Capstone Project – Zhuoyu Zhu

Q7. Handle archiving of results files for free users after 30 minutes

Approach used:
Each time the specified job request is completed and is about to send confirmation email to the user, I first check if the current user is a free user role by checking the data received from SQS. If the user is a free user, I use subprocess method and schedule an archive task in 30 minutes by using sched library. In addition, in case the specified user decides to upgrade to the premium user role within 30 minutes frame, to prevent archiving results file in this scenario, I check the user role again when the scheduled task is about to execute. If the user has upgraded to premium, I don't need to archive this result file any more otherwise we need to archive this result file to Amazon Glacier.

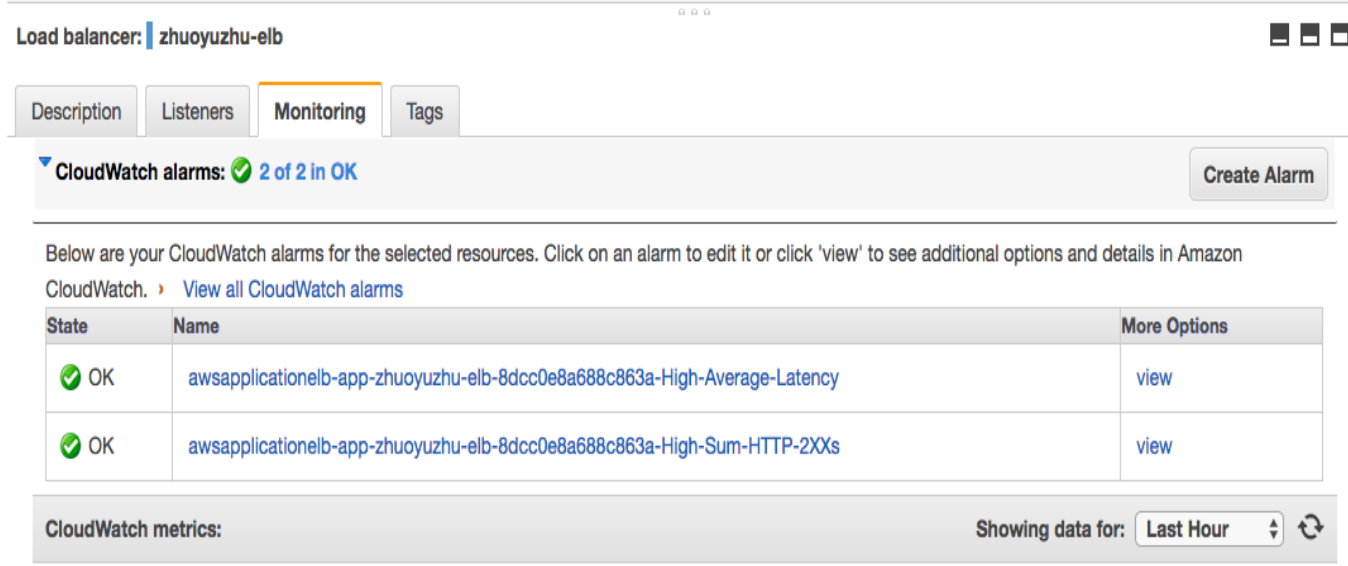
Reason why I used this approach:
Using task scheduler and subprocess make the entire archive process workload more distributed without slowing down user experience on our platform. On the other hand, if you check the time passed within a loop for each job request, you might either need to read data from DynamoDB or SQS. Using these approaches will slow down the annotation platform significantly since the cost of reading data constantly from DynamoDB or SQS within a loop is very expensive. Moreover, the size of SQS is limited, if our platform has hundreds of million of job ready for archive, we have to spend additional efforts to maintain the SQS or distribute some of the workload to another SQS. Using task scheduler and subprocess will solve all of these issues.

Q9. Restore data for users that upgrade to Premium
Approach used:
After the user has upgraded their membership successfully without throwing Stripe API exception, I initiate the archive restoration job request for any result files that were generated while the user was a free user. Then I used an Amazon SQS (zhuoyuzhu_glacier) to accept these request within mpcs_app.py file. After this, within the restore instance, I retrieve these request messages and check status. When the restoration is ready, the python script writes contents onto the result file and upload to S3 result bucket.

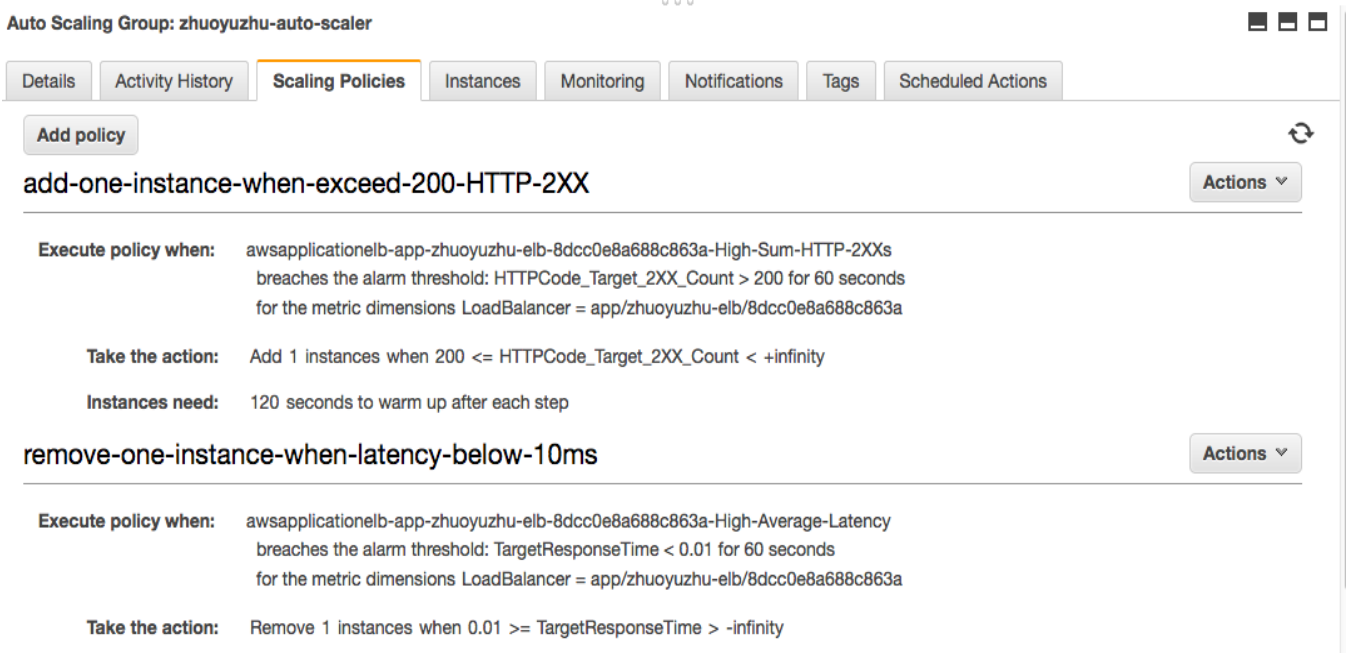
Reason why I used this approach:
Since the restoration requests for the given user are relatively lightweight compared to submitting annotation job request, so using Amazon SQS queue should be sufficient for this distributed cloud-based application. The chance about reaching the size limit of SQS is relatively small.

Q11. Add auto-scaling rules to the web server farm

CloudWatcch for High-Average-Latency and High-Sum-HTTP-2XXs:

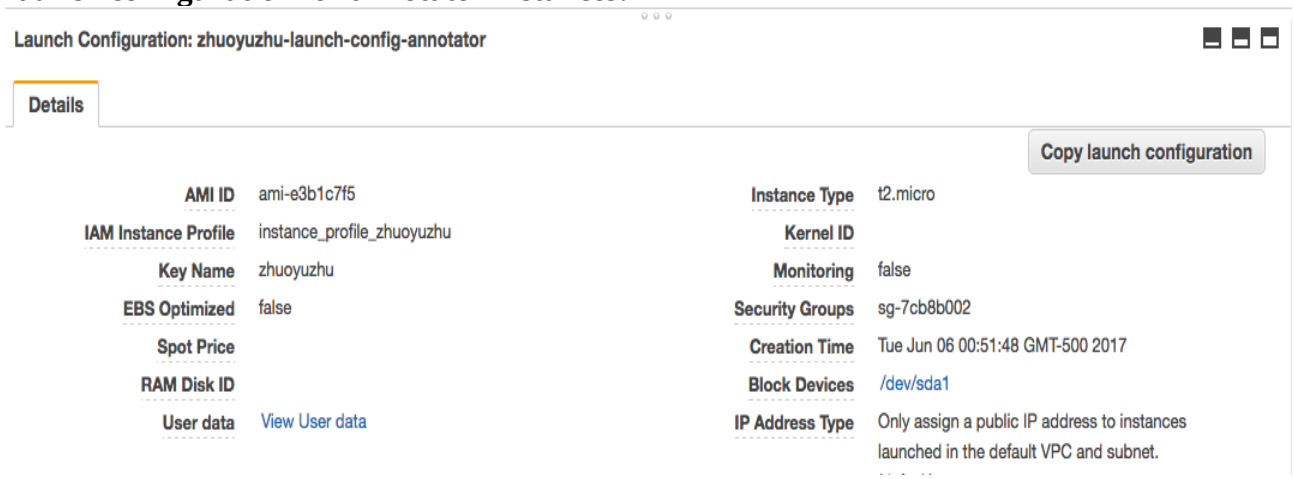


Scaling Policy:

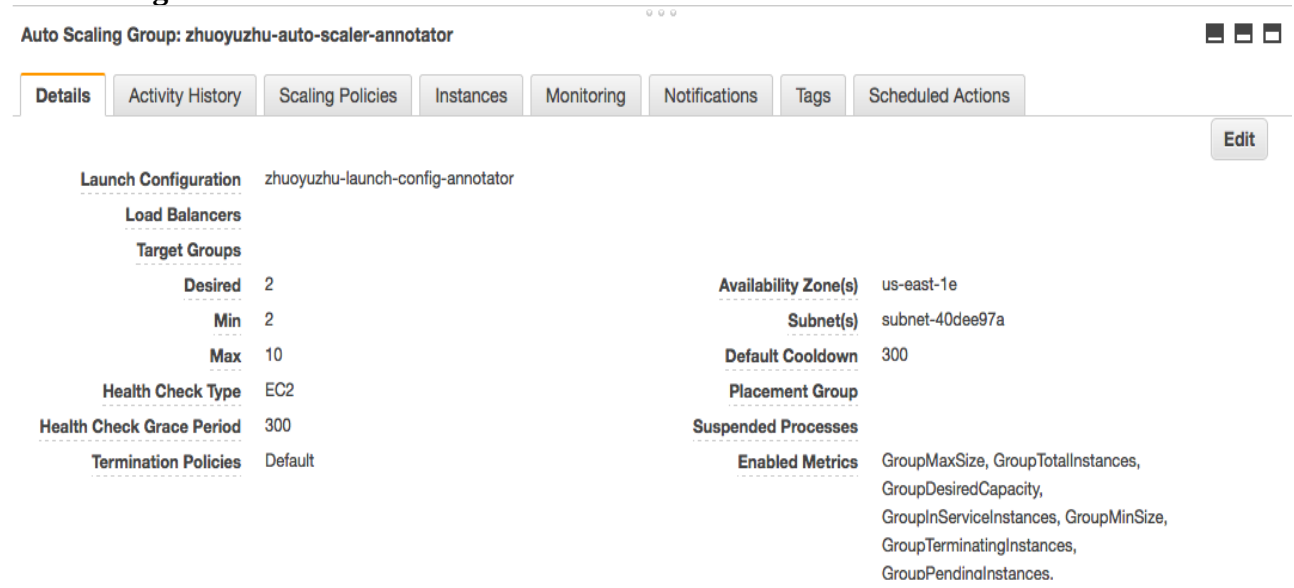


Q12. Add scaling rules to the annotator

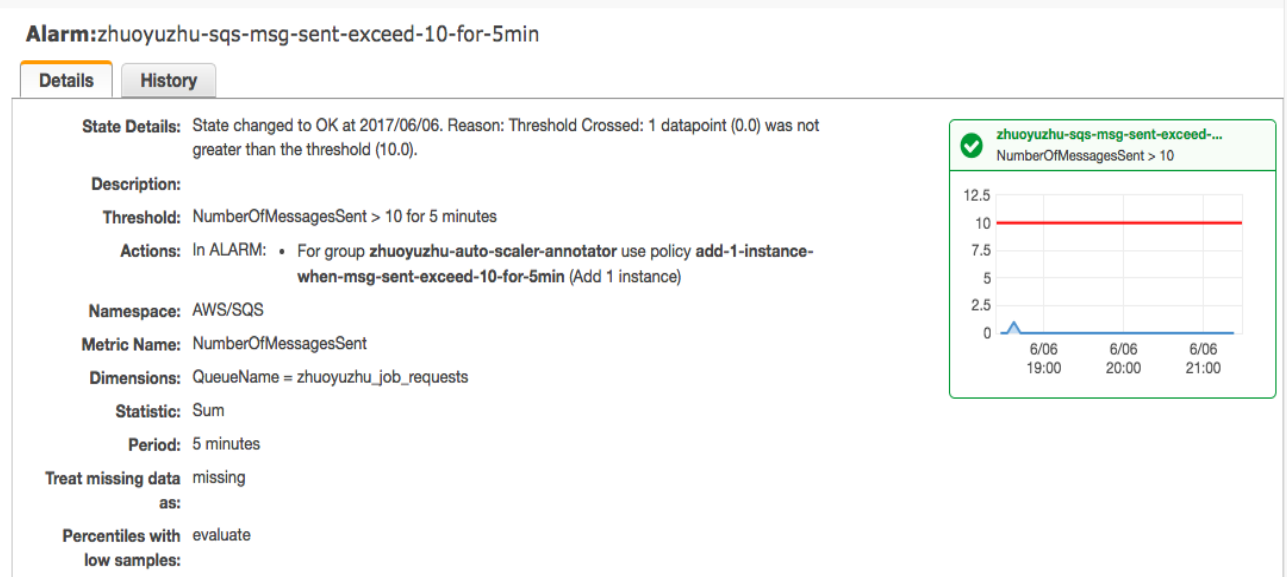
Launch configuration for annotator instances:



Auto-scaling for annotator instance:



CloudWatch for annotator:



Alarm:zhuoyuzhu-sqs-msg-received-below-5-for-2min

Details | History

State Details: State changed to ALARM at 2017/06/06. Reason: Threshold Crossed: 1 datapoint (0.0) was less than the threshold (5.0).

Description:

Threshold: NumberOfMessagesSent < 5 for 2 minutes

Actions: In ALARM:
- For group **zhuoyuzhu-auto-scaler-annotator** use policy **remove-1-instance-when-msg-received-below-5-for-2min** (Remove 1 instance)

Namespace: AWS/SQS

Metric Name: NumberOfMessagesSent

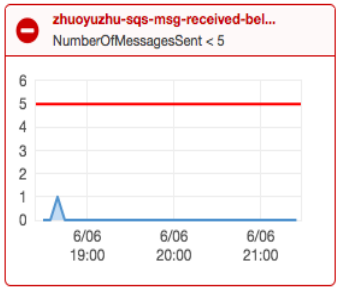
Dimensions: QueueName = zhuoyuzhu_job_requests

Statistic: Sum

Period: 1 minute

Treat missing data as: missing

Percentiles with low samples: evaluate



Auto-scaling policy for annotator:

Add policy

add-1-instance-when-msg-sent-exceed-10-for-5min

Actions

Execute policy when: zhuoyuzhu-sqs-msg-sent-exceed-10-for-5min
breaches the alarm threshold: NumberOfMessagesSent > 10 for 300 seconds
for the metric dimensions QueueName = zhuoyuzhu_job_requests

Take the action: Add 1 instances when 10 <= NumberOfMessagesSent < +infinity

Instances need: 120 seconds to warm up after each step

remove-1-instance-when-msg-received-below-5-for-2min

Actions

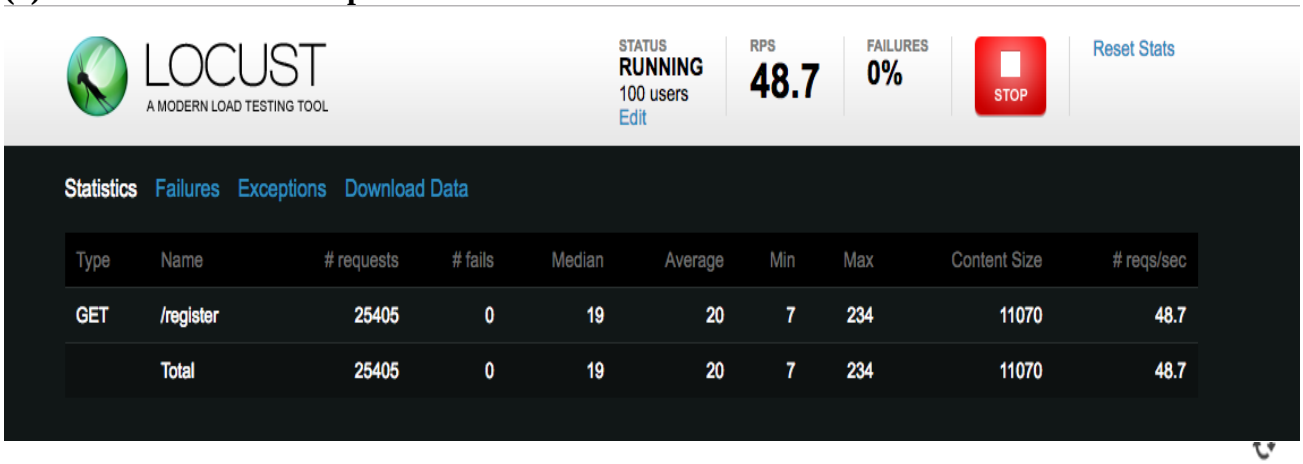
Execute policy when: zhuoyuzhu-sqs-msg-received-below-5-for-2min
breaches the alarm threshold: NumberOfMessagesSent < 5 for 2 consecutive periods of 60 seconds
for the metric dimensions QueueName = zhuoyuzhu_job_requests

Take the action: Remove 1 instances when 5 >= NumberOfMessagesSent > -infinity

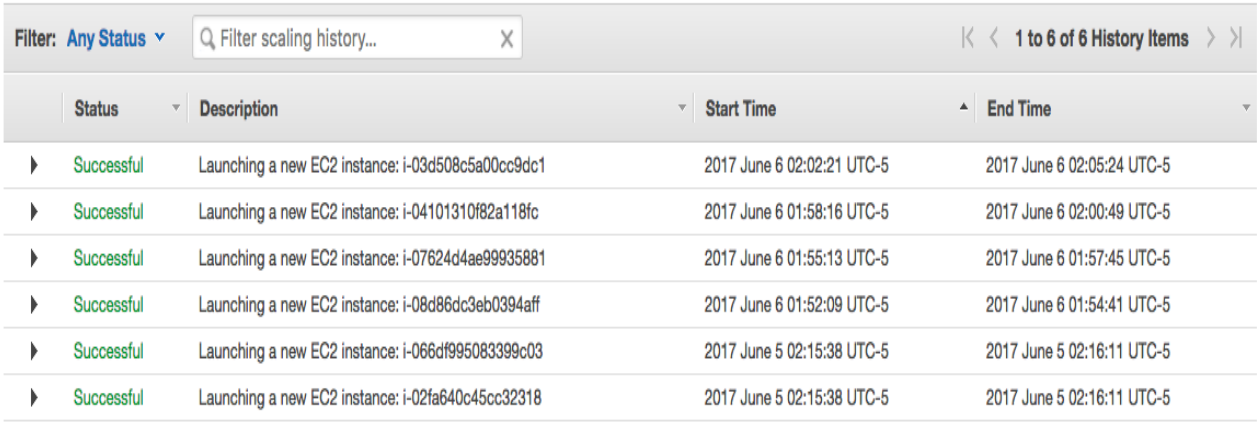
Q13. Test under load using Locust (Conclusion are at the bottom of this question)

1. Register template

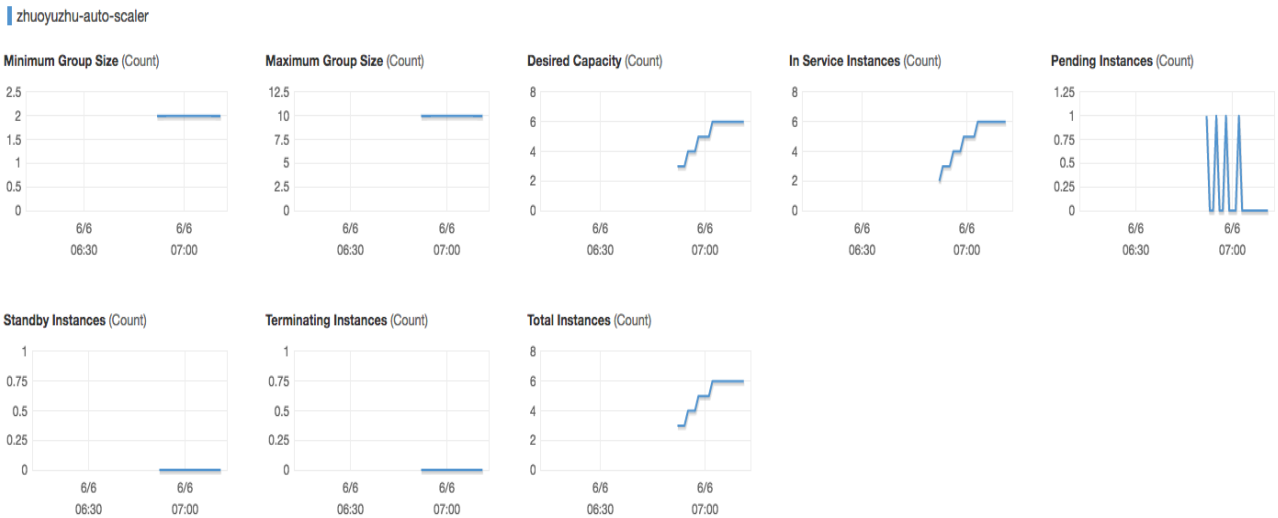
(1) 100 users and 5 users per second



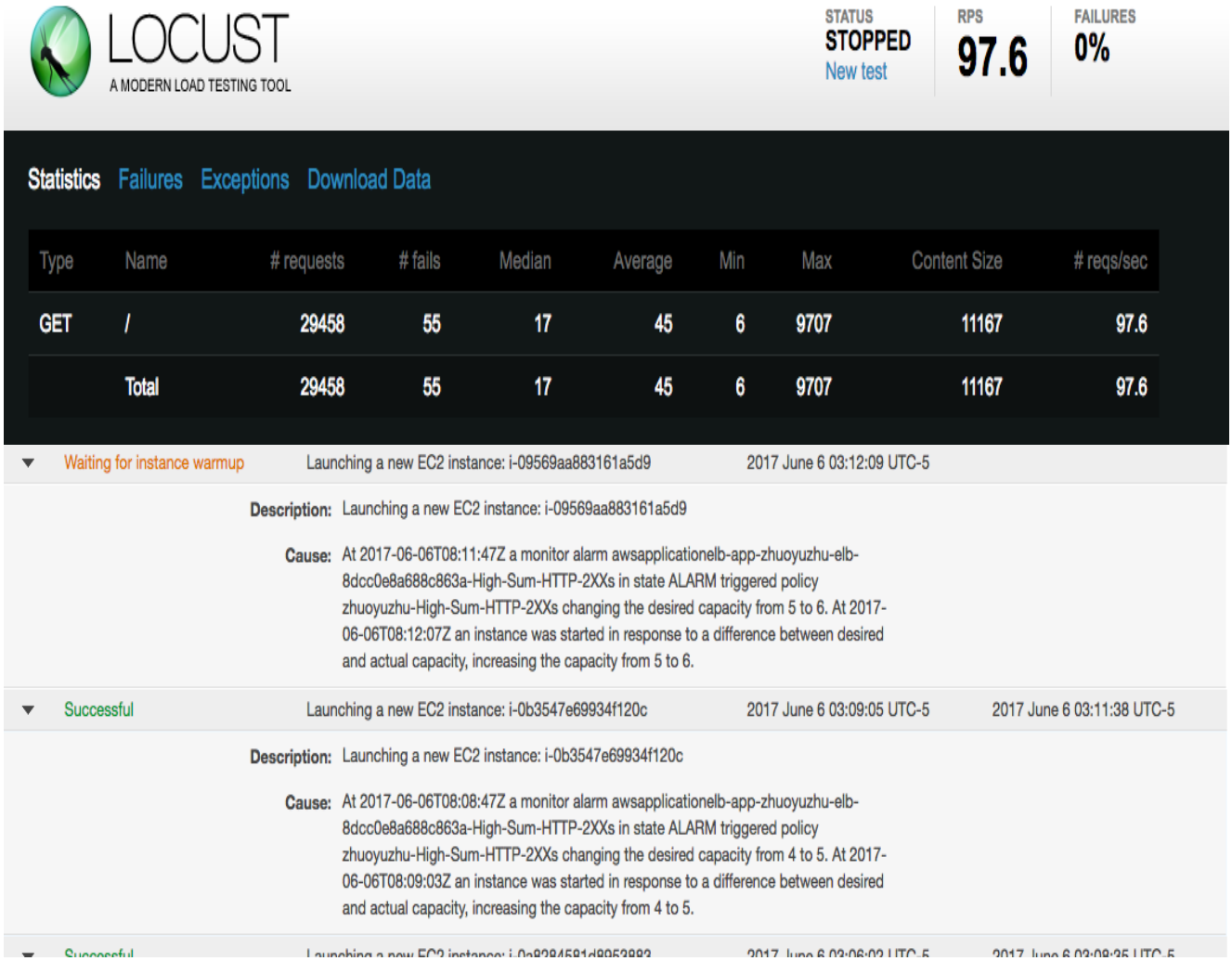
Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/register	25405	0	19	20	7	234	11070	48.7
	Total	25405	0	19	20	7	234	11070	48.7

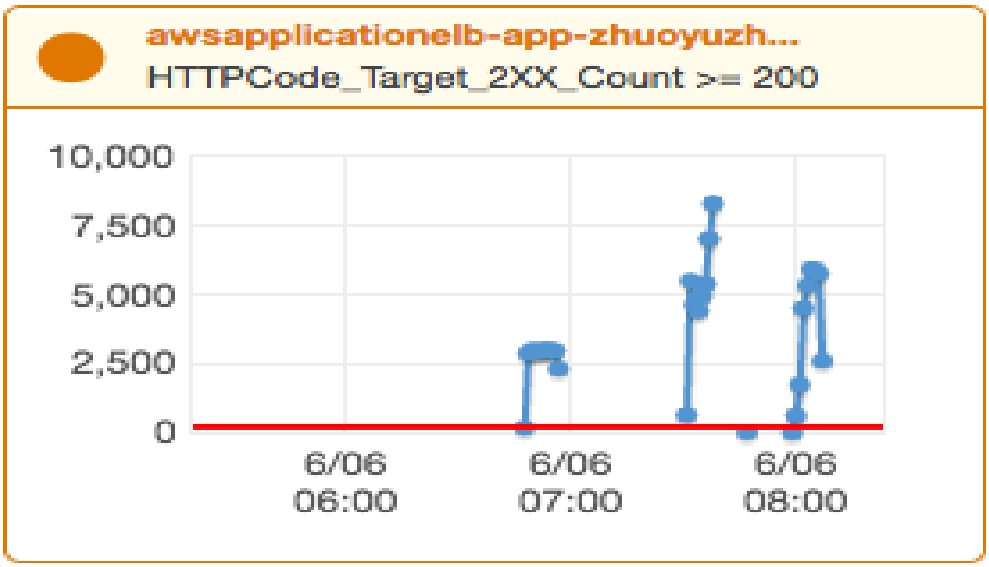


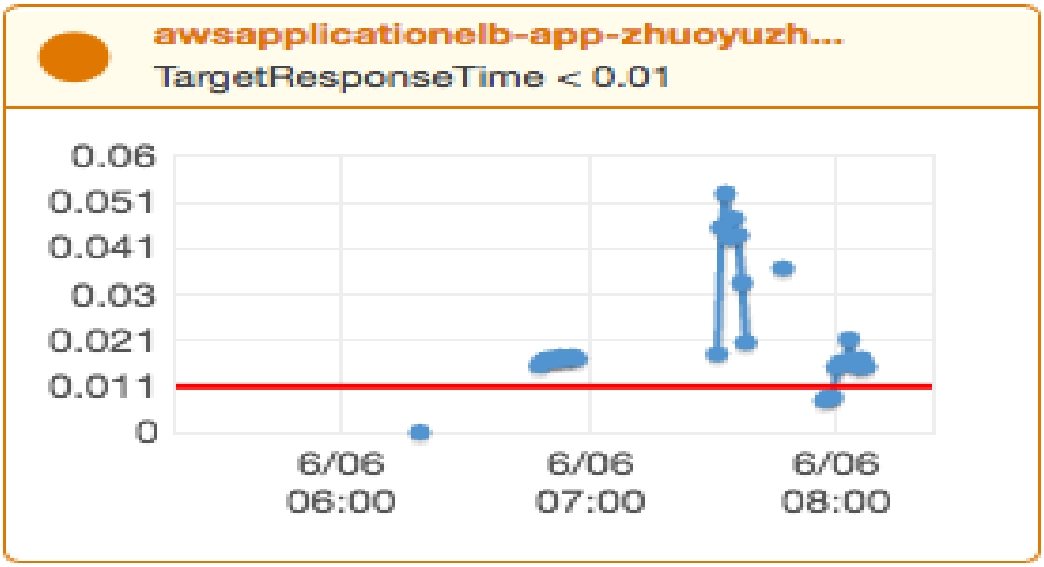
Status	Description	Start Time	End Time
Successful	Launching a new EC2 instance: i-03d508c5a00cc9dc1	2017 June 6 02:02:21 UTC-5	2017 June 6 02:05:24 UTC-5
Successful	Launching a new EC2 instance: i-04101310f82a118fc	2017 June 6 01:58:16 UTC-5	2017 June 6 02:00:49 UTC-5
Successful	Launching a new EC2 instance: i-07624d4ae99935881	2017 June 6 01:55:13 UTC-5	2017 June 6 01:57:45 UTC-5
Successful	Launching a new EC2 instance: i-08d86dc3eb0394aff	2017 June 6 01:52:09 UTC-5	2017 June 6 01:54:41 UTC-5
Successful	Launching a new EC2 instance: i-066df995083399c03	2017 June 5 02:15:38 UTC-5	2017 June 5 02:16:11 UTC-5
Successful	Launching a new EC2 instance: i-02fa640c45cc32318	2017 June 5 02:15:38 UTC-5	2017 June 5 02:16:11 UTC-5



2. Home Page
(1) 200 users and 20 users per second







Observation: The majority of the HTTP response code returns as 2XXs which mean our cloud-based annotation application works well when the workload is huge. It keep launch new instances when the alarm is triggered based on the specified timeframe. On the other hand, the other alarm average latency was never triggered. It means when the workload becomes huge, it will takes time to handle these amount of http request.

Q14. Python script for annotator load test

Please refer to the python file, annTest. Run out of time to capture the annotator test screenshots. The approach is using a while loop to keep send job request to the Amazon SNS topic and SQS queue. I used a count variable to record the current number of job requests sent. If the number of job requests exceeds 10, the python script will pause for 120 seconds otherwise it will pause for 25 second after each job request is sent. This will make sure to trigger the scale-out and scale-in policy for annotator to add and remove 1 instance respectively.