Stress Testing and Bottleneck Analysis

Overview

In this experiment, I deployed a single AWS ECS Fargate task (0.25 vCPU, 512 MB memory) running the product search API. The service stores 100,000 products in memory and processes exactly 100 product checks per request. I conducted six stress tests using Locust with progressively increasing user counts and request rates to determine how the system behaves under load and to identify its performance bottleneck.

The goal of this stress testing was to determine:

1. What happens as load increases
2. Whether performance issues are CPU-bound or memory-bound
3. Whether the problem would be solved through optimization or scaling
4. How CloudWatch metrics support scaling decisions

Stress Test Summary

I conducted six progressively heavier stress tests:

Test	Users	Request Rate	Duration (mins)
1	500	50	3
2	700	70	3
3	1000	50	3
4	1000	70	3
5	1000	100	5
6	3000	200	7

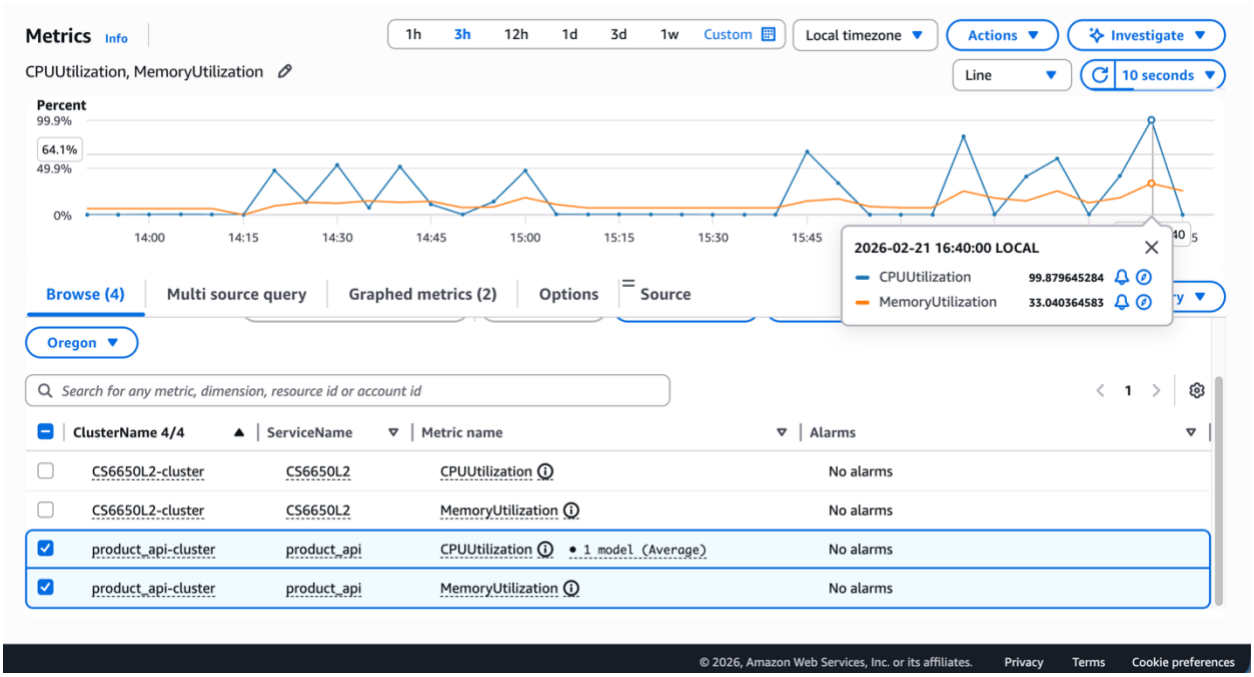
What Happened as Load Increased

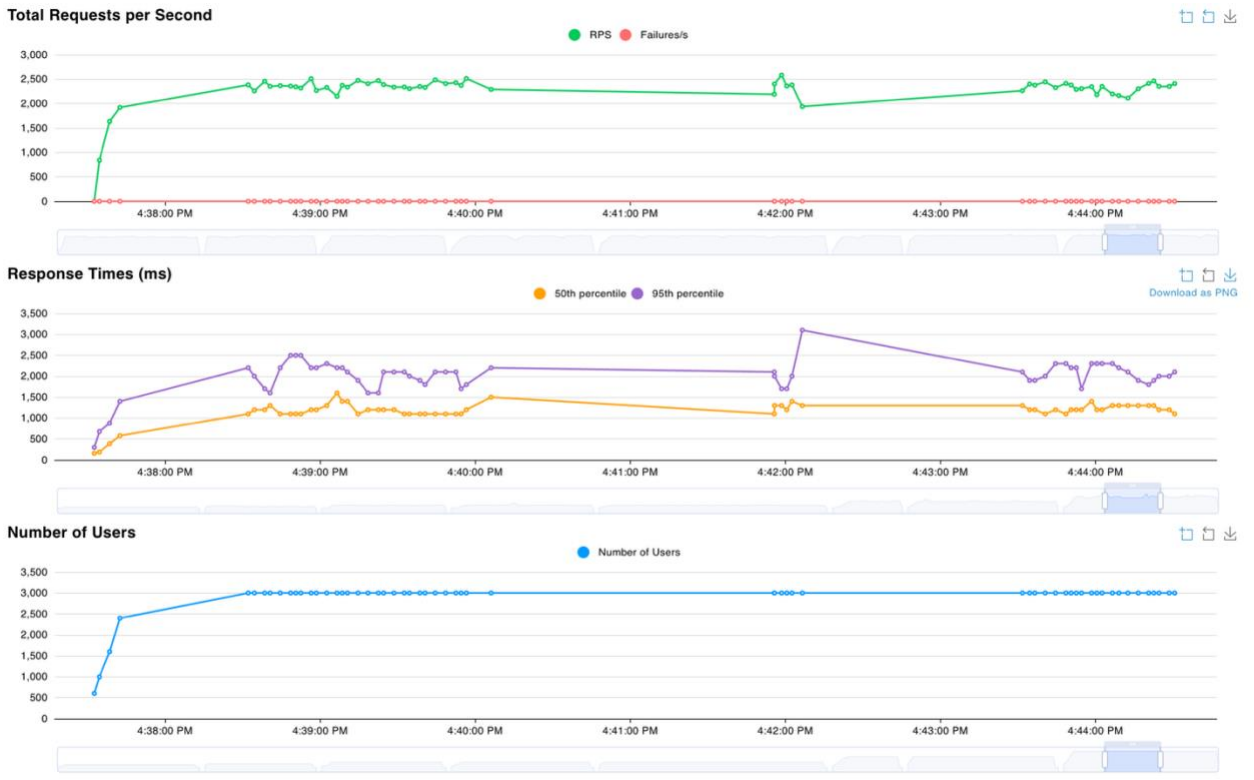
As load increased, the following behavior was observed:

- CPU utilization increased steadily across tests.
- Memory utilization remained relatively low and stable.
- Throughput (Requests Per Second) increased initially but eventually plateaued.
- Response times (especially p95 and p99) increased significantly under heavy load.

In the final stress test (3000 users at 200 req/s):

- CPU utilization peaked at approximately **99.87%**
- Memory utilization remained around **33%**
- Throughput plateaued at approximately **2,200–2,300 RPS**
- Average latency increased from ~222 ms (Test 1) to ~1247 ms (Test 6)
- Tail latency (p95/p99) exceeded 2–2.7 seconds





This behavior clearly indicates resource saturation.

Identifying the Bottleneck

The evidence strongly shows that the system became **CPU-bound**:

1. CPU utilization reached nearly 100%.
2. Memory utilization remained well below its limit (never exceeded ~33% of 512 MB).
3. Increasing load further did not increase throughput.
4. Latency increased significantly while RPS remained flat.

This pattern demonstrates a classic CPU saturation scenario:

- When CPU reached ~100%, the system could not process requests any faster.
- Additional load caused requests to queue, increasing response times.
- Memory was not the limiting factor because it remained stable and far from exhaustion.

Therefore, the bottleneck is clearly CPU.

Optimization vs Scaling

Based on CloudWatch metrics and Locust results, this problem is not memory-bound and does not require memory optimization.

The workload performs fixed computation per request (100 product checks), meaning:

- Each request consumes predictable CPU cycles.
- The system runs on a single 0.25 vCPU Fargate task.
- Once that CPU core is saturated, no additional performance can be achieved.

This suggests two viable scaling solutions:

Vertical Scaling

Increase CPU allocation:

- 0.25 vCPU → 0.5 vCPU
- Expected result: approximately higher throughput ceiling

Horizontal Scaling

Increase the number of tasks:

- 1 task → 2 tasks
- Throughput should approximately double
- CPU utilization per task should decrease under the same load

Because memory utilization remained low, increasing memory would not meaningfully improve performance.

CloudWatch metrics directly support this scaling decision by showing:

- CPU saturation near 100%
- Stable and low memory utilization
- Clear correlation between high CPU and latency increase\

This demonstrates correct use of CloudWatch metrics to make infrastructure scaling decisions.

Creative Stress Testing Approaches

Beyond the required baseline and breaking-point tests, I performed multiple progressively heavier stress tests, including:

- Increasing concurrent users while holding request rate steady
- Increasing request rate while holding user count steady
- Running extended-duration tests (5–7 minutes) to observe sustained load effects
- Testing extreme load conditions (3000 users, 200 req/s)

These variations allowed me to identify the exact load level where CPU utilization approached saturation and where throughput stopped scaling.

This systematic approach provided strong evidence of the system's performance ceiling.

Final Conclusion

Through six progressively heavier stress tests, I determined that the deployed ECS Fargate service becomes CPU-bound under high load. When CPU utilization reached approximately 99.87%, throughput plateaued around 2,200–2,300 RPS and latency increased significantly.

Memory utilization remained below 40%, confirming that memory is not the limiting resource.

The bottleneck can therefore be addressed through vertical or horizontal scaling rather than memory optimization.

This experiment demonstrates how load testing combined with CloudWatch metrics can be used to make informed infrastructure scaling decisions.