**Distributed Objects for Cloud Computing Project**

**Title: JPetStore Application**

Kuttabadkar Yogeeta Monica ykutta2@uic.edu

TABLE OF CONTENTS

[Application Overview: JPetStore 3](#_Toc437615062)

[Technical Characteristics of Application: 4](#_Toc437615063)

[Goal of the Report: 5](#_Toc437615064)

[Local Deployment 5](#_Toc437615065)

[Running JpetStore with Tomcat: 5](#_Toc437615066)

[Cloud Deployment of the application: 5](#_Toc437615067)

[Business Transactions of JPetStore6: 7](#_Toc437615068)

[Description of Business Transactions: 7](#_Toc437615069)

[JMeter Scripts: 12](#_Toc437615070)

[General JMeter Settings: 12](#_Toc437615071)

[JMeter Scripts Detailed Descriptions: 15](#_Toc437615072)

[Transaction# 1 Transaction Name: Register New Users 15](#_Toc437615073)

[Transaction# 2 Transaction Name: Browsing, Adding to Cart and Checkout 21](#_Toc437615074)

[Transaction# 3 Transaction Name: Cart Operations (Update\Remove) items from cart 24](#_Toc437615075)

[Transaction# 4 Transaction Name: Manage Account 28](#_Toc437615076)

[Transaction# 5 Transaction Name: View orders 29](#_Toc437615077)

[Auto Scaling Strategies: 33](#_Toc437615078)

[Transaction #1: Register New Users 33](#_Toc437615079)

[Local Machine Results: 33](#_Toc437615080)

[CPU % Rules: 34](#_Toc437615081)

[Cloud Results: Input: 10,000 Samples, Ramp Up: 10 seconds 34](#_Toc437615082)

[Cloud Results: Input: 10,000 Samples, Ramp Up: 10 minutes 36](#_Toc437615083)

[Optimal Strategies Found: 37](#_Toc437615084)

[Memory % Rules: 41](#_Toc437615085)

[Optimal Strategy Found: 42](#_Toc437615086)

[Transaction #2: Browsing & Adding to Cart 42](#_Toc437615087)

[Local Machine Results: 42](#_Toc437615088)

[CPU % Results: 43](#_Toc437615089)

[Cloud Results: Input: 20,000 Samples, Ramp Up: 10 Minutes 43](#_Toc437615090)

[Optimal Strategies Found: 44](#_Toc437615091)

[Memory % Rules: 46](#_Toc437615092)

[Most Optimal Strategies found are: 47](#_Toc437615093)

# 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.

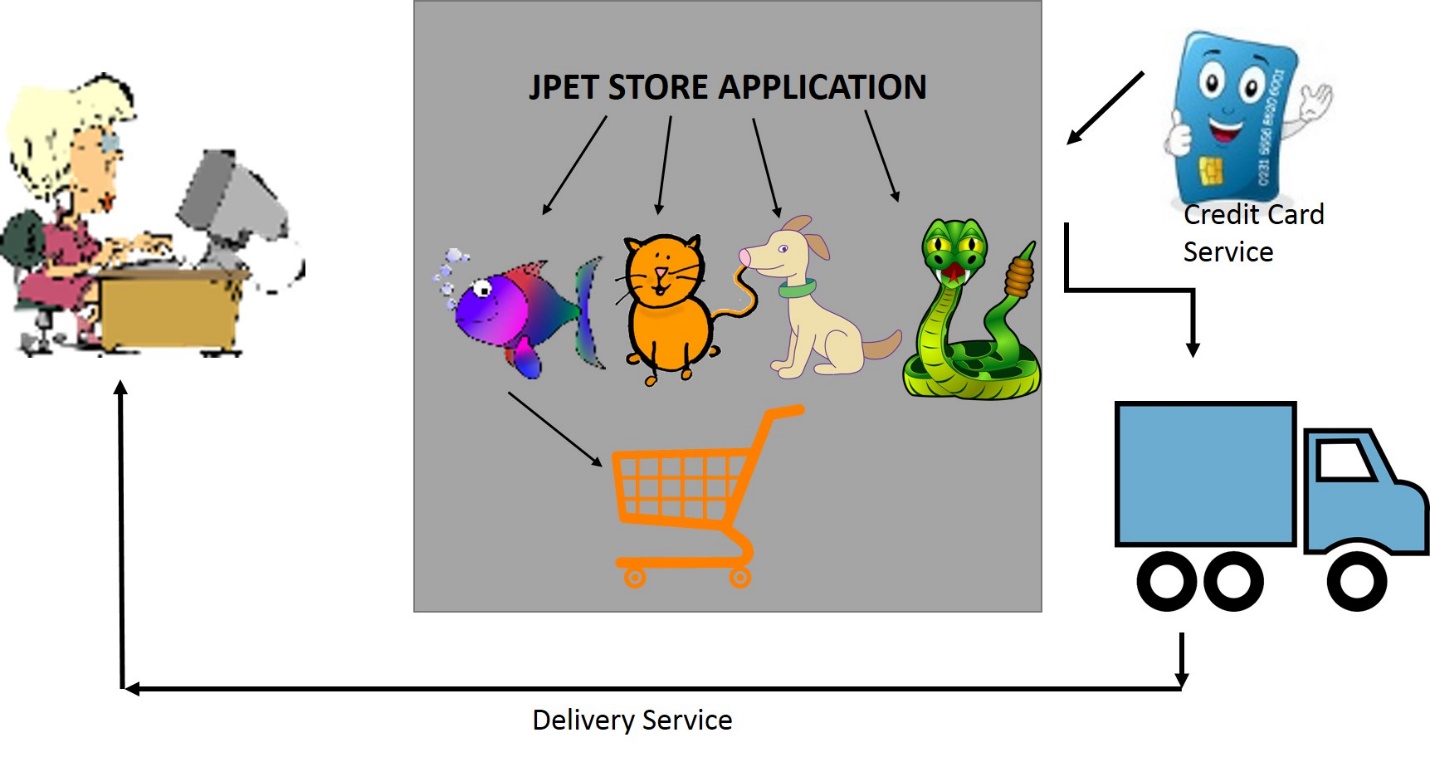


Fig: Overview of JPetStore application

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:**

Since, the JPetStore application is not a heavy weight application, we do not need complete control over the deployment environment. Hence we choose to deploy the application as a Web App and concentrate on writing provisioning rules.

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)

1. Import jpetstore.war file inside this directory.
2. 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 | 1. Enter Username 2. Enter Password 3. Enter Password again for confirmation 4. 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 | 1. On catalog page, select “User Sign-In” option 2. Enter Username 3. Enter Password 4. 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 times) | 1. Select Type of as Fish/Dog/Cat/Bird 2. Select from the sub items of main category 3. Select “Add to cart” option 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 | 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/ 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 | 1. Select the “My Account” option from the top of the page 2. Select “My Orders” option present on bottom of the page | Verify that “My orders” are displayed for the user |
| 9 | View Order | 1. Select confirmed order ID 2. 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 | 1. 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:

1. **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.

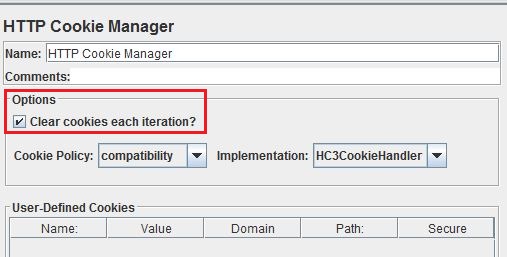


Fig: Cookie Manager Setting

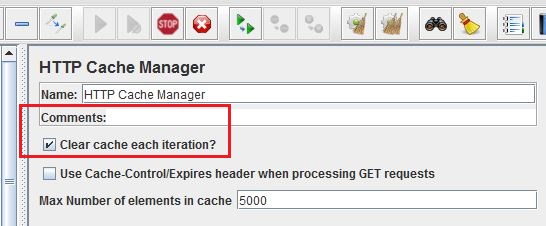
1. **HTTP Cache Manager:** For each iteration the cache is also cleared. Since, caching speeds up response rates, removing this ensures we test the application for peak loads. In the presence of caching the performance metrics would improve. 

Fig: Cache Manager Setting

1. **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.
2. **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

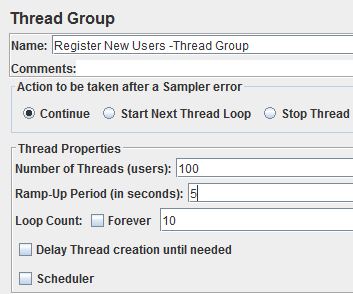


Fig: Thread Group Settings

1. **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**](http://localhost:8080/jpetstore)

**Usage: ${PATH}/catalog.action**

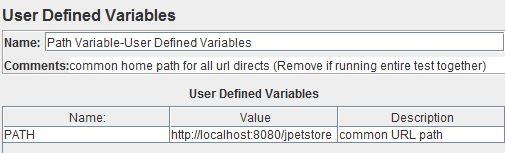


Fig: User Defined “PATH” variable

1. **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}

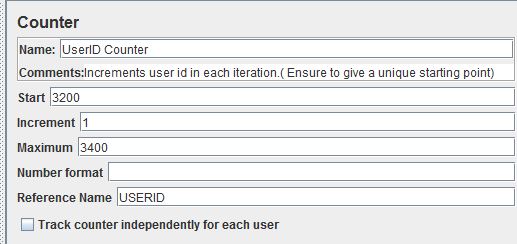


Fig: Counter for “USERID” variable

1. **Assertions**: Assertions are used to fulfil all validations in each step. Mostly Response Assertions verifying the Text Response to desired values.
2. **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.

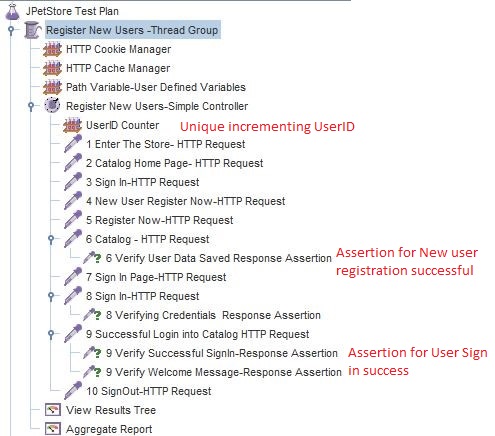


Fig: Screen shot showing steps of New User Registration

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

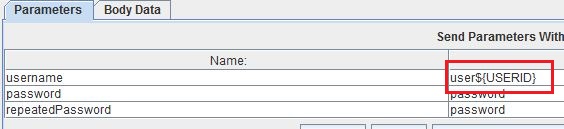


Fig: Screen Shot for Usage of UserID counter

1. **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

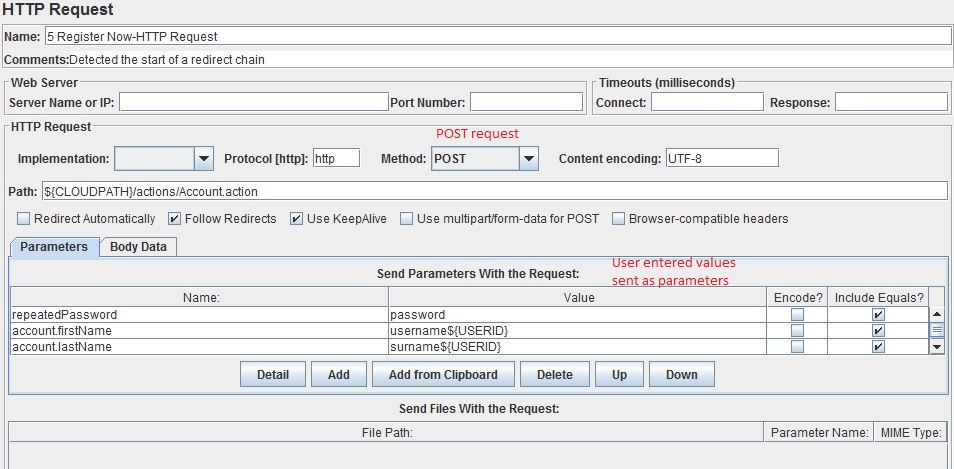


Fig: Screenshot of HTTP Request-Register Now

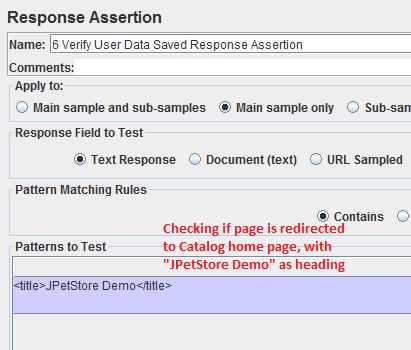


Fig: Screenshot of HTTP Request-Register Now Assertion

1. **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.

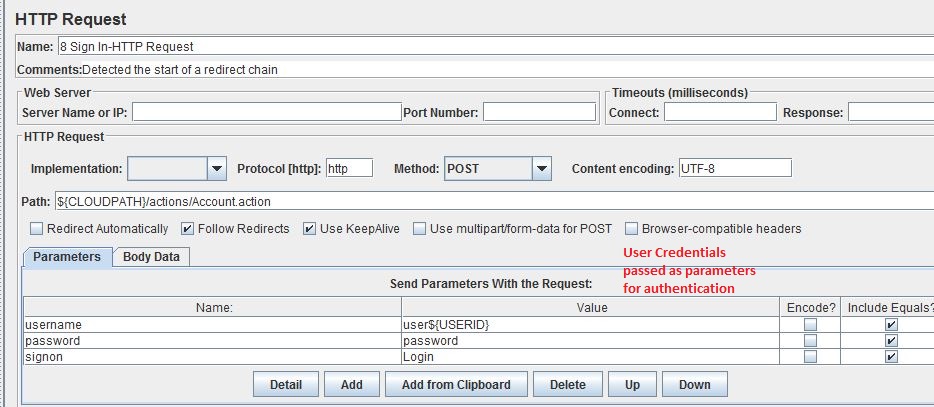


Fig: Screenshot of HTTP Request-Sign In of new user

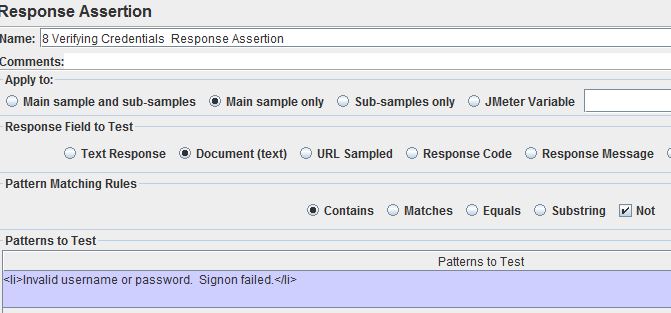


Fig: Screenshot of HTTP Response Assertion-Checking that Invalid username message is NOT present in response

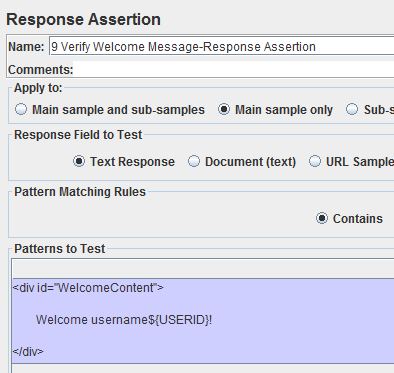


Fig: Screenshot of HTTP Response Assertion-Checking Welcome <Username> is present in response

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.

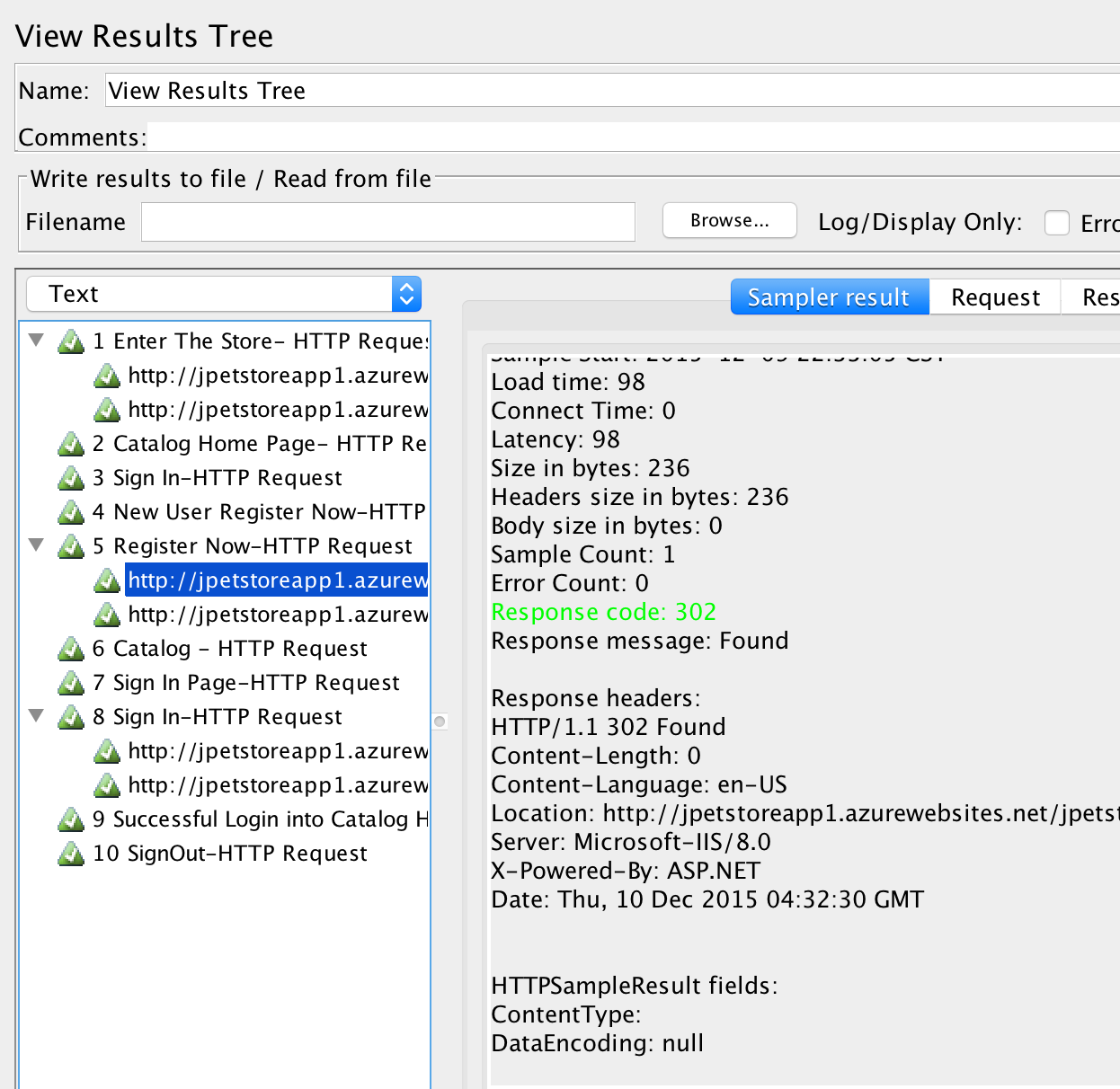


Fig: Screenshot of Thread Group Run- displaying all URL and their assertion values

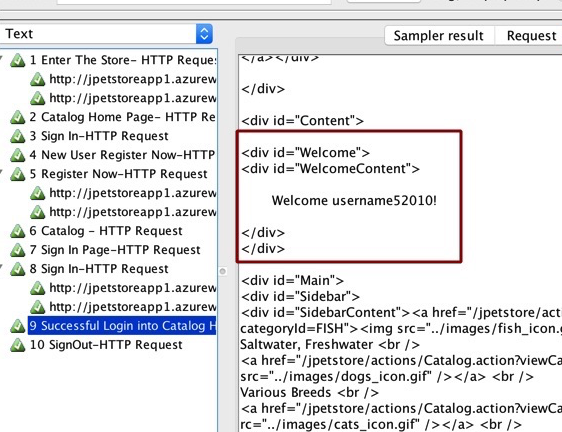
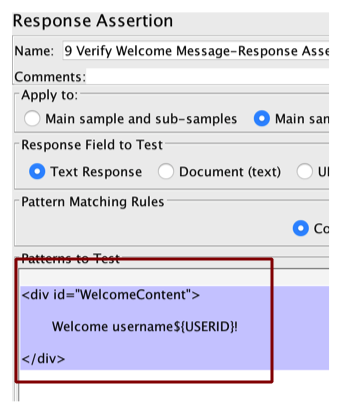


Fig: Screenshot of Thread Group Run- displaying Assertion in rule and the assertion value

## 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:

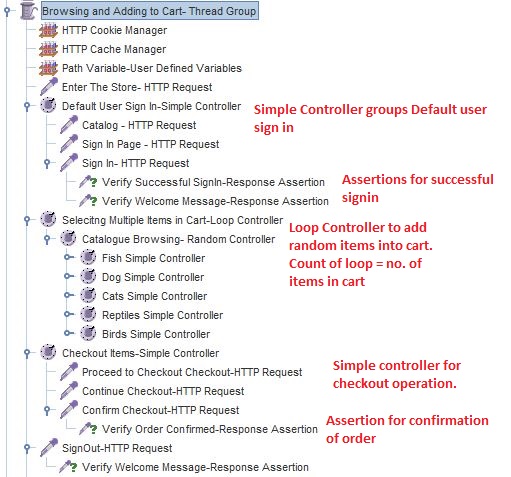


Fig: Screenshot of complete Transaction#2

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:

1. By selecting the “Add to Cart” next to the listed item
2. 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.

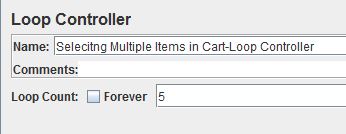


Fig: Loop Controller Count

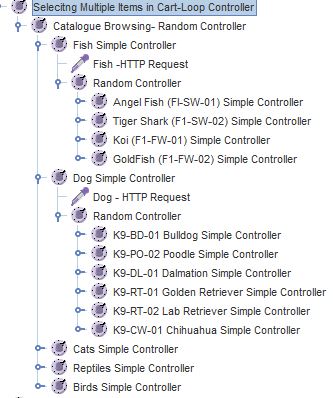


Fig: Screen shot – Selecting random items into cart

1. 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.

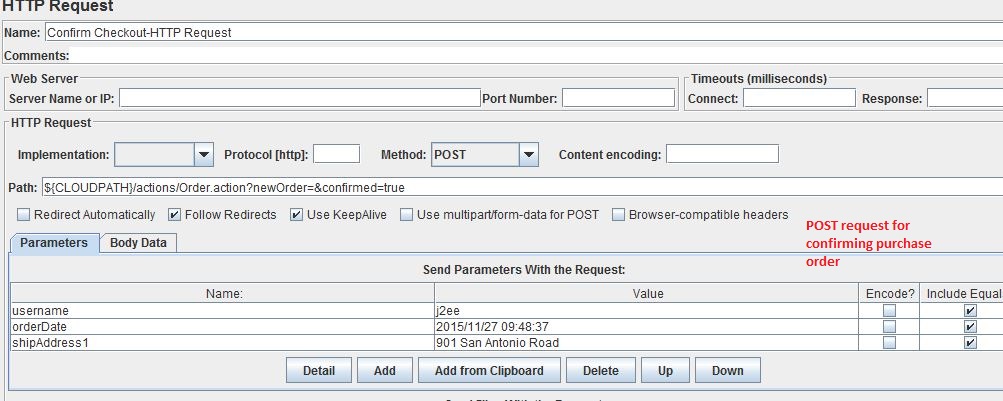


Fig: Screen shot – HTTP Request to confirm checkout

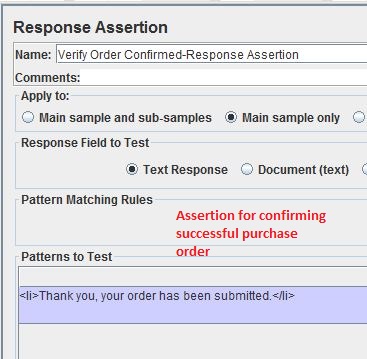


Fig: Screen shot – Assertion to check validation of confirmation message

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.

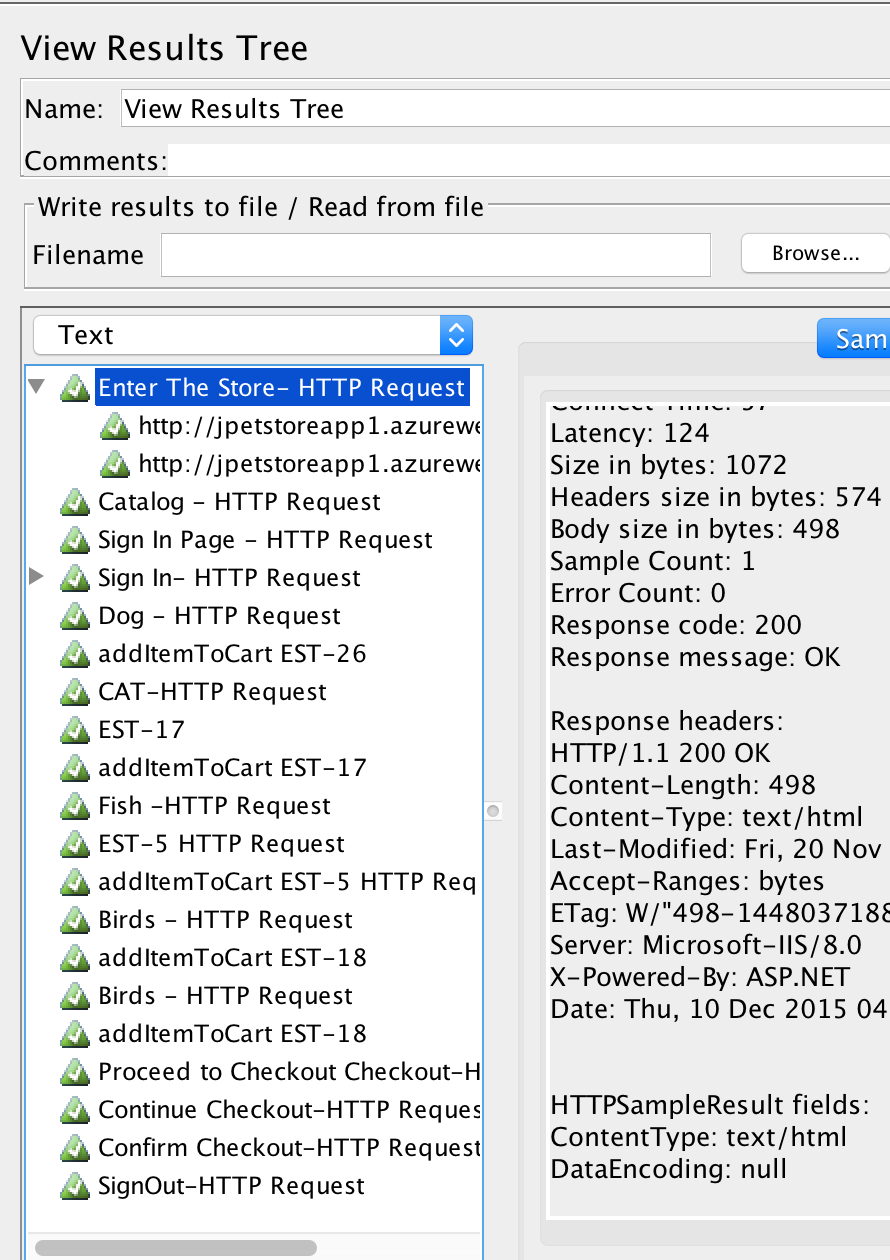
****

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.

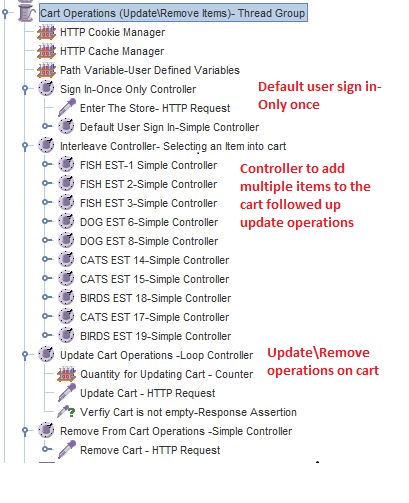


Fig: Screen shot- Thread group for Update Cart operations

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.

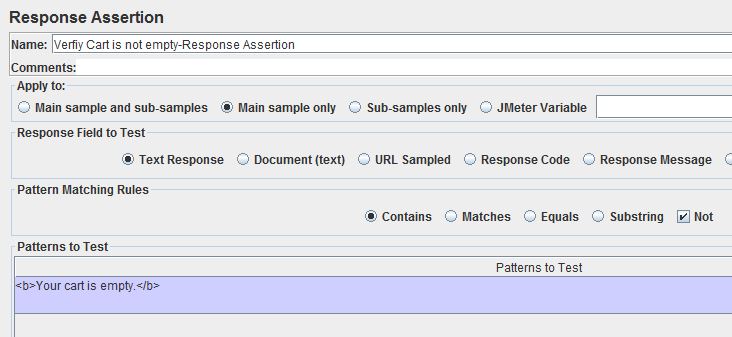


Fig: Assertion checking that cart is NOT empty

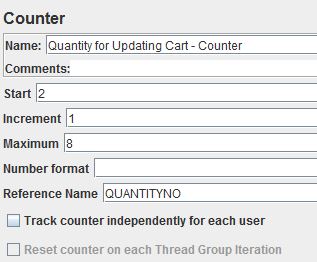


Fig: Counter to increment the number of items in the cart

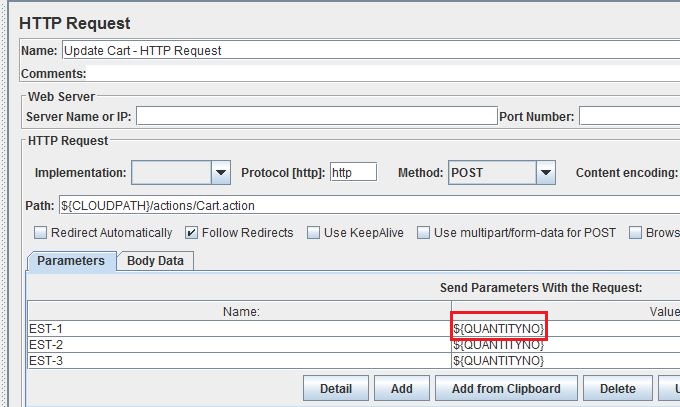
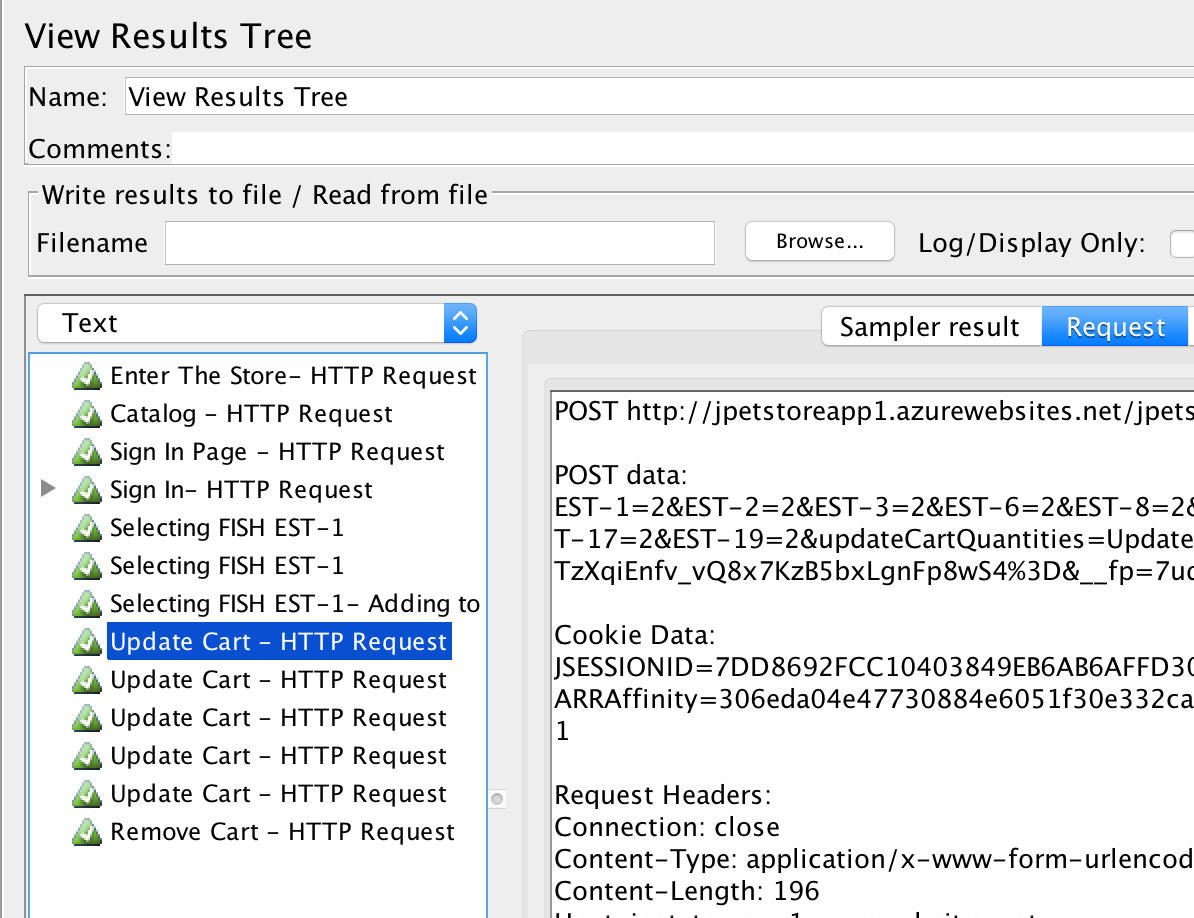


Fig: Using the counter value to update items in the cart

1. **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

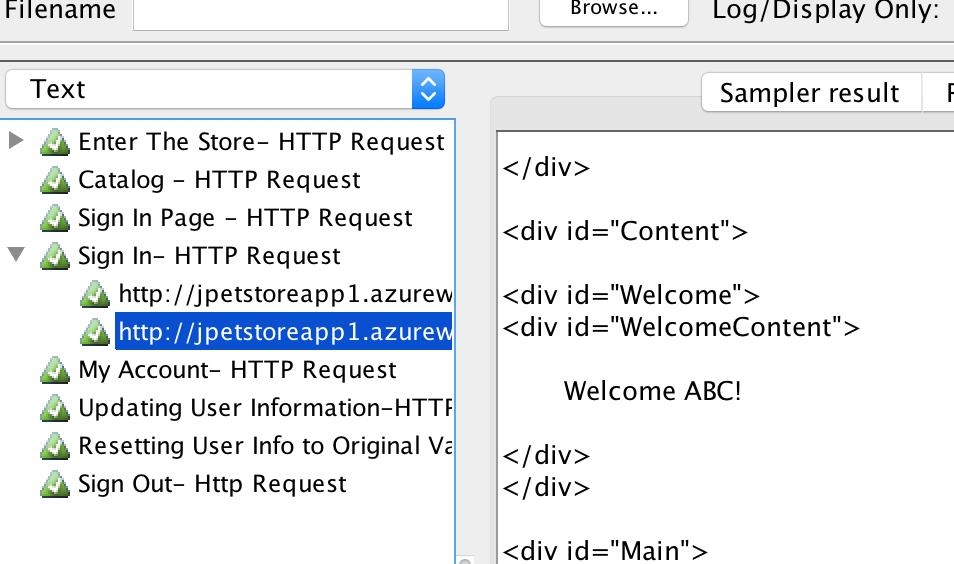
****

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.

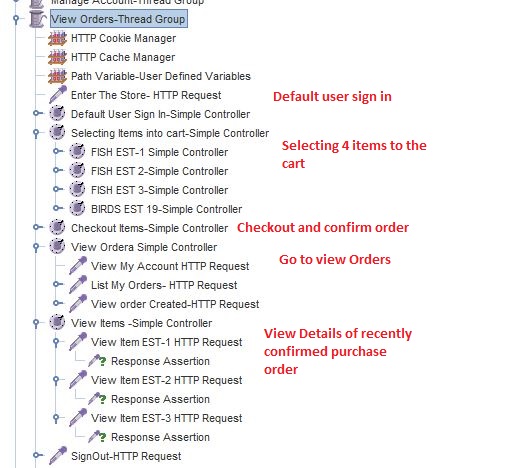


Fig: Screen shot- Thread group for transaction 5

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.

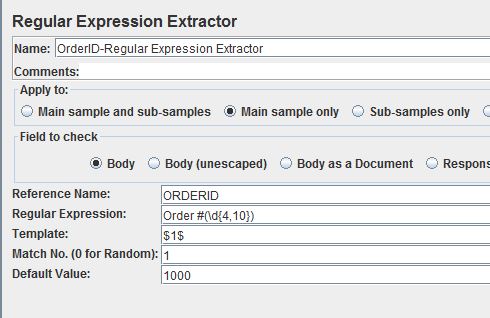
****

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

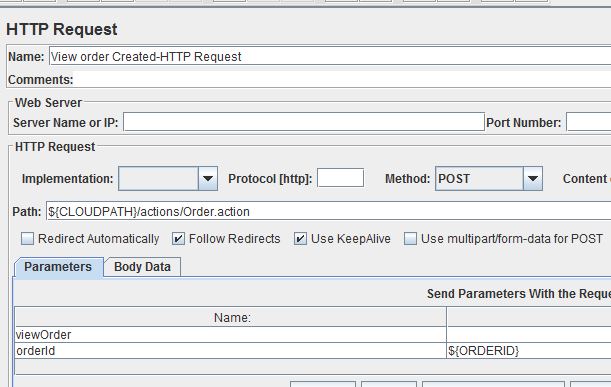


Fig: Using the Order ID to View the confirmed order

1. **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.

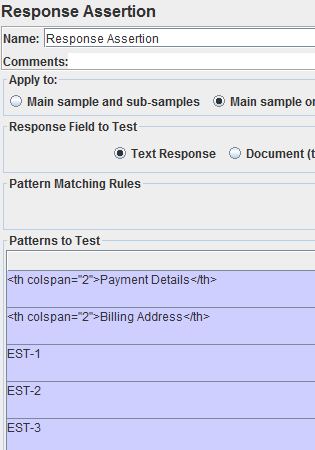


Fig: Assertion to check the items of the order

1. **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.

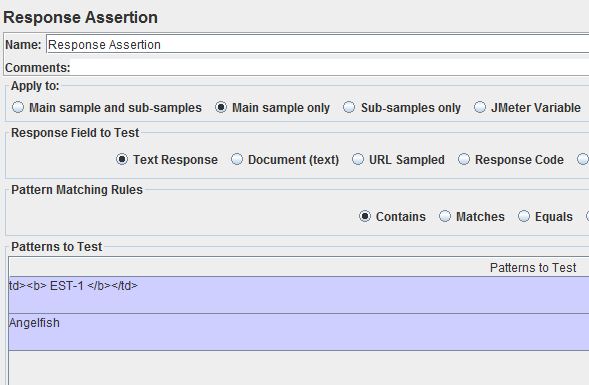
****

Fig: Assertion to check the correct item is displayed

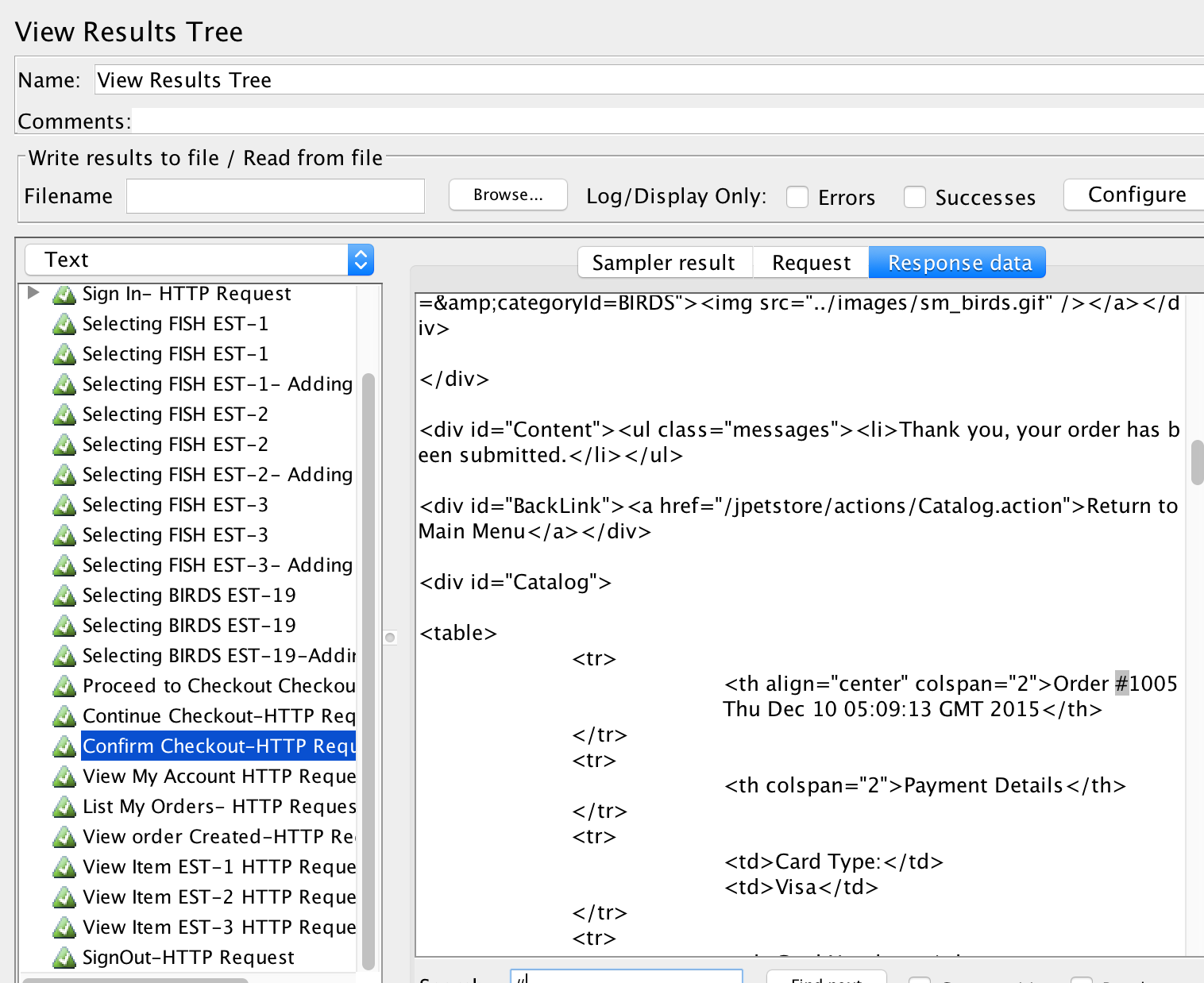
****

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

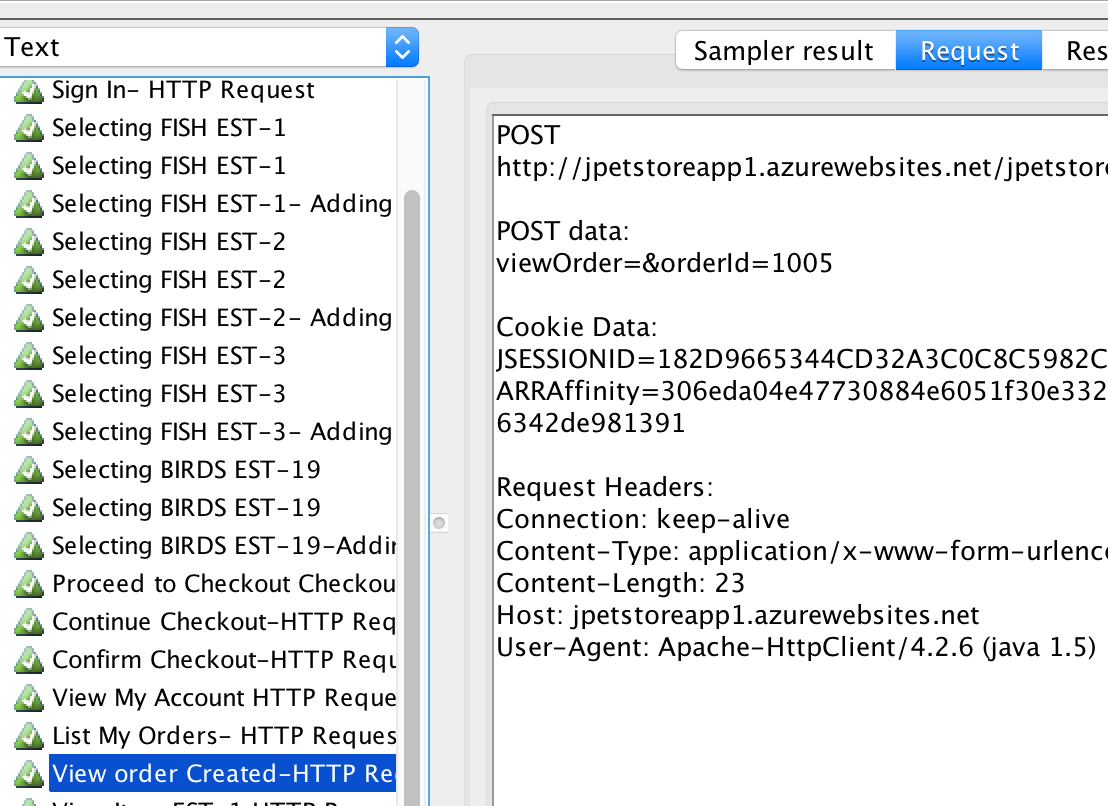
****

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

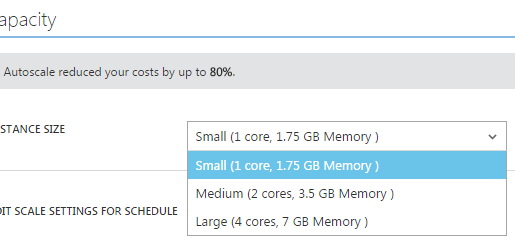


Fig: MS Azure types of Standard Instances

## 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.

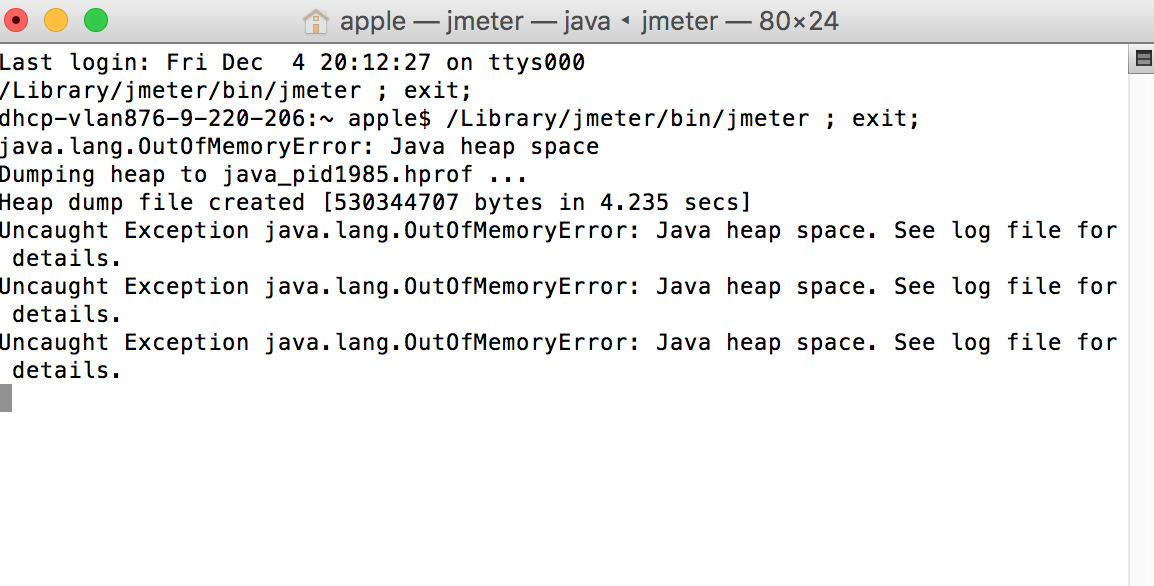


Fig: Out of Memory Exception throw by JMeter

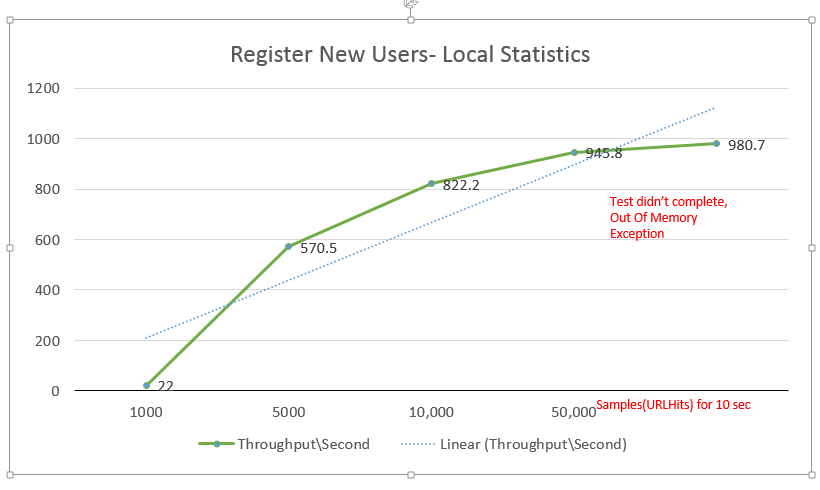


Fig: Graph showing Local statistics of Transaction – 1

## 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

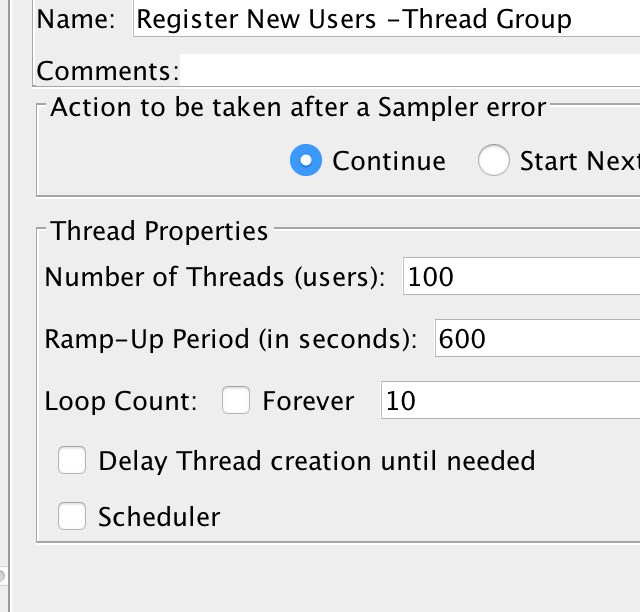


Fig: JMeter setting for Run

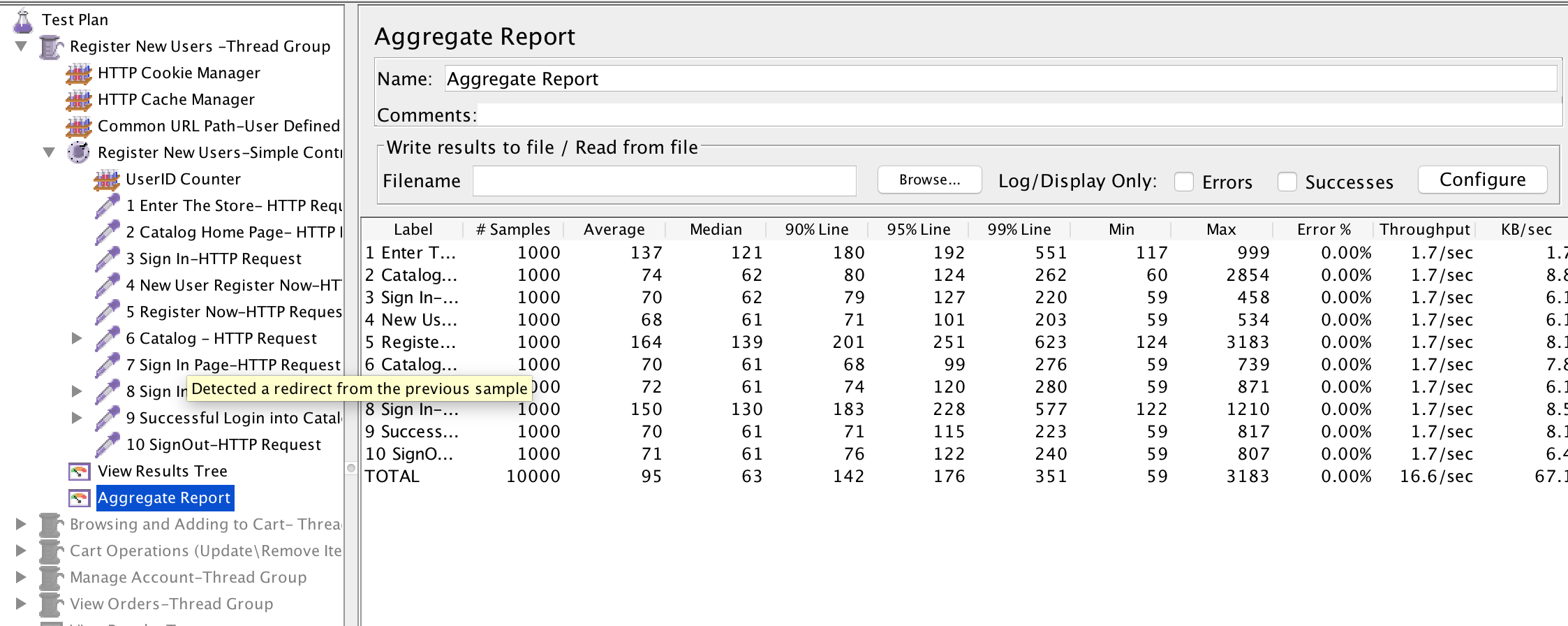


Fig: JMeter Results



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

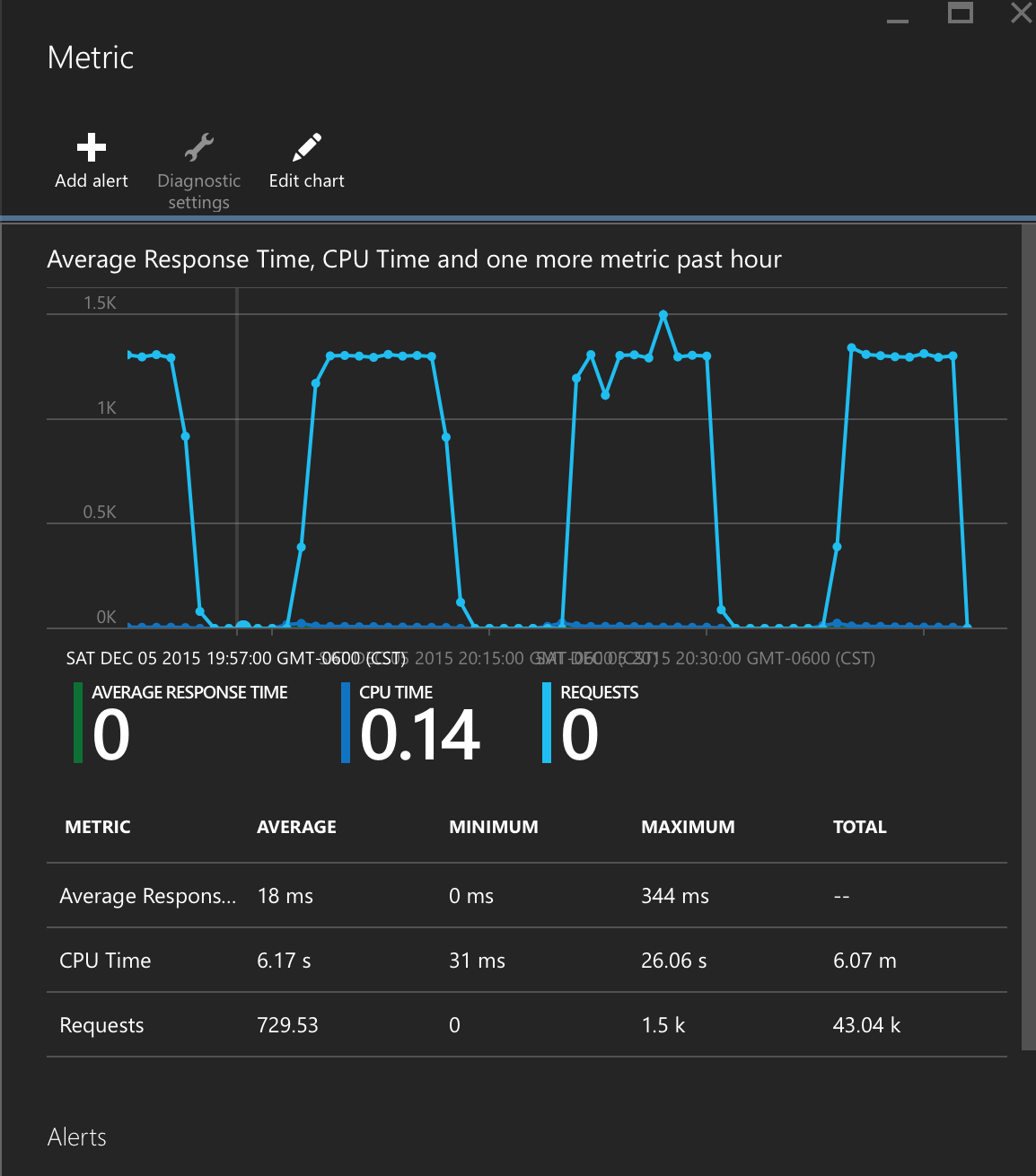
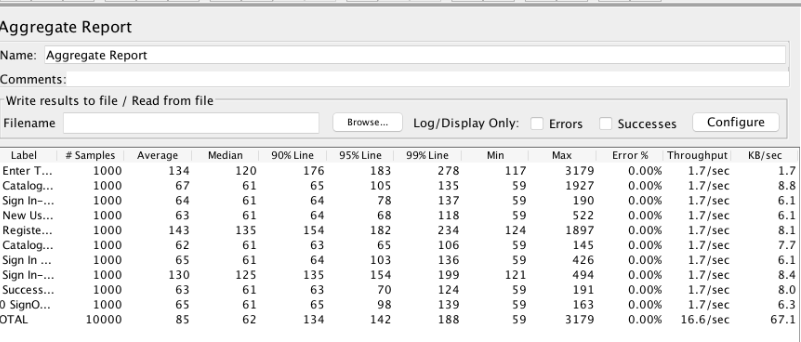


Fig: Metrics from MS Azure

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

# 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

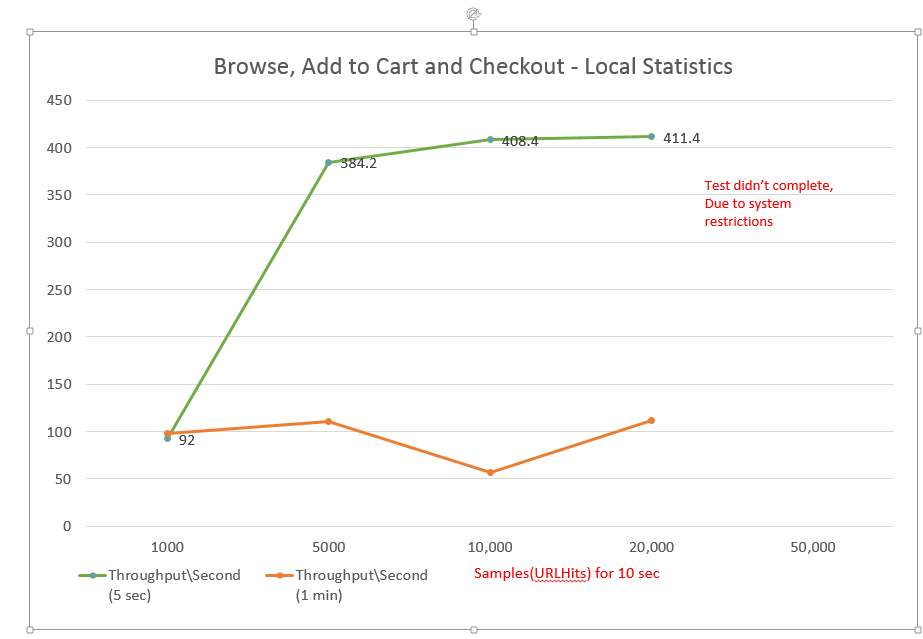


Fig: Graph plotted for throughput of local run

## 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

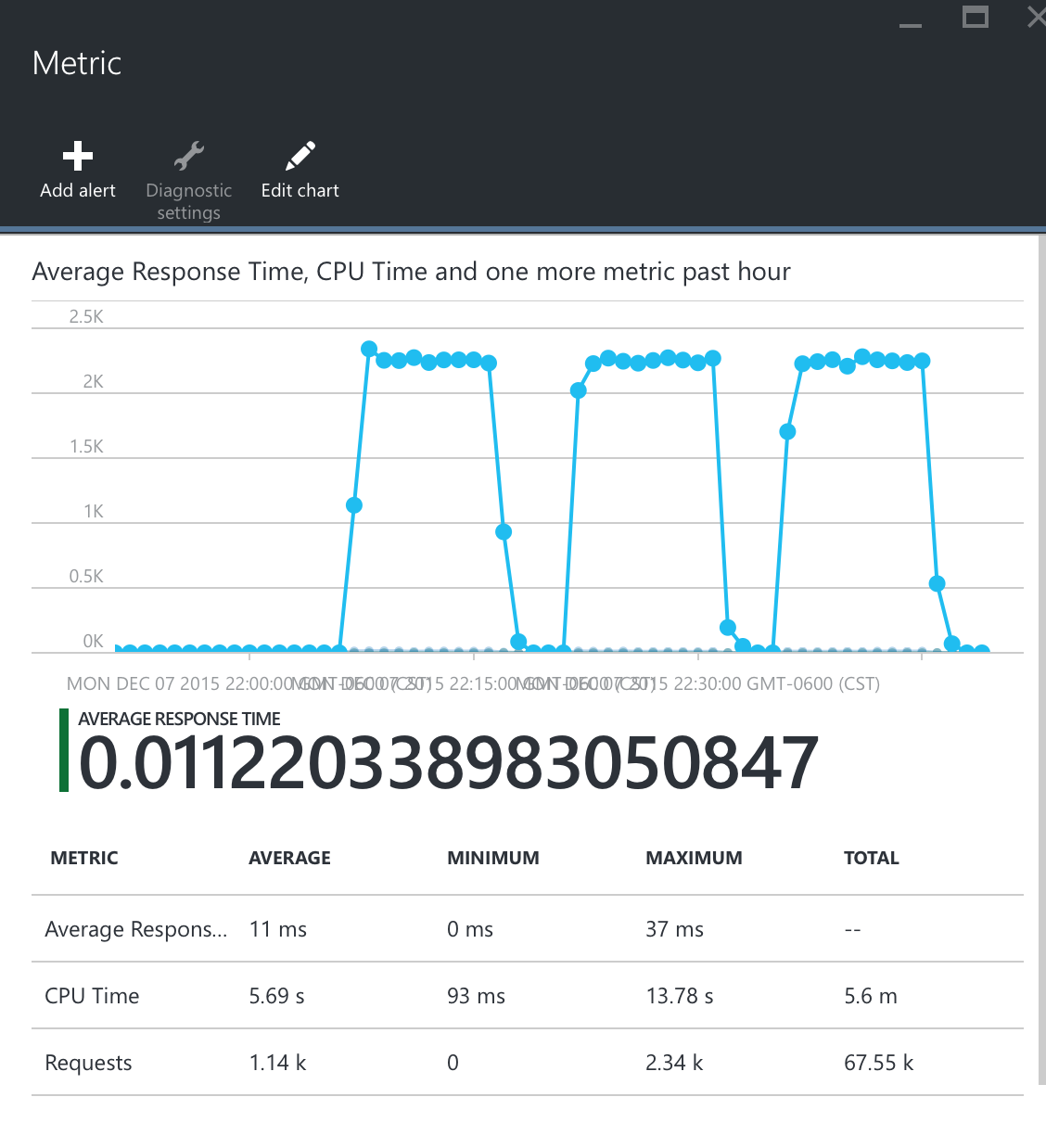


Fig: Metrics from MS Azure- Small Instance

**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

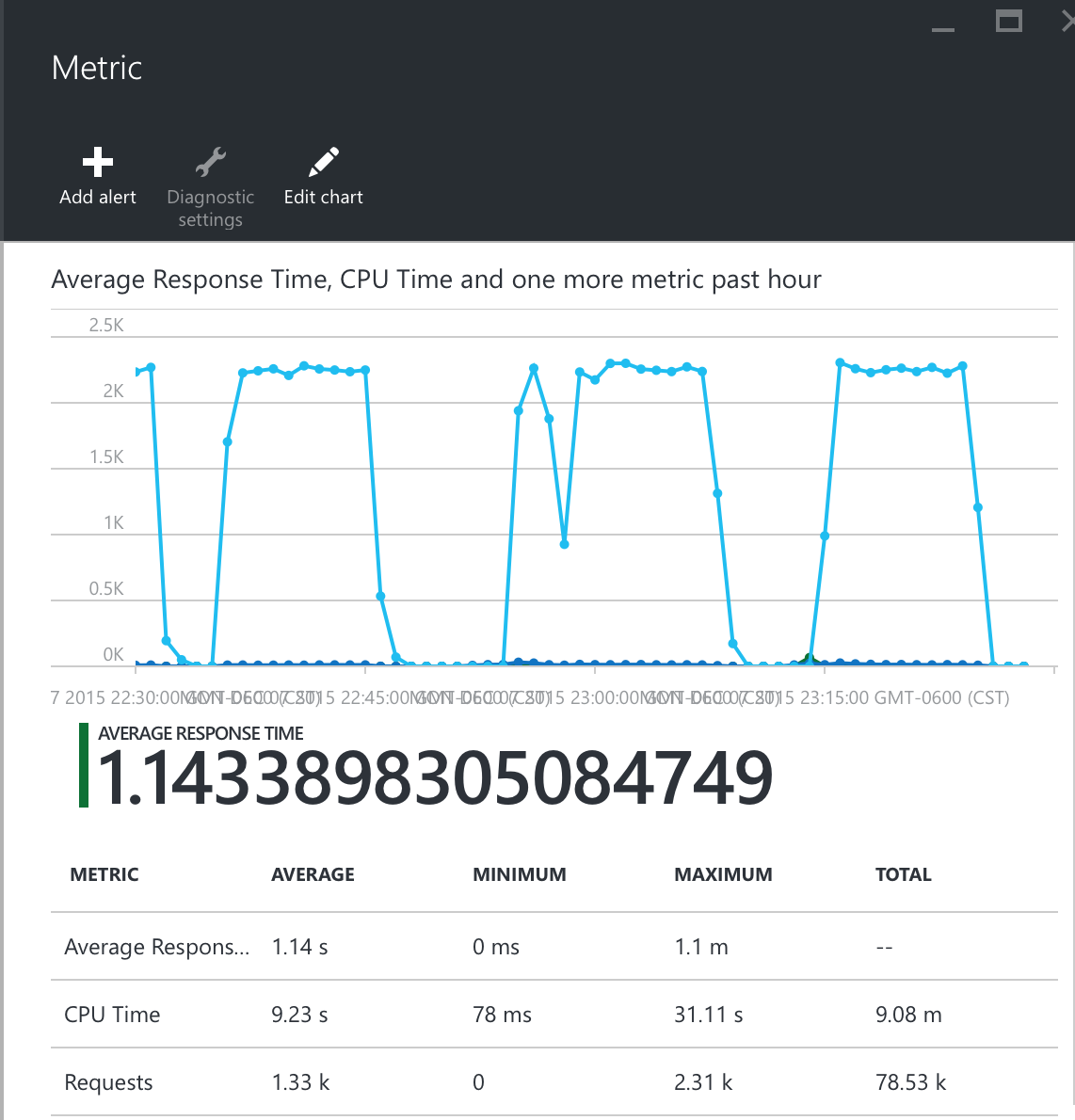


Fig: Metrics from MS Azure – Medium Scale

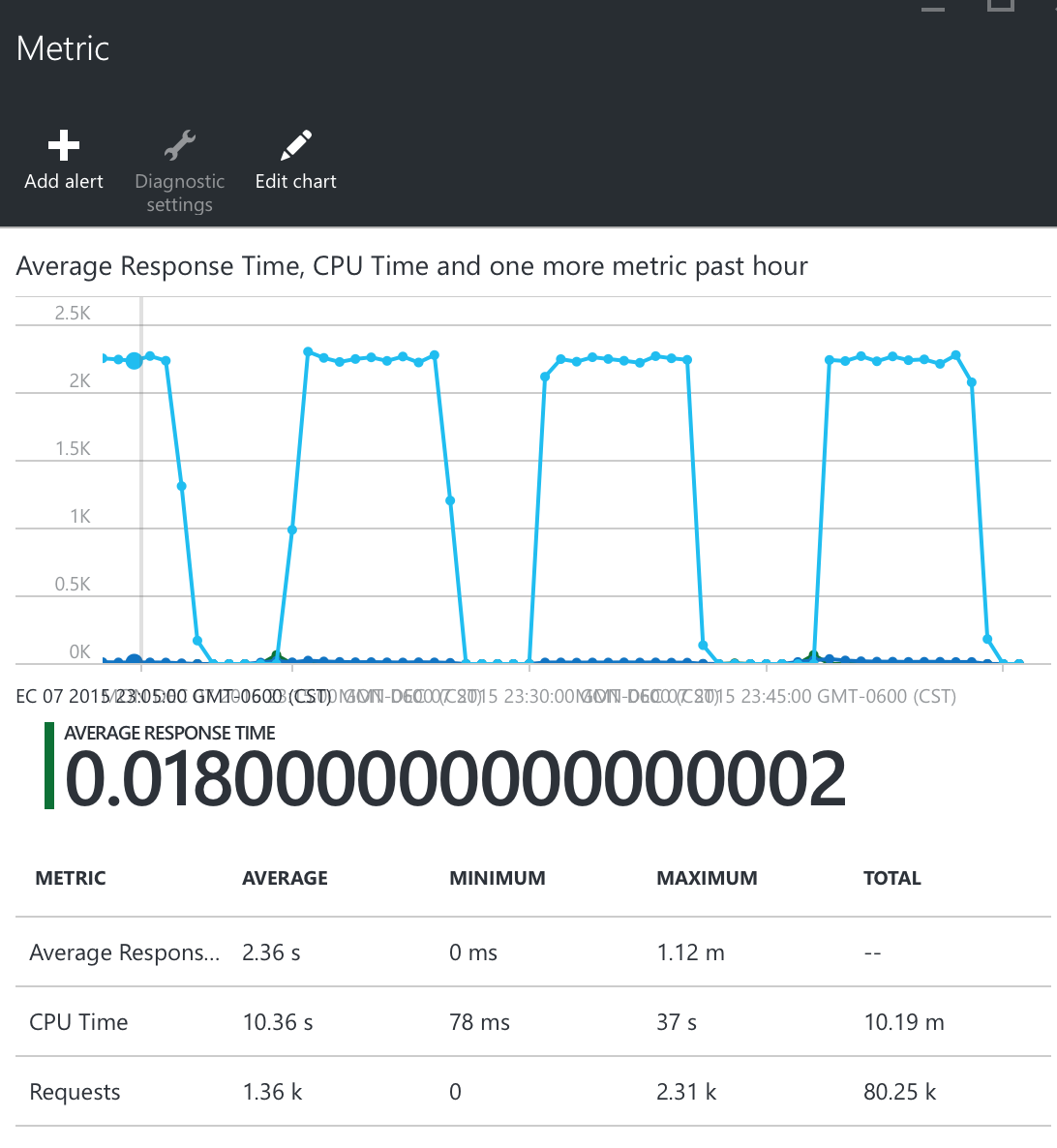


Fig: Metrics from MS Azure – Large Scale

## 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.

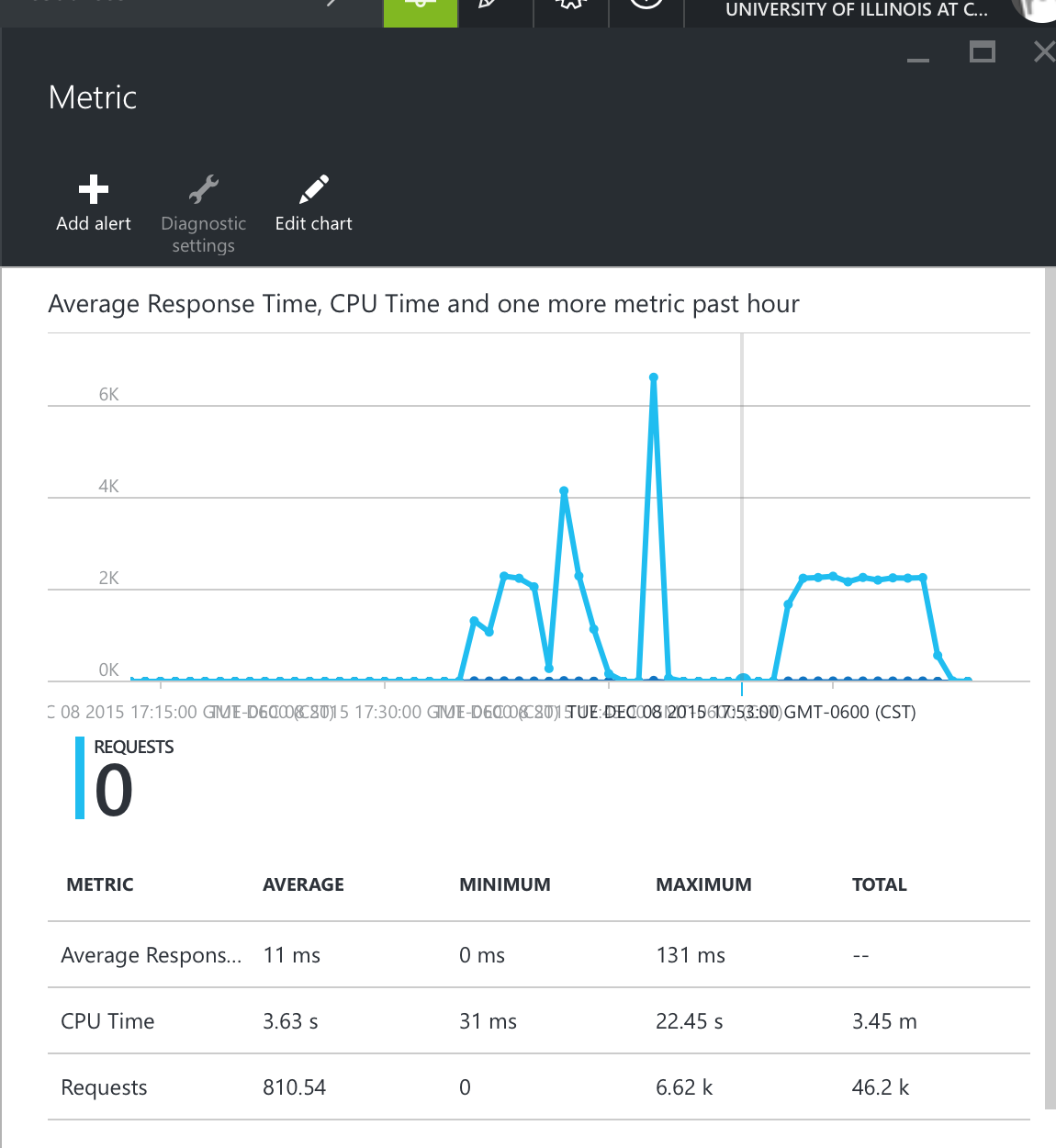


Fig: Metrics from MS Azure – Small Scale



Fig: Metrics from MS Azure – Medium Scale

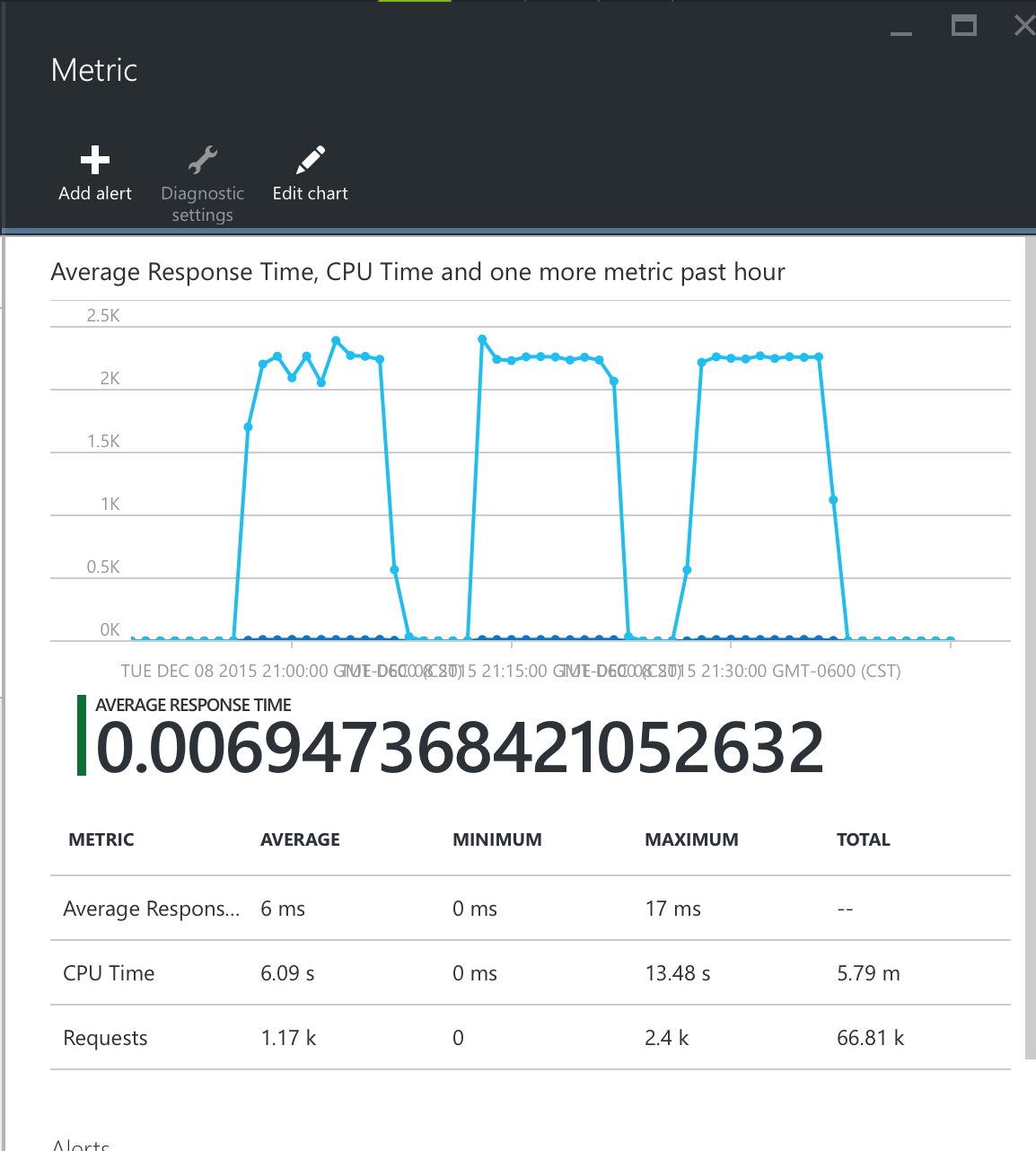


Fig: Metrics from MS Azure – Large Scale