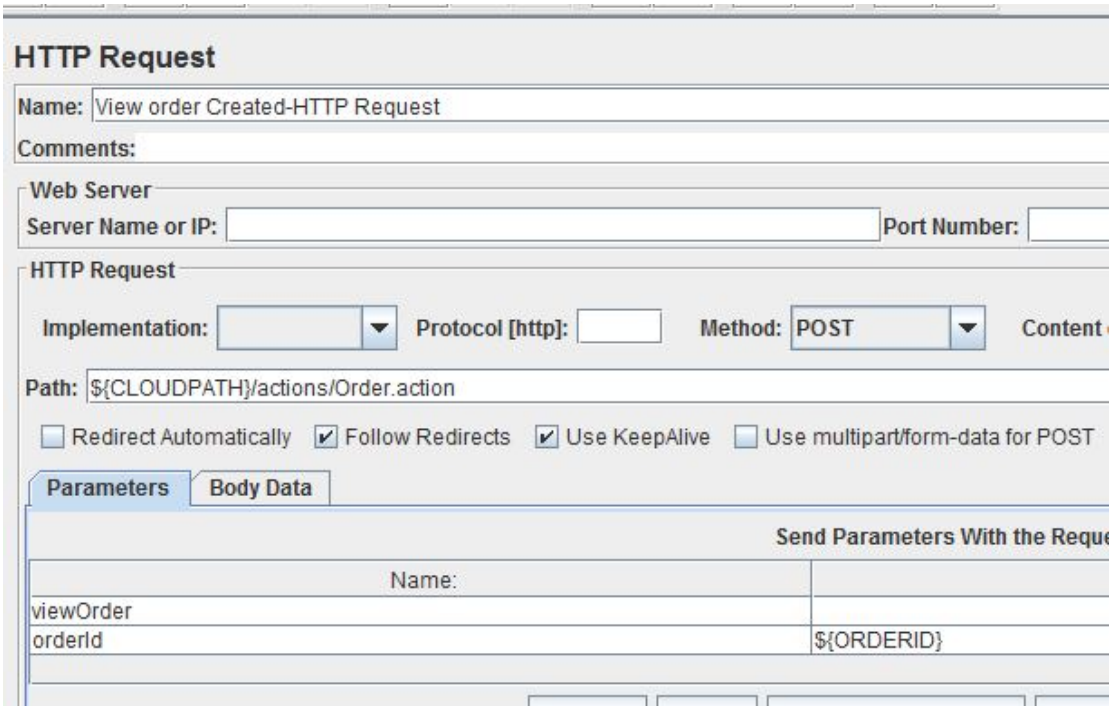


Fig: Using the Order ID to View the confirmed order

2) **View Orders Simple Controller:** This controller groups together steps to view the recently confirmed order from the "My Account"-> "My Orders" section. Assertion checks that the items are added to my orders section under the specific ordered.

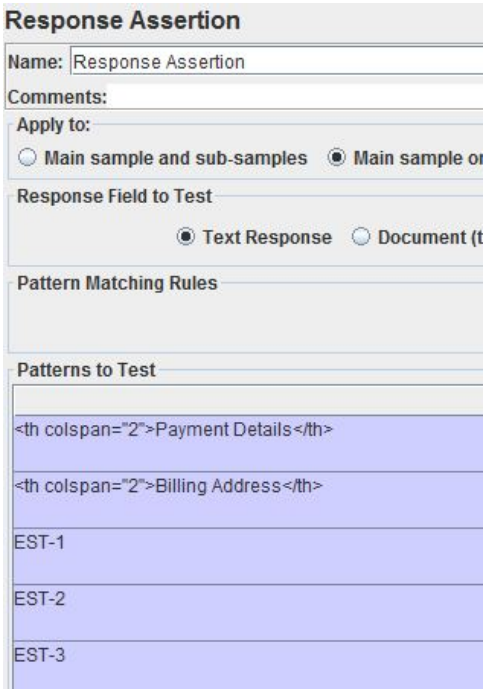


Fig: Assertion to check the items of the order

3) **View Items-Simple Controller:** These requests view the individual details of the items in the order. Assertions are added to check if the correct item is visible.

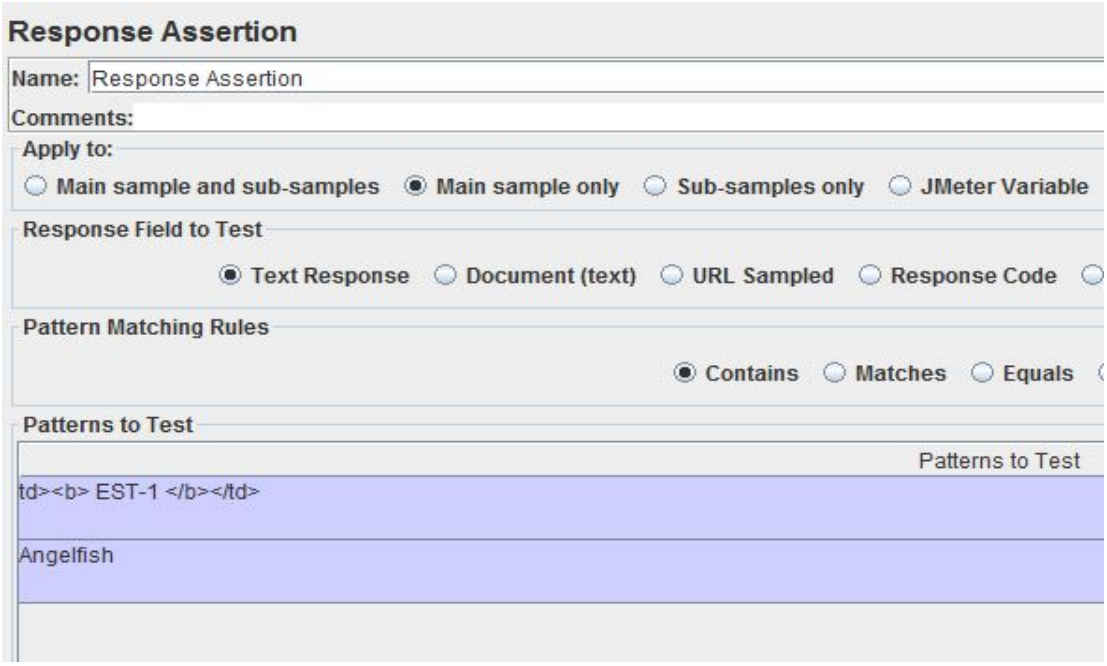


Fig: Assertion to check the correct item is displayed

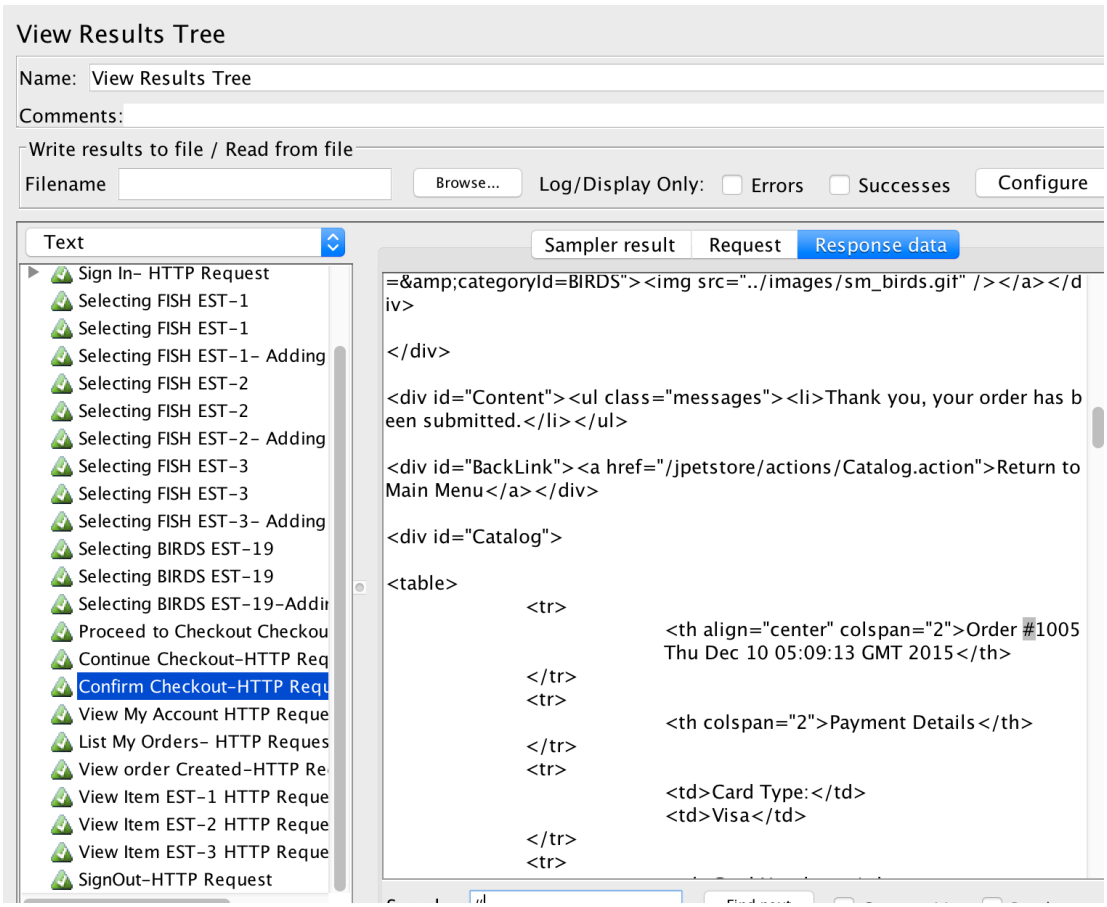


Fig: Screenshot of Thread Group Run- displaying all URL and their assertion values

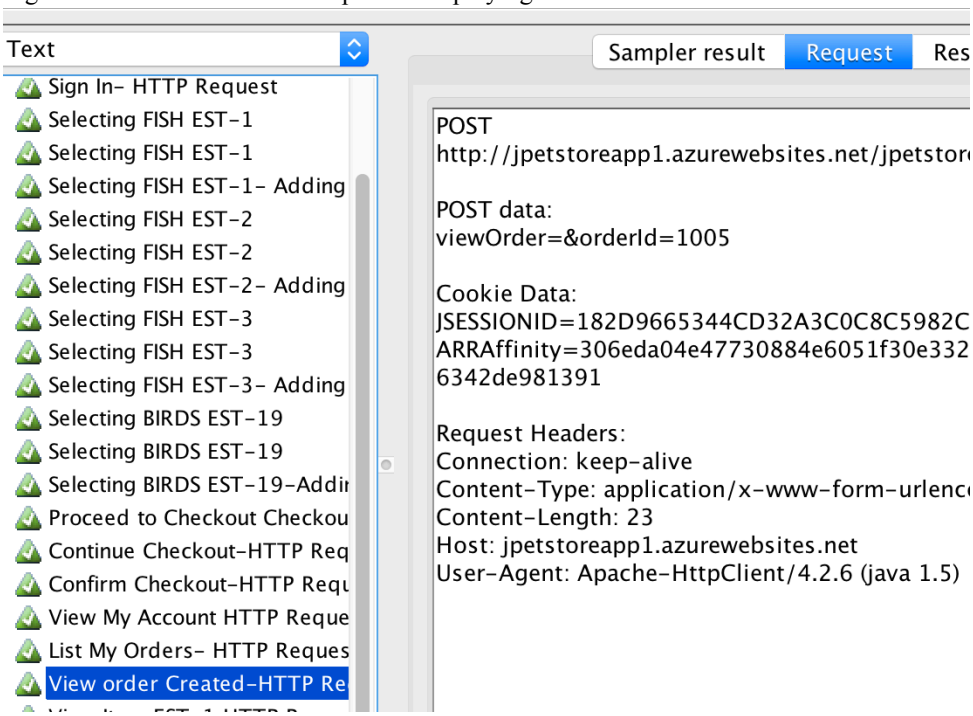


Fig: Screenshot of Thread Group Run- Passing Parameter Order ID as received from previous steps

Auto Scaling Strategies:

Each transaction is tested individually on the cloud and the results are documented with best possible strategies for 2 parameters

1. CPU Utilization
2. Memory Percentage

MS Azure WebApps provides 3 types of possible configurations for scaling, each with different pricing, since in MS Azure we cannot really scape up\down we use these configurations to give us an idea of how different core and memory instances can scale in\out based on our rules.

1. Small
2. Medium
3. Large

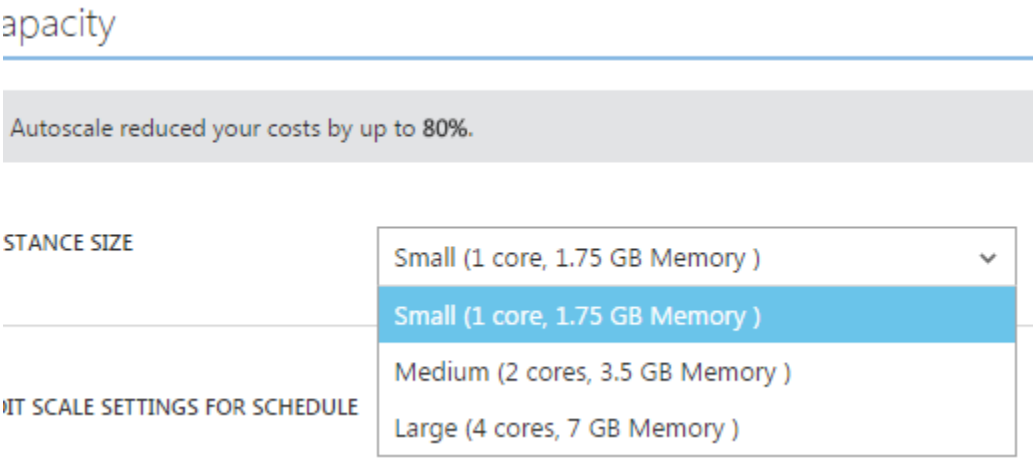


Fig: MS Azure types of Standard Instances

Transaction #1: Register New Users

Local Machine Results:

JMeter script was run for various inputs of samples for a duration of 10 minutes and the following throughput values were noted. For the input of 100 Users * 10 iterations * 10 URL\iteration = 10,000 Samples for a ramp up period of 10 seconds, an out of memory exception was thrown by Java. This was considered as input for testing in the cloud.

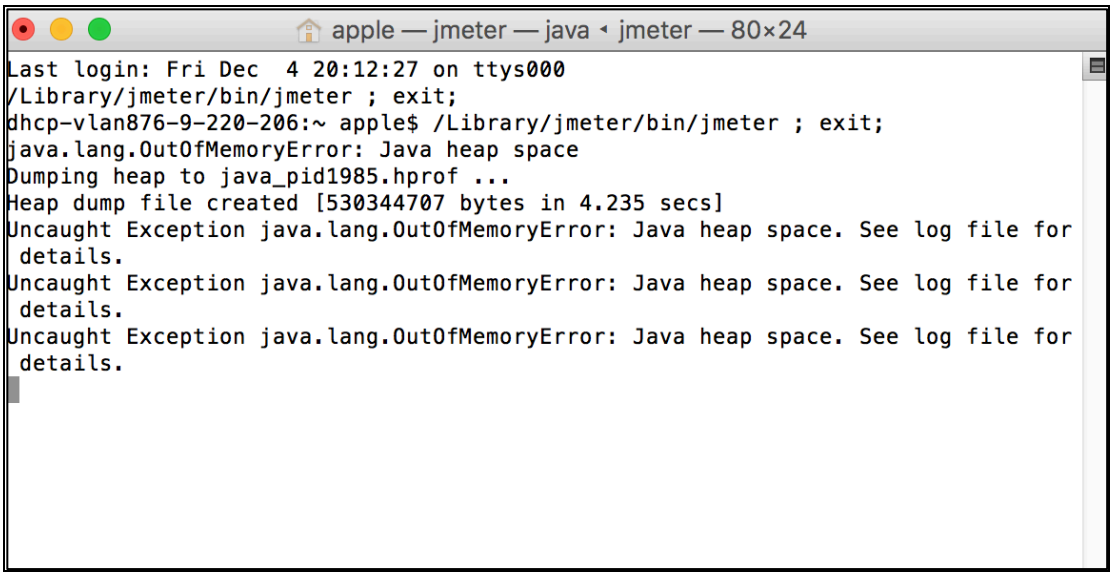


Fig: Out of Memory Exception throw by JMeter

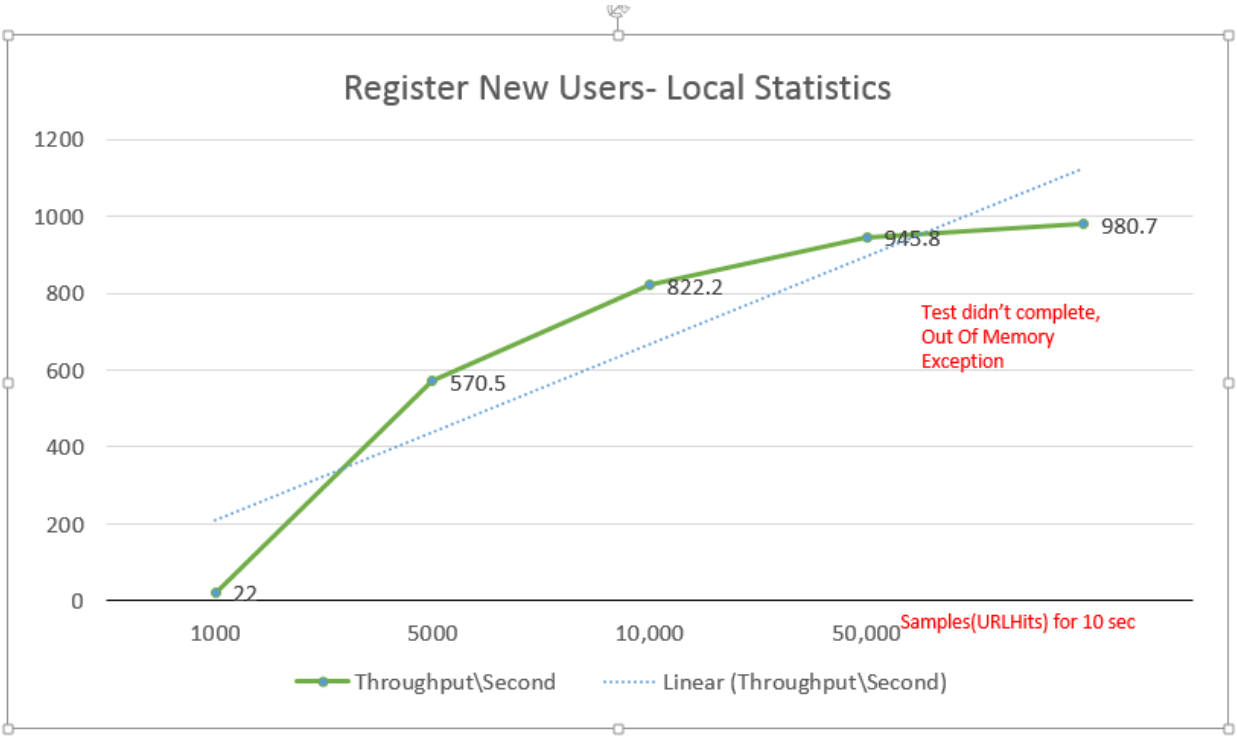


Fig: Graph showing Local statistics of Transaction – 1

CPU % Rules:

Cloud Results: **Input: 10,000 Samples, Ramp Up: 10 seconds**

In the cloud, the first run was done for 10,000 samples on a ramp up period of 10 seconds. Various combinations of CPU% were taken on Small, Medium, Large instances scaling rules

were written only for scale up, keeping the scale down value to default CPU<=60%. The results are documents in the below table.

TABLE: 10,000 SAMPLES\10 SECONDS RUN			
Metrics	**SMALL**	**MEDIUM**	**LARGE**
CPU%	**75%**	**75%**	**75%**
Average Response Time:	**668 ms**	**2.89s**	**Not Recorded as can stand higher %**
Throughput\sec:	**130.7**	**174.7**	
CPU Time in sec:	**5.48**	**6.01**	
CPU%	**80%**	**80%**	**80%**
Average Response Time:	**1.87**	**3.32**	**1.55**
Throughput\sec:	**79.4**	**342.8**	**306.6**
CPU Time in sec:	**2.83**	**8.38**	**15.21**
CPU%	**85%**	**85%**	**85%**
Average Response Time:	**2.51**	**951ms**	**1.67**
Throughput\sec:	**44.7**	**183.6**	**310.9**
CPU Time in sec:	**6.28**	**6.01**	**8.65**
CPU%	**90%**	**90%**	**90%**
Average Response Time:	**Not Recorded as cannot stand higher%**	**Not Recorded as cannot stand higher%**	**2.18**
Throughput\sec:			**235.5**
CPU Time in sec:			**7.16**
CPU%	**95%**	**95%**	**95%**
Average Response Time:	**Not Recorded as cannot stand higher%**	**Not Recorded as cannot stand higher%**	**3.83**
Throughput\sec:			**316.4**
CPU Time in sec:			**8.38**

Fig: Table showing metric values for 10,000 samples run for 10 seconds ramp up time (Screen shots attached in zip file "10,000Samples- 10 sec-CPU%")- Scale down values CPU<=60%

Highlighted sections show that for the below values the response times is below 2 seconds:

1. Small Instances: CPU%>=75
2. Medium Instances: CPU%>=85%
3. Large Instances: CPU%>=90%

Taking these values the scale down CPU% was adjusted for SMALL instances and results are noted below.

TABLE: 10,000 SAMPLES\10 SECONDS RUN		
Metrics	SMALL	SMALL
CPU%>= (Scale Up)	75%	75%
CPU%<= (Scale Down)	55%	65%
Average Response Time:	3.91	5.21s
Throughput\sec:	70.3	83.7
CPU Time in sec:	12.7	11.98

Fig: Table showing metric values for 10,000 samples run for 10 seconds ramp up time (Screen shots attached in zip file- 10,000Samples- 10 sec-CPU%)- Scale up values CPU>=75%

Cloud Results: **Input: 10,000 Samples, Ramp Up: 10 minutes**

Once a vague idea of the load and scaling strategies of the application was figured, the test on the cloud was conducted for higher ramp up times of 10 minutes and results are noted below:

TABLE: 10,000 SAMPLES\10 Minutes Runs			
Metrics	SMALL	MEDIUM	LARGE
CPU%>= (Scale up)	75%	80%	85%
CPU%<= (Scale down)	60%	60%	65%
No of Requests at server:	885.6	650	730
Average Response Time:	2.19	165ms	18ms
Throughput\sec:	16.6	16.6	16.6
CPU Time in sec:	6.01	5.61	6.17
No of Requests at server:	1.81K	652	
CPU%>= (Scale up)	80%	85%	
CPU%<= (Scale down)	55%	55%	
Average Response Time:	151ms	25ms	
Throughput\sec:	16.6	16.6	
CPU Time in sec:	5.42	5.86	
No of Requests at server:	1.26K		
CPU%>= (Scale up)	80%		
CPU%<= (Scale down)	60%		
Average Response Time:	164ms		
Throughput\sec:	16.6		
CPU Time in sec:	5.36s		

Fig: Table displaying run on cloud for 10,000 samples – 10 minutes (Screen shots attached in zip file- "10,000Samples- 10 min-CPU%")

Optimal Strategies Found:

The best found strategies are shown below:

Small Instance:

- CPU% >=75% Increase count by 1
- CPU% <=55% Decrease count by 1

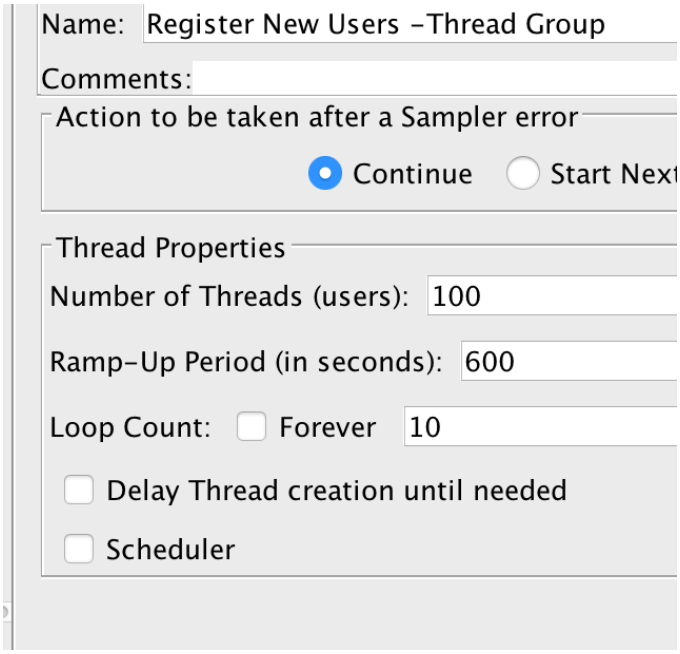


Fig: JMeter setting for Run

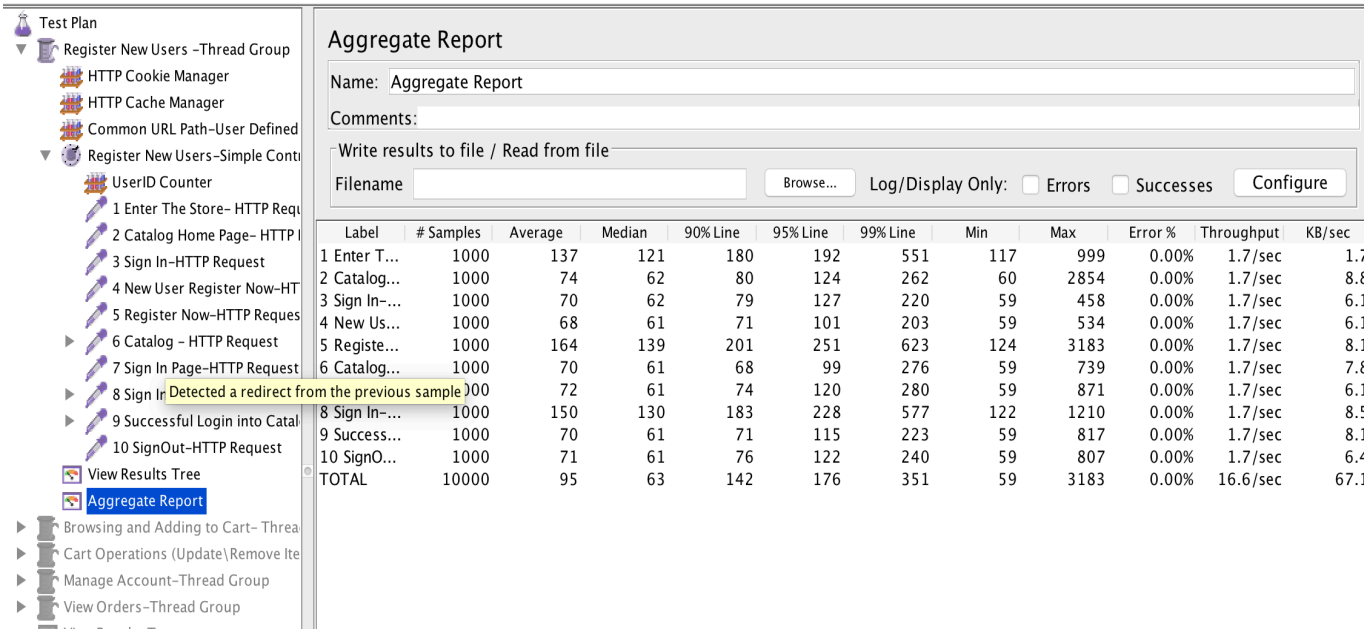


Fig: JMeter Results

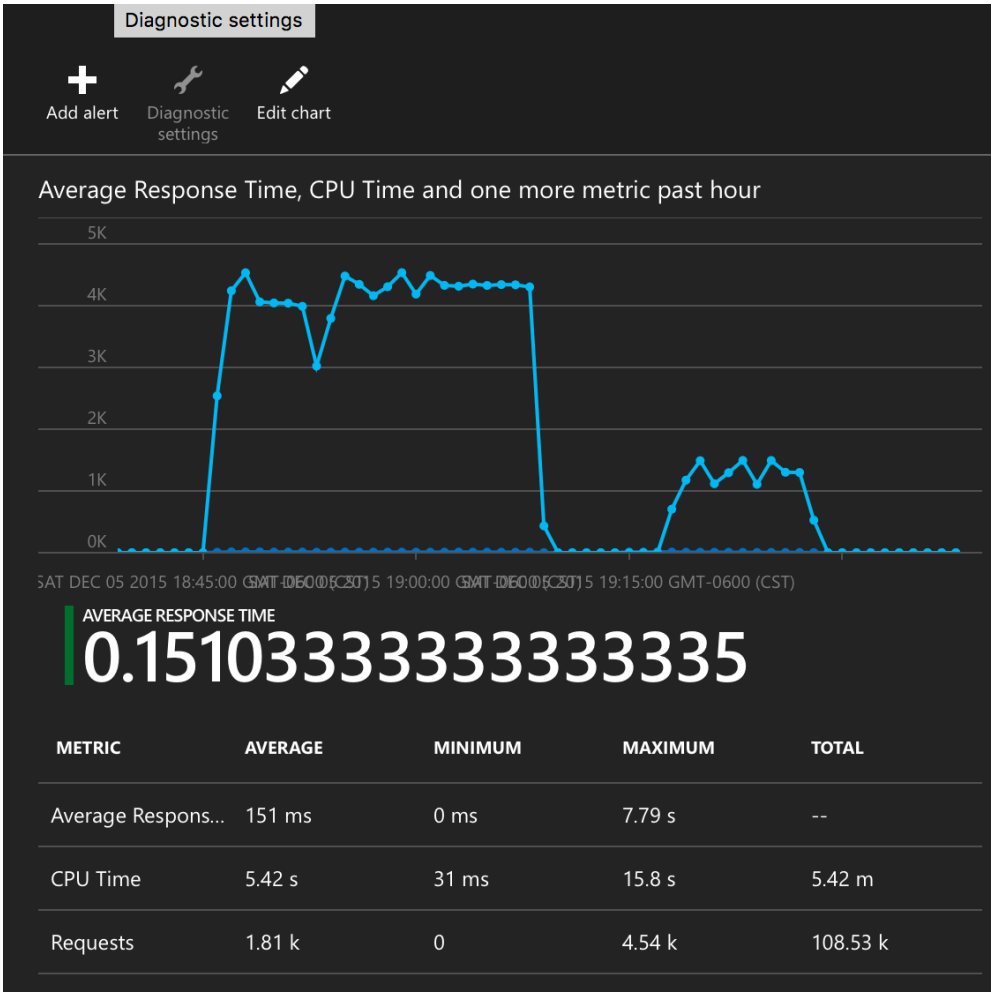


Fig: Metrics from MS Azure

Medium Instance:

Strategy:

- CPU% >=85% Increase count by 1
- CPU% <=55% Decrease count by 1

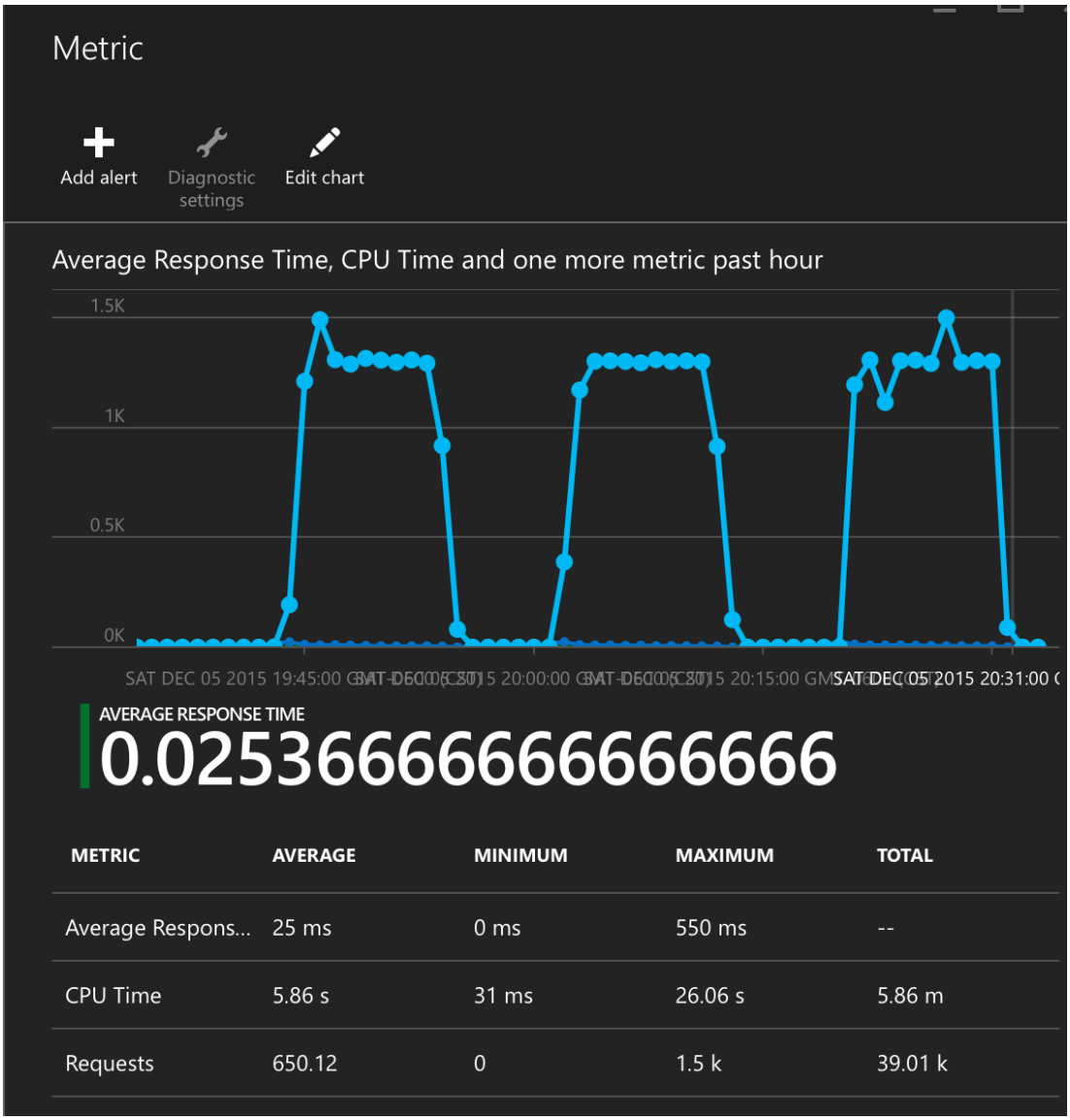


Fig: Metrics from MS Azure

Large Instance:

Strategy:

- CPU% >=85% Increase count by 1
- CPU% <=65% Decrease count by 1

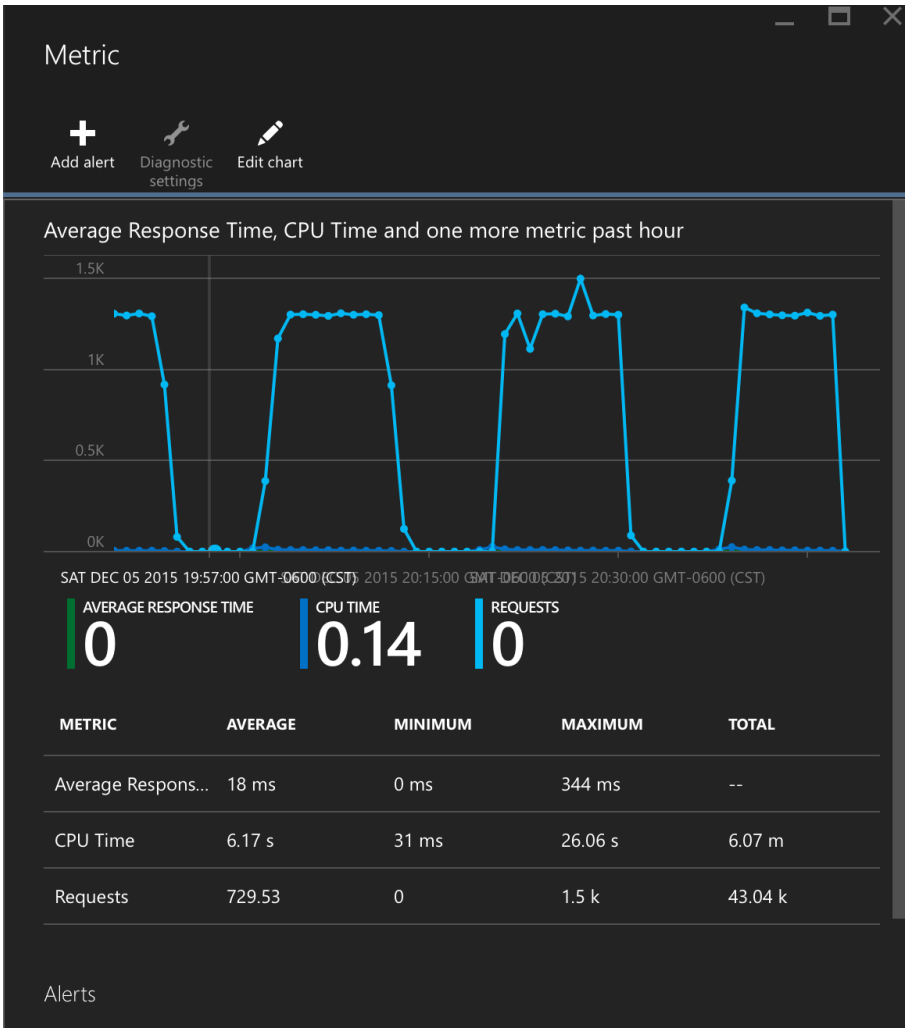
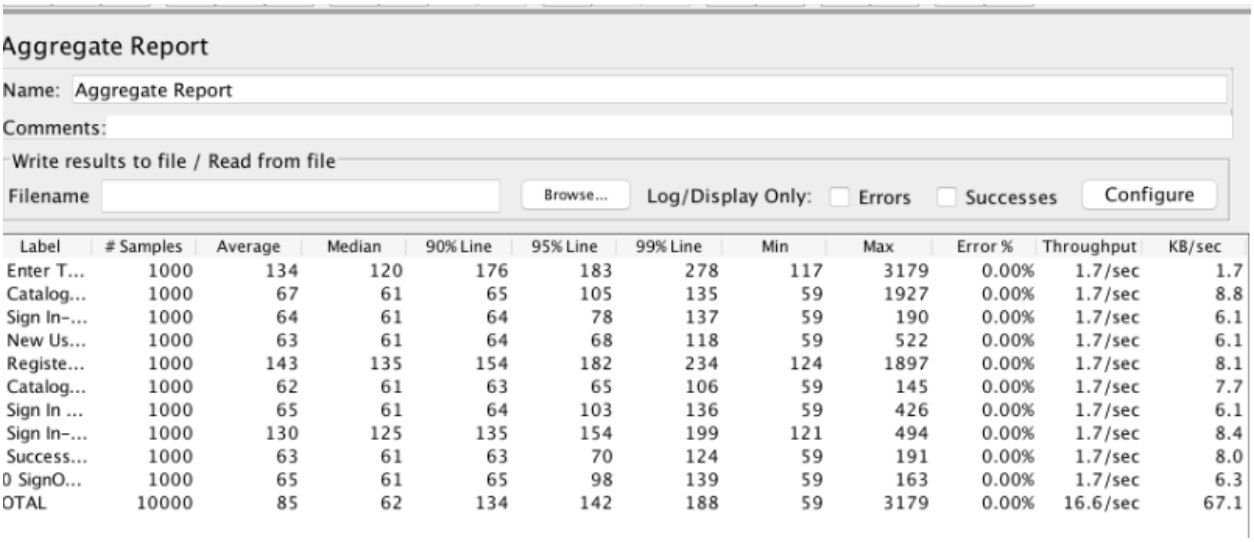


Fig: Metrics from MS Azure



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
Enter T...	1000	134	120	176	183	278	117	3179	0.00%	1.7/sec	1.7
Catalog...	1000	67	61	65	105	135	59	1927	0.00%	1.7/sec	8.8
Sign In-...	1000	64	61	64	78	137	59	190	0.00%	1.7/sec	6.1
New Us...	1000	63	61	64	68	118	59	522	0.00%	1.7/sec	6.1
Registe...	1000	143	135	154	182	234	124	1897	0.00%	1.7/sec	8.1
Catalog...	1000	62	61	63	65	106	59	145	0.00%	1.7/sec	7.7
Sign In ...	1000	65	61	64	103	136	59	426	0.00%	1.7/sec	6.1
Sign In-...	1000	130	125	135	154	199	121	494	0.00%	1.7/sec	8.4
Success...	1000	63	61	63	70	124	59	191	0.00%	1.7/sec	8.0
0 SignO...	1000	65	61	65	98	139	59	163	0.00%	1.7/sec	6.3
OTAL	10000	85	62	134	142	188	59	3179	0.00%	16.6/sec	67.1

Fig: JMeter Results

Observations:

1. Requests to the server are almost between the same ranges
2. Throughput is constant
3. Average Response time is below 1 second (observe last line)
4. The graph remains almost constant with less peaks
5. Max No of Instances=5, No of instances used: 2
6. No server errors encountered

Memory % Rules:

The next parameter tested are the memory % for various parts. As observed from the below table of values, for SMALL scale instances, the values of Avg response time, throughput and CPU time are almost similar. Hence, any one of the strategies can be adopted as a optimal strategy.

TABLE: 10,000 SAMPLES\10 Minutes Runs		
Metrics/Instances	**SMALL**	**MEDIUM**
Memory %>= (Scale up)	**80%**	**85%**
Memory %<= (Scale down)	**45%**	**55%**
No of Requests at server:	**652**	**842**
Average Response Time:	**2.31**	**56ms**
Throughput\sec:	**16.6**	**16.6**
CPU Time in sec:	**4.36**	**3.12**
No of Requests at server:	**652**	**842**
Memory %>= (Scale up)	**80%**	**85%**
Memory %<= (Scale down)	**55%**	**50%**
Average Response Time:	**2.31**	**22ms**
Throughput\sec:	**16.6**	**16.6**
CPU Time in sec:	**3.17**	**3.01**
No of Requests at server:	**650**	
Memory %>= (Scale up)	**80%**	
Memory %<= (Scale down)	**60%**	
Average Response Time:	**2.31**	
Throughput\sec:	**16.6**	
CPU Time in sec:	**2.68**	
No of Requests at server:	**880**	
Memory %>= (Scale up)	**85%**	
Memory %<= (Scale down)	**55%**	
Average Response Time:	**2.31**	
Throughput\sec:	**16.6**	
CPU Time in sec:	**2.89**	

Fig: Table displaying run on cloud for 10,000 samples – 10 minutes (Screen shots attached in zip file- "10,000Samples- 10 sec-Memory%)

Optimal Strategy Found:

Small Instances: Almost all values give similar results, hence any one can be choose

Medium Instances:

- Memory % >=85% Increase count by 1
- Memory % <=50% Decrease count by 1

Pricing in Cloud:

Since we use a azure pass, the pricing isn't accurate to document. Even though we do have basic pricing criteria found based upon the instances. Upon running the optimized strategy rules the pricing shown is:

Fig: Pricing for Small Instances

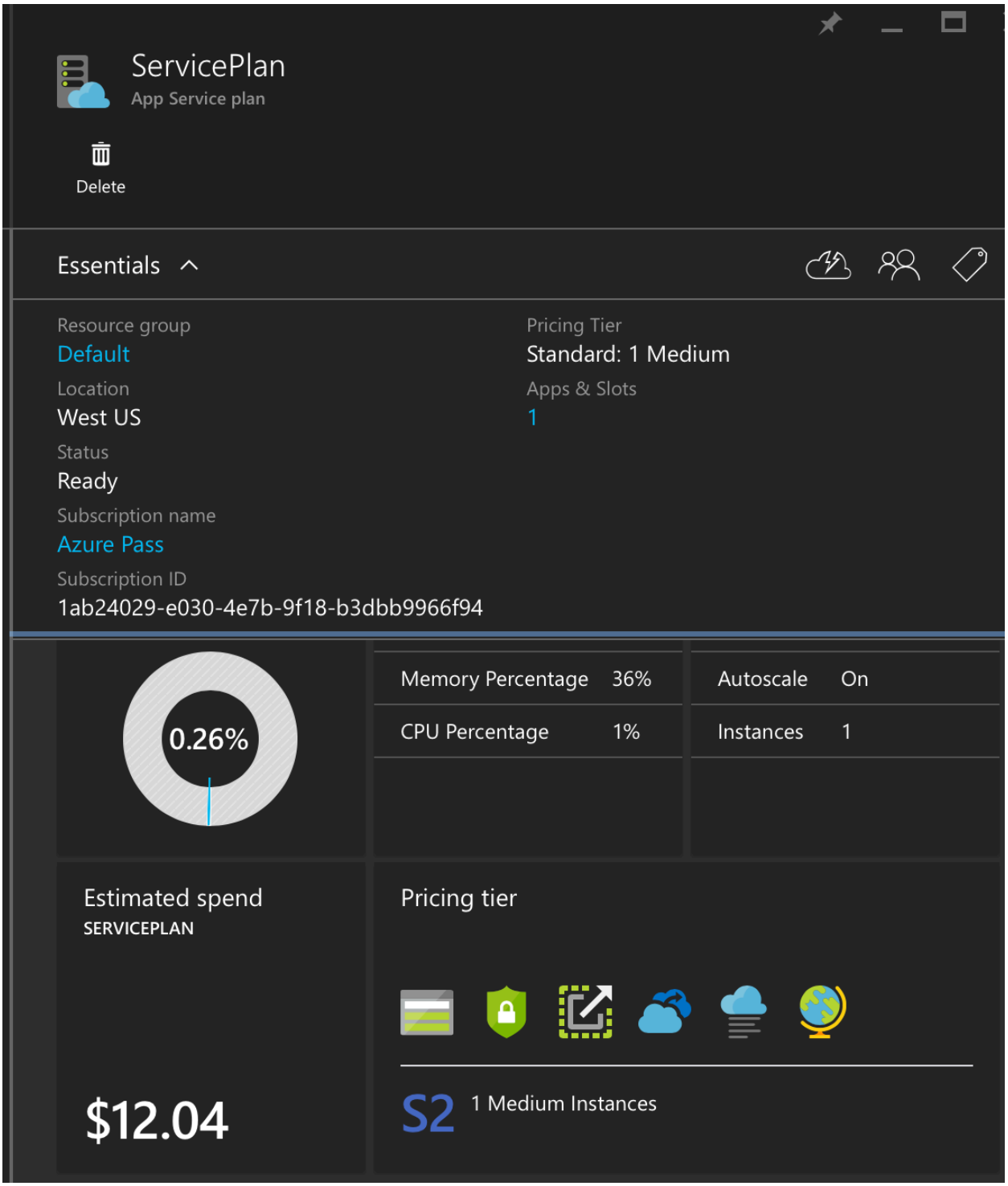


Fig: Pricing for Medium Instances

Fig: Pricing for Large Instances

Transaction #2: Browsing & Adding to Cart

Local Machine Results:

The below graph shows the throughput noted on the jmeter run in the local system. There wasn't any out of memory or bound on this operation due to system constraints. Hence, the input for the cloud testing was maintained the same as

100 users * 10 iterations * 20 URL\iteration = 20,000 Samples ---- 10 Mins

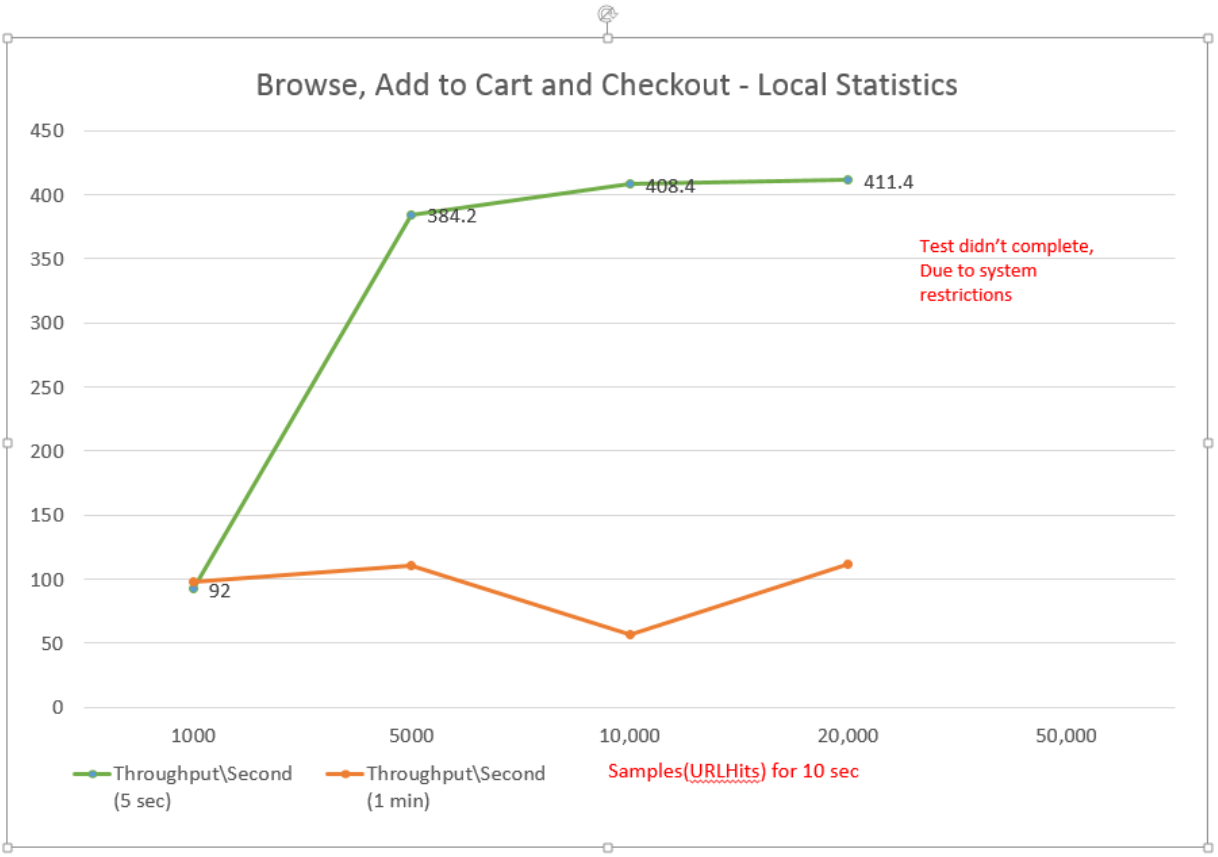


Fig: Graph plotted for throughput of local run

CPU % Results:

Cloud Results: **Input: 20,000 Samples, Ramp Up: 10 Minutes**

After understanding the application behavior, the second transaction was tested in the cloud with 20,000 samples on a ramp up period of 10 minutes. Various combinations of CPU% were taken on Small, Medium, Large instances scaling rules were tested and optimal strategies concluded.

TABLE: 20,000 SAMPLES\10 Minutes Runs			
Metrics	**SMALL**	**MEDIUM**	**LARGE**

CPU %>= (Scale up)	75%	85%	90%
CPU %<= (Scale down)	60%	60%	60%
No of Requests at server:	902.77	1.33K	1.36K
Average Response Time:	3.24	1.14	2.36
Throughput\sec:	33.6	33.6	33.6
CPU Time in sec:	12.62	9.23	10.36
No of Requests at server:	1.31K	1.31K	
CPU %>= (Scale up)	80%	90%	
CPU %<= (Scale down)	60%	60%	
Average Response Time:	1.99s	1.21	
Throughput\sec:	33.6	33.6	
CPU Time in sec:	12.71	9.34	
No of Requests at server:	1.1K4		
CPU %>= (Scale up)	85%		
CPU %<= (Scale down)	60%		
Average Response Time:	11ms		
Throughput\sec:	33.6		
CPU Time in sec:	5.96		
No of Requests at server:	1.47K		
CPU %>= (Scale up)	80%		
CPU %<= (Scale down)	55%		
Average Response Time:	22ms		
Throughput\sec:	33.6		
CPU Time in sec:	8.46		

Fig: Table displaying run on cloud for 10,000 samples – 10 minutes (Screen shots attached in zip file- "10,000Samples- 10 sec-CPU%)

Optimal Strategies Found:

The best found strategies are shown below:

Small Instance:

- CPU% >=85% Increase count by 1 (or) CPU% >=80% Increase count by 1
- CPU% <=60% Decrease count by 1 (or) CPU% >=55% Decrease count by 1

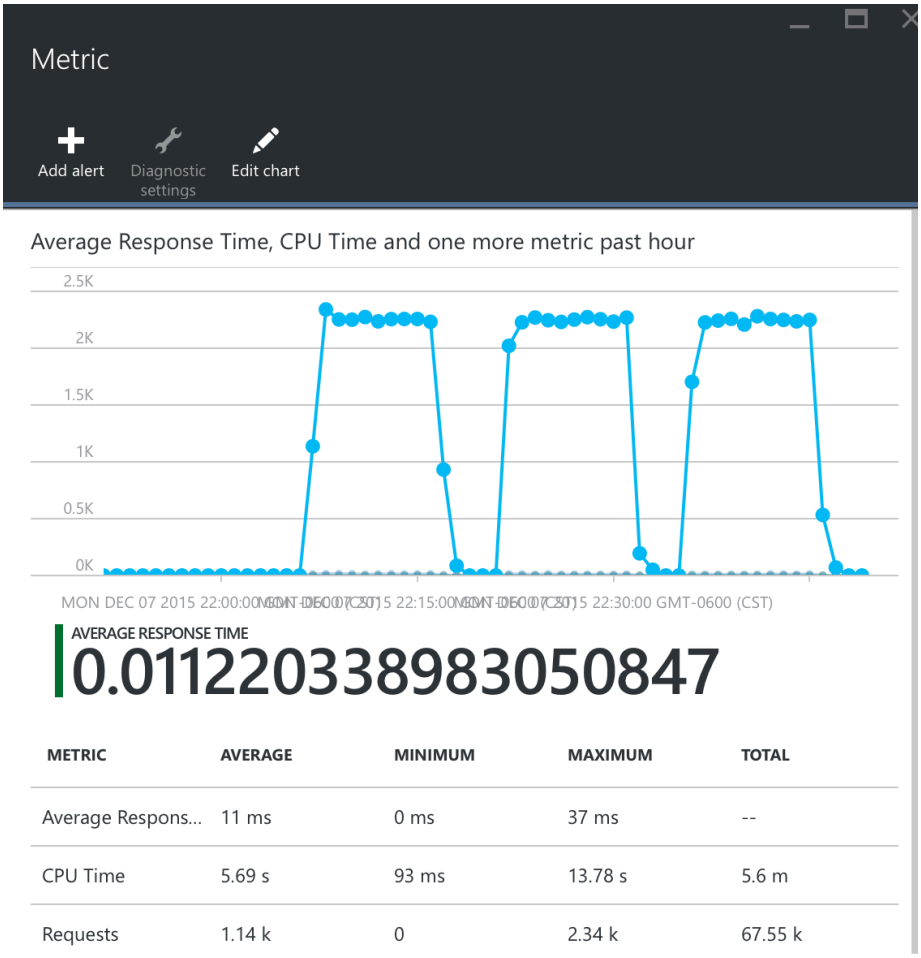


Fig: Metrics from MS Azure- Small Instance

Medium Instance:

Strategy:

- CPU% >=85% Increase count by 1
- CPU% <=60% Decrease count by 1

Large Instance:

Strategy:

- CPU% >=90% Increase count by 1
- CPU% <=60% Decrease count by 1

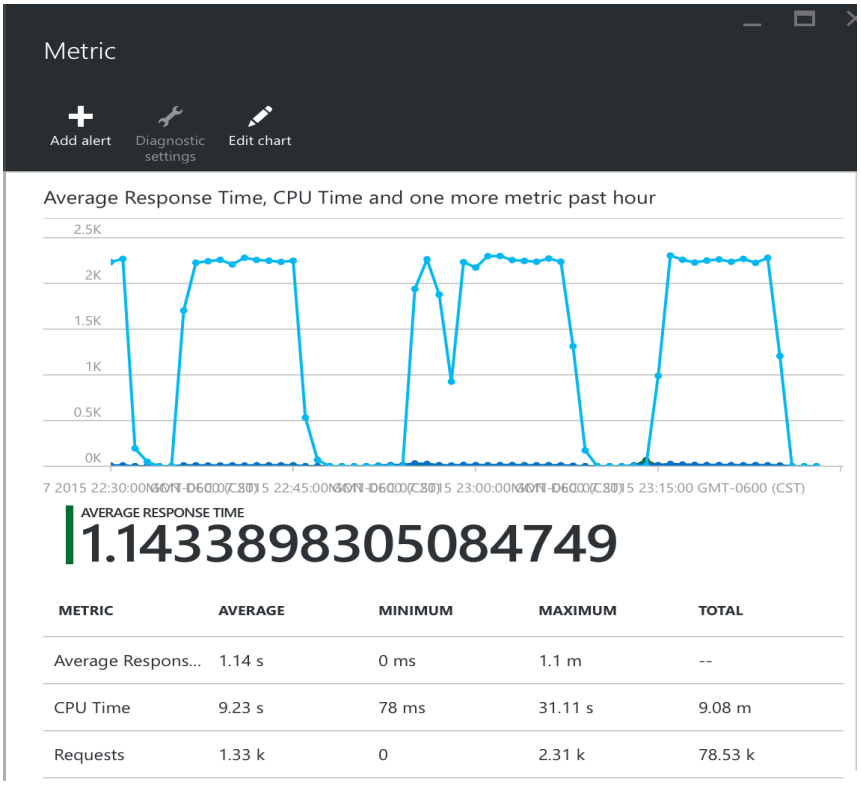


Fig: Metrics from MS Azure – Medium Scale

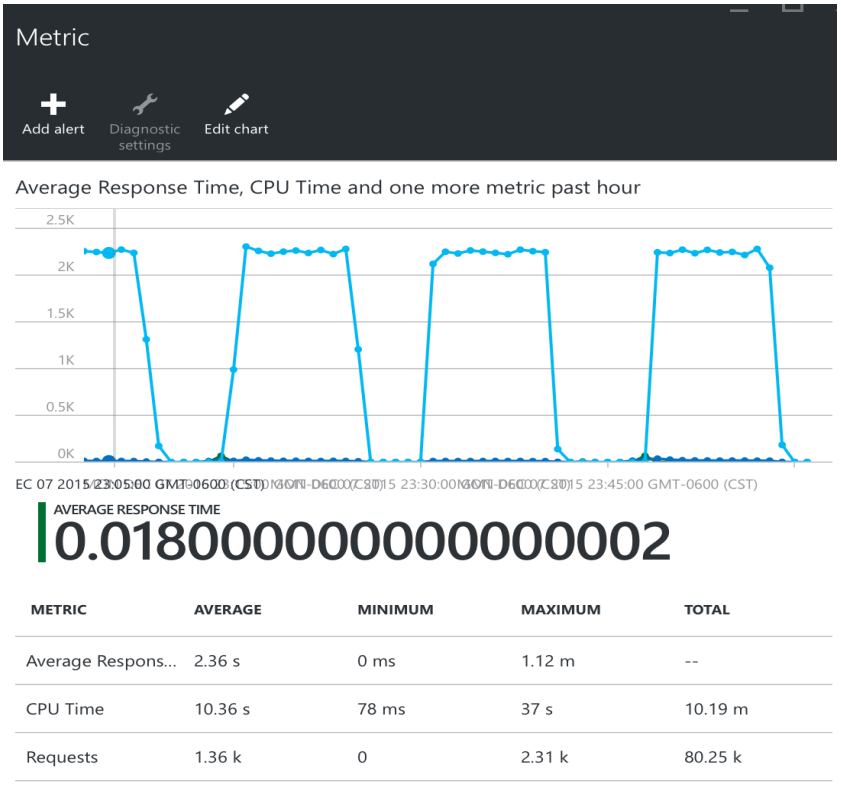


Fig: Metrics from MS Azure – Large Scale

Memory % Rules:

TABLE: 20,000 SAMPLES\10 Minutes Runs			
Metrics	**SMALL**	**MEDIUM**	**LARGE**
Memory %>= (Scale up)	**80%**	**85%**	**90%**
Memory %<= (Scale down)	**60%**	**60%**	**60%**
No of Requests at server:	**678**	**775.36**	**748.32**
Average Response Time:	**8ms**	**1.55**	**6ms**
Throughput\sec:	**33.6**	**33.6**	**33.6**
CPU Time in secs:	**1.87**	**4.11**	**4.63**
No of Requests at server:	**810.54**	**1.16K**	**1.14K**
Memory %>= (Scale up)	**80%**	**90%**	**95%**
Memory %<= (Scale down)	**65%**	**55%**	**60%**
Average Response Time:	**11ms**	**1.55**	**8ms**
Throughput\sec:	**33.6**	**33.6**	**33.6**
CPU Time in secs:	**3.63**	**5.42**	**6.66**
No of Requests at server:	**1.21K**	**1.53K**	**1.17K**
Memory %>= (Scale up)	**85%**	**90%**	**95%**
Memory %<= (Scale down)	**45%**	**60%**	**55%**
Average Response Time:	**15ms**	**10ms**	**6ms**
Throughput\sec:	**33.6**	**33.6**	**33.6**
CPU Time in secs:	**4.85**	**6.37s**	**6.09**

Fig: Table displaying run on cloud for 10,000 samples – 10 minutes (Screen shots attached in zip file- "10,000Samples- 10 sec-Memory%)

Most Optimal Strategies found are:

Small Instance:

- Memory% >=80% Increase count by 1
- Memory % <=60% Decrease count by 1

Medium Instance:

- Memory % >=90% Increase count by 1
- Memory % <=60% Decrease count by 1

Large Instance:

- Memory % >=95% Increase count by 1
- Memory % <=55% Decrease count by 1

Observations:

1. We have choose to keep the scale down memory value at a high threshold because the JPetStore application does not get over loaded easily. Also since this is not a shared cloud, multiple applications aren't hosted on the same cloud. Hence, once the instance is removed if halve of the remaining work can be distributed among other instances, it is considered as an effective cost strategy.

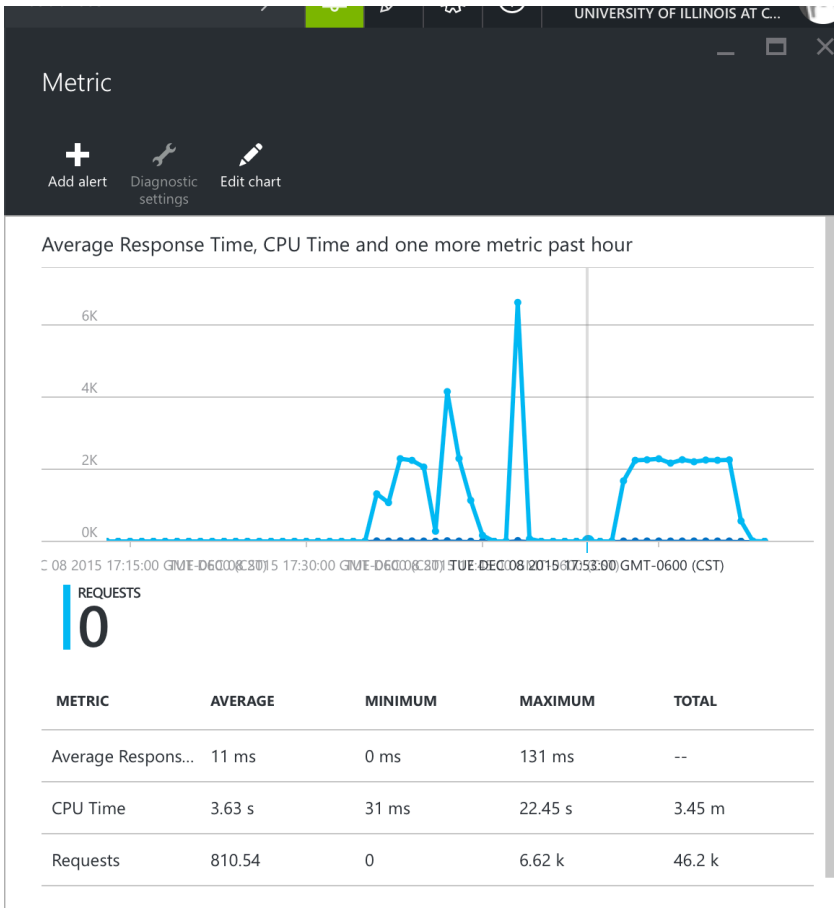


Fig: Metrics from MS Azure – Small Scale

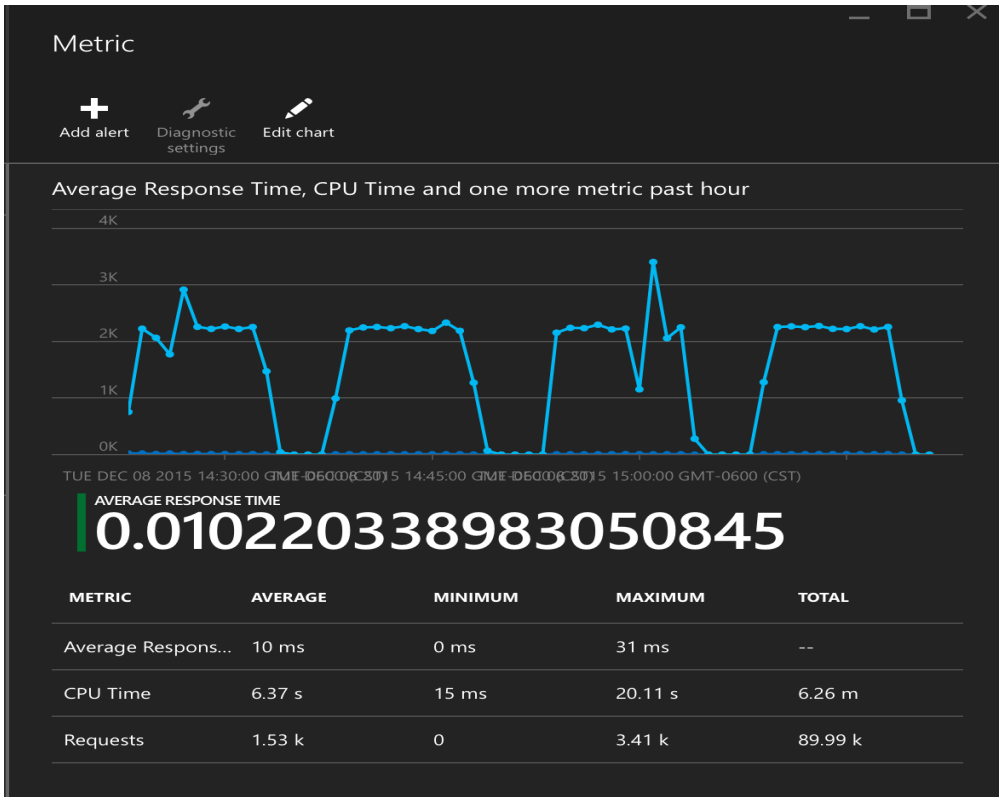


Fig: Metrics from MS Azure – Medium Scale

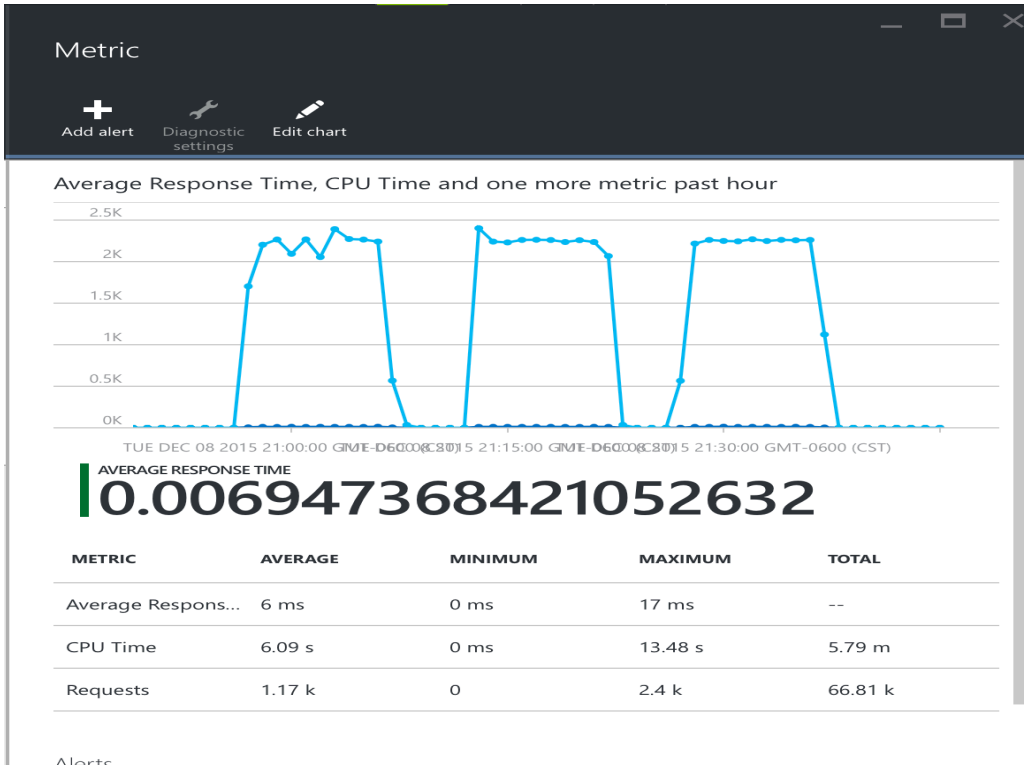


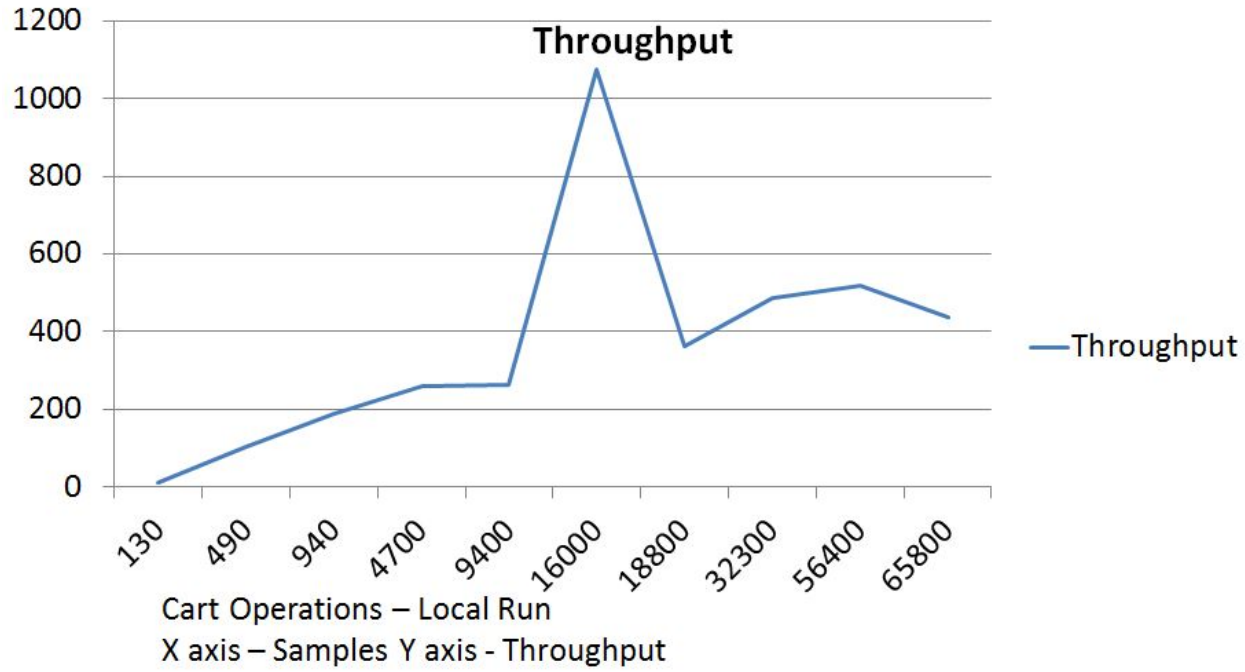
Fig: Metrics from MS Azure – Large Scale

Transaction #3: Cart Operations

Local Machine Results:

The below graph shows the throughput noted on the jmeter run in the local system. There wasn't any out of memory or bound on this operation due to system constraints. Hence, the input for the cloud testing was maintained the same as

700 users * 10 iterations * 32 URL\iteration = 56,000 Samples ---- 1 Min



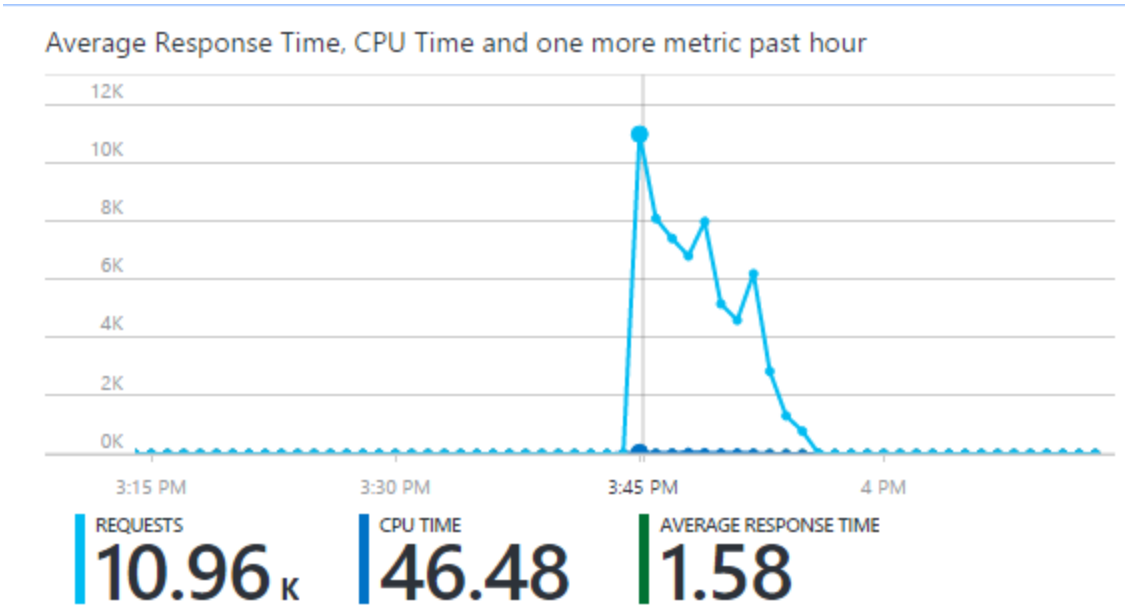
CPU % Results:

Cloud Results: Input: 56,000 Samples, Ramp Up: 1 Minute

After understanding the application behavior, the second transaction was tested in the cloud with 56,000 samples on a ramp up period of 1 minute. Various combinations of CPU% were taken on Small, Medium, Large instances scaling rules were tested and optimal strategies concluded.

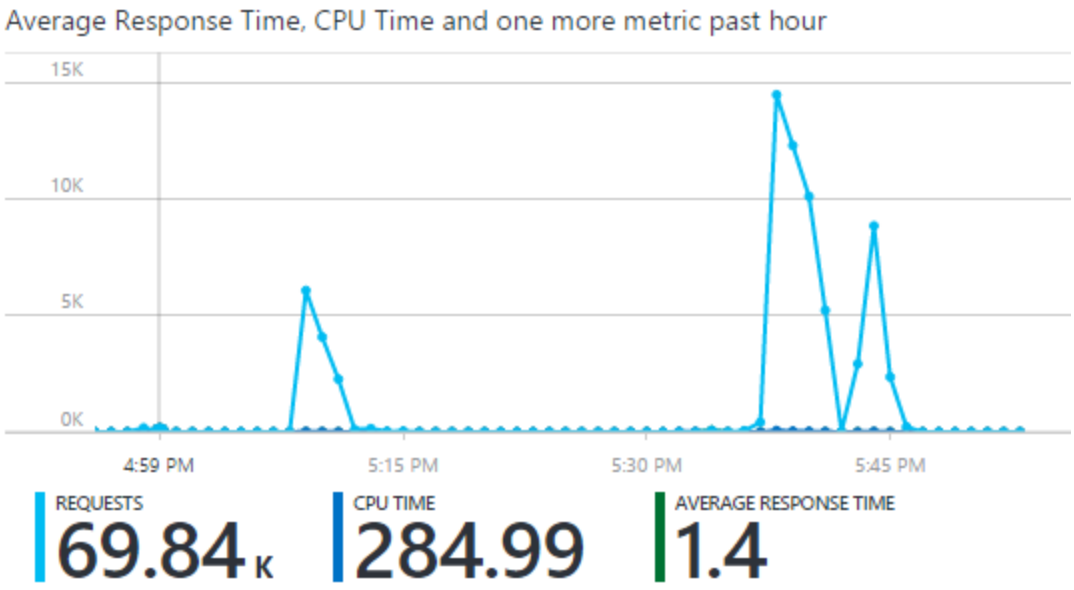
TABLE: 56,000 SAMPLES\10 SECONDS RUN			
Metrics	**SMALL**	**MEDIUM**	**LARGE**
CPU%	**75%**	**75%**	**75%**
Average Response Time:	**288 ms**	**2.3**	**3.5**
Throughput\sec:	**91.8**	**96.6**	**130.4**
CPU Time in sec:	**5.43**	**6.50**	**8.3**

CPU%	80%	80%	80%
Average Response Time:	350 ms	1.4	Not Recorded as can stand higher %
Throughput\sec:	85.5	110.6	
CPU Time in sec:	6.2	4.91	
CPU%	85%	85%	85%
Average Response Time:	578ms	1.02	2.6
Throughput\sec:	44.7	98.65	315.2
CPU Time in sec:	6.28	6.01	6.2
CPU%	90%	90%	90%
Average Response Time:	Not Recorded as cannot stand higher%	Not Recorded as cannot stand higher%	5.6
Throughput\sec:			238.9
CPU Time in sec:			10.2



METRIC	AVERAGE	MINIMUM	MAXIMUM	TOTAL
Average Respons...	198 ms	0 ms	2.82 s	--
CPU Time	5.43 s	15 ms	52.86 s	5.43 m
Requests	1.03 k	0	10.96 k	62.03 k

Fig: Metrics from MS Azure – Small Scale



METRIC	AVERAGE	MINIMUM	MAXIMUM	TOTAL
Average Respons...	1.4 s	0 ms	1.19 m	--
CPU Time	4.91 s	46 ms	1.09 m	4.75 m
Requests	1.2 k	0	14.45 k	69.84 k

Fig: Metrics from MS Azure – Medium Scale

Small Instance:

Strategy:

- CPU% >=75% Increase count by 1
- CPU% <=60% Decrease count by 1

Medium Instance:

Strategy:

- CPU% >=85% Increase count by 1
- CPU% <=60% Decrease count by 1

Large Instance:

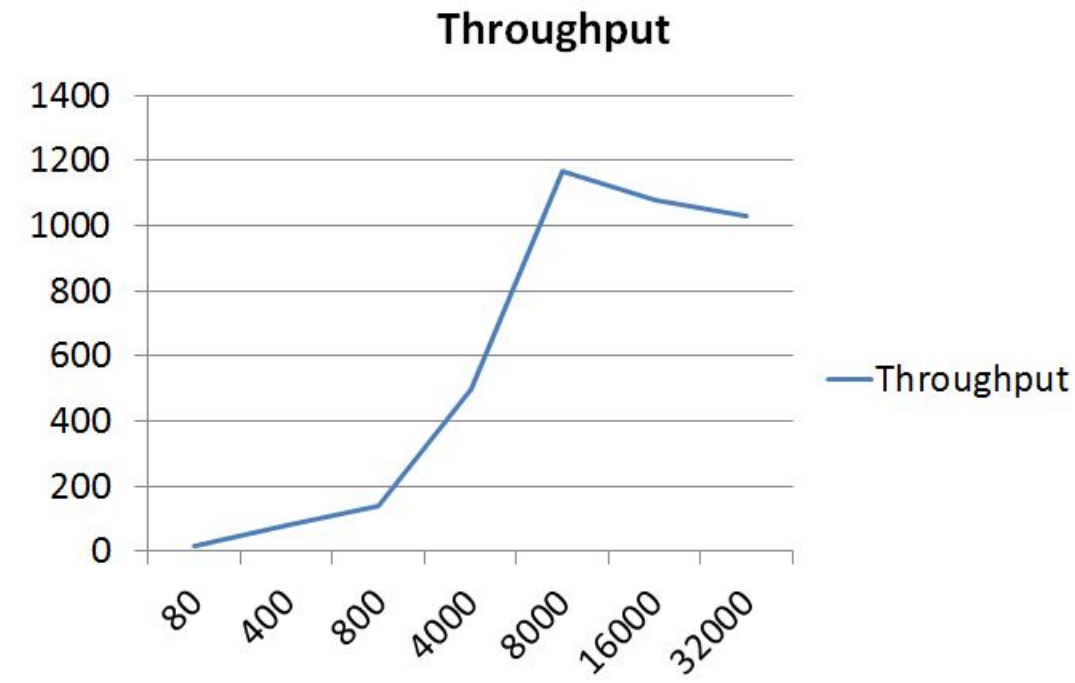
Strategy:

- CPU% >=90% Increase count by 1
- CPU% <=60% Decrease count by 1

Transaction #4: Manage Accounts

Local Machine Results:

The JMeter was unresponsive when the users was 500. When the users were specified as 450 for 5 iterations and 10 seconds, the throughput decreased drastically to 6.8/s and hence this is the minimum load needed to test the application.



Manage Account – Local Run
X axis – Samples Y axis - Throughput

CPU % Results:

Cloud Results: **Input: 36,000 Samples, Ramp Up: 10 seconds**

In the cloud, the run was done for 36000 samples on a ramp up period of 10 seconds. Various combinations of CPU% were taken on Small, Medium, Large instances scaling rules were written only for scale up, keeping the scale down value to default CPU<=60%. The results are documents in the below table.

TABLE: 36000 SAMPLES\10 SECONDS RUN			
Metrics	SMALL	MEDIUM	LARGE

CPU%	70%	70%	70%
Throughput\sec:	82.2/s	200.7/s	Not Recorded as can stand higher %
CPU%	80%	80%	80%
Throughput\sec:	120.4/s	161.8/s	132.4/s
CPU%	85%	85%	85%
Throughput\sec:	125.7/s	160.6/s	130.9/s
CPU%	90%	90%	90%
Throughput\sec:	Not Recorded as cannot stand higher%	Not Recorded as cannot stand higher%	144.5
CPU%	95%	95%	95%
Throughput\sec:	Not Recorded as cannot stand higher%	Not Recorded as cannot stand higher%	238.4

Optimal Strategies Found:

The best found strategies are shown below:

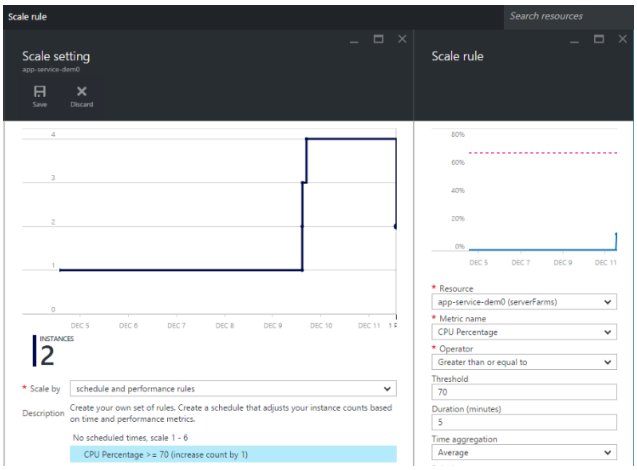
Rule: When the CPU utilization >= 70 increase the instance by 1 (Small Core)

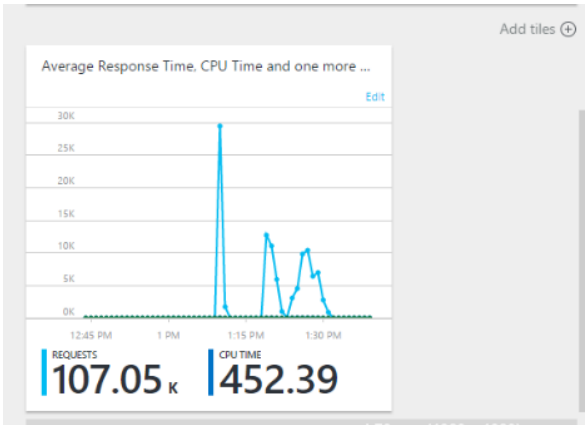
Results from Aggregate Report:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
Enter The Stor...	4500	7342	6524	15910	17992	21235	76	24631	0.02%	12.6/sec	13.2
Catalog - HTT...	4500	4838	3531	13017	14987	16726	41	18628	0.00%	12.5/sec	66.6
Sign In Page - ...	4500	4899	3399	12176	15685	22520	39	22667	0.00%	12.6/sec	46.0
Sign In- HTTP ...	4500	12444	12245	21698	23342	25962	88	39034	0.00%	12.2/sec	62.1
My Account- H...	4500	5799	5010	13342	14625	18437	40	19185	0.00%	12.6/sec	76.1
Updating User...	4500	4965	5162	9802	11049	14833	43	20256	0.00%	12.4/sec	75.4
Reseting Use...	4500	5261	5244	9718	11210	14988	44	18644	0.00%	12.5/sec	75.8
Sign Out- Http ...	4500	4793	3531	10405	12496	18361	39	18621	0.00%	12.9/sec	49.4
TOTAL	36000	6293	5090	14625	17697	22860	39	39034	0.00%	82.2/sec	381.2

Instance Graph

Average Response time, CPU time

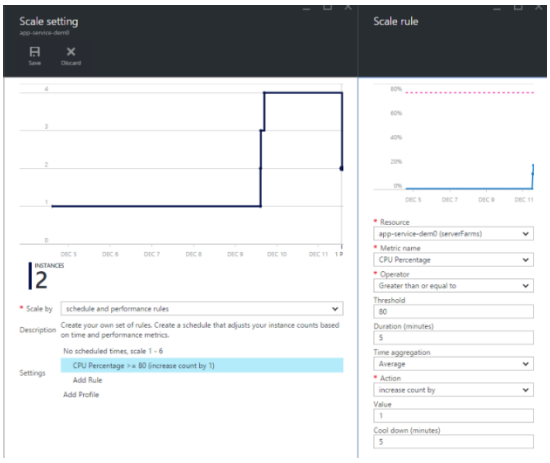




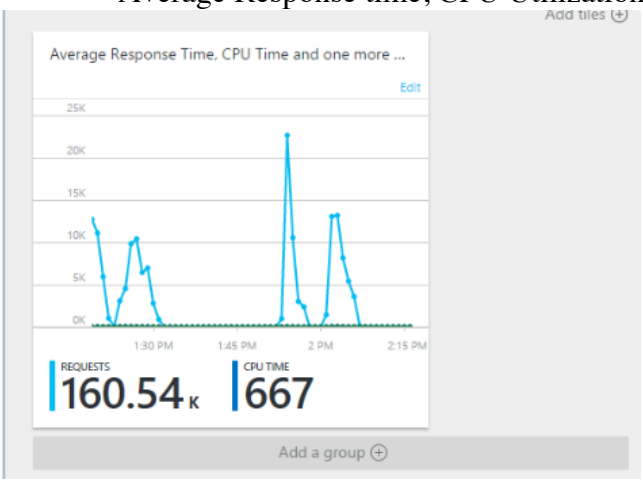
Rule 2: When CPU utilization>=80, increase the instance by 1 (Small Core).
Aggregate Report:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
Enter The Stor...	4500	5146	573	17628	28737	42663	75	54168	0.00%	15.6/sec	16.3
Catalog - HTT...	4500	1749	292	5650	7829	11187	38	14609	0.00%	15.4/sec	82.0
Sign In Page - ...	4500	1887	383	5966	7674	12094	36	16694	0.00%	15.4/sec	56.0
Sign In- HTTP ...	4500	3374	978	11154	16579	18535	79	21307	0.00%	15.3/sec	78.0
My Account- H...	4500	2132	624	6592	8598	10637	40	19290	0.00%	15.3/sec	92.5
Updating User...	4500	1996	776	4980	6910	10157	40	21132	0.00%	15.3/sec	92.7
Reseting Use...	4500	1815	758	4600	6167	10971	40	21149	0.00%	15.3/sec	92.7
Sign Out- Http ...	4500	1278	280	4489	5474	6891	38	15059	0.00%	15.4/sec	59.1
TOTAL	36000	2422	530	6578	10016	24589	36	54168	0.00%	120.7/sec	559.5

Instances Created



Average Response time, CPU Utilization:

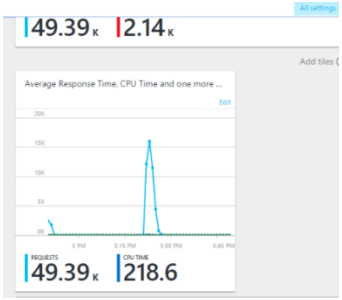


Rule: When CPU utilization > = 80 , increase an instance by 1. Medium Cores
Instance to be added are medium (2 cores)

Aggregate Report:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
Enter The Stor...	4500	3124	2201	6673	13498	20146	74	34551	0.00%	21.1/sec	22.0
Catalog - HTT...	4500	1067	591	2681	3022	5545	39	8519	0.00%	20.9/sec	111.0
Sign In Page - ...	4500	1236	831	2691	3882	6568	40	7233	0.00%	20.6/sec	75.1
Sign In- HTTP ...	4500	2342	2409	4570	5065	8390	83	9825	0.00%	20.4/sec	103.7
My Account- H...	4500	1198	1084	2534	2793	3800	40	11182	0.02%	20.4/sec	123.2
Updating User...	4500	1265	1056	2717	3362	6430	39	7284	0.02%	20.4/sec	123.8
Resetting Use...	4500	1233	933	2745	3258	3984	40	7576	0.00%	20.4/sec	123.4
Sign Out- Http ...	4500	896	457	2131	2696	4384	38	18954	0.04%	20.4/sec	78.3
TOTAL	36000	1545	1007	3373	4453	8961	38	34551	0.01%	161.3/sec	747.9

Average Response time, CPU Time



Instance Graph

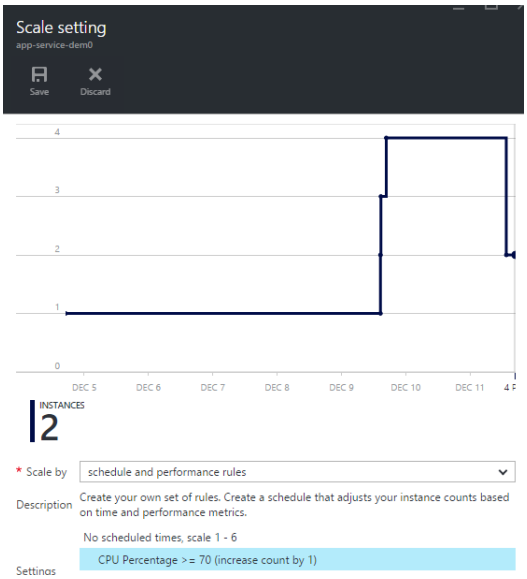
Rule: When CPU utilization >=70 increase an instance by 1 and use medium core processors
USers 900, Ramp up time 10, Iterations 5

Number of Instances created are 2

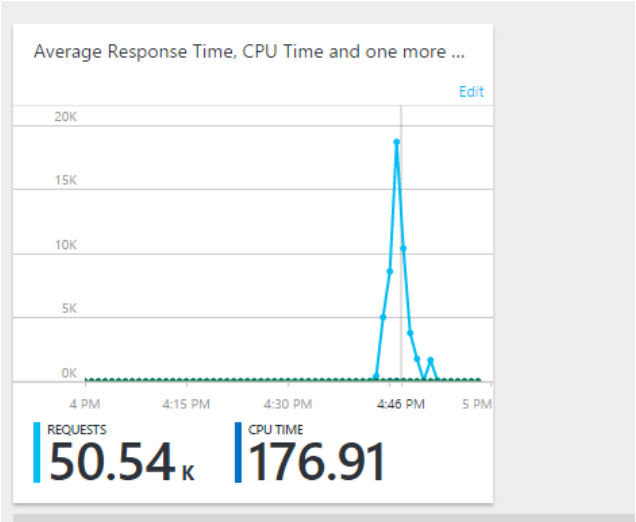
Aggregate Report

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
Enter The Stor...	4500	5417	2268	7163	10141	125680	78	136378	89.22%	25.5/sec	11.4
Catalog - HTT...	4500	2097	2207	2347	2403	4668	41	16298	89.22%	25.5/sec	20.1
Sign In Page - ...	4500	2070	2210	2406	2471	2641	42	121971	89.22%	25.4/sec	15.5
Sign In- HTTP ...	4500	2142	2201	2368	2435	2532	83	132383	89.22%	25.5/sec	19.5
My Account- H...	4500	2096	2192	2436	2534	4644	43	11026	89.09%	28.5/sec	24.9
Updating User...	4500	3025	2235	8379	10500	14647	44	17898	78.16%	28.7/sec	44.7
Resetting Use...	4500	3155	2210	7041	9673	15585	42	21449	71.38%	29.1/sec	55.5
Sign Out- Http ...	4500	2245	2182	3185	4632	7610	39	11339	71.38%	29.4/sec	36.3
TOTAL	36000	2781	2212	2568	6289	12574	39	136378	83.36%	200.5/sec	205.2

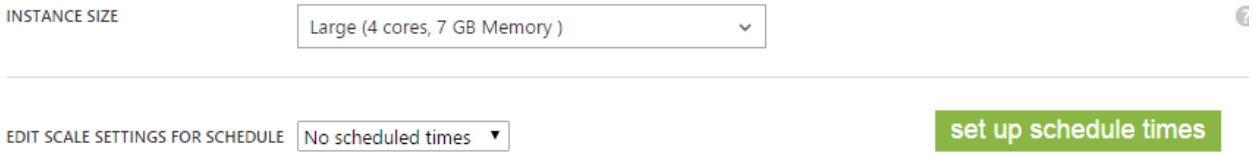
Instance graph



Average Response time, CPU Time



Rule: When CPU utilization >=80 increase an instance by 1 and use large 4 core processors
USers 900, Ramp up time 10, Iterations 5

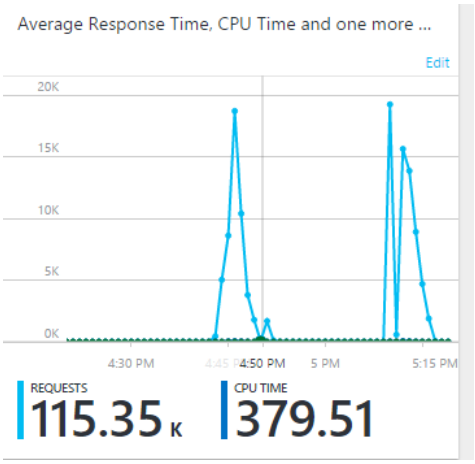


Aggregate Report

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
Enter The Stor...	4500	628	138	2419	2950	3467	72	10638	0.00%	17.9/sec	18.8
Catalog - HTT...	4500	693	83	2469	3262	5628	37	14446	0.00%	17.0/sec	90.6
Sign In Page - ...	4500	473	75	1916	2319	3674	35	7227	0.00%	17.1/sec	62.1
Sign In- HTTP ...	4500	1062	152	4757	5376	6474	77	11165	0.00%	17.0/sec	86.6
My Account- H...	4500	409	79	1530	1825	3478	38	15804	0.00%	17.2/sec	103.9
Updating User...	4500	473	74	1701	2022	4185	40	14711	0.00%	17.4/sec	105.9
Resetting Use...	4500	543	75	1690	3658	4698	40	11803	0.00%	17.4/sec	105.7
Sign Out- Http ...	4500	308	62	1264	1669	2832	36	4164	0.04%	17.5/sec	67.3
TOTAL	36000	574	96	1898	3056	5240	35	15804	0.01%	132.4/sec	613.8

Instance graph

Average Response time, CPU Time

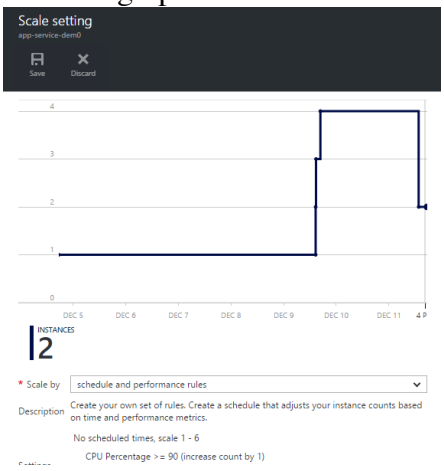


Rule: When CPU utilization >=90 increase an instance by 1 and use large core processors
USers 900, Ramp up time 10, Iterations 5

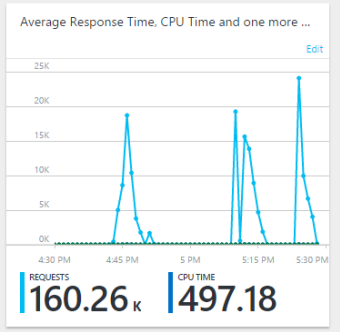
Aggregate Report

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
Enter The Stor...	4500	403	158	1070	1213	3431	72	4652	0.00%	20.8/sec	21.7
Catalog - HTT...	4500	248	83	501	758	2199	39	18936	0.02%	19.6/sec	104.0
Sign In Page - ...	4500	167	68	416	583	1246	36	5360	0.00%	19.0/sec	69.3
Sign In- HTTP ...	4500	348	152	914	1146	2167	77	4703	0.00%	18.6/sec	94.7
My Account- H...	4500	211	81	547	680	2020	39	6828	0.00%	18.4/sec	111.0
Updating User...	4500	180	75	493	593	1585	39	4696	0.00%	18.3/sec	110.9
Resetting Use...	4500	186	75	482	599	1673	40	14776	0.00%	18.2/sec	110.2
Sign Out- Http ...	4500	151	64	368	552	1486	36	11304	0.00%	18.2/sec	69.9
TOTAL	36000	237	96	572	914	1970	36	18936	0.00%	144.7/sec	670.7

Instance graph



Average Response time, CPU Time



Summary:

For small core processors, the rule “Increase an instance by 1 when the CPU utilization >= 80%” is a better provisioning strategy since it produces higher throughput of 120.7/s

For medium core processors, the rule “Increase an instance by 1 when the CPU utilization >= 70%” is a better provisioning strategy since it produces higher throughput of 200.5/s

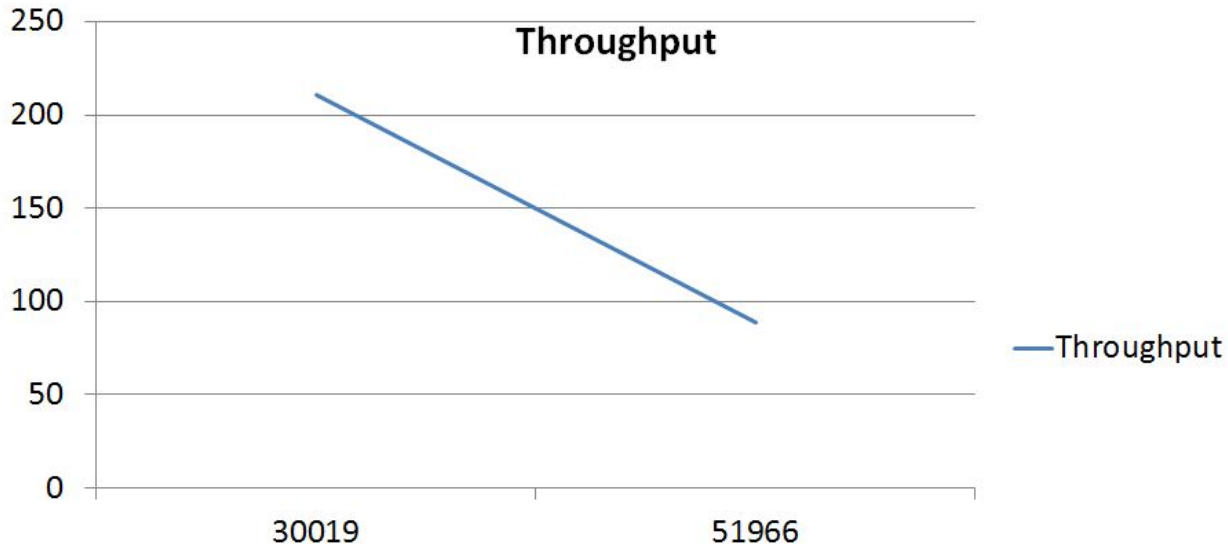
For larger core processors, the rule “Increase an instance by 1 when the CPU utilization >= 90%” is a better provisioning strategy since it produces higher throughput of 144.5/s

Transaction #5: View Orders

Local Machine Results:

The below graph shows the throughput noted on the jmeter run in the local system. There wasn't any out of memory or bound on this operation due to system constraints. Hence, the input for the cloud testing was maintained the same as

100 users * 10 iterations * 20 URL\iteration = 20,000 Samples



View Orders – Local Run
X axis – Samples Y axis - Throughput

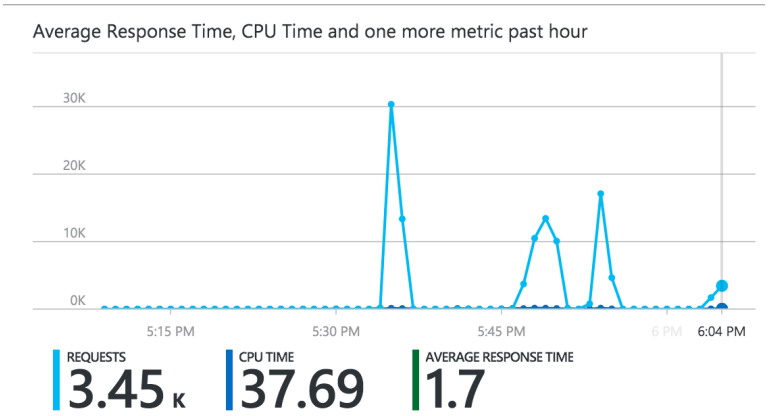
CPU % Results:

Cloud Results: Input: 56,000 Samples, Ramp Up: 1 Minute

After understanding the application behavior, the second transaction was tested in the cloud with 20,000 samples on a ramp up period of 1 minute. Various combinations of CPU% were taken on Small, Medium, Large instances scaling rules were tested and optimal strategies concluded.

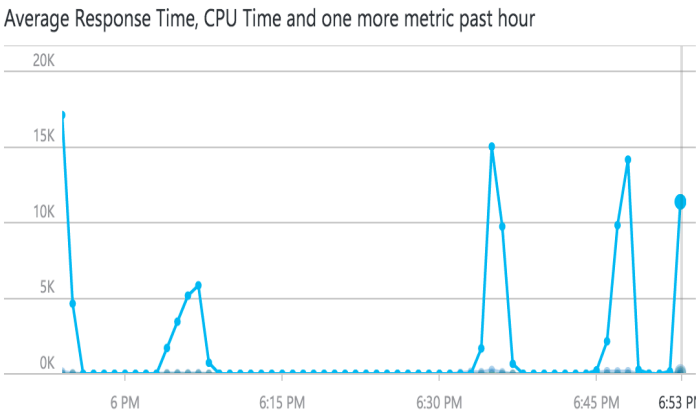
TABLE: 20,000 SAMPLES\10 seconds Runs			
Metrics	**SMALL**	**MEDIUM**	**LARGE**
CPU %>= (Scale up)	**75%**	**85%**	**90%**

CPU %<= (Scale down)	60%	60%	60%
No of Requests at server:	1.92K	1.73K	1.51K
Average Response Time:	3	617ms	640ms
Throughput\sec:	218.5	413.2	236.1
CPU Time in sec:	22.34	26.88	26.88
No of Requests at server:	1.77K	1.79K	
CPU %>= (Scale up)	85%	90%	
CPU %<= (Scale down)	60%	60%	
Average Response Time:	2.3	682ms	
Throughput\sec:	220.8	228	
CPU Time in sec:	20.12	22.35	
No of Requests at server:	1.7K		
CPU %>= (Scale up)	80%		
CPU %<= (Scale down)	60%		
Average Response Time:	1.2		
Throughput\sec:	221		
CPU Time in sec:	19.32		



METRIC	AVERAGE	MINIMUM	MAXIMUM	TOTAL
Average Respons...	3 s	0 ms	1.36 m	--
CPU Time	22.34 s	0 ms	2.49 m	21.23 m
Requests	1.92 k	0	30.35 k	109.59 k

Fig: Metrics from MS Azure- Small Instance



METRIC	AVERAGE	MINIMUM	MAXIMUM	TOTAL
Average Respons...	617 ms	0 ms	9.2 s	--
CPU Time	29.8 s	0 ms	4.3 m	29.8 m
Requests	1.73 k	0	17.11 k	104 k

Fig: Metrics from MS Azure- Medium Instance

Optimal Strategies Found:
Small Instance:

Strategy:

- CPU% >=75% Increase count by 1
- CPU% <=60% Decrease count by 1

Medium Instance:

Strategy:

- CPU% >=85% Increase count by 1
- CPU% <=60% Decrease count by 1

Large Instance:

Strategy:

- CPU% >=90% Increase count by 1
- CPU% <=60% Decrease count by 1

CONCLUSION:

The JPetStore application is a good fit for the cloud due to the following reasons:

- It doesn't depend on a heavy database, no transactions to save huge amounts of data
- It doesn't maintain user sessions, hence no requirements of spawning new threads\persistent objects

The architecture is loosely coupled and works perfectly with no issues or bottlenecks. The load on the application had to be increased to huge amounts inorder to break it. The provisioning rules can be extremely flexible based on consumer requirements.