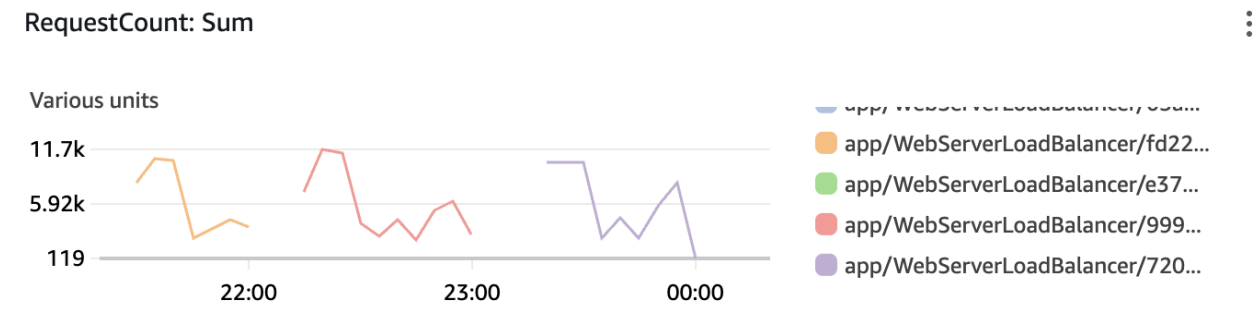
(a)

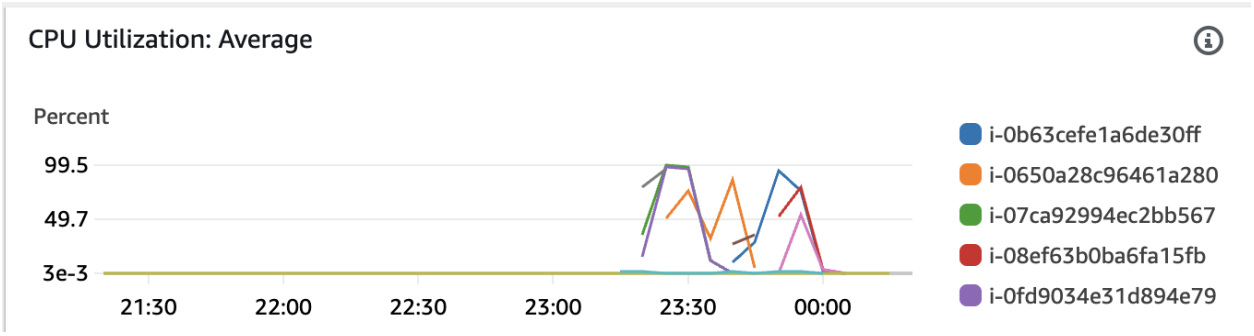


The load pattern observed on the Elastic Load Balancer (ELB) from the load generator exhibits an initial spike in requests, indicating a surge of traffic at the beginning of the test. This is followed by a significant drop, suggesting the conclusion of an intensive test phase. Subsequently, there is a smaller increase, which could represent a simulation of normal operational load, before dropping off again to a lower level of requests.

This pattern was deduced by analyzing the Request Count metrics from the ELB on the AWS CloudWatch dashboard. The metrics show the number of requests over time, with the peaks and troughs indicating changes in the load. The similarity in the pattern between the second and third intervals reinforces the consistency of the load testing process.

(b)

I set the Scale Up policy to trigger when the CPU utilization exceeds 60%, indicating high demand, and the Scale Down policy to trigger when CPU utilization falls below 40%, indicating a decrease in demand (At first, the down policy was set lower like 20% but it turns out that it leads to higher instance hours and kind of waste). Based on the CPU utilization graph which indicates fluctuating demand, my Auto Scaling Group policies were modeled to dynamically adjust the number of instances in response to changes in CPU load.



The CPU utilization insights from the load generator tests, which show peaks and troughs, informed these threshold settings. The scale-up policy uses a cooldown period of 30 seconds to prevent rapid, frequent scaling actions that could lead to instability, while the scale-down policy has a shorter cooldown period of 15 seconds to reduce costs when demand is low rapidly. These cooldown settings help to balance cost against the need for responsiveness and availability.