



Using RTI and CloudWatch to Control Scaling in AWS EC2

January 2014

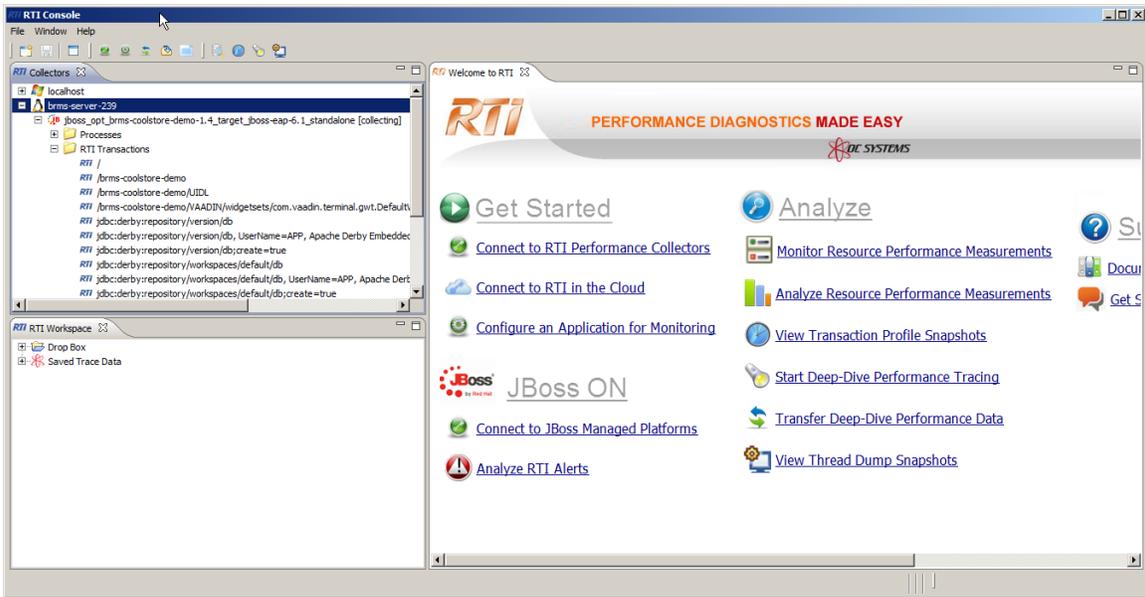
This paper describes the results of an experiment using RTI metrics to trigger CloudWatch scaling policies to control the scaling of a JBoss web application in the Amazon Web Services (AWS) Elastic Compute Cloud (EC2). We will discuss the setup and process and the final results. The primary purpose of this experiment was to investigate the mechanisms of cloud instance scaling with an eye towards using more application-specific metrics to control the scale up/down decision.

Setup

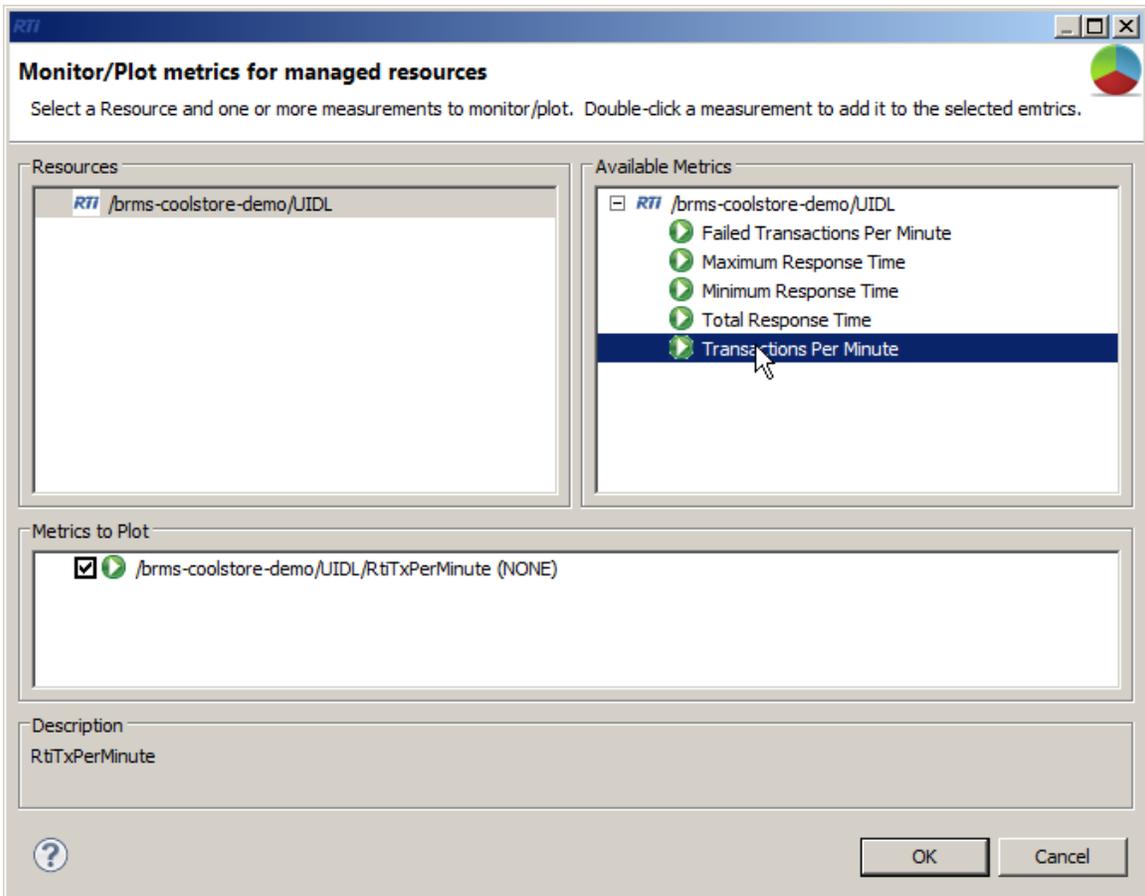
This experiment was run in the AWS EC2 using an Elastic Load Balancer (ELB) as the front-end connected to between 1 and 3 instances of an AMI running the JBoss BRMS CoolStore demo application, all contained in an Auto Scaling Group (ASG). Scale-up and scale-down policies were defined for the ASG. RTI was installed on the JBoss instances as well as a script to periodically extract RTI metrics and post them to CloudWatch. JMeter was set up on an external system to provide the test load.

Detailed Setup

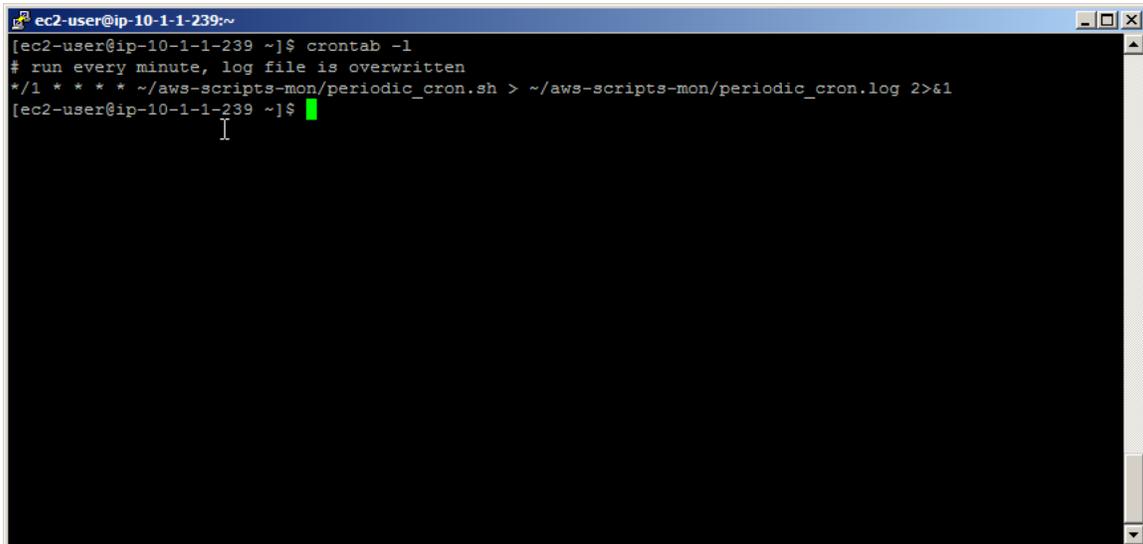
This is the RTI Collector for the “base” BRMS instance viewed from the RTI console:



This is the selected request, /brms-coolstore-demo/UIDL, and the selected metric, Transactions Per Minute, viewed from the RTI console:

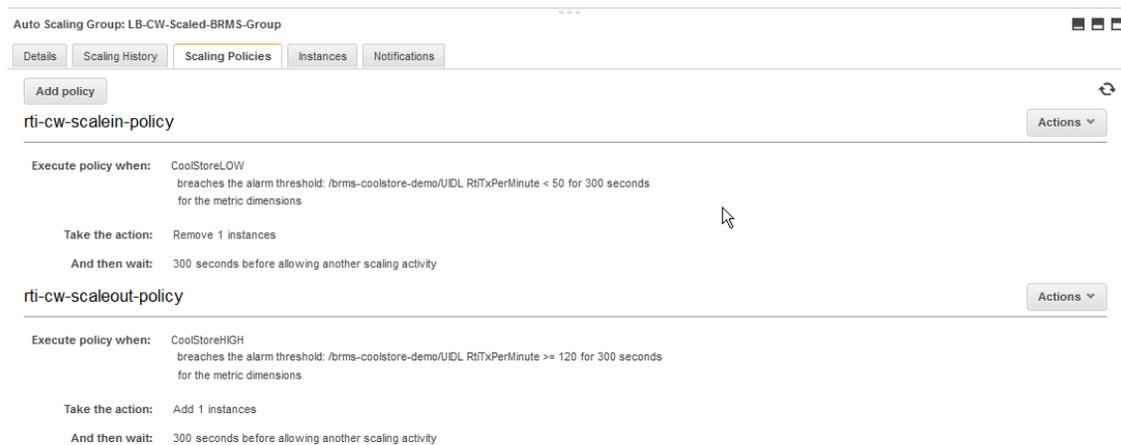


This is the crontab entry used to periodically execute the script to extract the selected RTI metric and forward it to CloudWatch:



```
ec2-user@ip-10-1-1-239:~  
[ec2-user@ip-10-1-1-239 ~]$ crontab -l  
# run every minute, log file is overwritten  
*/1 * * * * ~/aws-scripts-mon/periodic_cron.sh > ~/aws-scripts-mon/periodic_cron.log 2>&1  
[ec2-user@ip-10-1-1-239 ~]$
```

These are the ASG scale-up and scale-down policies. Scale-up (scale-out) triggers when the metric value exceeds 120 transactions per minute for 5 minutes and scale-down (scale-in) triggers when the metric value falls below 50 transactions per minute for 5 minutes:



Auto Scaling Group: LB-CW-Scaled-BRMS-Group

Details | Scaling History | **Scaling Policies** | Instances | Notifications

Add policy

rli-cw-scalein-policy Actions

Execute policy when: CoolStoreLOW
breaches the alarm threshold: /brms-coolstore-demo/UIDL_RITxPerMinute < 50 for 300 seconds for the metric dimensions

Take the action: Remove 1 instances

And then wait: 300 seconds before allowing another scaling activity

rli-cw-scaleout-policy Actions

Execute policy when: CoolStoreHIGH
breaches the alarm threshold: /brms-coolstore-demo/UIDL_RITxPerMinute >= 120 for 300 seconds for the metric dimensions

Take the action: Add 1 instances

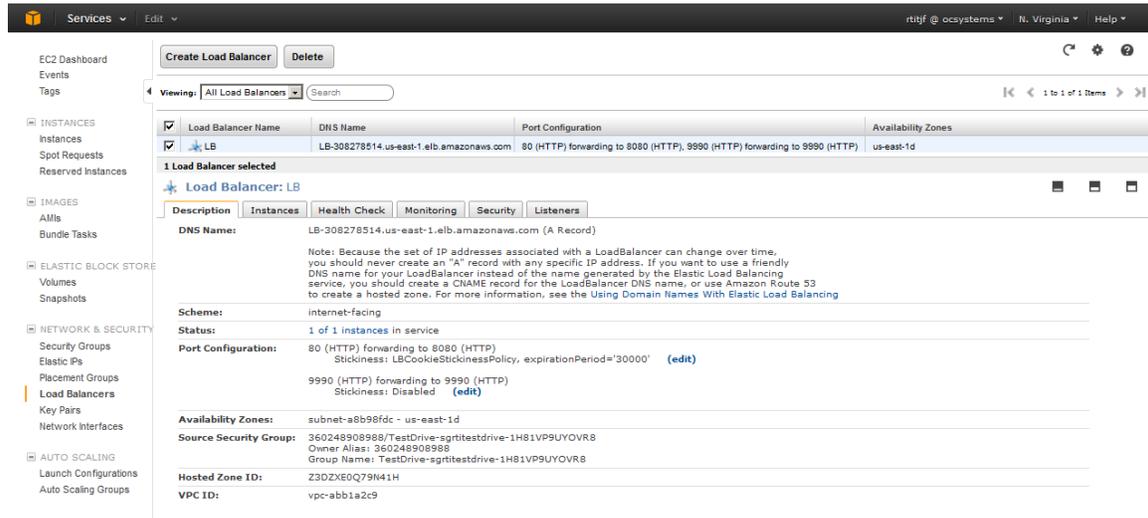
And then wait: 300 seconds before allowing another scaling activity

The Test

The general approach was to set up the scaling system, define the scale-up and scale-down policies for the ASG, and then run a load test, which would trigger the scale-up

(high traffic) and then the scale-down (low traffic) via RTI metrics posted to CloudWatch.

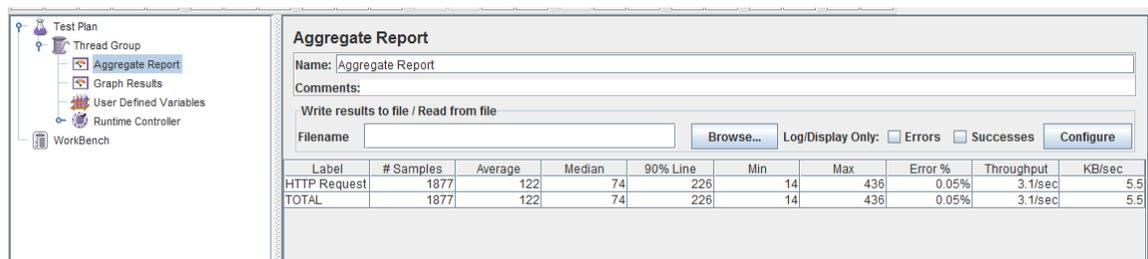
This is the Elastic Load Balancer, which serves as the front-end of the test system. The ELB will distribute incoming requests to the available back-end JBoss instances:



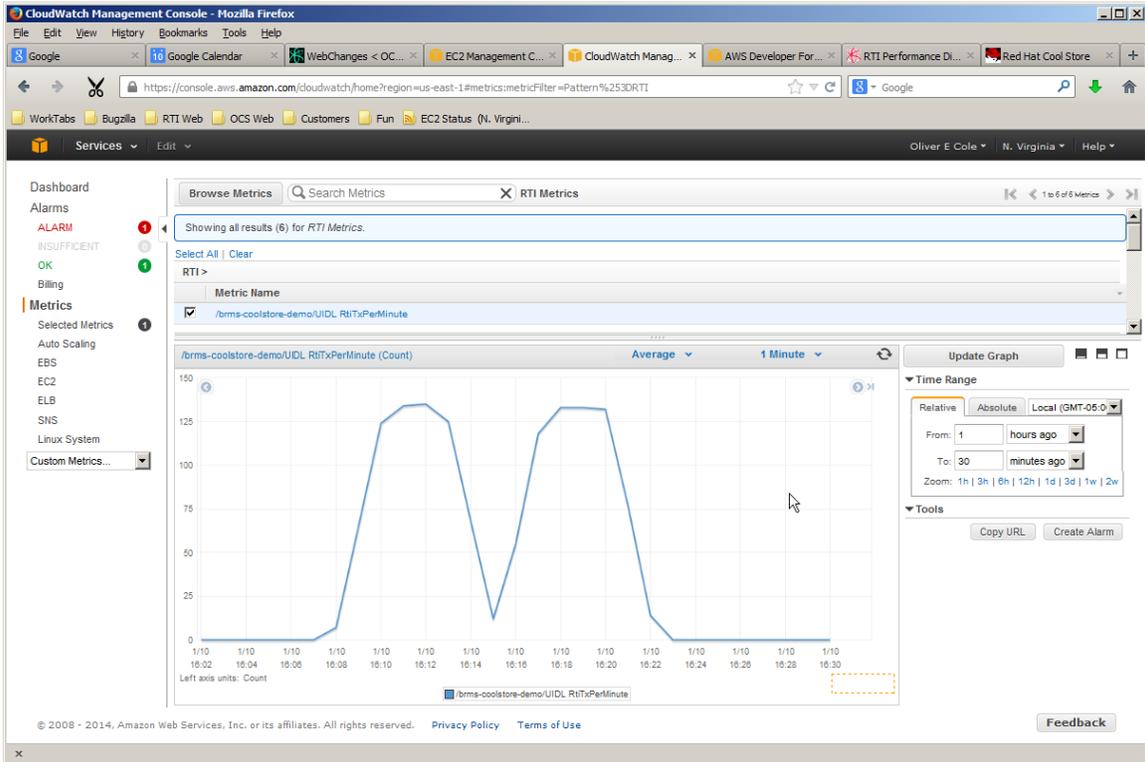
This is the Auto Scaling Group for the BRMS application showing one instance running (the base instance) at the start of the test:



The load test was started in JMeter to produce a high traffic load:



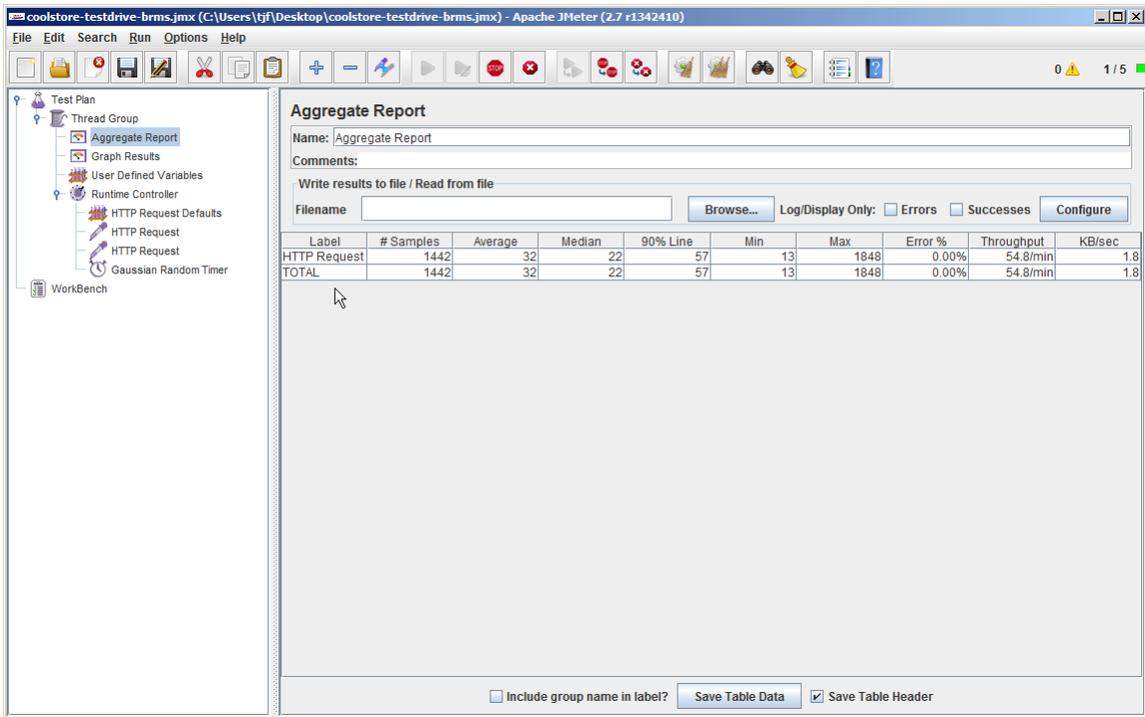
This is the CloudWatch display of the RTI metric showing the number of requests per minute has exceeded the threshold of 120 (it peaks at 140 per minute at 16:12):



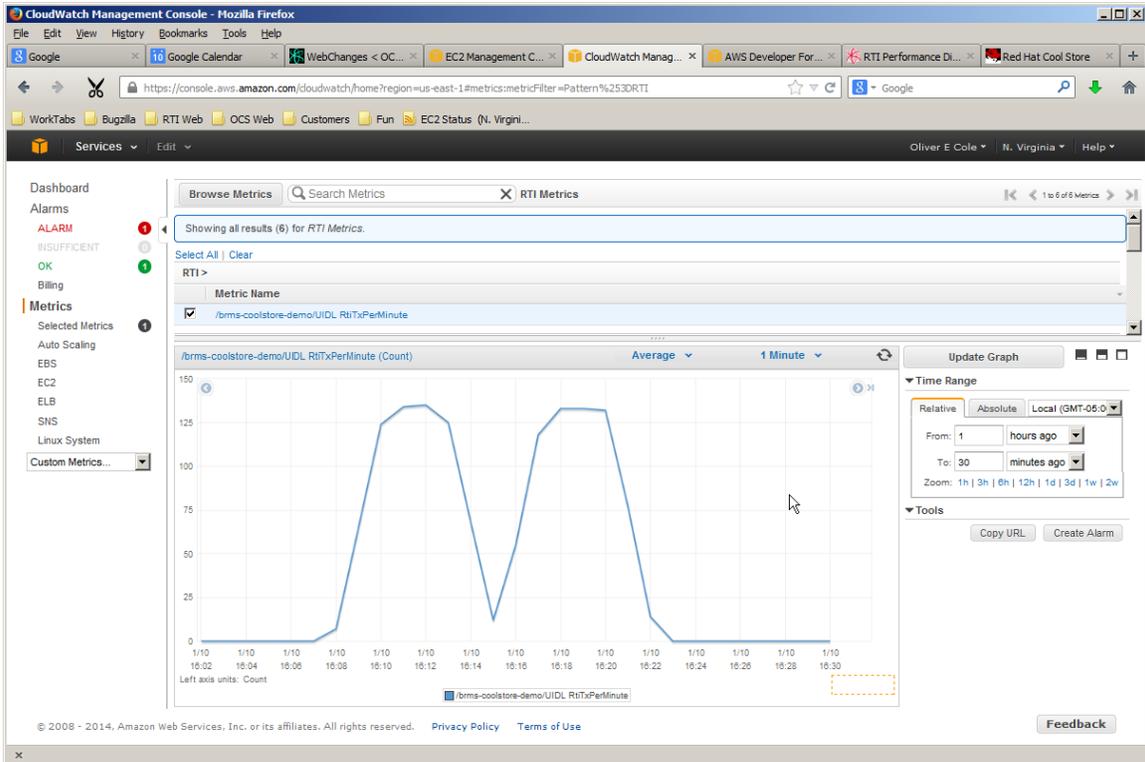
These are the CloudWatch notifications indicating a scale-up alert at 16:11 and a launch complete at 16:12 (and 16:18):

ALARM: "CoolStoreLOW" in US - N. Virginia	☆	AWS Notifications	6.2 KB	4:03 PM
ALARM: "CoolStoreHIGH" in US - N. Virginia	☆	AWS Notifications	6.3 KB	4:11 PM
Auto Scaling: launch for group "LB-CW-Scaled-BRMS-Group"	☆	AWS Notifications	6.3 KB	4:12 PM
Auto Scaling: launch for group "LB-CW-Scaled-BRMS-Group"	☆	AWS Notifications	6.3 KB	4:18 PM
ALARM: "CoolStoreLOW" in US - N. Virginia	☆	AWS Notifications	6.2 KB	4:25 PM
Auto Scaling: termination for group "LB-CW-Scaled-BRMS-Group"	☆	AWS Notifications	6.4 KB	4:27 PM

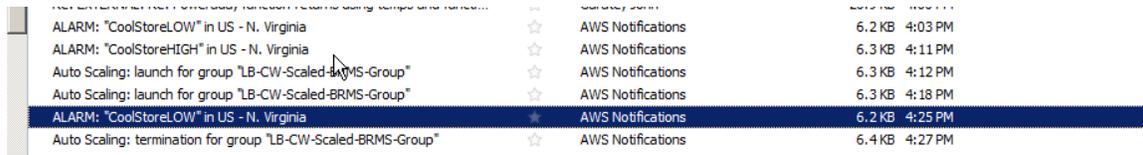
The JMeter load test tapered off at 16:20 to reduce traffic:



This is the CloudWatch display of the RTI metric showing the load dropped below 50 requests per minute after 16:21):



The CloudWatch notifications indicating a scale-down alarm at 16:25 is highlighted and the actual scale-down is logged at 16:27:



The screenshot shows a list of AWS CloudWatch notifications. The notification for the 'CoolStoreLOW' alarm is highlighted in blue, indicating it was triggered at 4:25 PM. The following notification shows the auto-scaling group 'LB-CW-Scaled-BRMS-Group' being terminated at 4:27 PM.

Notification Title	Sender	Size	Time
ALARM: "CoolStoreLOW" in US - N. Virginia	AWS Notifications	6.2 KB	4:03 PM
ALARM: "CoolStoreHIGH" in US - N. Virginia	AWS Notifications	6.3 KB	4:11 PM
Auto Scaling: launch for group "LB-CW-Scaled-BRMS-Group"	AWS Notifications	6.3 KB	4:12 PM
Auto Scaling: launch for group "LB-CW-Scaled-BRMS-Group"	AWS Notifications	6.3 KB	4:18 PM
ALARM: "CoolStoreLOW" in US - N. Virginia	AWS Notifications	6.2 KB	4:25 PM
Auto Scaling: termination for group "LB-CW-Scaled-BRMS-Group"	AWS Notifications	6.4 KB	4:27 PM

Results

The exercise demonstrated how RTI metrics can be used to control scaling in AWS EC2 by creating CloudWatch custom metrics. Application-specific RTI metrics may provide a more accurate way to control application scaling than using “external” throughput or load metrics, allowing scale up/down decisions to be made before application performance degrades in user-impacting ways.