

Scale-Out

When observing the Auto Scaling Group during the load test, we can see insights into how the system reacts to increased demand:

- As the load increases, the ASG responds by adding instances. This behavior is expected as the system detects higher request counts and attempts to maintain performance. The incremental addition of instances, as seen in the stepwise increase, shows a controlled scaling approach to avoid overwhelming the system with too many new instances at once.
- The xiat_HighRequestCount policy, which triggers when requests exceed 200 per minute, proves effective in maintaining service levels. This policy ensures that the system scales out before performance degrades, showing proactive management of resources. The fact that instances are added incrementally highlights the system's ability to handle increasing load smoothly, without sudden spikes in resource allocation, which can lead to instability.
- The elastic nature of the cloud infrastructure is evident as the ASG scales out to match the demand. This demonstrates the flexibility and scalability of cloud resources, ensuring that the application can handle varying loads efficiently.

Scale-In

Upon stopping the load test, we gain insights into the system's ability to scale in and reduce resources:

- As the load decreases, the system scales in by reducing the number of instances. This is driven by the modified xiat_lowRequestCount_Test policy since the request response time is hard to lower than 0.01s as project guided, which triggers when request counts fall below 20 per minute. This behavior shows a focus on cost-efficiency, as unnecessary instances are terminated to save resources and costs.
- The stepwise reduction in instances reflects a cautious approach to scaling in, ensuring that performance is not compromised by removing too many instances too

quickly. This gradual reduction also prevents potential disruptions that could arise from sudden capacity changes, maintaining a stable service level.


- The system's ability to add and remove instances based on real-time demand highlights the balance between maintaining performance and optimizing costs. By scaling out during high load and scaling in during low load, the system ensures that resources are used efficiently, demonstrating the effectiveness of the auto-scaling policies in managing cloud resources dynamically.

Conclusion

The observations from the load test provide valuable insights into the behavior of the Auto Scaling Group under varying loads. The ability to scale out in response to increased demand and scale in when the demand drops ensures that the system remains performant and cost-effective. These insights highlight the importance of well-configured auto-scaling policies in leveraging the full benefits of cloud elasticity.

Screenshots

Test under load using Locust



LOCUST

HOST

<https://xiat.mpcs-cc.com>

STATUS

STOPPED

[New test](#)

RPS

149.7

FAILURES

0%

Statistics

Charts

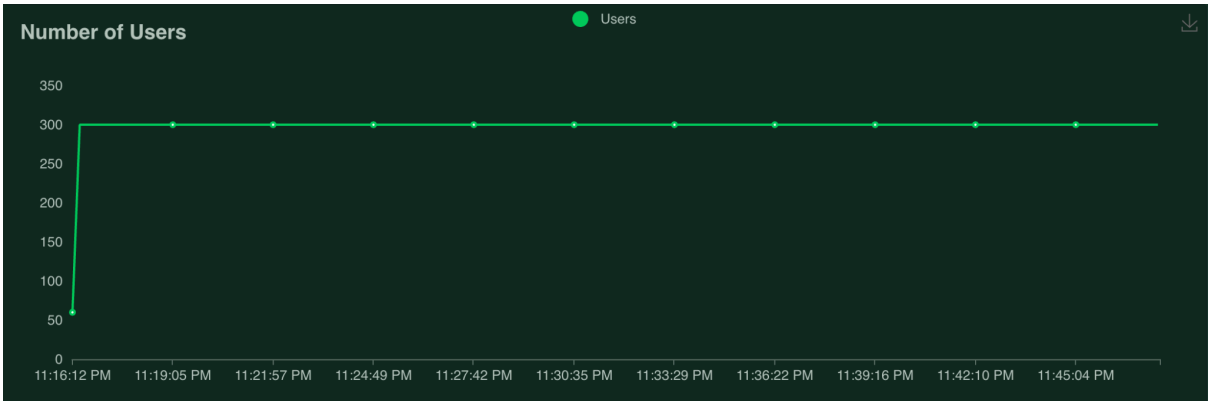
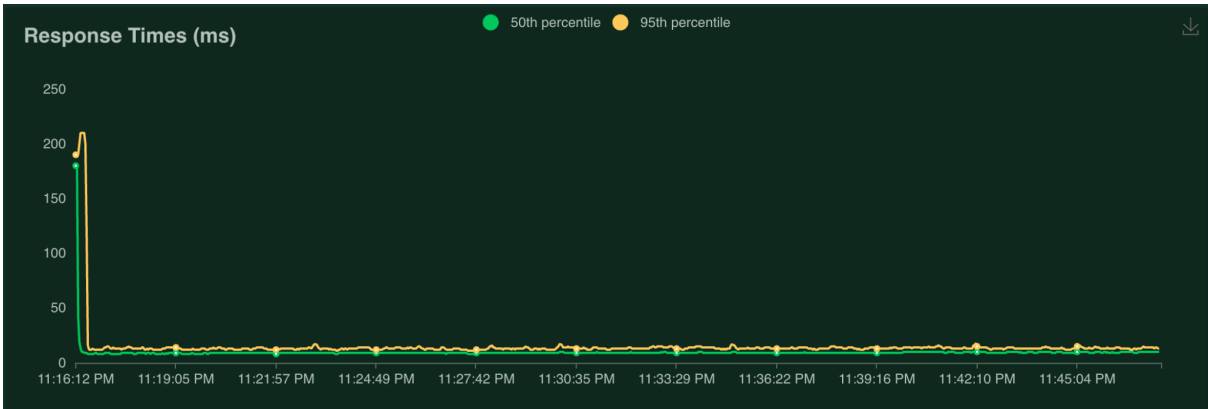
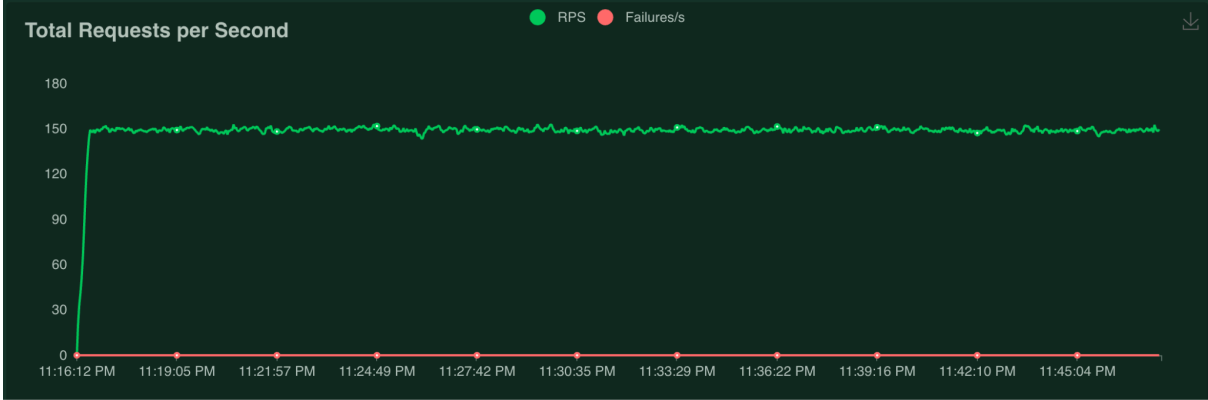
Failures

Exceptions

Current ratio

Download Data

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	/	279797	0	9	12	19	10	6	247	8342	149.7	0
Aggregated		279797	0	9	12	19	10	6	247	8342	149.7	0



In service instances changes using CloudWatch

