Project Report

CS 441 – Distributed Objects for Cloud Computing

Provisioning Resources in Azure Cloud

Application Under Test: JPetstore

By:

Name: Krishna Madhur Philkhana

UIN: 651032229

Master’s in Computer Science

Table of Contents:

1. Introduction:

When applications are deployed in the cloud environment, the resources should be allocated based on the resource usage of the application. Hence the user will pay only for those resources which he/she will be using for hosting the application. So, it is very important for the user to performance test his/her web application thoroughly before deploying on the cloud platform.

Another important property of deploying applications on cloud platform is on demand network access to a shared pool of resources without much user intervention, in short auto-scaling. This can be achieved by writing provisioning rules in cloud.

The current project is to identify the potential provisioning rules that can be employed to reduce the performance bottlenecks of the AUT – Jpetstore. This will increase the performance of the application.

2. Application Under Test (AUT) - JPetStore – A Pet Store Application :

a. Description:

JPetStore is a typical e-commerce web application used by customers to buy pets. The application uses HSQL database to store the data and is developed using java based web technologies using maven build system.

b. Motivation for choosing JPetStore:

I have done a similar project during my training in NTT Data Americas Company. At that time, I did not deploy it on any web server apart from the localhost. So, deployment of this selected AUT effectively on a cloud platform will complete the part that I left during my training. Also, it brings me one step closer towards becoming a full stack programmer.

3. Instructions to Build the AUT :

a. Download the source code ZIP file from this link: https://github.com/mybatis/jpetstore-6
b. AUT uses Eclipse IDE for the build. Just import the zip file that is downloaded from the above link into the Eclipse IDE.

4. Deploy AUT in Microsoft Azure :

a. Deploy it as a Cloud Service:
   Azure plug-in should be installed in the eclipse using the resource:
   WindowsAzurePlugin4EJ - http://dl.msopentech.com/eclipse

   Once the plug-in is installed, right click on the JPetStore-6 folder in the projects section of the workspace. In the Options, please select ‘Azure’ -> ‘Select New

Azure'Publish to Azure'. After providing your subscription details, and selecting the configuration for the deployment, the application will be packaged and deployed in the cloud. The deployment of web application in the cloud is done as cloud service.

b. Which method I chose? Why?
I chose to deploy my web application as cloud service. My application under test (AUT) is a light weight application. The database used in the application is HSQL DB, a database for Java Eco System. So my application requires very low customization. Hence, it would not be ideal to deploy it using the IaaS service.

The application is not deployed as a web application because it gives minimal control over the application's software stack. Hence I found Azure's Cloud Service to be ideal deployment method. Also, this type of deployment is very easy and hence the concentration will be on improving performance rather than deployment of the application.

5. Objective and Methodology :

The objective is to arrive at a set of provision rules for the AUT- JPetStore, where the user can deploy his/her application in the cloud infrastructure optimally. The AUT's performance- availability, consistency and throughput should be increased after we apply the provision rules.

To perform the performance testing, we use JMeter to simulate users to create the load. We measure the amount of CPU usage from the cloud and throughput, response time and error rate for the requested input transactions from the JMeter results. We then formulate the provision rules to increase the throughput, decrease the error rate and minimize the response time.

6. Project Requirements:

a. It requires a computer with a i5 or i7 processor running windows (7,8,10) or Ubuntu.
b. It requires the installation of the load and stress testing tool called JMeter from jmeter.apache.org.
c. The cloud environment in our case, it is Windows Azure.

7. JMeterScripts :

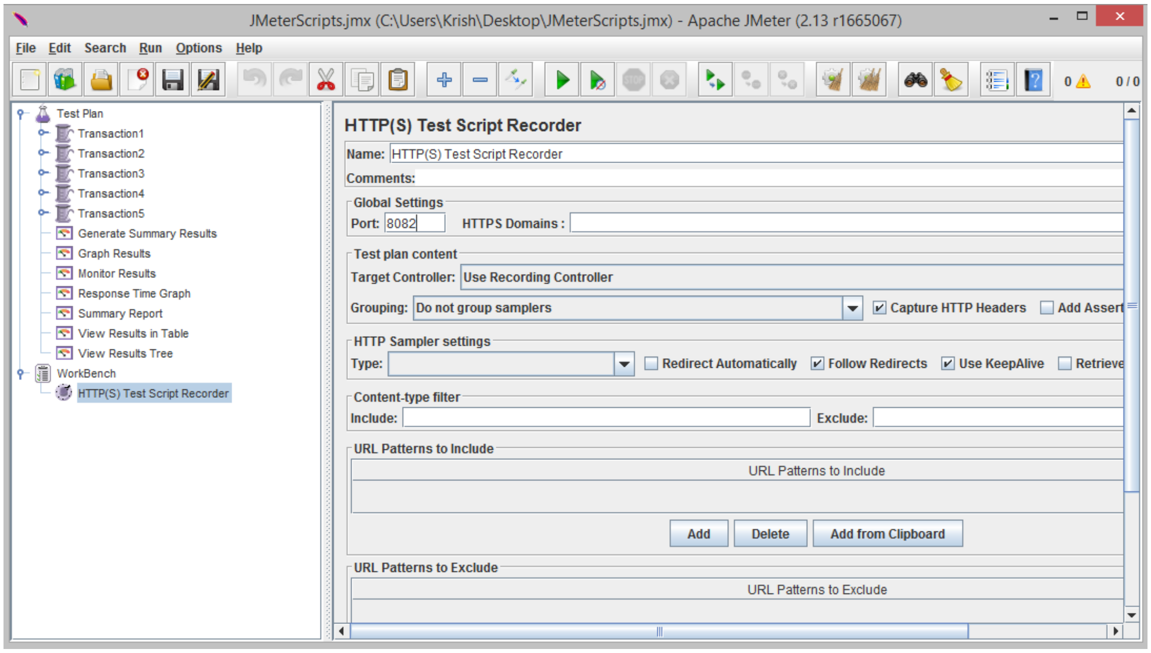
a. Building scripts :
The JMeter scripts are very useful when we want to stress test an application. We need to provide the number of threads (users) that we want to hit the particular transaction with. A transaction here would be a set of URLs of the Application Under Test (AUT).

We also add the Listeners to the Thread Groups that we create. These listeners capture all the important data that we would like to analyze. Summary Report gives us the details of the throughput values, error percentage, kb/sec and other metrics.

We will make use of the metrics that we get in the JMeter and the metrics we get in the Azure cloud to design the provisioning rules.

b. Recording Scripts :

I. Create a Non Test Element in the WorkBench- HTTP(s) Script Recorder. We need to start this recorder to start recording our scripts. Give the port number as 8082(Any unused port number can be given).



HTTP(S) Test Script Recorder

Name: HTTP(S) Test Script Recorder

Comments:

Global Settings

Port: 8082 HTTPS Domains :

Test plan content

Target Controller: Use Recording Controller

Grouping: Do not group samplers ☑ Capture HTTP Headers □ Add Assert

HTTP Sampler settings

Type: □ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive □ Retrieve

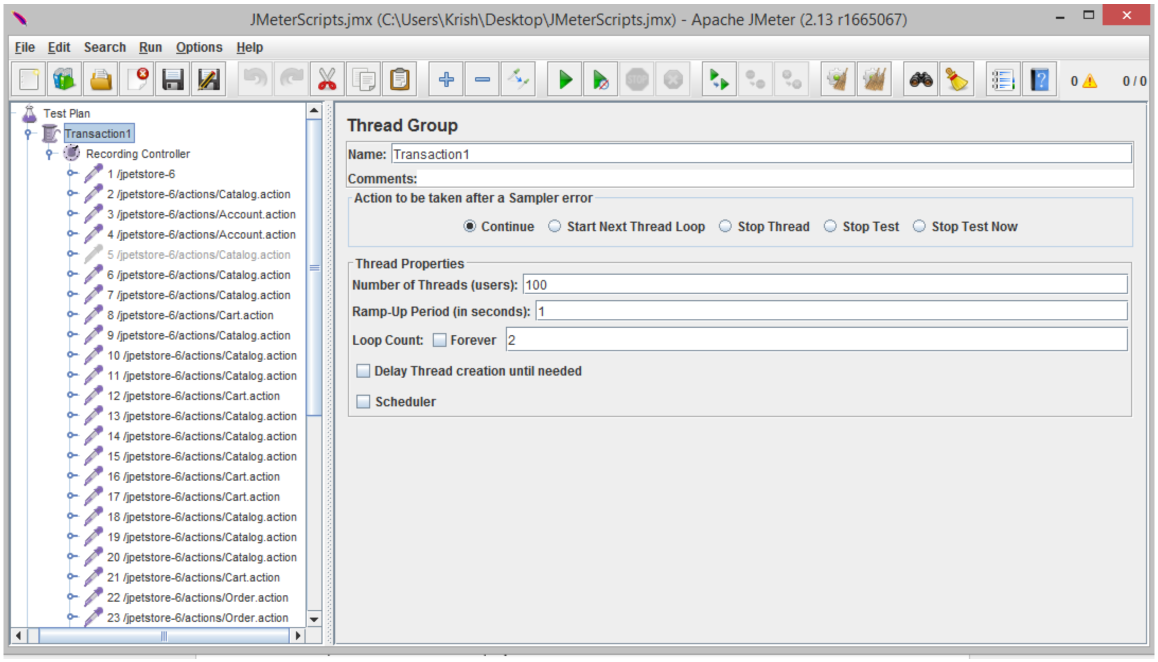
Content-type filter

Include: Exclude:

URL Patterns to Include

URL Patterns to Exclude

II. Create a Logic Controller->Recording Controller in the Test Group. The transactions done in the web browser will be stored in the controller here.



JMeterScripts.jmx (C:\Users\Krish\Desktop\JMeterScripts.jmx) - Apache JMeter (2.13 r1665067)

- Test Plan
  - Transaction1
    - Recording Controller
      - 1 /jpetstore-6
      - 2 /jpetstore-6/actions/Catalog.action
      - 3 /jpetstore-6/actions/Account.action
      - 4 /jpetstore-6/actions/Account.action
      - 5 /jpetstore-6/actions/Catalog.action
      - 6 /jpetstore-6/actions/Catalog.action
      - 7 /jpetstore-6/actions/Catalog.action
      - 8 /jpetstore-6/actions/Cart.action
      - 9 /jpetstore-6/actions/Catalog.action
      - 10 /jpetstore-6/actions/Catalog.action
      - 11 /jpetstore-6/actions/Catalog.action
      - 12 /jpetstore-6/actions/Cart.action
      - 13 /jpetstore-6/actions/Catalog.action
      - 14 /jpetstore-6/actions/Catalog.action
      - 15 /jpetstore-6/actions/Catalog.action
      - 16 /jpetstore-6/actions/Cart.action
      - 17 /jpetstore-6/actions/Cart.action
      - 18 /jpetstore-6/actions/Catalog.action
      - 19 /jpetstore-6/actions/Catalog.action
      - 20 /jpetstore-6/actions/Catalog.action
      - 21 /jpetstore-6/actions/Cart.action
      - 22 /jpetstore-6/actions/Order.action
      - 23 /jpetstore-6/actions/Order.action

Thread Group

Name: Transaction1

Comments:

Action to be taken after a Sampler error

● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

Number of Threads (users): 100

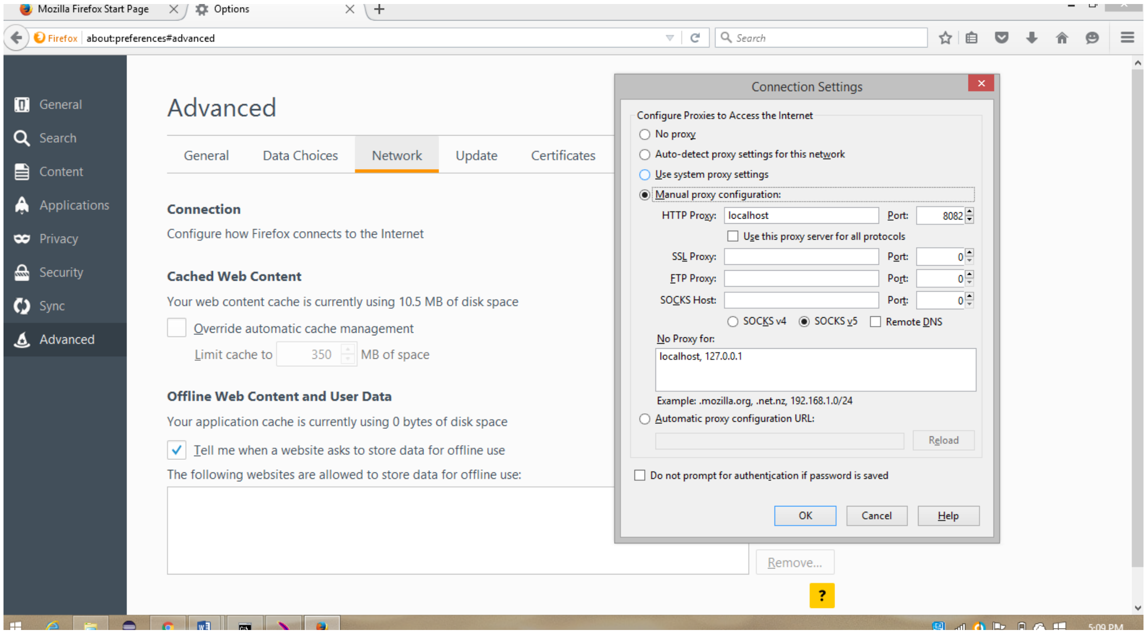
Ramp-Up Period (in seconds): 1

Loop Count: □ Forever 2

□ Delay Thread creation until needed

□ Scheduler

III. In MozillaFirefox Browser, go to Options-> Advanced-> Networks-> Settings-> Connection Settings
HTTP Proxy: localhost
Port: 8082 (This must match with the port number given in the HTTP(s) Script Recorder in step 1.



Mozilla Firefox Start Page
Options
Firefox about:preferences#advanced
Search
General
Search
Content
Applications
Privacy
Security
Sync
Advanced

Advanced

General Data Choices Network Update Certificates

Connection

Configure how Firefox connects to the Internet

Cached Web Content

Your web content cache is currently using 10.5 MB of disk space

□ Override automatic cache management

Limit cache to 350 MB of space

Offline Web Content and User Data

Your application cache is currently using 0 bytes of disk space

☑ Tell me when a website asks to store data for offline use

The following websites are allowed to store data for offline use:

Connection Settings

Configure Proxies to Access the Internet

○ No proxy

○ Auto-detect proxy settings for this network

○ Use system proxy settings

● Manual proxy configuration:

HTTP Proxy: localhost Port: 8082

□ Use this proxy server for all protocols

SSL Proxy: Port: 0

FTP Proxy: Port: 0

SOCKS Host: Port: 0

○ SOCKS v4 ● SOCKS v5 □ Remote DNS

No Proxy for:

localhost, 127.0.0.1

Example: .mozilla.org, .net.nz, 192.168.1.0/24

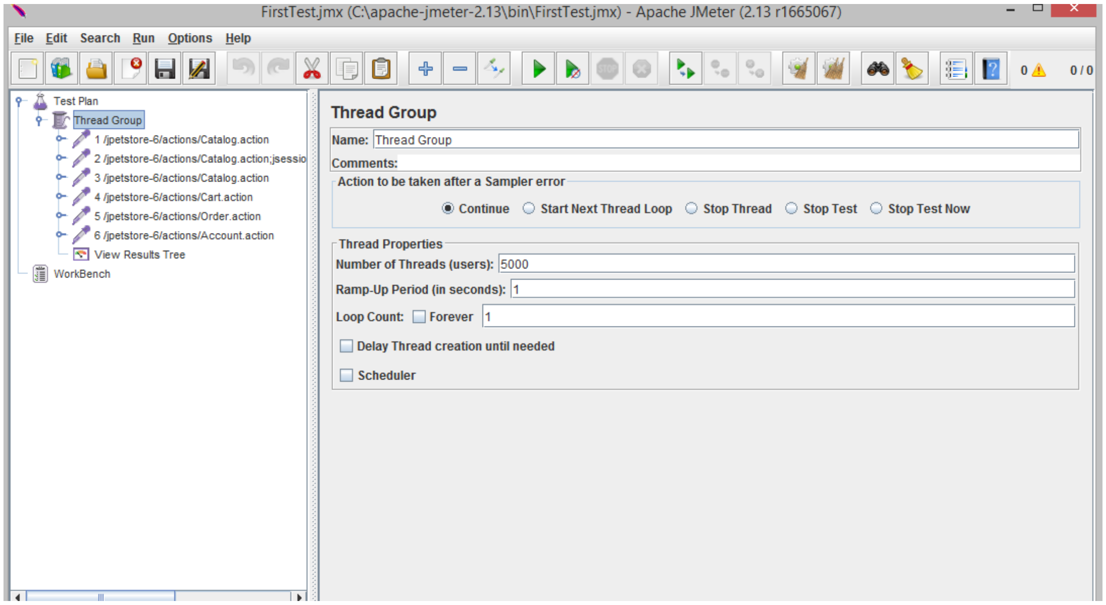
○ Automatic proxy configuration URL:

□ Do not prompt for authentication if password is saved

8. Experiments:

Experiment 1: AUT deployed on the local host

The AUT is deployed on the local host. The JMeter script is run on the local host with one transaction with 5000 user threads. LocalHost is an i3 processor with 500GB permanent storage and 4GB RAM Storage.


FirstTest.jmx (C:\apache-jmeter-2.13\bin\FirstTest.jmx) - Apache JMeter (2.13 r1665067)
File Edit Search Run Options Help
0 ⚠ 0 / 0
Test Plan
Thread Group
1 /jpetstore-6/actions/Catalog.action
2 /jpetstore-6/actions/Catalog.action;jsessio
3 /jpetstore-6/actions/Catalog.action
4 /jpetstore-6/actions/Cart.action
5 /jpetstore-6/actions/Order.action
6 /jpetstore-6/actions/Account.action
View Results Tree
WorkBench

Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

Number of Threads (users): 5000

Ramp-Up Period (in seconds): 1

Loop Count: □ Forever 1

□ Delay Thread creation until needed

□ Scheduler

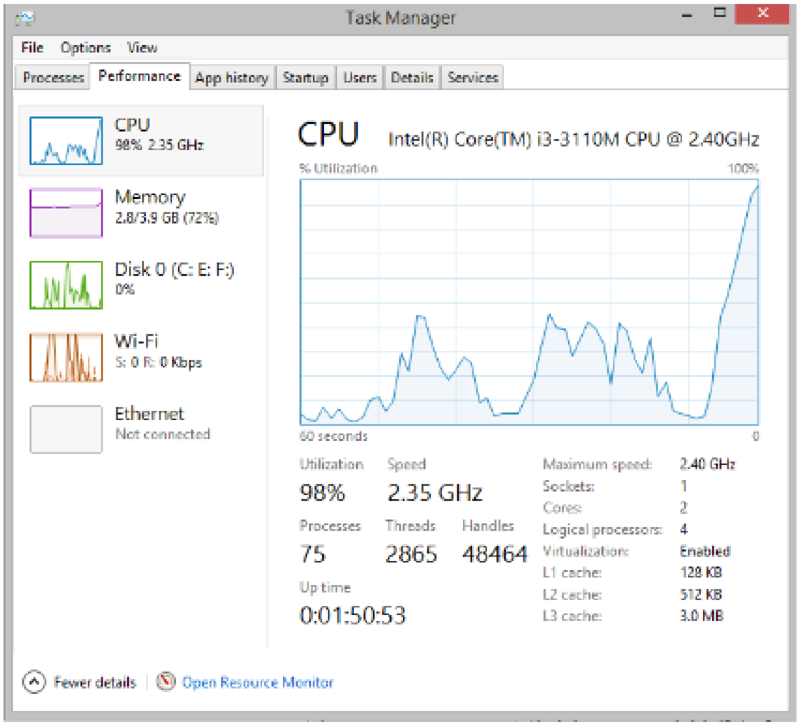
Results and Observations:

The CPU percentage crossed 95% immediately after I started my test. The hardware that my system has, is not at all sufficient to handle so many threads. Also, being a localhost, I do not have the option of scaling up my resources.

So, if this application is hosted on a similar machine like mine, then the application would lose the availability and consistency properties that are needed for the web applications.

Choosing Cloud Storage for such e-commerce applications is very advisable because the traffic for such sites is very unpredictable. Also, paying huge money for the hardware when the application is not using is also not advisable.

It will be ideal if such e-commerce web applications are deployed in a cloud.



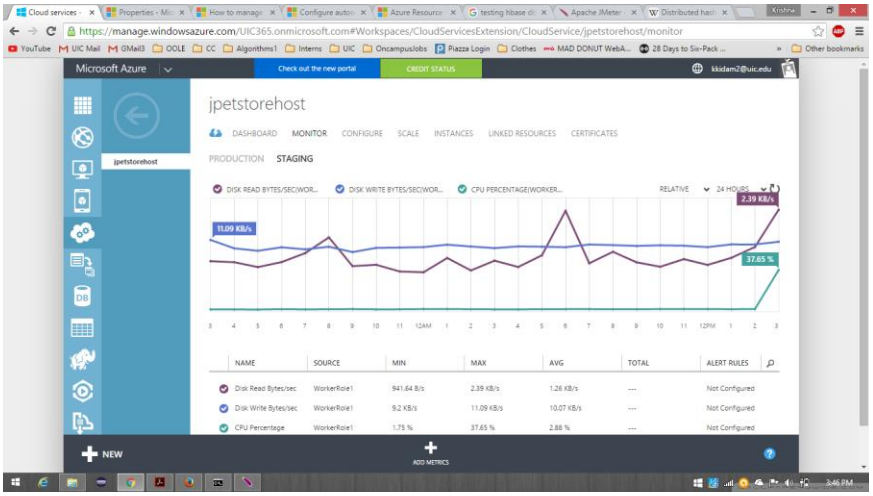
Utilization	Speed		Maximum speed:	2.40 GHz
98%	2.35 GHz		Sockets:	1
			Cores:	2
Processes	Threads	Handles	Logical processors:	4
75	2865	48464	Virtualization:	Enabled
			L1 cache:	128 KB
Up time			L2 cache:	512 KB
0:01:50:53			L3 cache:	3.0 MB

Experiment 2: On AUT deployed as Cloud Service -1

The AUT is deployed on the cloud. The link to the web application is: fb13b39934f64ddb944a8c3d7f198165.cloudapp.net/jpetstore-6/

The JMeter script is run on the cloud host with one transaction with 5000 user threads. Script: FirstTest.jmx

Initial Metrics:



NAME	SOURCE	MIN	MAX	AVG	TOTAL	ALERT RULES
Disk Read Bytes/sec	WorkerRole1	941.64 B/s	2.39 KB/s	1.26 KB/s	---	Not Configured
Disk Write Bytes/sec	WorkerRole1	9.2 KB/s	11.09 KB/s	10.07 KB/s	---	Not Configured
CPU Percentage	WorkerRole1	1.75 %	37.65 %	2.88 %	---	Not Configured

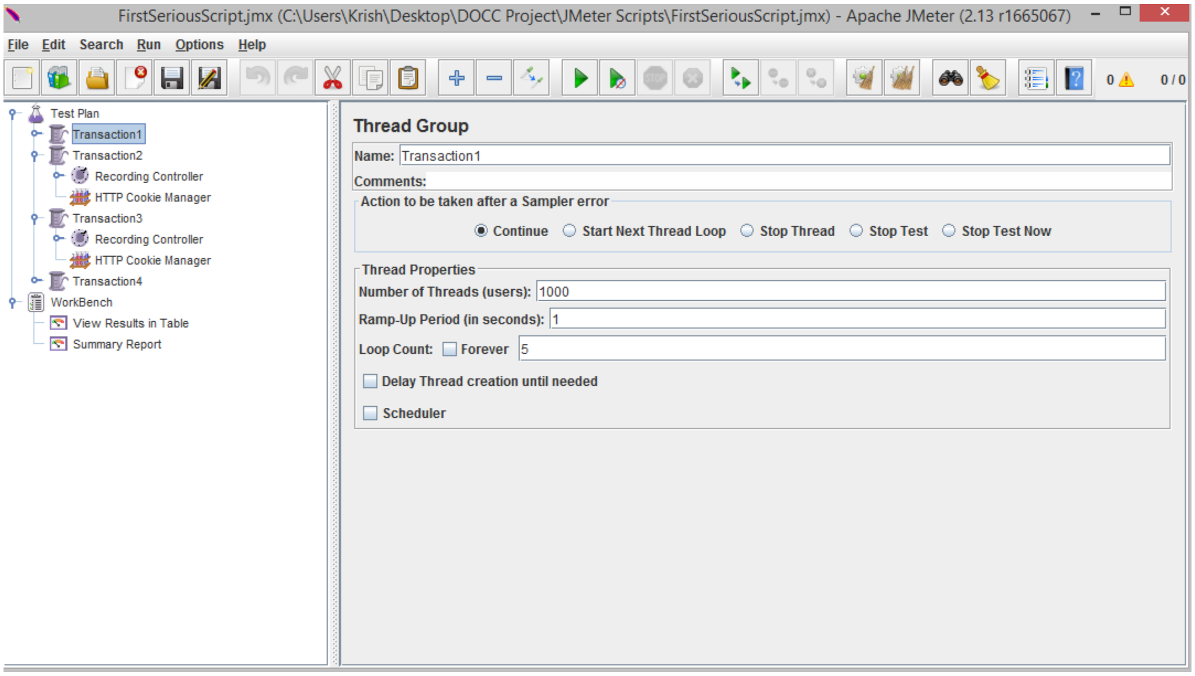
There were no errors in the Summary Report of the JMeter also.

Results and Observations:

This states that there is no problem to the consistency and availability properties of the Application when put under such a load. So in the next experiment, I increased the number of transactions.
Summary Report: JMeterReports\FirstTestReport.csv

Experiment 3: On AUT deployed as Cloud Service-2

I used a new JMeter script with increased number of transactions and the loop counts for each of these transactions. Script used: FirstSeriousScript



FirstSeriousScript.jmx (C:\Users\Krish\Desktop\DOCC Project\JMeter Scripts\FirstSeriousScript.jmx) - Apache JMeter (2.13 r1665067)

Thread Group

Name: Transaction1

Comments:

Action to be taken after a Sampler error

● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

Number of Threads (users): 1000

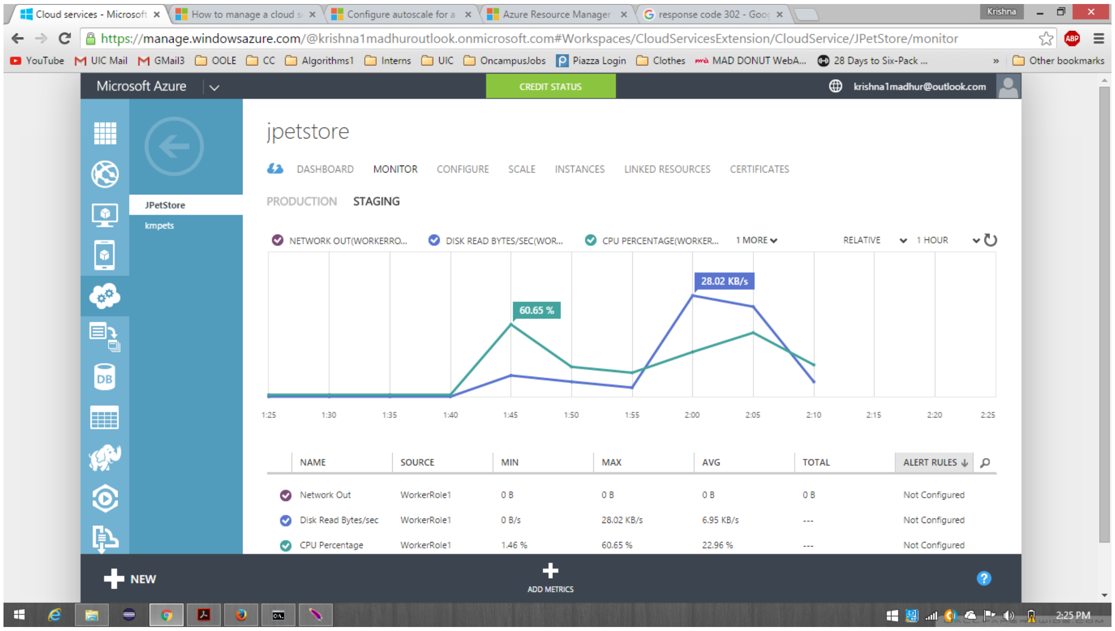
Ramp-Up Period (in seconds): 1

Loop Count: □ Forever 5

□ Delay Thread creation until needed

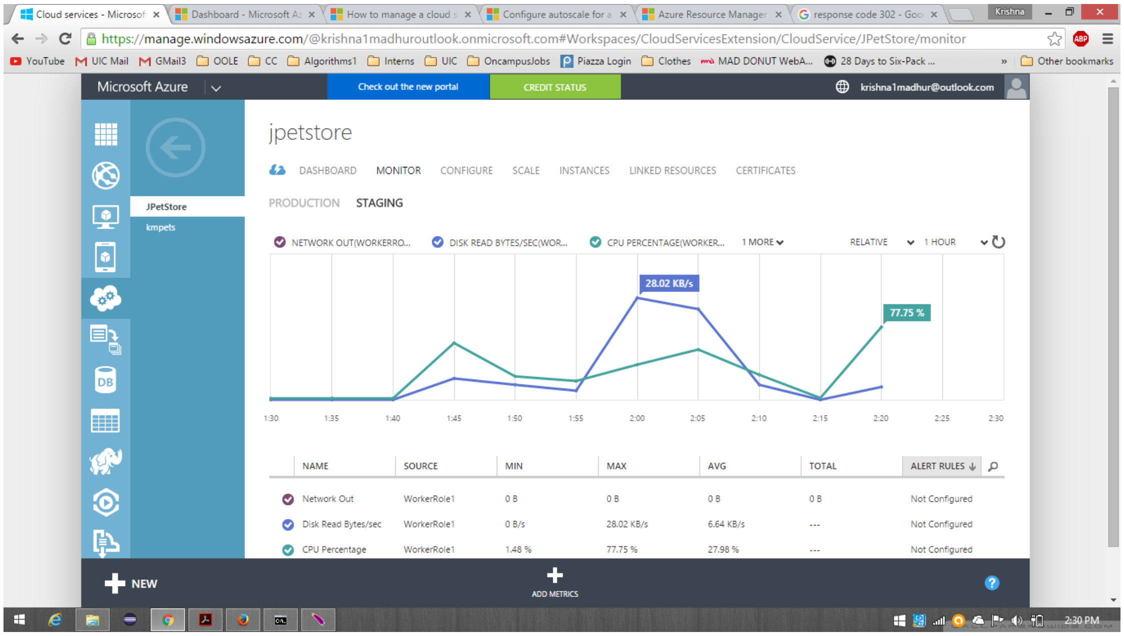
□ Scheduler

Results and Observations:



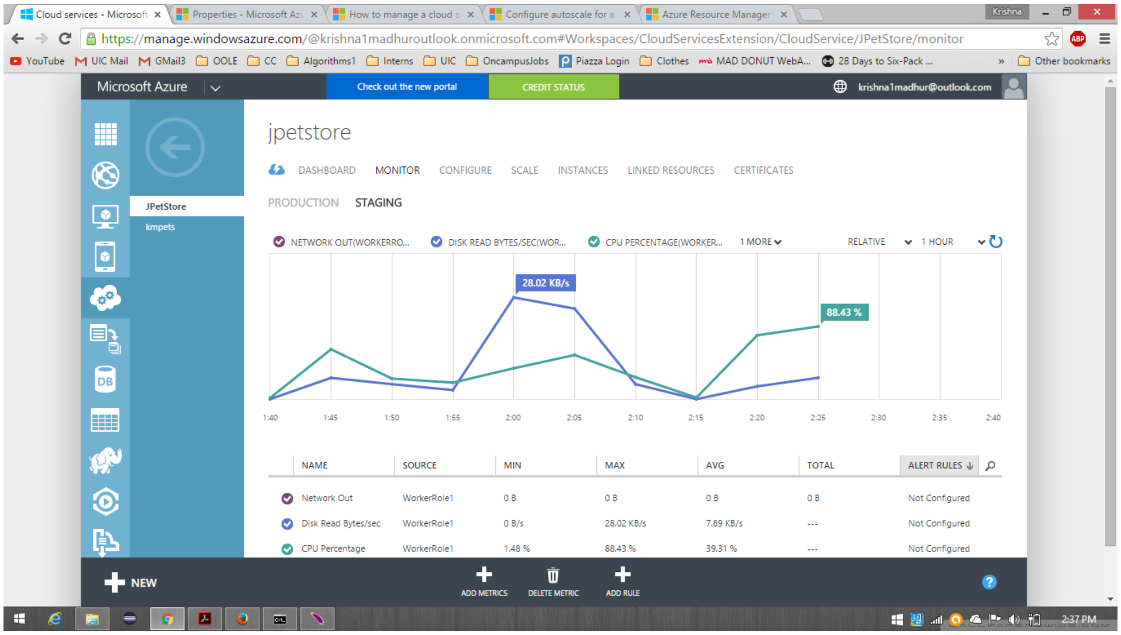
NAME	SOURCE	MIN	MAX	AVG	TOTAL	ALERT RULES
Network Out	WorkerRole1	0 B	0 B	0 B	0 B	Not Configured
Disk Read Bytes/sec	WorkerRole1	0 B/s	28.02 KB/s	6.95 KB/s	---	Not Configured
CPU Percentage	WorkerRole1	1.46 %	60.65 %	22.96 %	---	Not Configured

Fig: 2:10PM



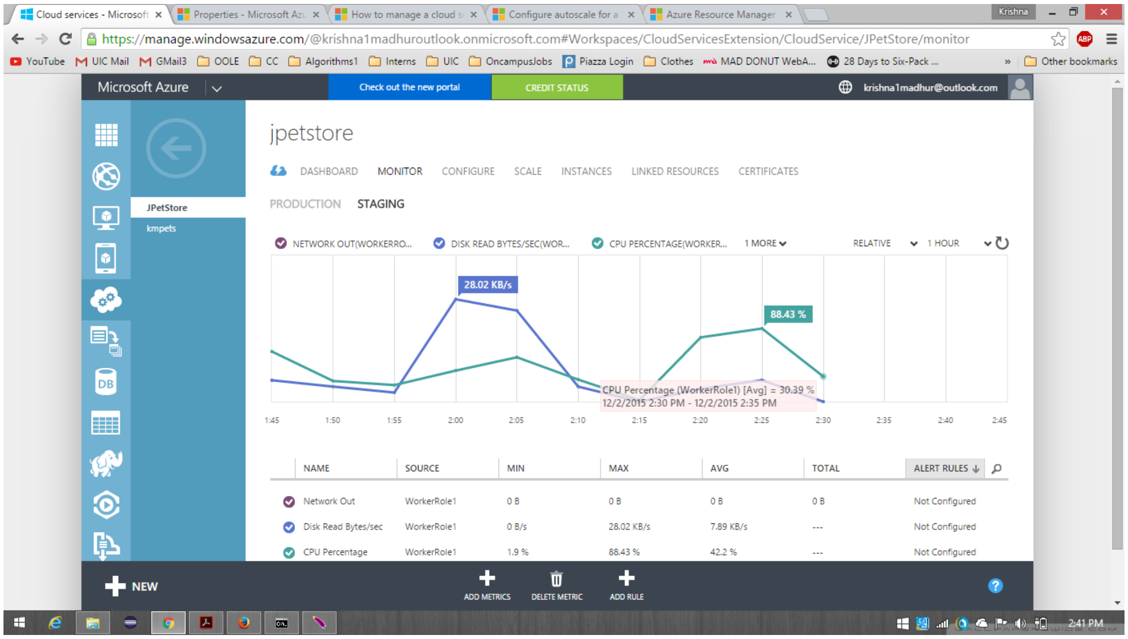
NAME	SOURCE	MIN	MAX	AVG	TOTAL	ALERT RULES
Network Out	WorkerRole1	0 B	0 B	0 B	0 B	Not Configured
Disk Read Bytes/sec	WorkerRole1	0 B/s	28.02 KB/s	6.64 KB/s	---	Not Configured
CPU Percentage	WorkerRole1	1.48 %	77.75 %	27.98 %	---	Not Configured

Fig: 2:20PM



NAME	SOURCE	MIN	MAX	AVG	TOTAL	ALERT RULES
Network Out	WorkerRole1	0 B	0 B	0 B	0 B	Not Configured
Disk Read Bytes/sec	WorkerRole1	0 B/s	28.02 KB/s	7.89 KB/s	---	Not Configured
CPU Percentage	WorkerRole1	1.48 %	88.43 %	39.31 %	---	Not Configured

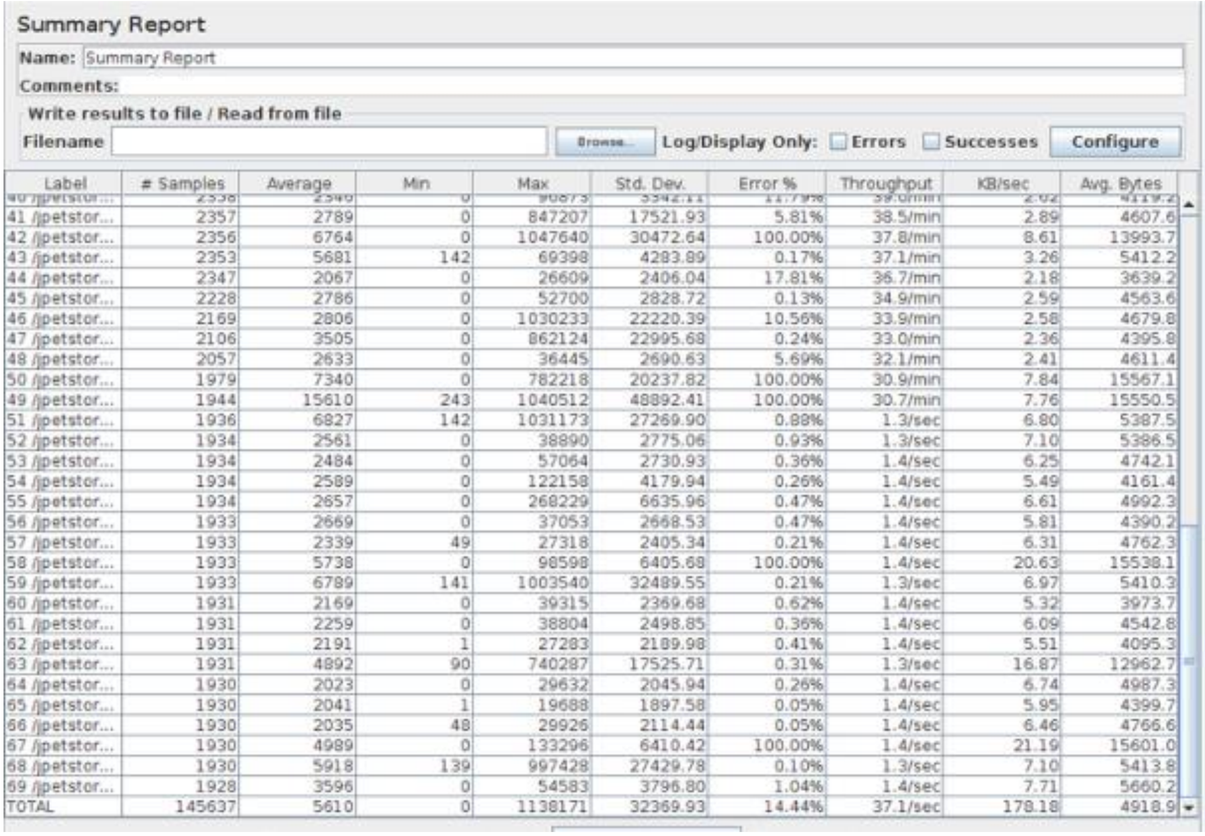
Fig: 2:25PM



NAME	SOURCE	MIN	MAX	AVG	TOTAL	ALERT RULES
Network Out	WorkerRole1	0 B	0 B	0 B	0 B	Not Configured
Disk Read Bytes/sec	WorkerRole1	0 B/s	28.02 KB/s	7.89 KB/s	---	Not Configured
CPU Percentage	WorkerRole1	1.9 %	88.43 %	42.2 %	---	Not Configured

If you observe from the screens above, the CPU percentage is increasing gradually. In my JMeter Summary Reports, I observed that many URLs of my transactions are encountering huge errors after the CPU percentage has gone past 45%. Also, throughput of my application decreased and it is **30.7 per minute**. The response time has gone to **~160000ms**. So the consistency and the availability properties of the web application have been down. This is a loss to the Web Application Provider.

The CPU percentage has gone past 88%. Then my JMeter crashed. And rightly so, my web application's CPU percentage has come down.



Summary Report

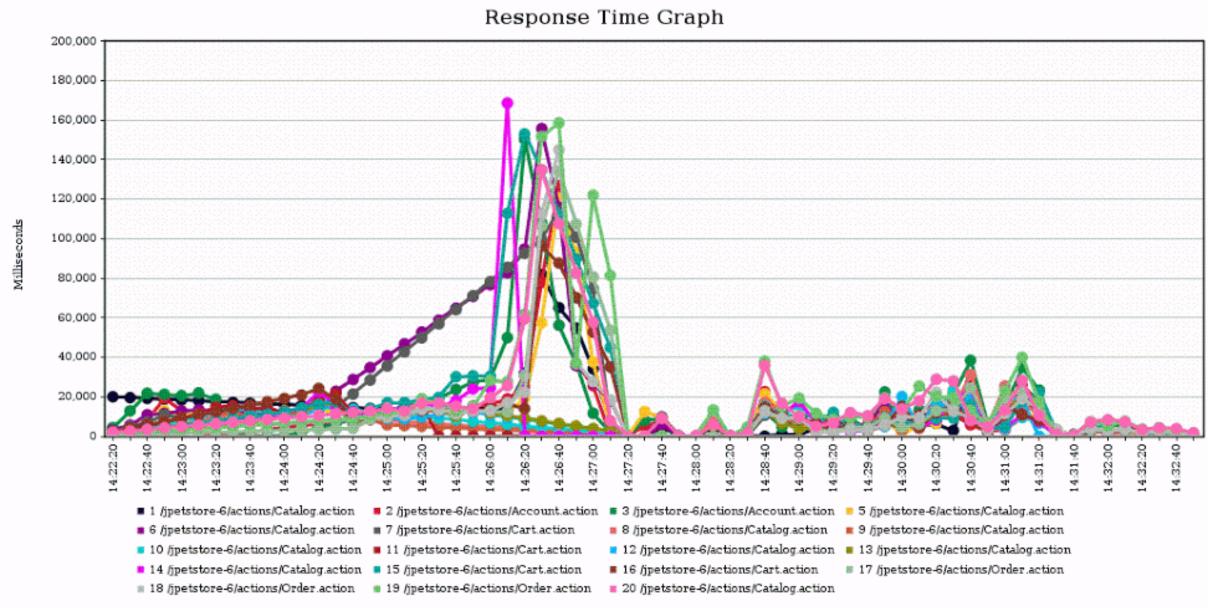
Name: Summary Report

Comments:

Write results to file / Read from file

Filename:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
40 /jpetstor...	2338	2340	0	90873	3342.11	11.79%	39.0/min	2.02	4119.2
41 /jpetstor...	2357	2789	0	847207	17521.93	5.81%	38.5/min	2.89	4607.6
42 /jpetstor...	2356	6764	0	1047640	30472.64	100.00%	37.8/min	8.61	13993.7
43 /jpetstor...	2353	5681	142	69398	4283.89	0.17%	37.1/min	3.26	5412.2
44 /jpetstor...	2347	2067	0	26609	2406.04	17.81%	36.7/min	2.18	3639.2
45 /jpetstor...	2228	2786	0	52700	2828.72	0.13%	34.9/min	2.59	4563.6
46 /jpetstor...	2169	2806	0	1030233	22220.39	10.56%	33.9/min	2.58	4679.8
47 /jpetstor...	2106	3505	0	862124	22995.68	0.24%	33.0/min	2.36	4395.8
48 /jpetstor...	2057	2633	0	36445	2690.63	5.69%	32.1/min	2.41	4611.4
50 /jpetstor...	1979	7340	0	782218	20237.82	100.00%	30.9/min	7.84	15567.1
49 /jpetstor...	1944	15610	243	1040512	48892.41	100.00%	30.7/min	7.76	15550.5
51 /jpetstor...	1936	6827	142	1031173	27269.90	0.88%	1.3/sec	6.80	5387.5
52 /jpetstor...	1934	2561	0	38890	2775.06	0.93%	1.3/sec	7.10	5386.5
53 /jpetstor...	1934	2484	0	57064	2730.93	0.36%	1.4/sec	6.25	4742.1
54 /jpetstor...	1934	2589	0	122158	4179.94	0.26%	1.4/sec	5.49	4161.4
55 /jpetstor...	1934	2657	0	268229	6635.96	0.47%	1.4/sec	6.61	4992.3
56 /jpetstor...	1933	2669	0	37053	2668.53	0.47%	1.4/sec	5.81	4390.2
57 /jpetstor...	1933	2339	49	27318	2405.34	0.21%	1.4/sec	6.31	4762.3
58 /jpetstor...	1933	5738	0	98598	6405.68	100.00%	1.4/sec	20.63	15538.1
59 /jpetstor...	1933	6789	141	1003540	32489.55	0.21%	1.3/sec	6.97	5410.3
60 /jpetstor...	1931	2169	0	39315	2369.68	0.62%	1.4/sec	5.32	3973.7
61 /jpetstor...	1931	2259	0	38804	2498.85	0.36%	1.4/sec	6.09	4542.8
62 /jpetstor...	1931	2191	1	27283	2189.98	0.41%	1.4/sec	5.51	4095.3
63 /jpetstor...	1931	4892	90	740287	17525.71	0.31%	1.3/sec	16.87	12962.7
64 /jpetstor...	1930	2023	0	29632	2045.94	0.26%	1.4/sec	6.74	4987.3
65 /jpetstor...	1930	2041	1	19688	1897.58	0.05%	1.4/sec	5.95	4399.7
66 /jpetstor...	1930	2035	48	29926	2114.44	0.05%	1.4/sec	6.46	4766.6
67 /jpetstor...	1930	4989	0	133296	6410.42	100.00%	1.4/sec	21.19	15601.0
68 /jpetstor...	1930	5918	139	997428	27429.78	0.10%	1.3/sec	7.10	5413.8
69 /jpetstor...	1928	3596	0	54583	3796.80	1.04%	1.4/sec	7.71	5660.2
TOTAL	145637	5610	0	1138171	32369.93	14.44%	37.1/sec	178.18	4918.9



Problem: Unfortunately, I had to kill the JMeter process as my system hung down after giving so many threads. So could not generate the Summary Report file for that particular test.

Fix: I started my next JMeter test by borrowing one of my colleague's laptop system which has an Intel i7 processor. The JMeter performance was better in the new system.

PROVISIONING RULE: R: **Increase the CPU instances to 2, when the CPU percentage crosses over 45%.**

sinst(a,i) when CPU percentage crosses 45%

Here, a is the 'WorkerRole1' and i is '1'.

Experiment 4: On AUT deployed as Cloud Service-3

This time I increased the number of transactions on the AUT in such a way that:

Transaction1- Only one insert operation performed on checkout - 27URLs.

Transaction2- Insert after every item purchase. This will increase the database interactions of the transactions which would further result in the increase of CPU processing time - 47URLs.

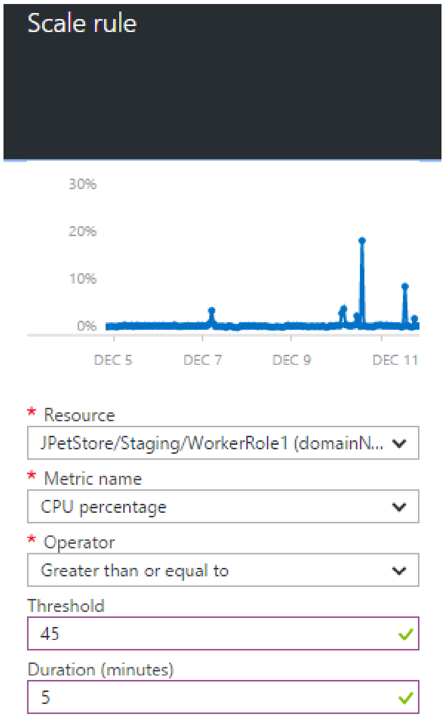
Transaction3- Search for an object and then check out. Only one insert but multiple search operations on the database - 43URLs.

Transaction4- Search and checkout simultaneously. Multiple search and insert operations – 32 URLs.

Transaction5- Update operations on the database - 36URLs.

Script: JMeterScripts\JMeterScripts.jmx

We will apply the provisioning rule that is mentioned above and then continue the experiment.



Scale rule

* Resource

JPetStore/Staging/WorkerRole1 (domainN...

* Metric name

CPU percentage

* Operator

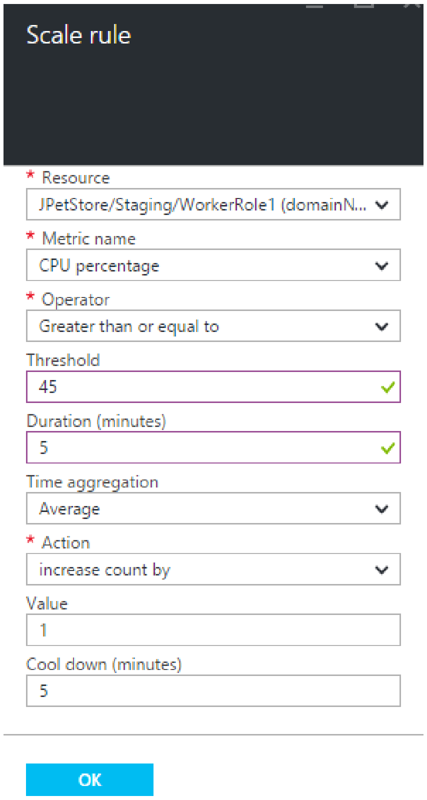
Greater than or equal to

Threshold

45

Duration (minutes)

5



Scale rule

* Resource

JPetStore/Staging/WorkerRole1 (domainN...

* Metric name

CPU percentage

* Operator

Greater than or equal to

Threshold

45

Duration (minutes)

5

Time aggregation

Average

* Action

increase count by

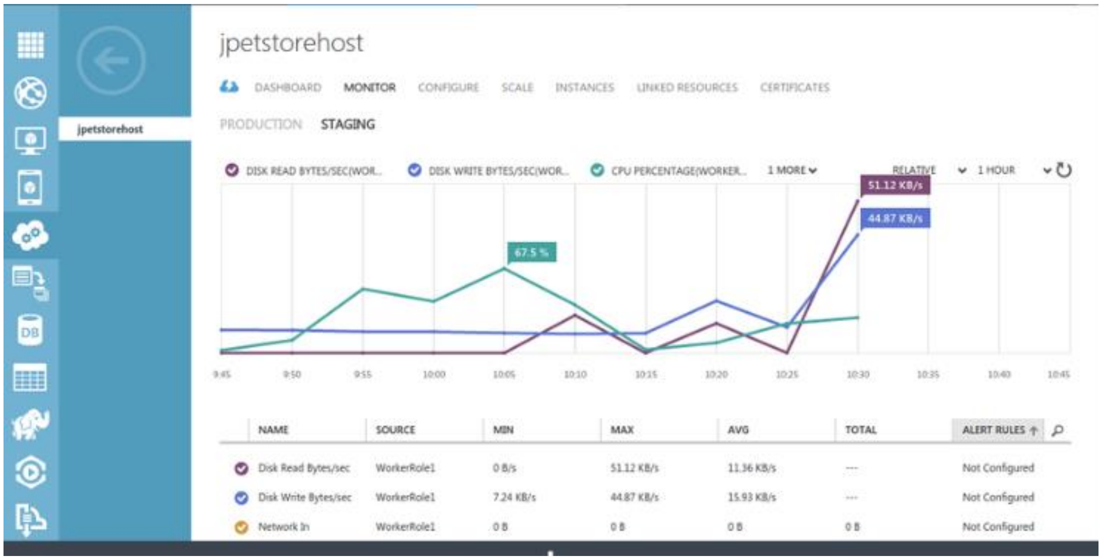
Value

1

Cool down (minutes)

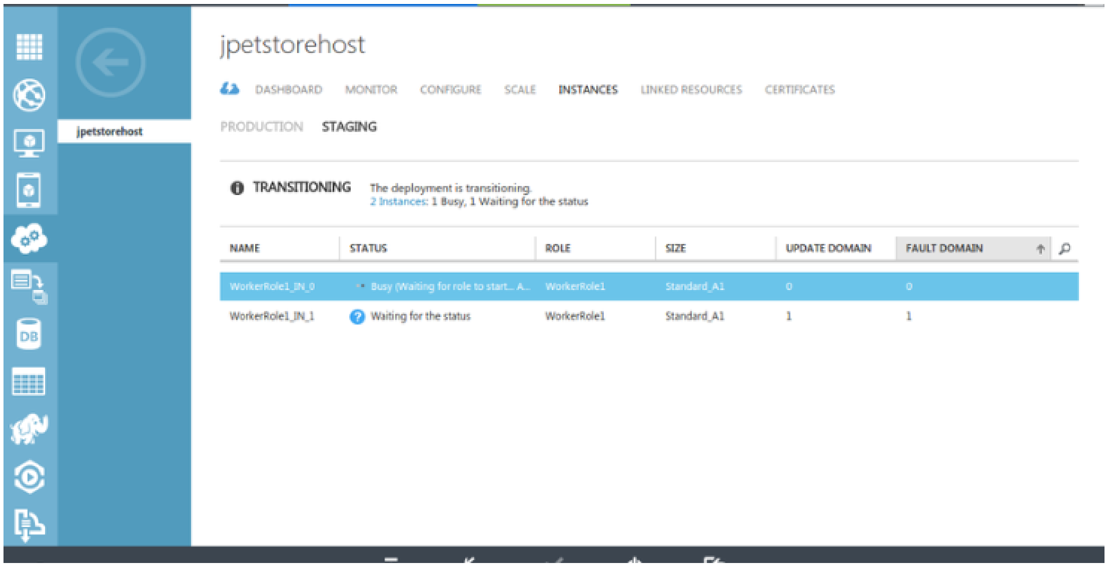
5

Results and Observations:



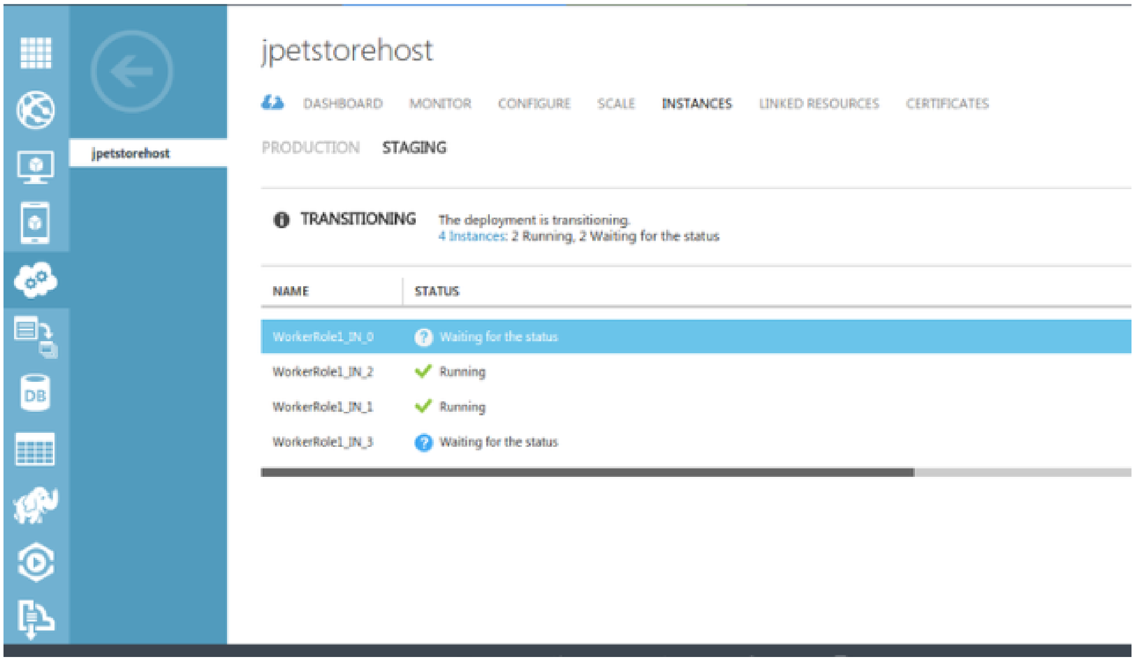
NAME	SOURCE	MIN	MAX	AVG	TOTAL	ALERT RULES
Disk Read Bytes/sec	WorkerRole1	0 B/s	51.12 KB/s	11.36 KB/s	---	Not Configured
Disk Write Bytes/sec	WorkerRole1	7.24 KB/s	44.87 KB/s	15.93 KB/s	---	Not Configured
Network In	WorkerRole1	0 B	0 B	0 B	0 B	Not Configured

New instances of the VM are created when CPU percentage goes past the 45%. These are shown in the figures below.



TRANSITIONING The deployment is transitioning.
2 Instances: 1 Busy, 1 Waiting for the status

NAME	STATUS	ROLE	SIZE	UPDATE DOMAIN	FAULT DOMAIN
WorkerRole1_IN_0	Busy (Waiting for role to start... A...	WorkerRole1	Standard_A1	0	0
WorkerRole1_IN_1	Waiting for the status	WorkerRole1	Standard_A1	1	1



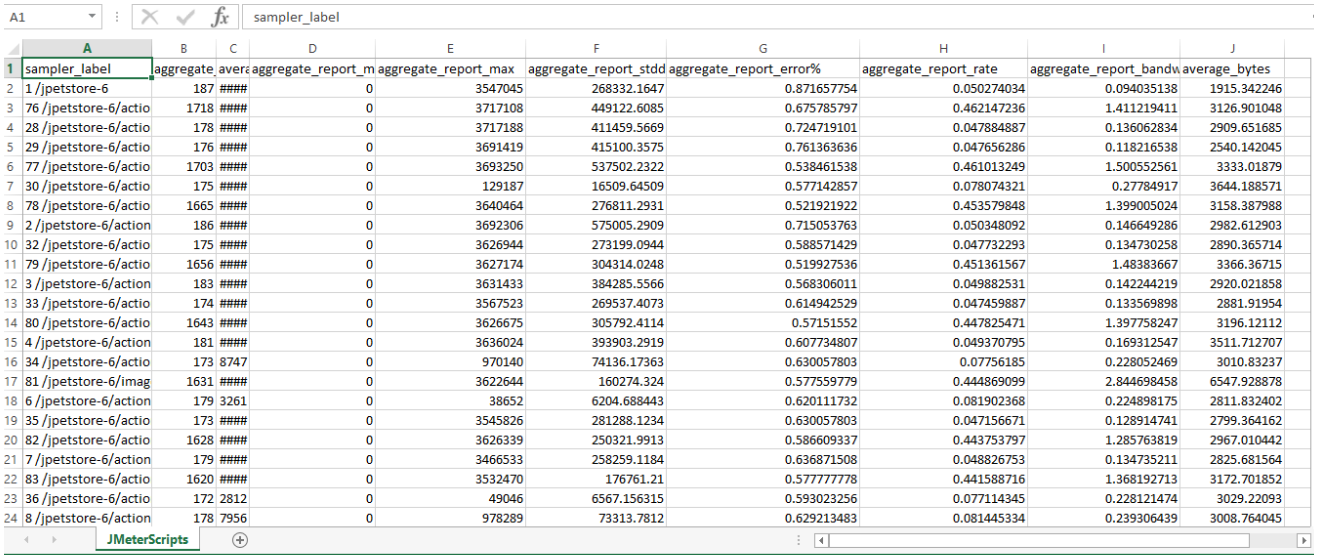
jpetstorehost

TRANSITIONING The deployment is transitioning.
4 Instances: 2 Running, 2 Waiting for the status

NAME	STATUS
WorkerRole1_IN_0	Waiting for the status
WorkerRole1_IN_2	✓ Running
WorkerRole1_IN_1	✓ Running
WorkerRole1_IN_3	Waiting for the status

The throughput and the error percentages are greatly reduced. These can be seen from the Summary report generated by the JMeter.

Report: DOCC Project\JMeter Reports\JMeterScripts.csv



sampler_label

	A	B	C	D	E	F	G	H	I	J
1	sampler_label	aggregate_	avera	aggregate_report_m	aggregate_report_max	aggregate_report_stdd	aggregate_report_error%	aggregate_report_rate	aggregate_report_bandw	average_bytes
2	1 /jpetstore-6	187	####	0	3547045	268332.1647	0.871657754	0.050274034	0.094035138	1915.342246
3	76 /jpetstore-6/actio	1718	####	0	3717108	449122.6085	0.675785797	0.462147236	1.411219411	3126.901048
4	28 /jpetstore-6/actio	178	####	0	3717188	411459.5669	0.724719101	0.047884887	0.136062834	2909.651685
5	29 /jpetstore-6/actio	176	####	0	3691419	415100.3575	0.761363636	0.047656286	0.118216538	2540.142045
6	77 /jpetstore-6/actio	1703	####	0	3693250	537502.2322	0.538461538	0.461013249	1.500552561	3333.01879
7	30 /jpetstore-6/actio	175	####	0	129187	16509.64509	0.577142857	0.078074321	0.27784917	3644.188571
8	78 /jpetstore-6/actio	1665	####	0	3640464	276811.2931	0.521921922	0.453579848	1.399005024	3158.387988
9	2 /jpetstore-6/action	186	####	0	3692306	575005.2909	0.715053763	0.050348092	0.146649286	2982.612903
10	32 /jpetstore-6/actio	175	####	0	3626944	273199.0944	0.588571429	0.047732293	0.134730258	2890.365714
11	79 /jpetstore-6/actio	1656	####	0	3627174	304314.0248	0.519927536	0.451361567	1.48383667	3366.36715
12	3 /jpetstore-6/action	183	####	0	3631433	384285.5566	0.568306011	0.049882531	0.142244219	2920.021858
13	33 /jpetstore-6/actio	174	####	0	3567523	269537.4073	0.614942529	0.047459887	0.133569898	2881.91954
14	80 /jpetstore-6/actio	1643	####	0	3626675	305792.4114	0.57151552	0.447825471	1.397758247	3196.12112
15	4 /jpetstore-6/action	181	####	0	3636024	393903.2919	0.607734807	0.049370795	0.169312547	3511.712707
16	34 /jpetstore-6/actio	173	8747	0	970140	74136.17363	0.630057803	0.07756185	0.228052469	3010.83237
17	81 /jpetstore-6/imag	1631	####	0	3622644	160274.324	0.577559779	0.444869099	2.844698458	6547.928878
18	6 /jpetstore-6/action	179	3261	0	38652	6204.688443	0.620111732	0.081902368	0.224898175	2811.832402
19	35 /jpetstore-6/actio	173	####	0	3545826	281288.1234	0.630057803	0.047156671	0.128914741	2799.364162
20	82 /jpetstore-6/actio	1628	####	0	3626339	250321.9913	0.586609337	0.443753797	1.285763819	2967.010442
21	7 /jpetstore-6/action	179	####	0	3466533	258259.1184	0.636871508	0.048826753	0.134735211	2825.681564
22	83 /jpetstore-6/actio	1620	####	0	3532470	176761.21	0.577777778	0.441588716	1.368192713	3172.701852
23	36 /jpetstore-6/actio	172	2812	0	49046	6567.156315	0.593023256	0.077114345	0.228121474	3029.22093
24	8 /jpetstore-6/action	178	7956	0	978289	73313.7812	0.629213483	0.081445334	0.239306439	3008.764045

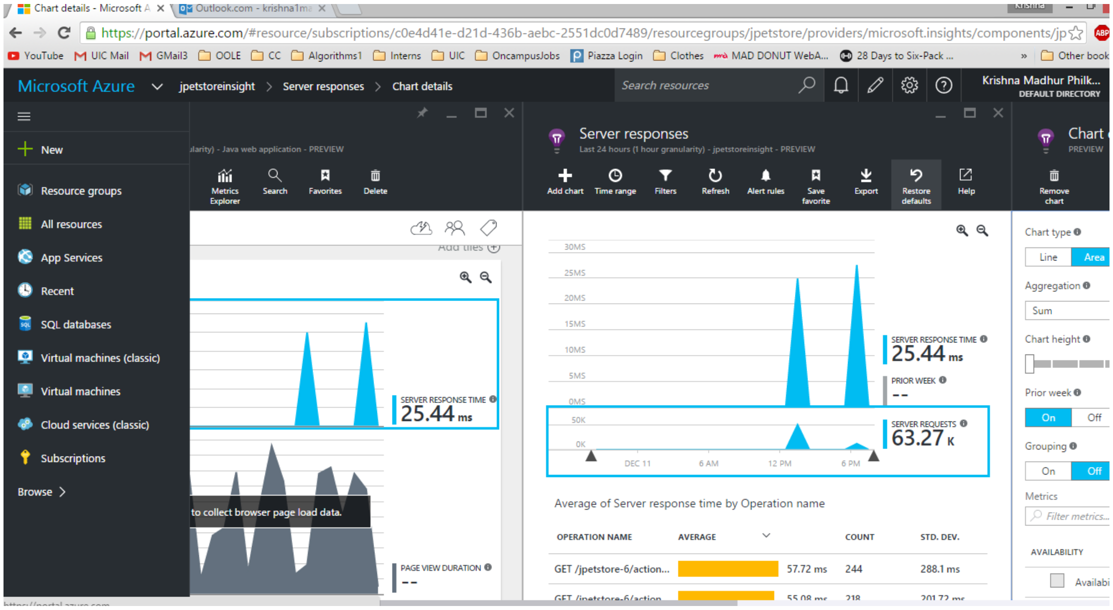
Experiment 5: Application Insights

I wanted to get the other metrics like the Server response Time and RAM memory usage etc.

So I employed the Applications Insight that is provided by the Microsoft Azure.

To Configure:

1. Create an Application Insight in the portal. Name it as JPetStoreApplicationInsight.
2. Give the resource name as 'JPetStore'
3. Go to the Eclipse from where you deployed the project. Right click on the application. Azure-> ApplicationInsights
4. Now Add the Instrumentation Key by copying the key from the portal. This will help the JPetStore cloud service to link the new metrics to the application.
5. Publish your project again. Refresh the portal. We can see the new metrics that come up in the application insight- JPetstoreApplicationInsight.



https://portal.azure.com/#resource/subscriptions/c0e4d41e-d21d-436b-aebc-2551dc0d7489/resourcegroups/jpetstore/providers/microsoft.insights/components/jp

Chart type: Line [✓ Area]
Aggregation: Sum
Prior week: [✓ On] Off
Grouping: On [✓ Off]

Average of Server response time by Operation name

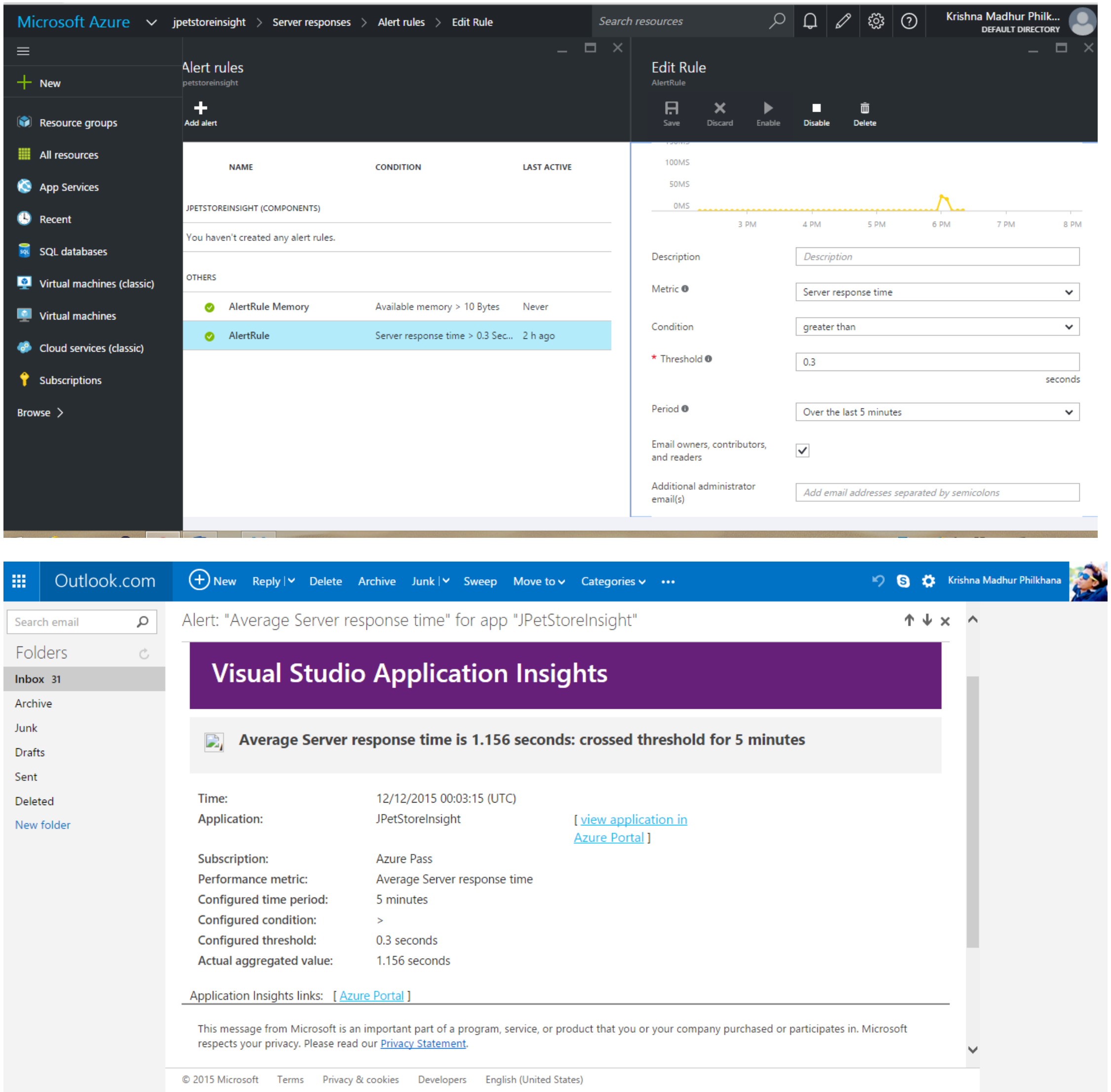
OPERATION NAME	AVERAGE	COUNT	STD. DEV.
GET /jpetstore-6/action...	57.72 ms	244	288.1 ms
GET /jpetstore-6/action...	55.08 ms	218	201.72 ms

On these application insights, we can specify ALERT rules.

The alert rules notify the administrator and whoever else is configured about the threshold alerts that the administrator places on these metrics.

The Application will not auto scale using these metrics. But these metrics are very useful for the maintenance of the web applications in Microsoft Azure cloud.

They help the user to reduce their costs and help them maintain their web application consistent.



NAME	CONDITION	LAST ACTIVE
JPETSTOREINSIGHT (COMPONENTS)		
You haven't created any alert rules.		
OTHERS		
✓ AlertRule Memory	Available memory > 10 Bytes	Never
✓ AlertRule	Server response time > 0.3 Sec...	2 h ago

Description: Description

Metric: Server response time

Condition: greater than

* Threshold: 0.3 seconds

Period: Over the last 5 minutes

Email owners, contributors, and readers: ☑

Additional administrator email(s): Add email addresses separated by semicolons

Alert: "Average Server response time" for app "JPetStoreInsight"

Visual Studio Application Insights

Average Server response time is 1.156 seconds: crossed threshold for 5 minutes

Time:	12/12/2015 00:03:15 (UTC)
Application:	JPetStoreInsight [view application in Azure Portal]
Subscription:	Azure Pass
Performance metric:	Average Server response time
Configured time period:	5 minutes
Configured condition:	>
Configured threshold:	0.3 seconds
Actual aggregated value:	1.156 seconds

Application Insights links: [Azure Portal]

This message from Microsoft is an important part of a program, service, or product that you or your company purchased or participates in. Microsoft respects your privacy. Please read our Privacy Statement.

© 2015 Microsoft   Terms   Privacy & cookies   Developers   English (United States)

Fig: Alert on Average Response Time

9. Conclusions:

The provisioning rule that we developed will result in the increase in availability and consistency of the JPetStore web application.

<u>**PROVISIONING RULE**</u>: R: **Increase the CPU instances to 2, when the CPU percentage crosses over 45%.**

sinst(a,i) when CPU percentage crosses 45%

Here, a is the ‘WorkerRole1’ and i is ‘1’.

Also, the alerts would greatly help the user who deploys his/her applications in the cloud platforms.

10. Learnings from the Project:

1. Deploying Web Applications in cloud
2. Concept of Performance Testing
3. JMeter
4. Azure Portal
5. Practical implementation of Elasticity of Cloud