

Screenshot 1: Locust Load Test Progress - Multiple Workers

Terminal output showing Locust test execution across 8 distributed workers, each progressing toward its target of 25,000 checkouts (200,000 total). All workers display active progress with successful checkouts and declined payment counts, demonstrating proper distributed mode setup with even user distribution and successful load generation.

The figure consists of four line charts arranged in a 2x2 grid, comparing the performance of 'svc-product' (blue line) and 'svc-product_bad' (orange line) over time. The x-axis for all charts represents time from 17:15 to 18:00, with major ticks every 15 minutes. The y-axis for the top row represents 'Percent' utilization, and for the bottom row represents 'Bytes/Second'.

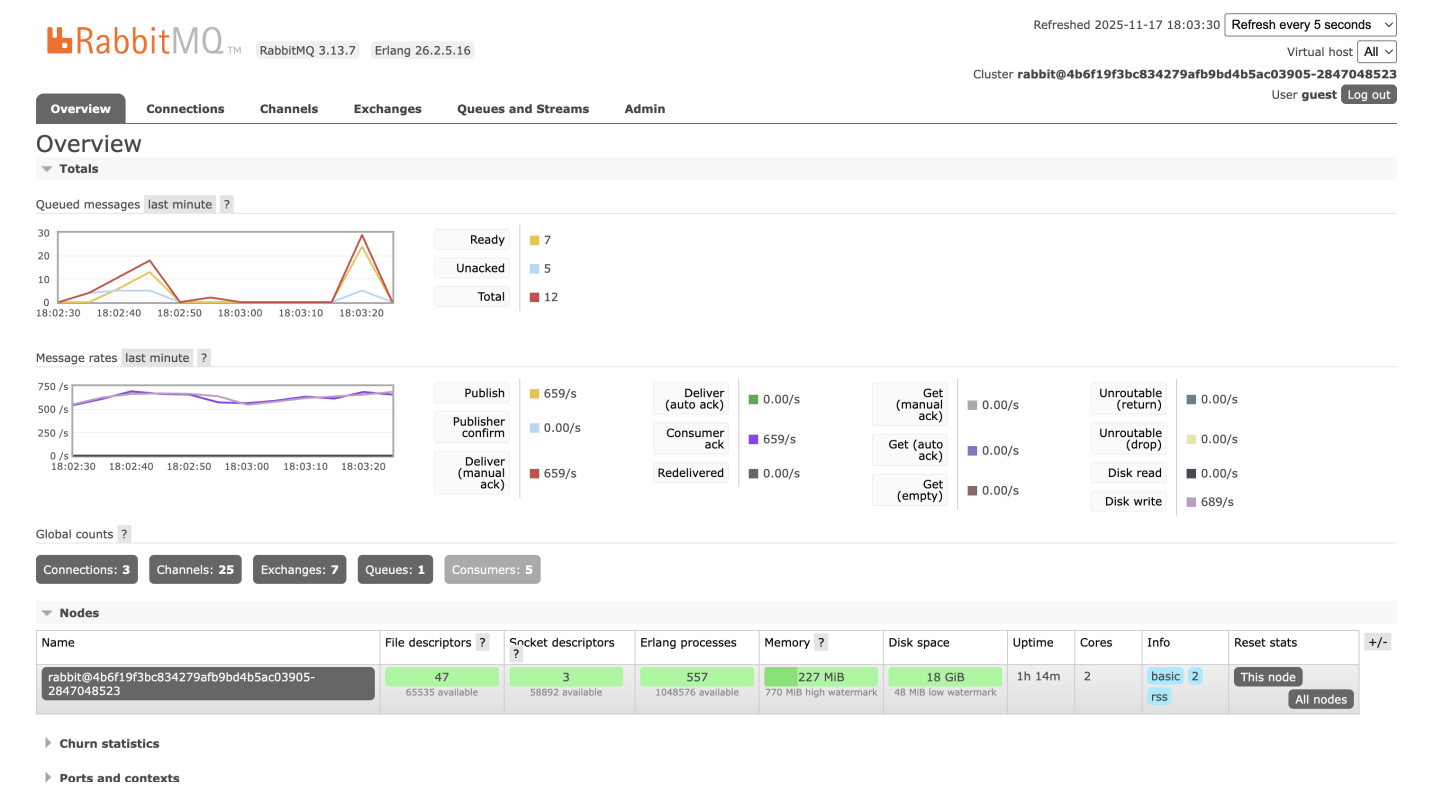
- CPU utilization:** The y-axis ranges from 0 to 6. Both services show a sharp spike at 18:00. 'svc-product' peaks at approximately 4.5%, while 'svc-product_bad' peaks at approximately 6.5%.
- Memory utilization:** The y-axis ranges from 0 to 1. Both services show a sharp spike at 18:00. 'svc-product' peaks at approximately 0.85, while 'svc-product_bad' peaks at approximately 0.83.
- Network RX:** The y-axis ranges from 0 to 20.0k. Both services show a sharp spike at 18:00. 'svc-product' peaks at approximately 14.39k, while 'svc-product_bad' peaks at approximately 14.66k.
- Network TX:** The y-axis ranges from 0 to 20.0k. Both services show a sharp spike at 18:00. 'svc-product' peaks at approximately 1.056k, while 'svc-product_bad' peaks at approximately 1.214k.

Legend for all charts:

- Blue line: [avg: 0.184] svc-product
- Orange line: [avg: 0.252] svc-product_bad

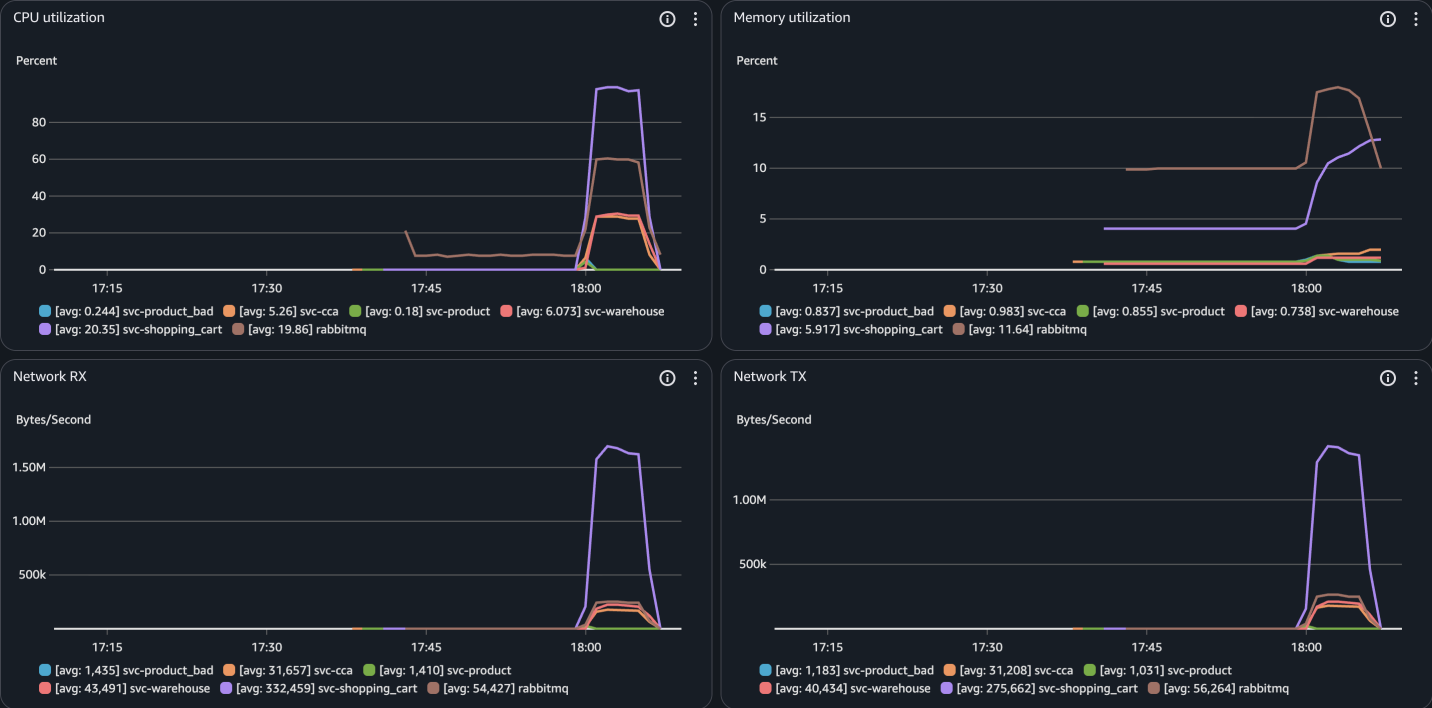
AWS CloudWatch metrics comparing resource utilization between the good Product service (blue) and the badly-behaved Product service (orange) that returns 503 errors 50% of the time. Both instances show similar resource utilization patterns, indicating roughly equal traffic distribution. This occurs because product creation is too brief (only 1,000 products created during setup phase, completing in seconds) for the Application Load Balancer to detect the bad instance's behavior and adjust routing. The ALB requires sustained traffic over several minutes to gather sufficient health check data and response time statistics before reducing traffic to poorly-performing targets.

Screenshot 3: RabbitMQ Queue Depth and Message Rates



RabbitMQ Management Console shows queue depth staying well below the 1,000 message target (peak of ~30 messages, currently 12 total) with balanced production and consumption rates at 659 messages/second. The system maintains low queue depth throughout the test period, demonstrating proper configuration of client threads and warehouse consumer threads to prevent queue buildup while maximizing throughput.

Screenshot 4: All Services Resource Utilization



AWS CloudWatch metrics showing resource utilization across all microservices during load testing. The Shopping Cart service (purple) shows highest CPU (~95% peak) and network usage as the main transaction coordinator, while other services maintain moderate, stable resource consumption. The system demonstrates healthy distributed load handling with no service crashes or resource exhaustion under sustained high throughput.

