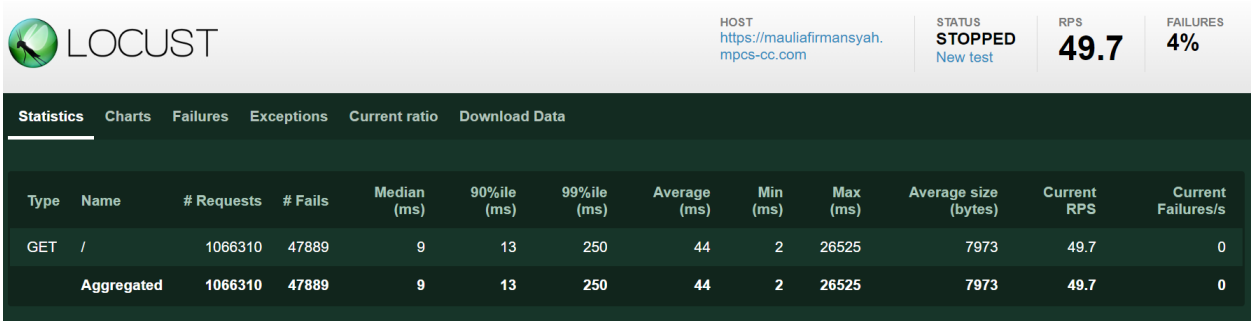
Additional Notes:

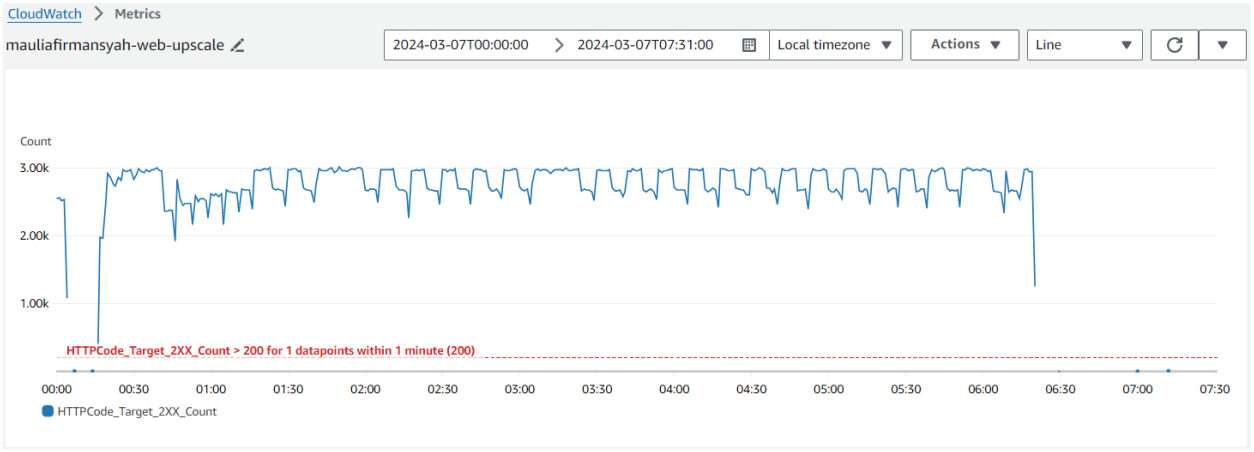
Muhammad Aulia Firmansyah
mauliafirmansyah@uchicago.edu

Test under load using Locust

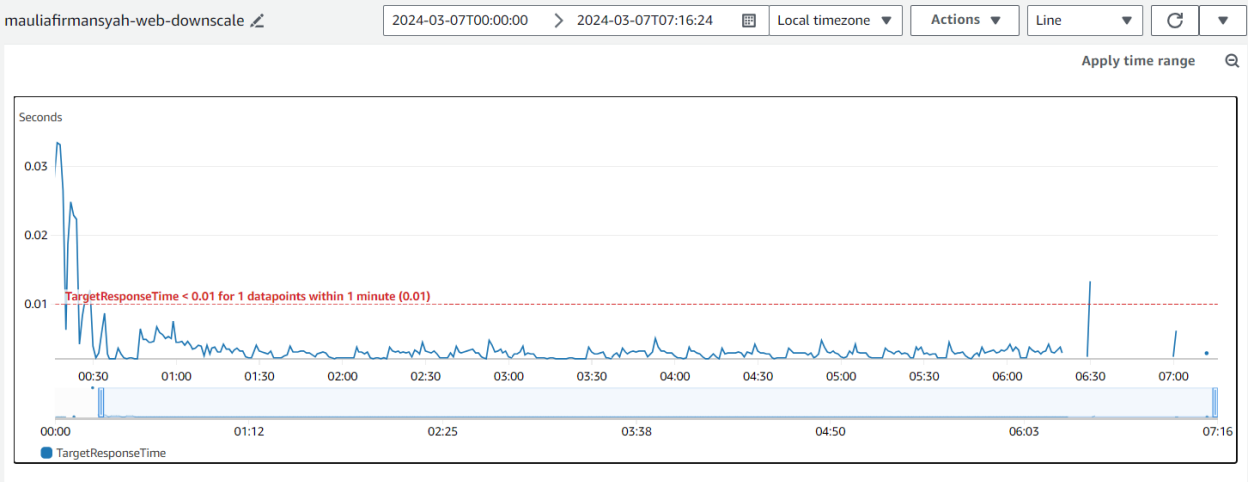


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	1066310	47889	9	13	250	44	2	26525	7973	49.7	0
	Aggregated	**1066310**	**47889**	**9**	**13**	**250**	**44**	**2**	**26525**	**7973**	**49.7**	**0**

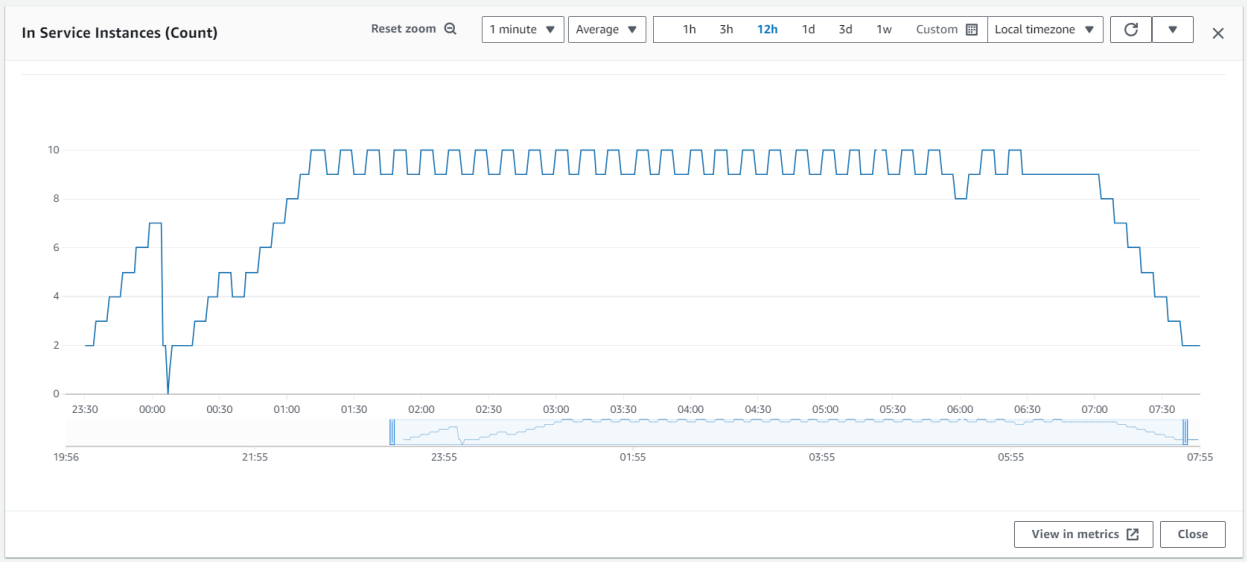
I run the locust test twice, because in the first run (23:30 to 00:00), the web downscaling alarm didn't trigger. In the second run (00:15 to 07.00), the locust was run using 100 users @ 5 users/second. These are several observations I found during the run:



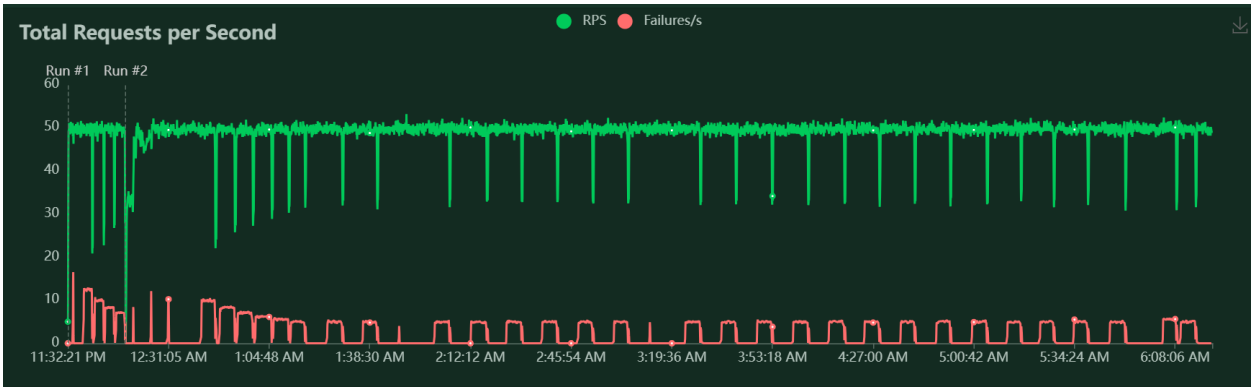
- The web upscale alarm was triggered as the request count reached 3000/minute, which is expected. Although, there was fluctuation where the requests dropped to around 2700.



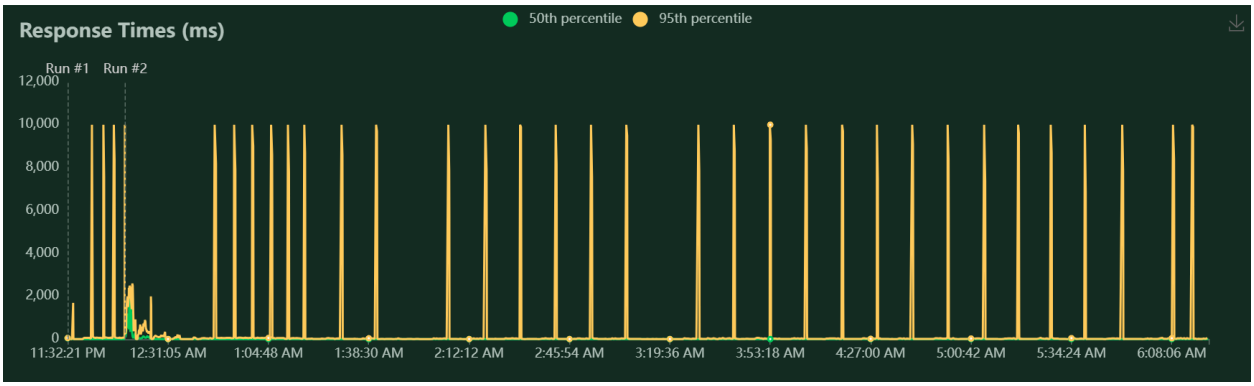
- Interestingly, the response time was only >= 0.01 for around 30 minutes (00:00 to 00:30). After that period, the response time remained < 0.01, thus trigger the web downscale alarm.



- Because web downscale alarm wasn't triggered from 00:00 to 00:30, the instances scaled up. After that, the web downscale alarm was triggered, prompting a small downscale. Although, after that, the instances keep scaling up to its maximum instances of 10 at 01:15.
- After that period, the number of instances keep fluctuating between 9 and 10 every 5 minutes. This is possibly because both upscale and downscale alarm was triggered at the same time.
- Finally, after the locust was stopped, the number of instances stated to go down to 2 in about 30-40 minutes. This is because the upscale alarm stopped while the downscale alarm continued.



- Another fascinating observation is the statistics from the locust itself. In the beginning of the run, the requests encountered 10 failures/second (2%). After the instances scaled up to 10, failures still fluctuated but at lower rate of 3-5 failures/second (<=1%). This means increasing number of instances actually improved the performance.



- Rather than the average time in CloudWatch, the response time graph in Locust used 50th and 95th percentile, which captured the occasional high response times of 10s. This is probably a timeout, though.

Load test the annotator farm

I created a file named 'ann_load.py' which was used to simulate load test request messages to annotator. It sends dummy loads at defined interval to the request SNS. The purpose is to trigger the upscale alarm from the CloudWatch monitoring.

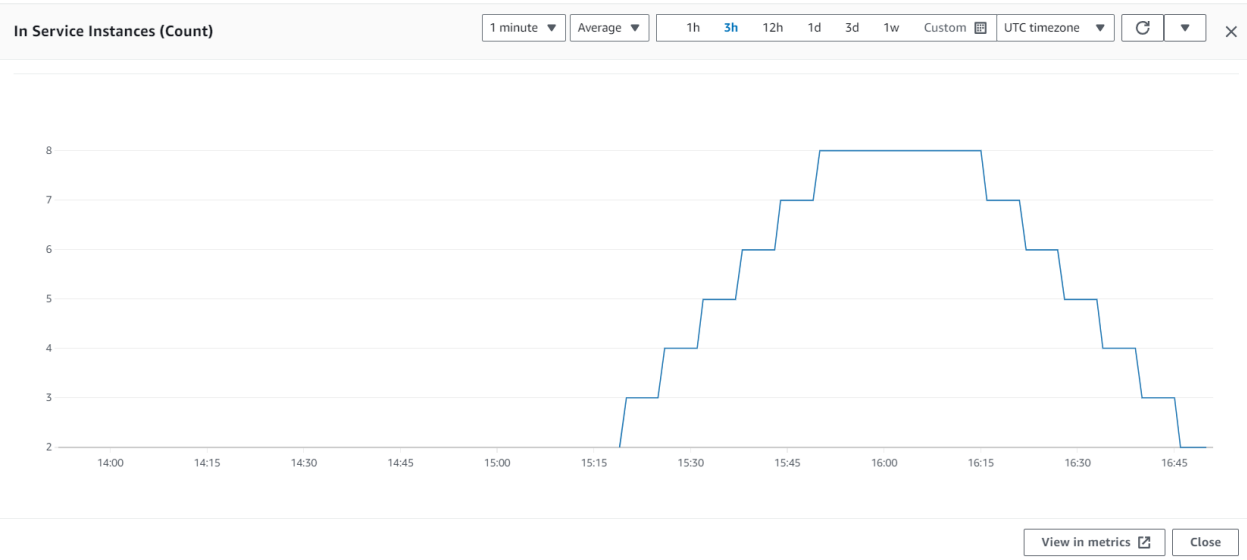
These are my scenario for the test:

* I increased the messages rate to 0.5-1 message per second, rising the messages rate shown in the CloudWatch sharply, reaching 300 messages/10 minutes.

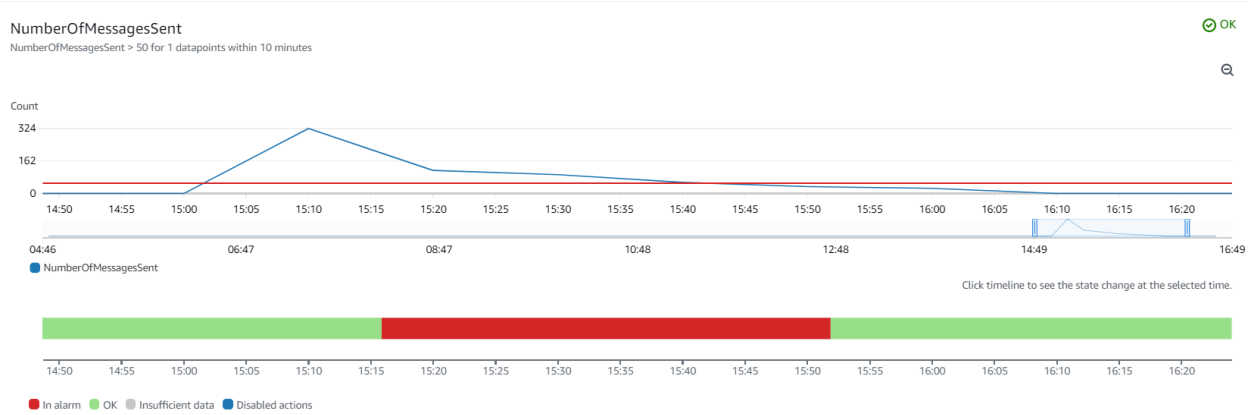
* After that, I tried to set my messages rate so that the number is lower than 50 but higher than 5.

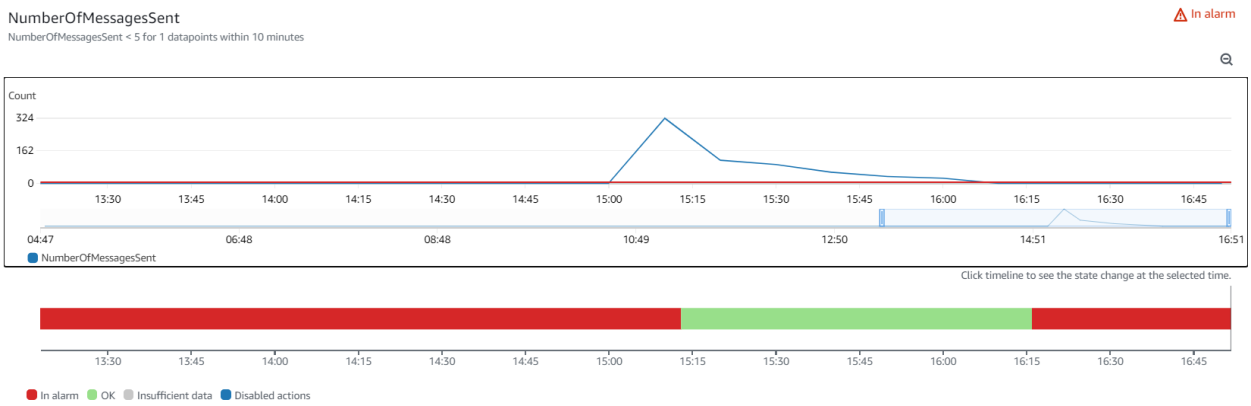
* Finally, I stopped the load test, lowering the number to 0.

These are several observations I found during the run:



* The behavior of the instances reflected the scenario consistently, rising to 8, staying there, then going down to 2.





* Interestingly, the CloudWatch alarm didn't trigger until 10 minutes after the increase, possibly because the alarm collected data in 10 minutes time frame.