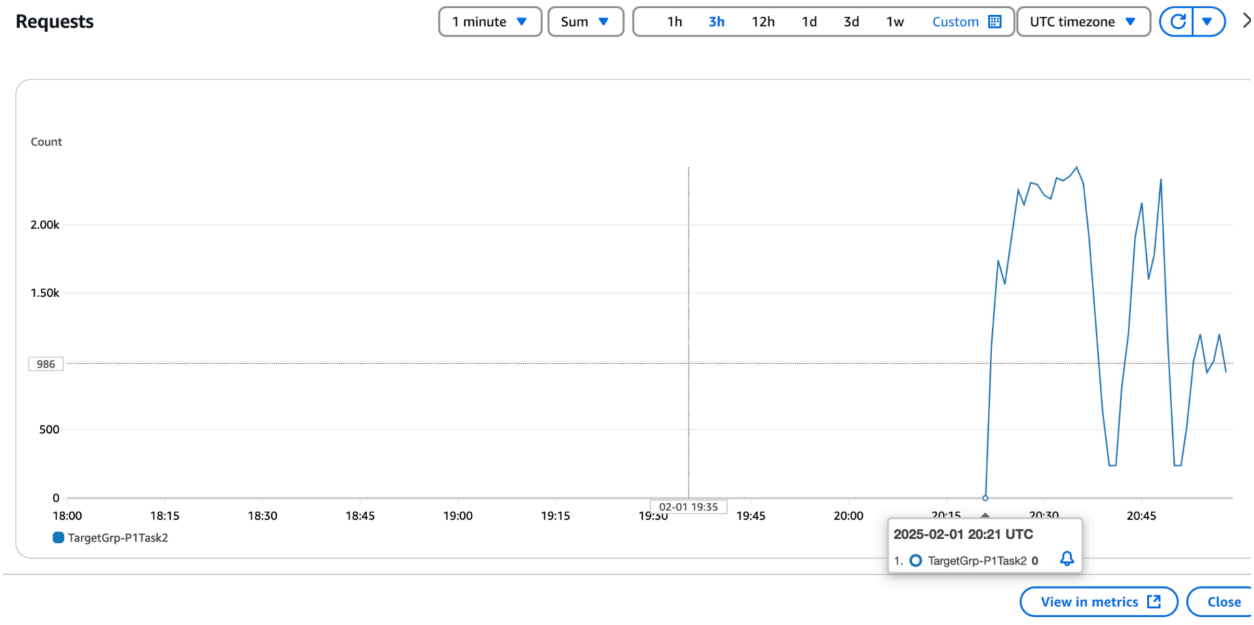
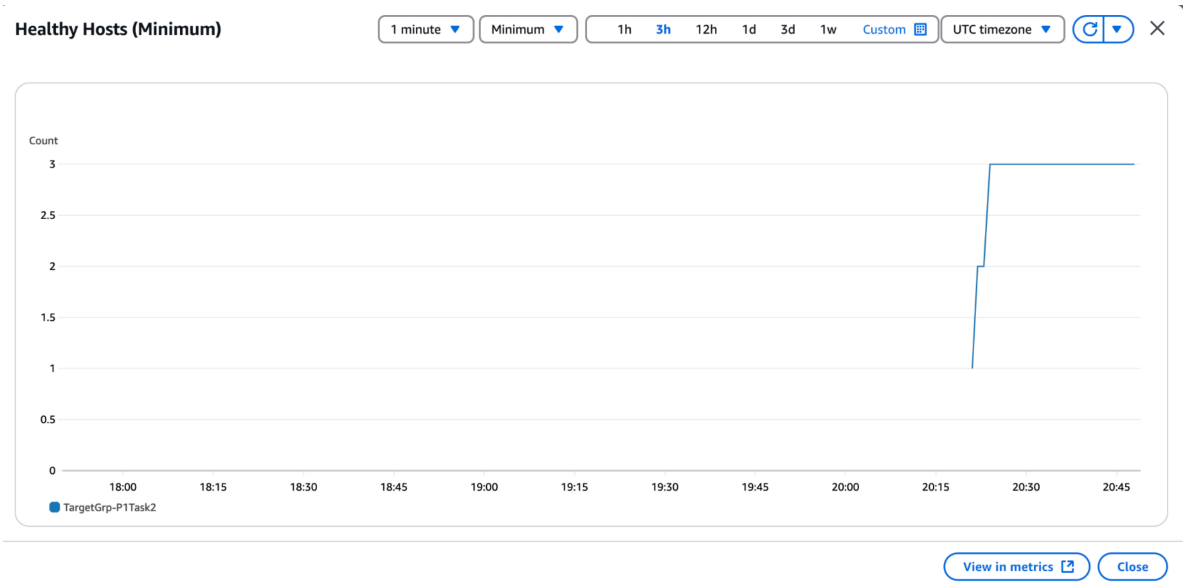
1. What traffic patterns did you see in the load sent to the ELB by the load generator? How did you actually figure out the load pattern? (Please provide appropriate screenshots from the AWS dashboard wherever necessary) (3 pts)

   I observed it over multiple runs of the test. The initial burst of requests per second reaches around 30 which also drops to 0 due scaling issues or unavailability of instances during health checks due to which I had to tweak the health check parameter and test if the loads have the same pattern. Around 18$^{th}$ to 22$^{nd}$ minute the load increased suddenly to hit the maximum RPS. The scaling in or scaling down period was crucial as the as the peak load subsided, but the instances were still up which increased the instance hours. I have monitored the Request Count, Health Hosts, Target Response Time with the CloudWatch alarms' graph which gave an idea of when the RPS varied, and the instances were scaled out or scaled in as per the scaling policy. I also analyzed the logs of the warm-up and auto-scaling test on multiple configurations from which I was able to correlate the values with the data.

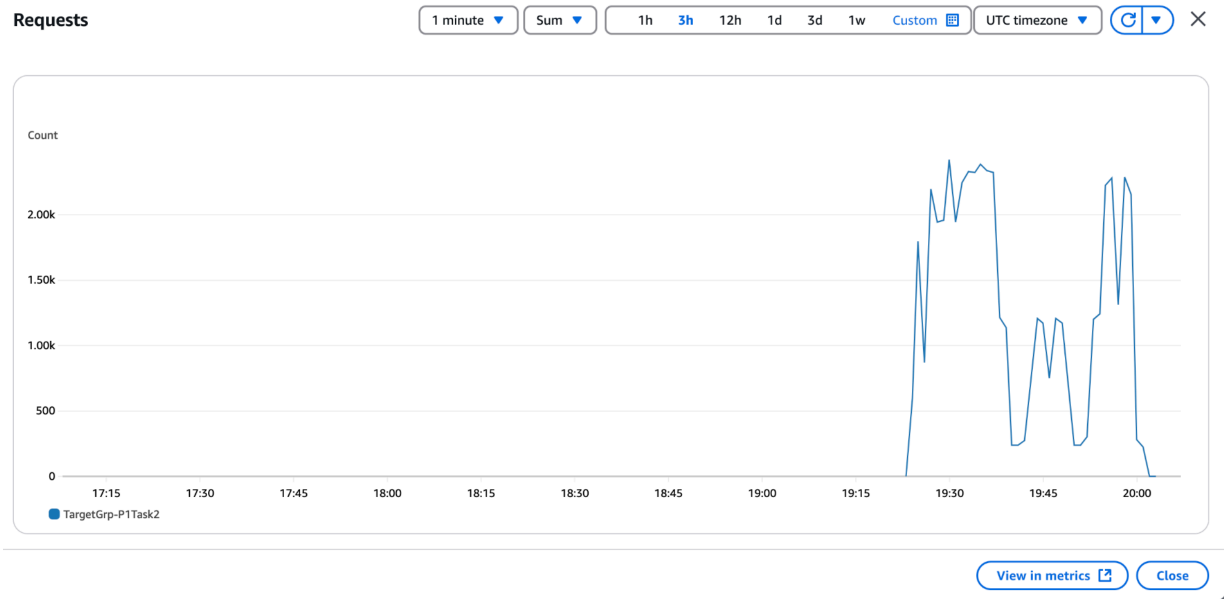
   I also realized that at the end of every test there was a pattern number signifies the pattern of load displayed which really helped me find a correlation and fine tune the parameters.




Shows the requests for the target group



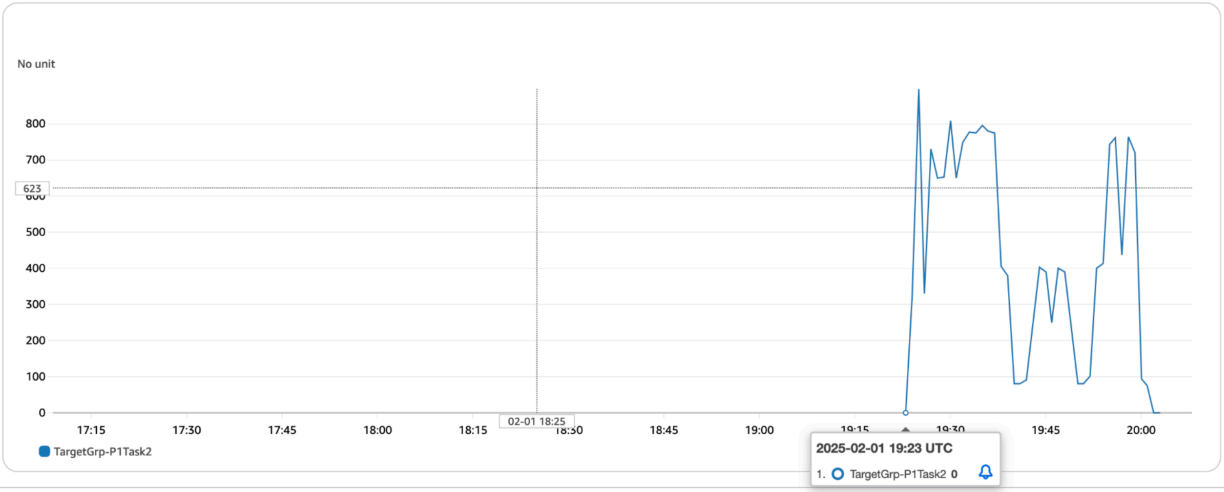

Healthy h




Spawning as per the loads

**Request Count Per Target**

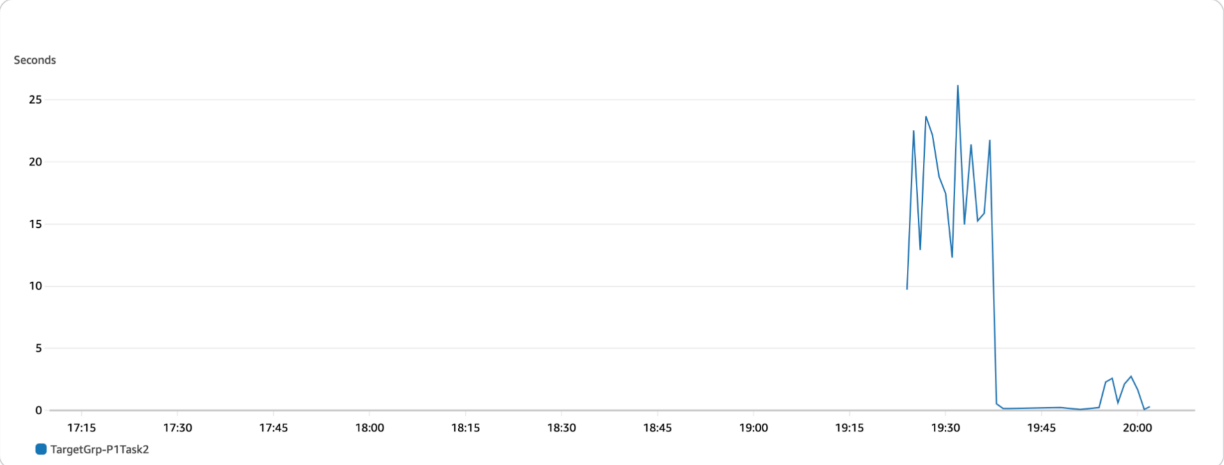
1 minute | Sum | 1h | 3h | 12h | 1d | 3d | 1w | Custom | UTC timezone



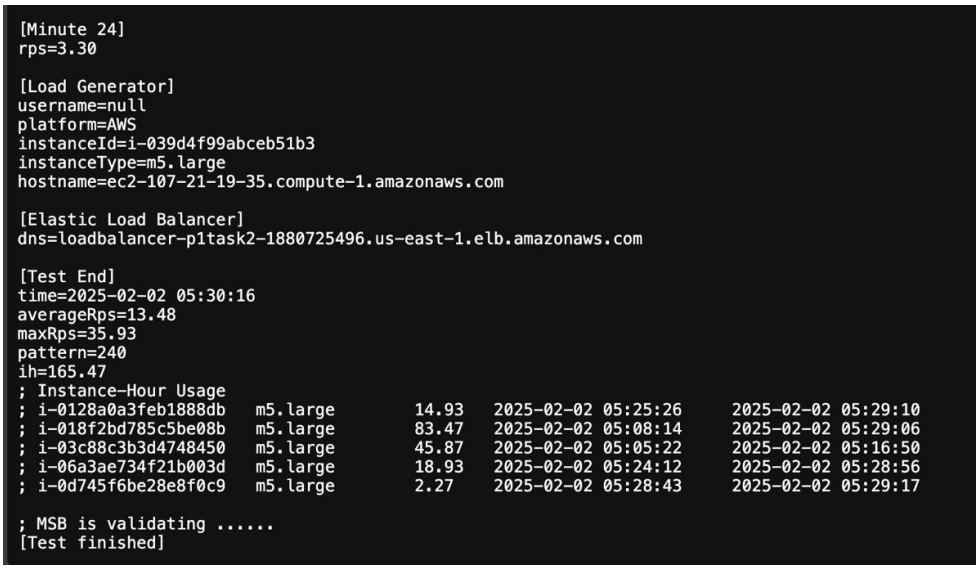

View in metrics | Close

**Target Response Time**

1 minute | Average | 1h | 3h | 12h | 1d | 3d | 1w | Custom | UTC timezone




View in metrics | Close




2. How did you model the policies for the Auto scaling Group in response to the insights gained in the above question? (2 pts)

On the basis of the traffic patterns that I observed, I had to optimized the Auto Scaling policies as:

I initially tried the test with 3 instances which sometimes had a lot of RPS during peak load, but the scale out increment during high CPU utilization was slow. I set the increment to 1 for gradual increase to maintain instance hours and handle sudden spikes. I tried scaling in quickly, even with -2 instances but I realized its better to gradually decrease it. I also had cooldown periods starting from 20 seconds to 60 seconds to monitor how quick reactions are needs. I also worked around CPU threshold to trigger scale-out at 80% and scale in at a lower threshold this helped me initialized new resources quickly and reduce them as needed while maintaining the instance hours for which I optimized the scaling policies to minimize instances during peak hours. In general, I realized that longer cool-down periods led to delays in scaling actions. Higher thresholds caused delayed scaling whereas having more instances also increased the RPS. I kept the cool-down period as default of 45 and 60 for scale in and scale out for faster response to variations in the load. I optimized it to achieve the max RPS, average RPS and instance hours with quick scale in and scale out policies.