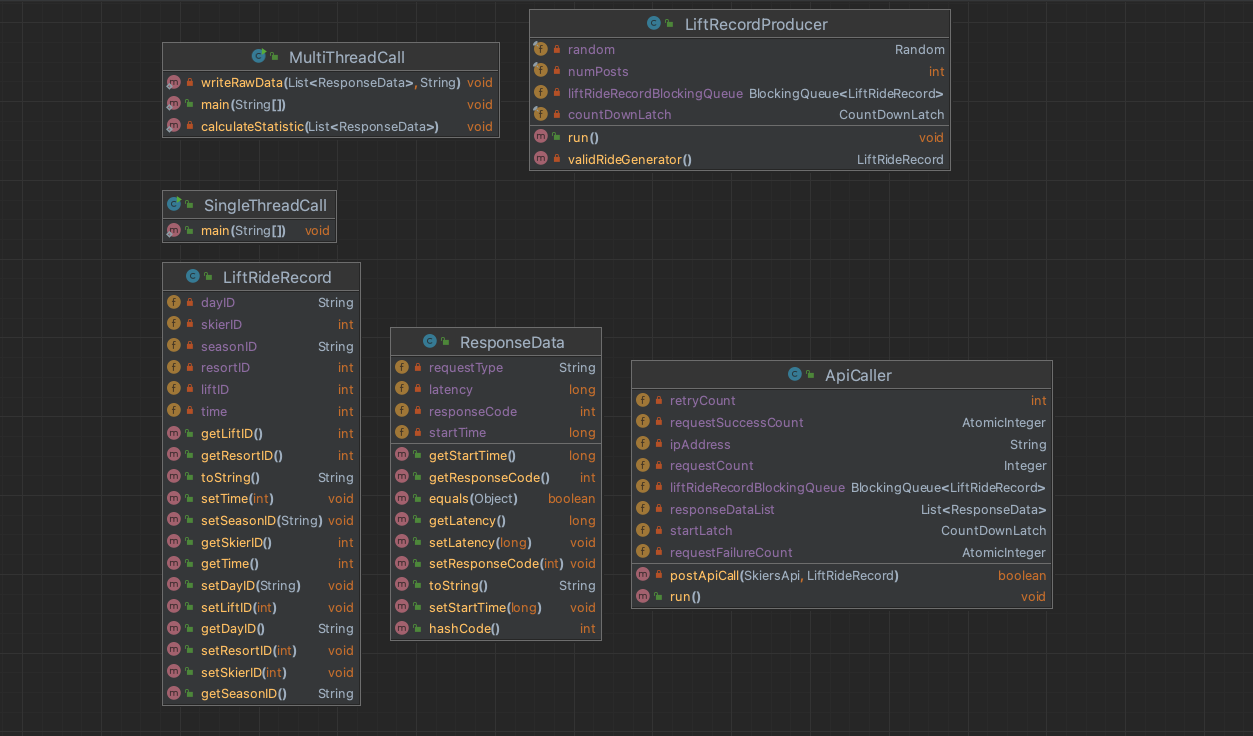
Assignment 2 Zegui Jiang

Client 2 – Remaining same with Assignment 1

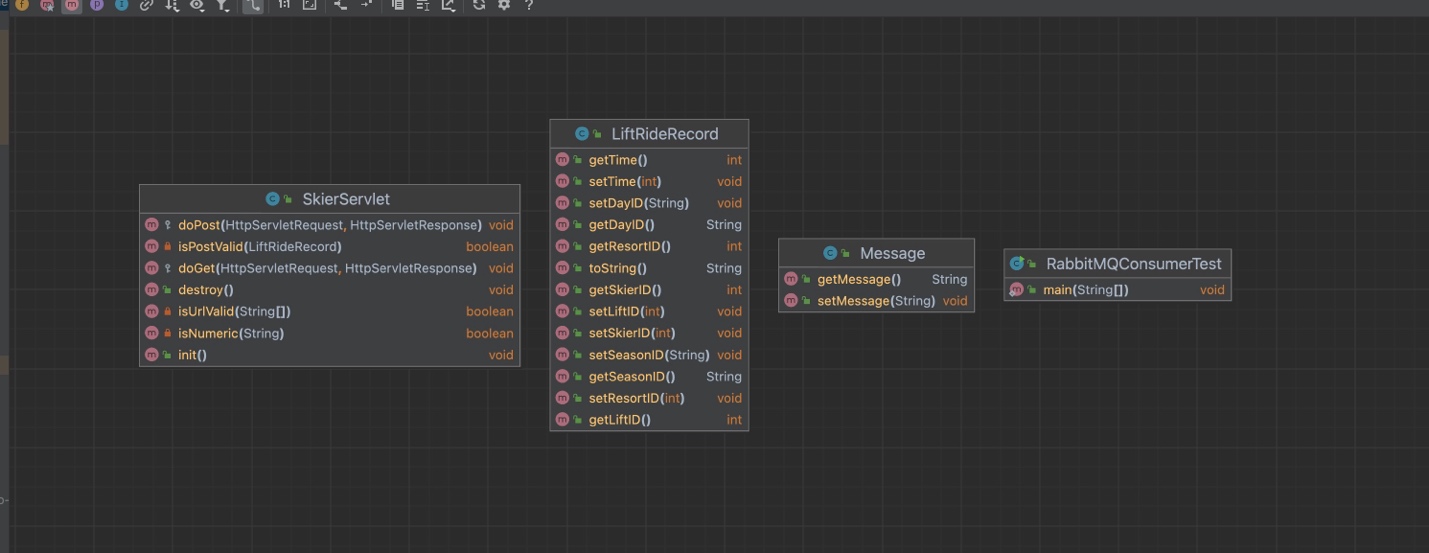
Client 2 extends the functionality of client 1 by enhancing the MultiThreadCall class to include capabilities for writing raw data to a CSV file and calculating statistical metrics. To fulfill these additional requirements, a new class named ResponseData is introduced to encapsulate the response records.

The MultiThreadCall class in client 2 would have these new responsibilities:

**Writing Response Records**: After each API call, the response details are captured in an instance of the ResponseData class. This object includes all relevant information that needs to be logged, such as timestamps, response status, and any payload data returned by the API.

**Calculating Statistics**: The class is responsible for computing various performance metrics based on the response data collected. These statistics might include the total number of requests, success rate, failure rate, average response time, and other relevant performance indicators.

LIftServer



The server upgraded the functionality for processing messages. Firstly, it connects to RabbitMQ, then it sends the received message queue to RabbitMQ.

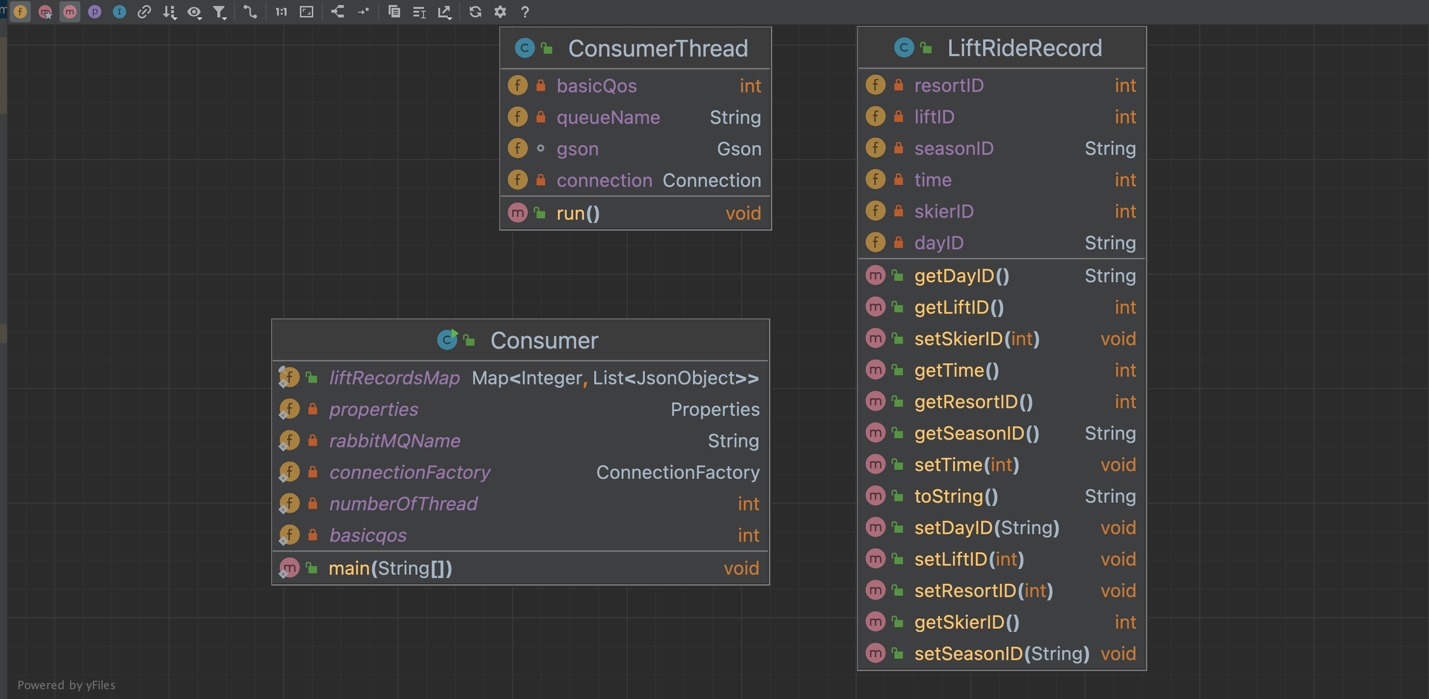
Init(): Reads and configures settings from the **rabbitmq.conf** configuration file found in resources. It extracts the RabbitMQ queue name and the number of channels from the properties file. All channels are stored in a channel pool, which is a Linked Blocking Queue.

Destroy(): Closes all open resources when the process is terminated.

IsPostValid and IsUrlValid: These functions are likely responsible for validating the incoming request's POST data and the URL, respectively.

doPost: Handles incoming POST requests by receiving a message and forwarding it to RabbitMQ.

RabbitMQ Consumer



ConsumerThread:

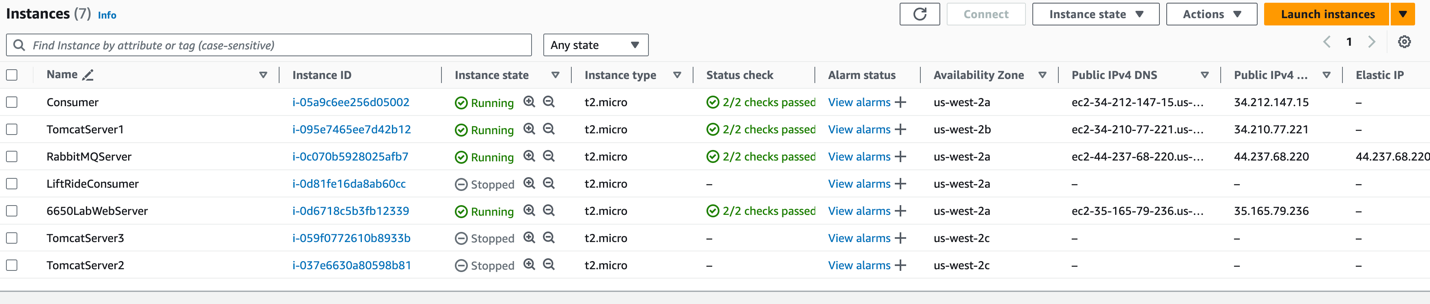
- It creates a new channel Thread on the provided connection.

* It declares a queue with the given queueName without any extra arguments
* It sets the basicQos for the channel, which controls how many messages the server will deliver to the consumer before acknowledgments are received.
* Attempts to parse and consume the message into a JsonObject and retrieves a SkierID, and store them to liftRecordsMap

Consumer:

* Loading Configuration: load a RabbitMQ configuration file (rabbitmq.conf). including settings such as host, port, username, password, queue name, the default number of threads for consumers numberOfThread, and the default basicQos
* **Setting Up Connection Factory and Starting Consumer Threads:** An ExecutorService with a fixed thread pool size, based on numberOfThread, is created to manage consumer threads. For each thread, a new ConsumerThread is executed. This thread is responsible for handling messages from the specified RabbitMQ queue (rabbitMQName) with the given basicQoS.

**With Single Server**

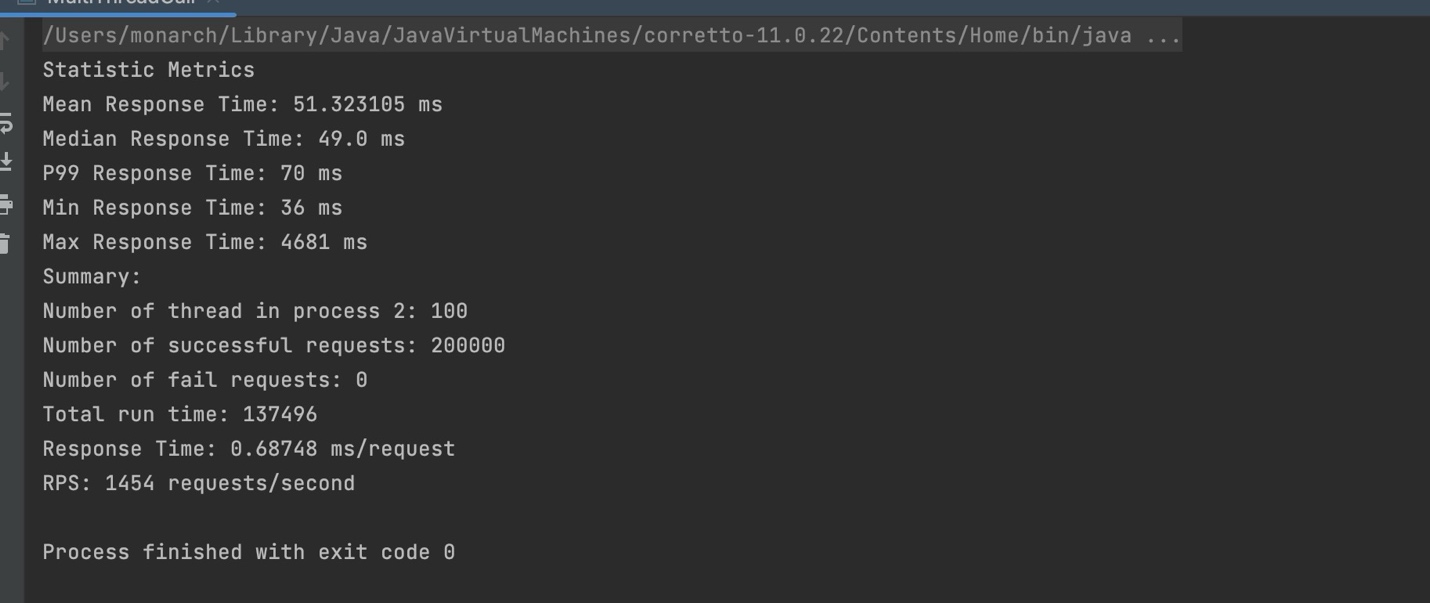
**Basic Structure, and all run in ec2:  
Client -> Tomcat -> RabbitMq -> Consumer**Following data points includes Overall throughput (should be greater than no load balancer), short queue sizes and profile

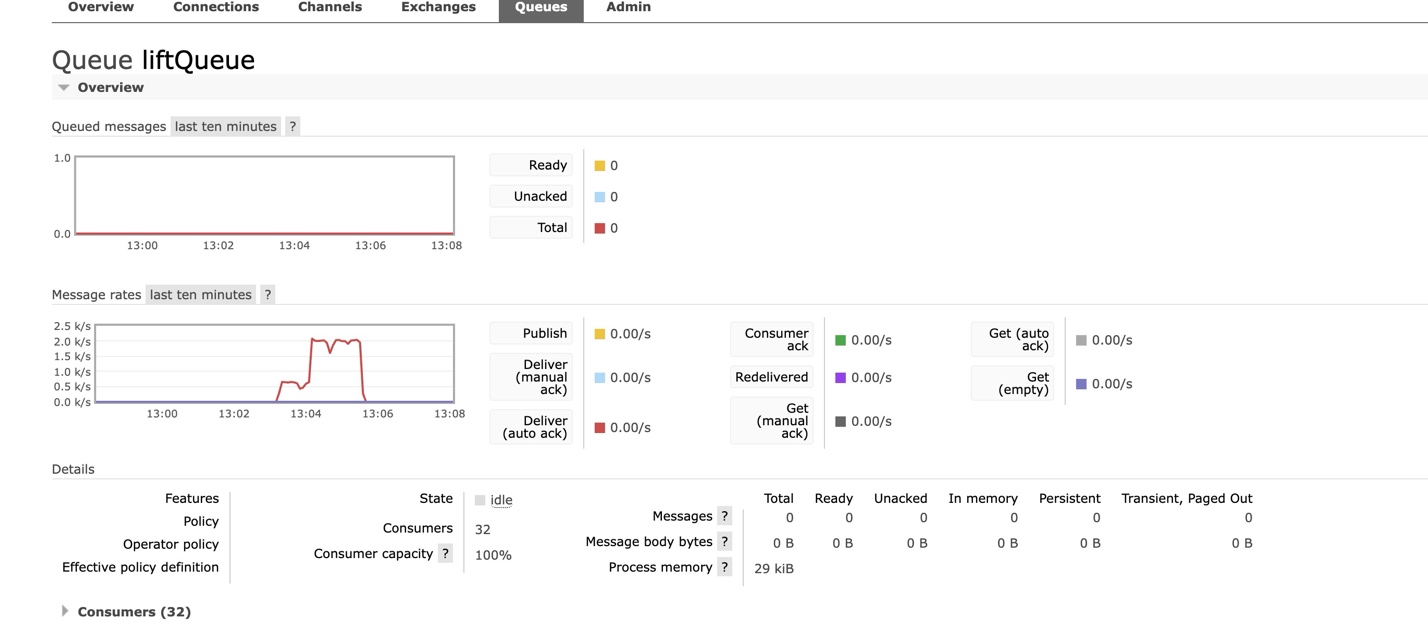
Basicqos and Thread in Consumer has test with 1 and 24 could have stable consuming performance.

**Number of Thread in Client: 100**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

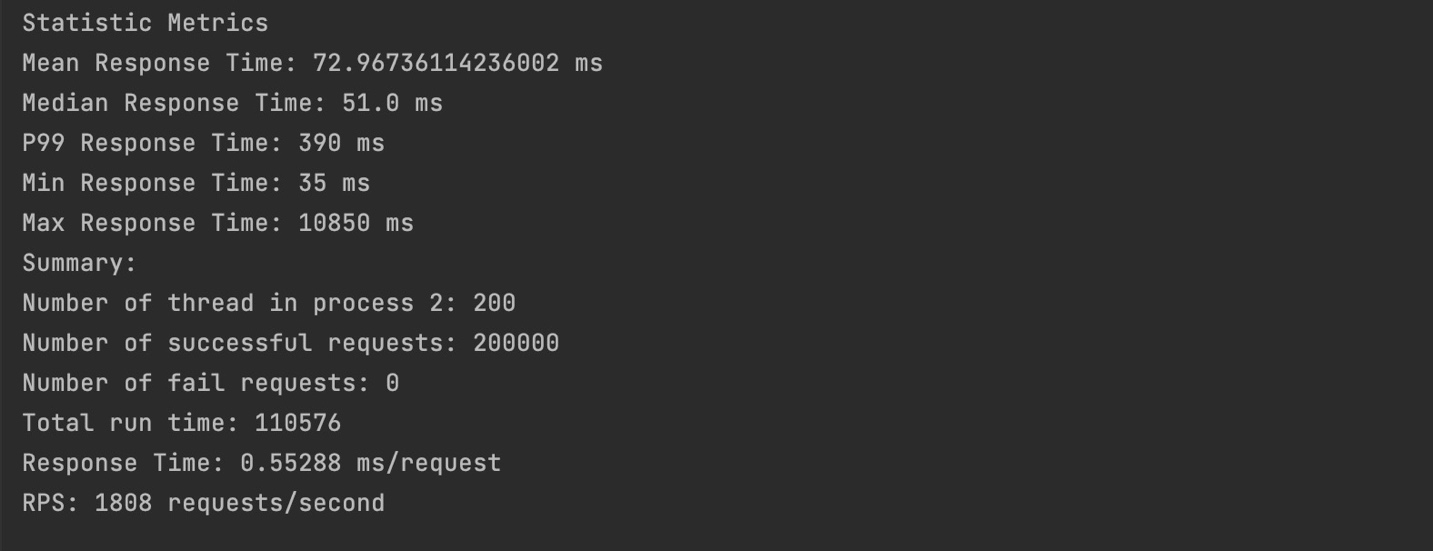
****

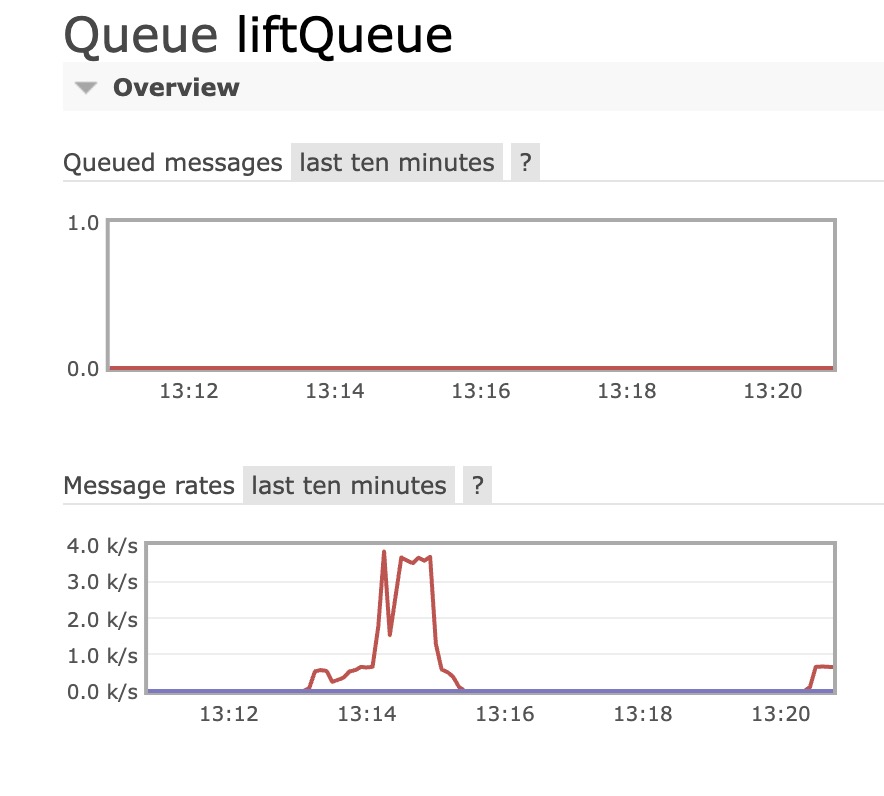
****

**Number of Thread in Client: 200**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

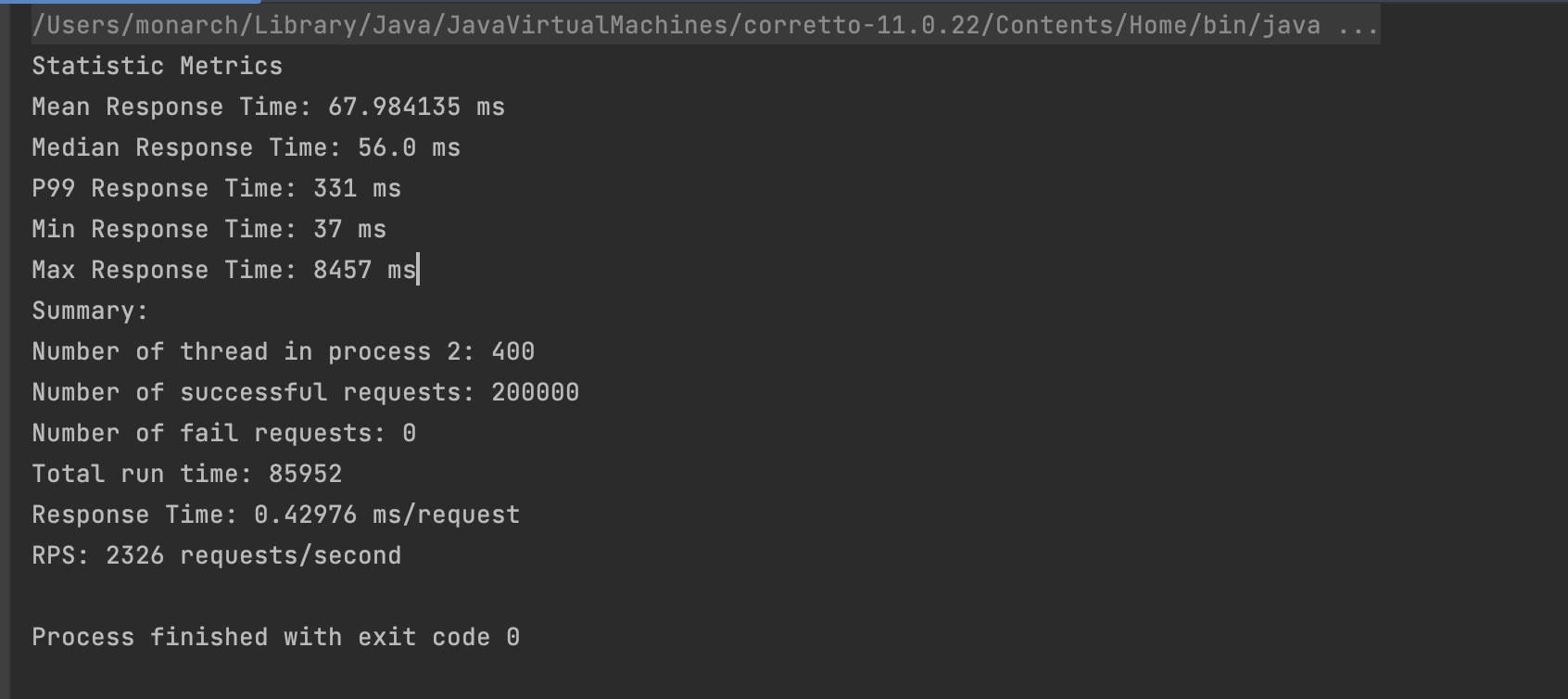
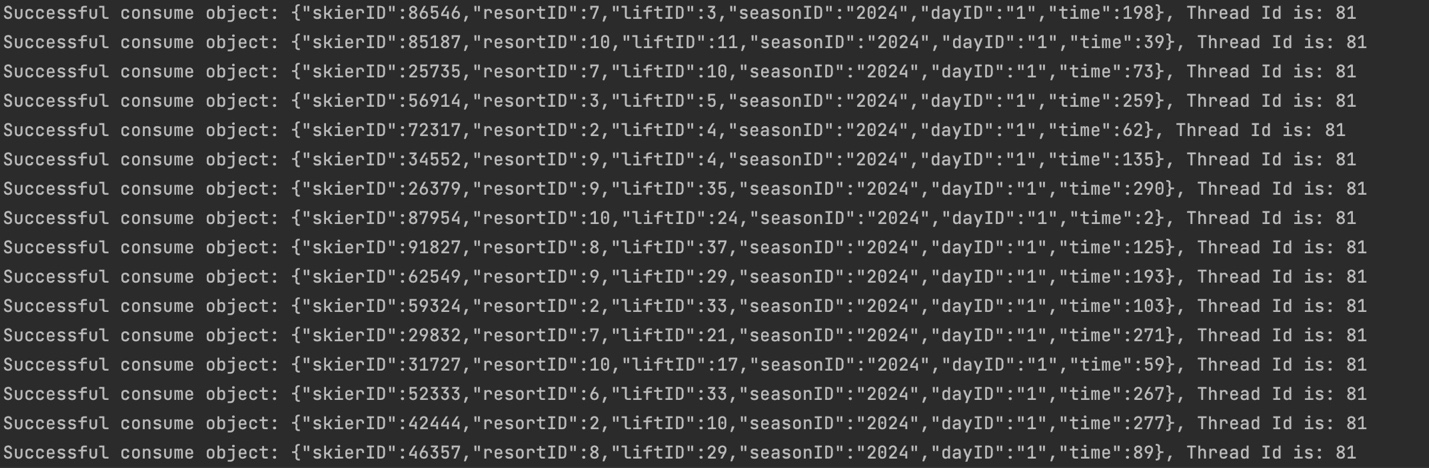
****

****

**Number of Thread in Client: 400**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

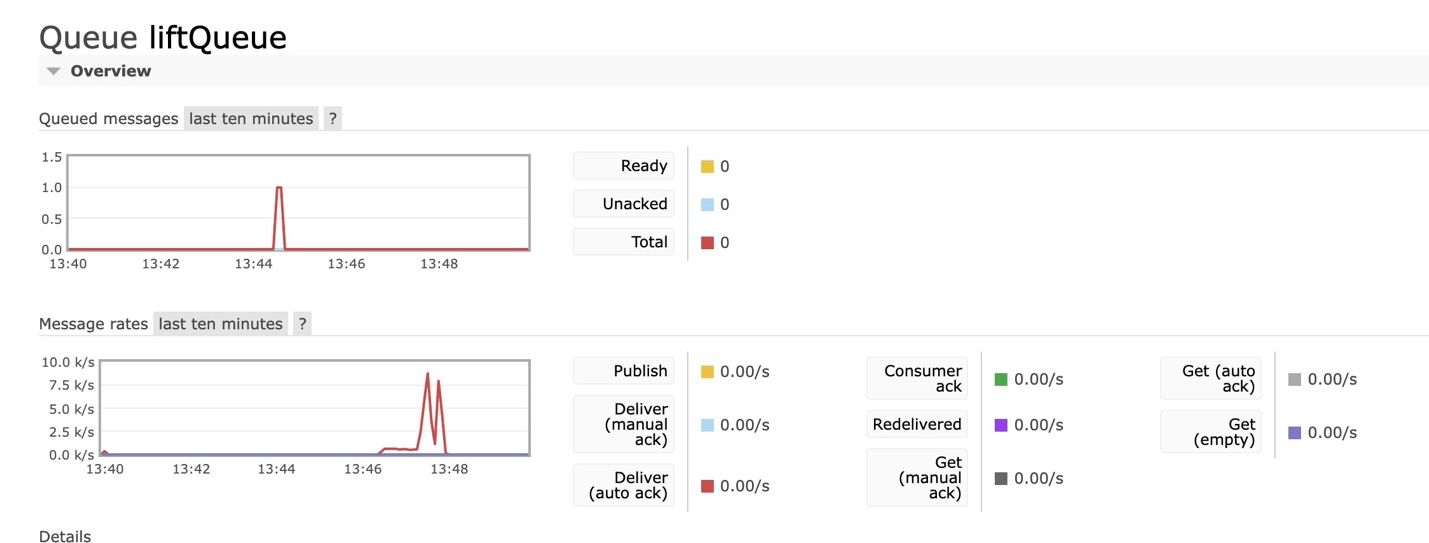
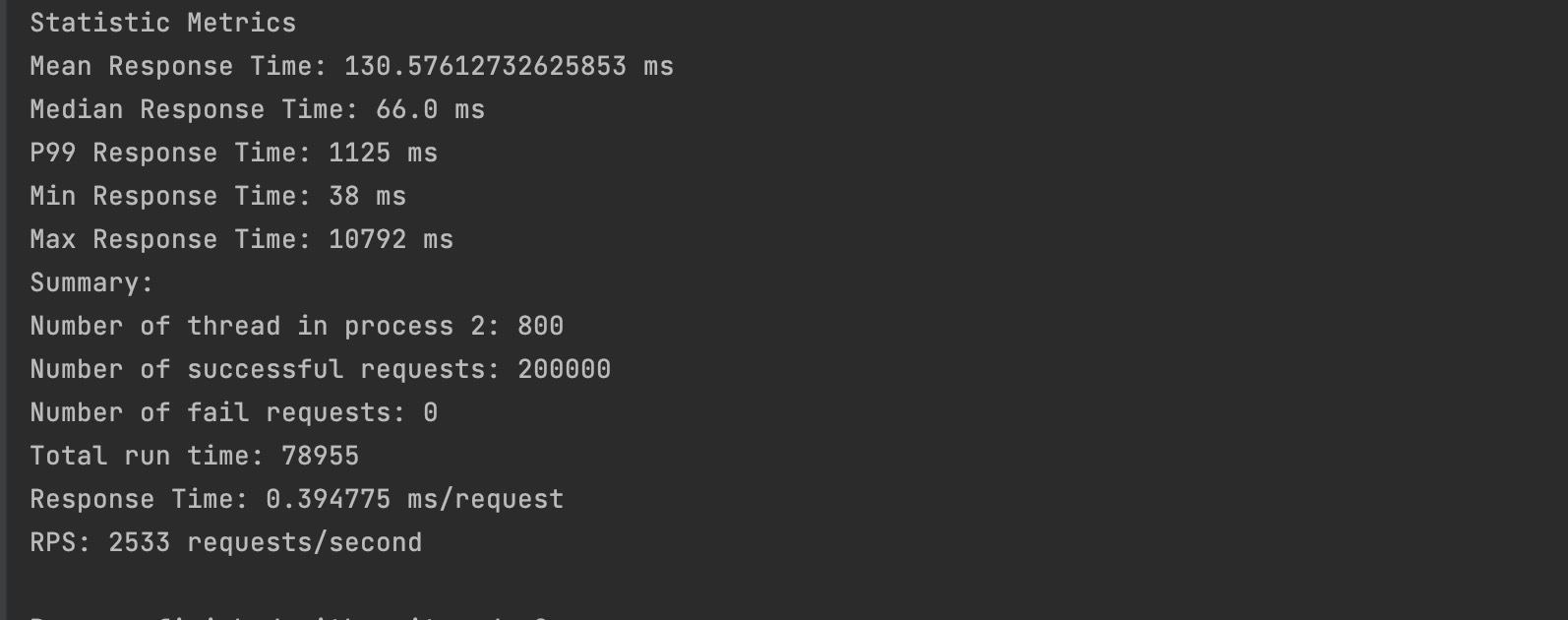
****

**Number of Thread in Client: 800**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

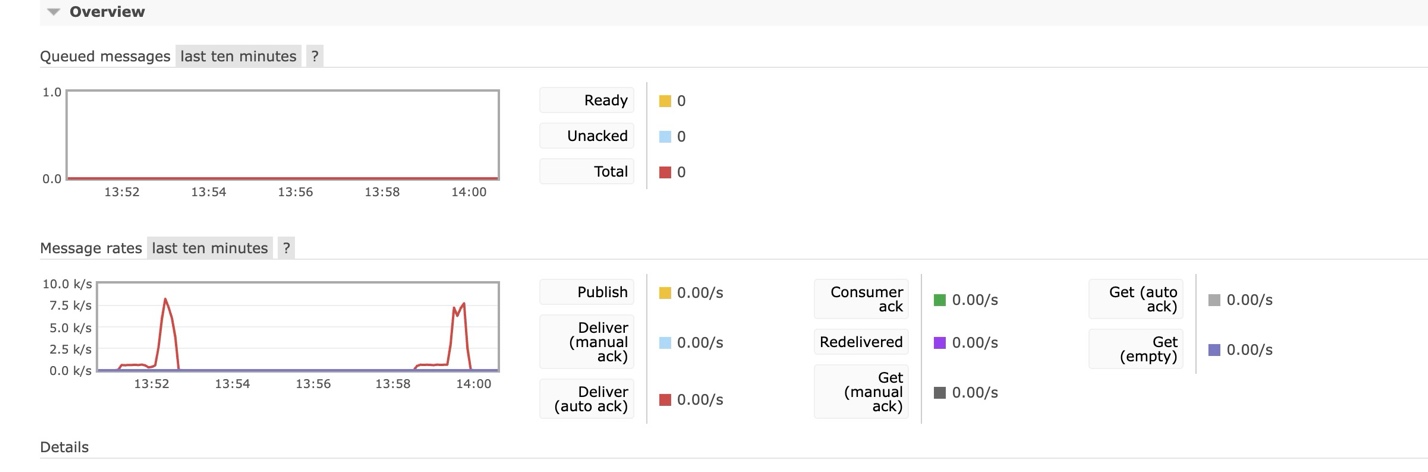
**Note: I have a test on connection so it has a peak in queue**

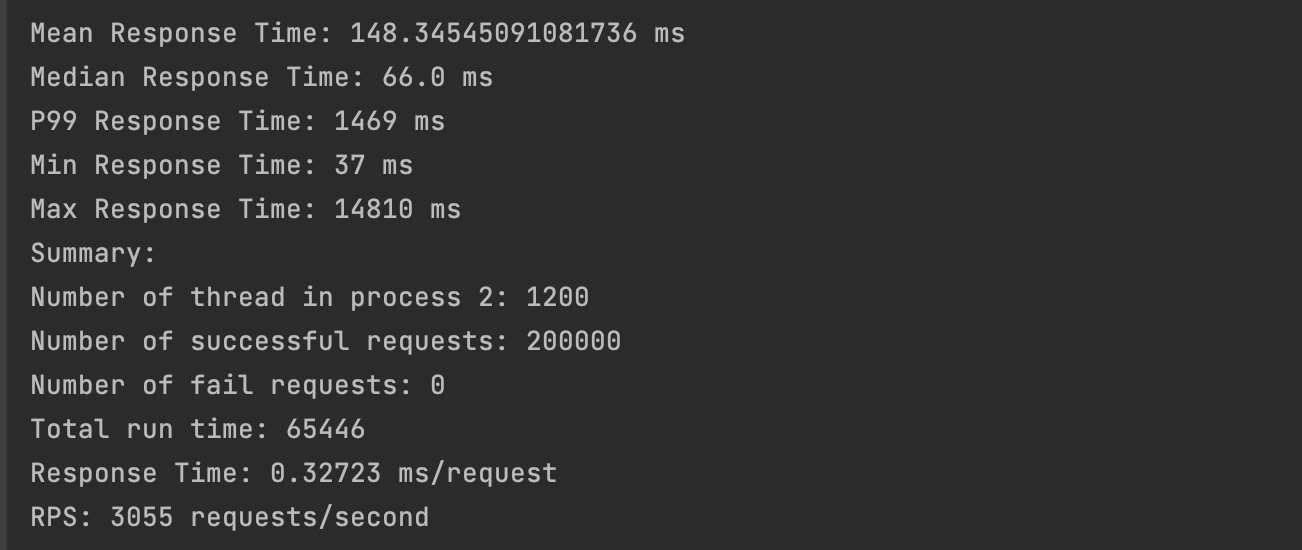
****

**Number of Thread in Client: 1200**

**Number of Thread in Consumer: 24**

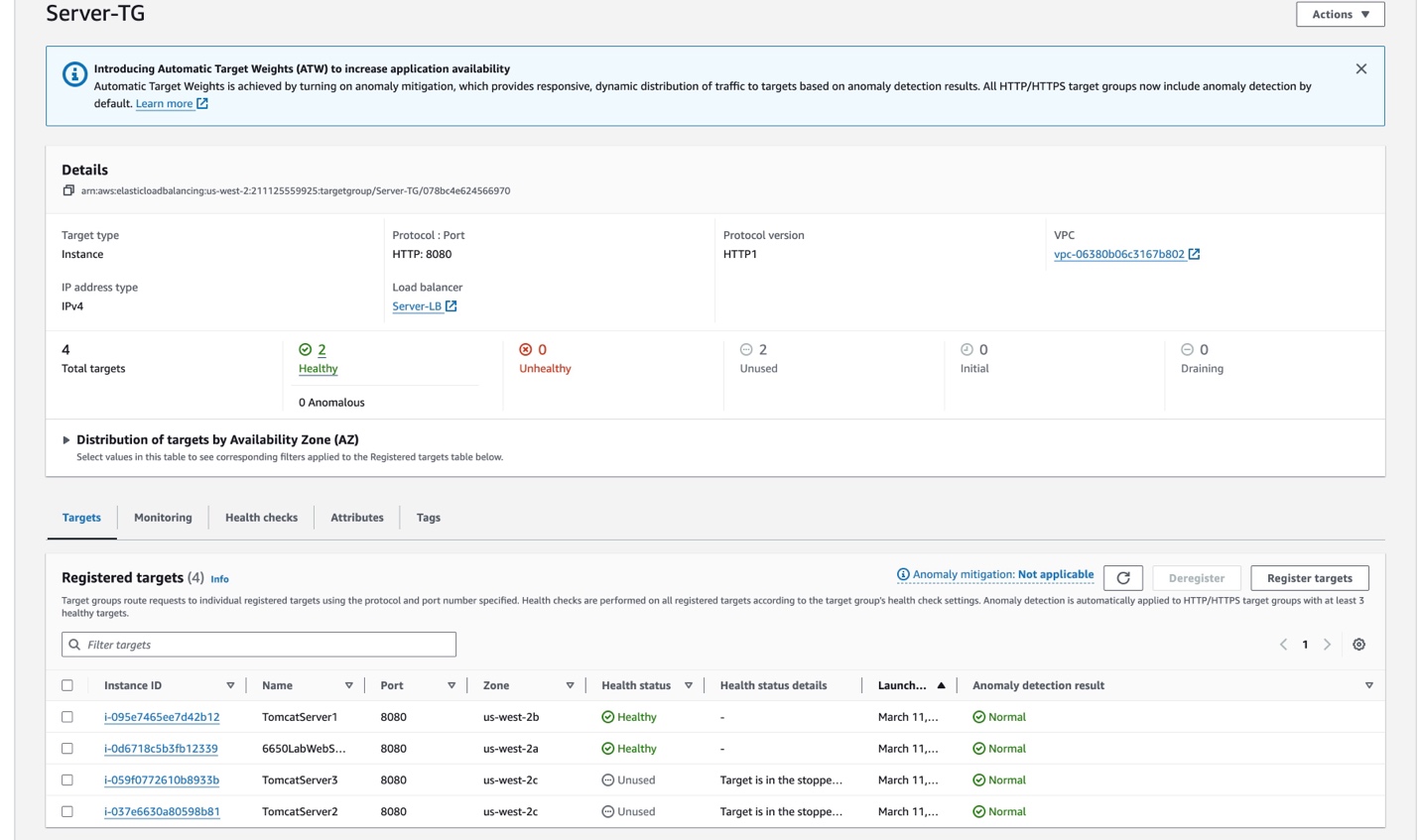
**Number of basicqos: 1**

****

****

**With Two Server**

Create image from first server and stat a new instance TomcatServer1 base on the image. Put the TomcatServer1 in to Target group with same vpc and create load balancer for the group. Checked healthy with port 8080. Let’s see performance

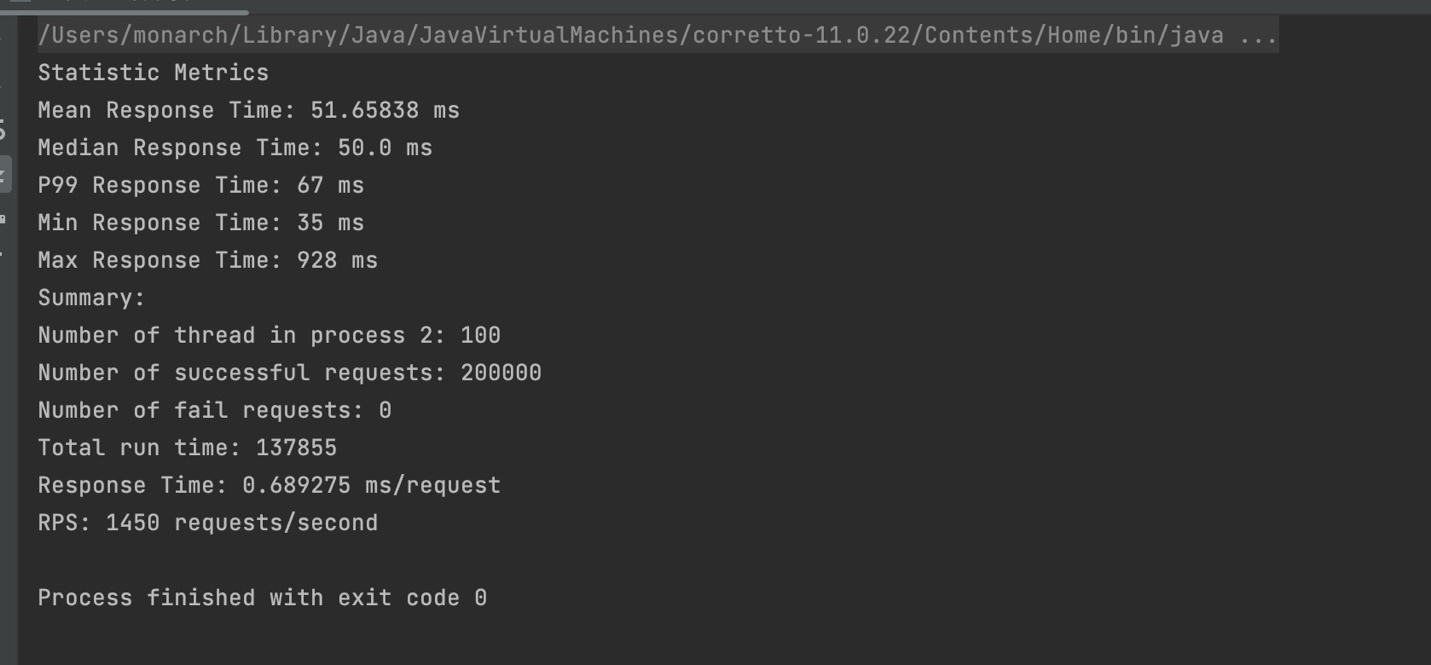
****

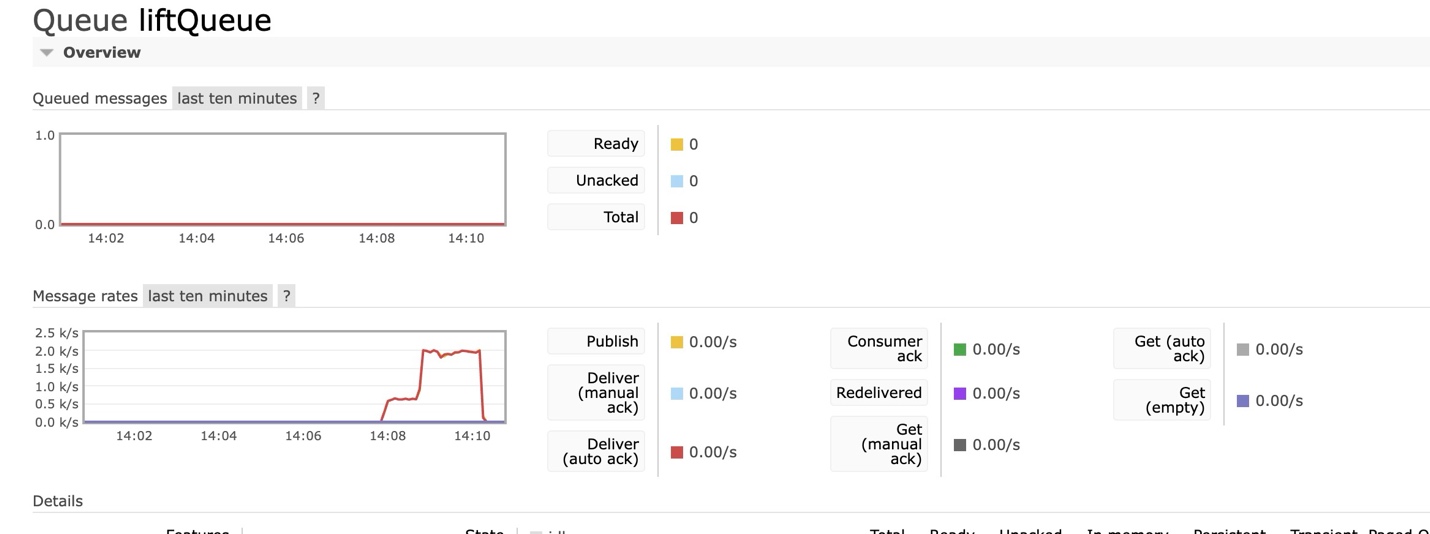
**Number of Thread in Client: 100**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

**Note : RPS does not change much with single server, but the connection is stable**

****

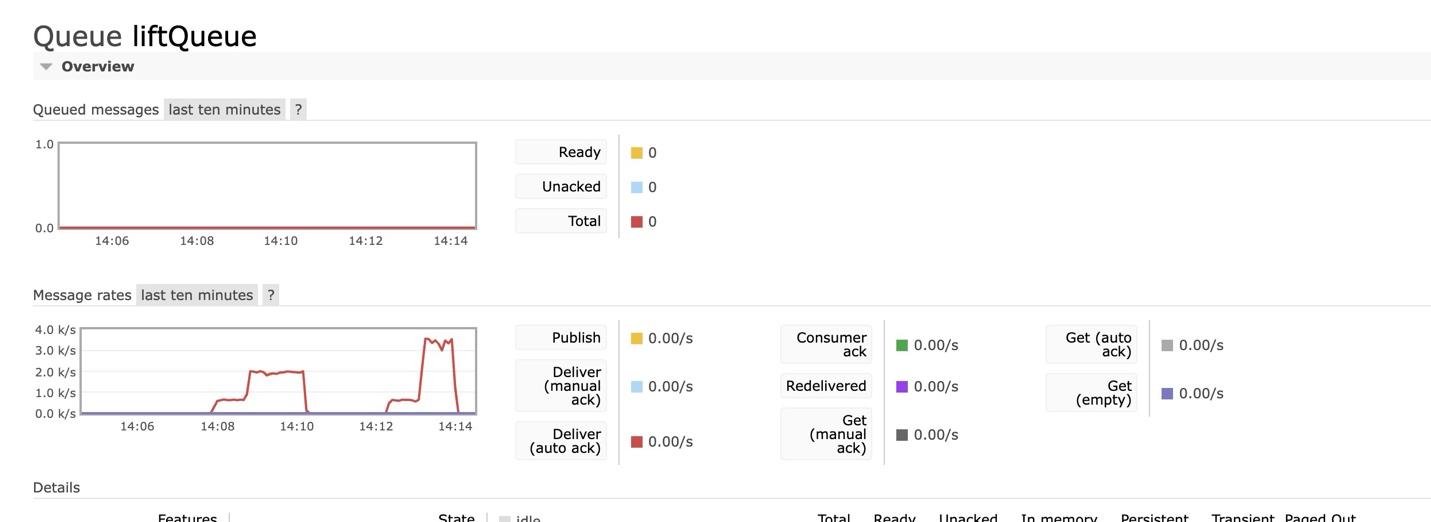
****

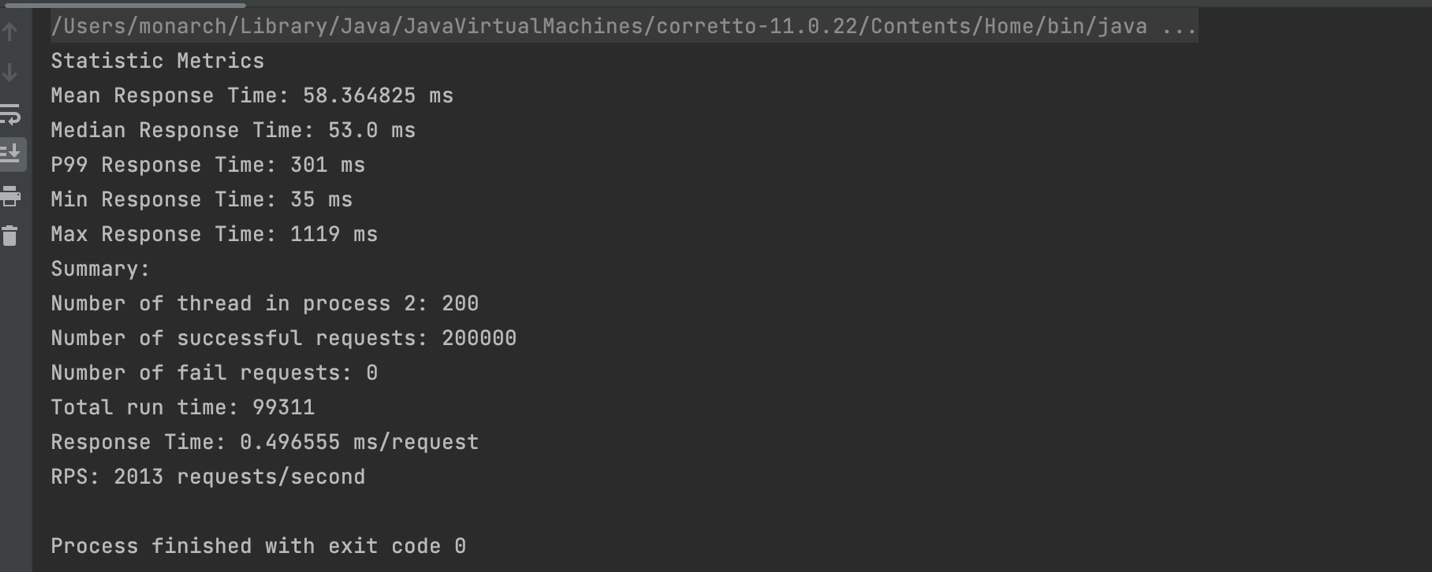
**Number of Thread in Client: 200**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

**Note: RPS start increased, which is higher than single server**

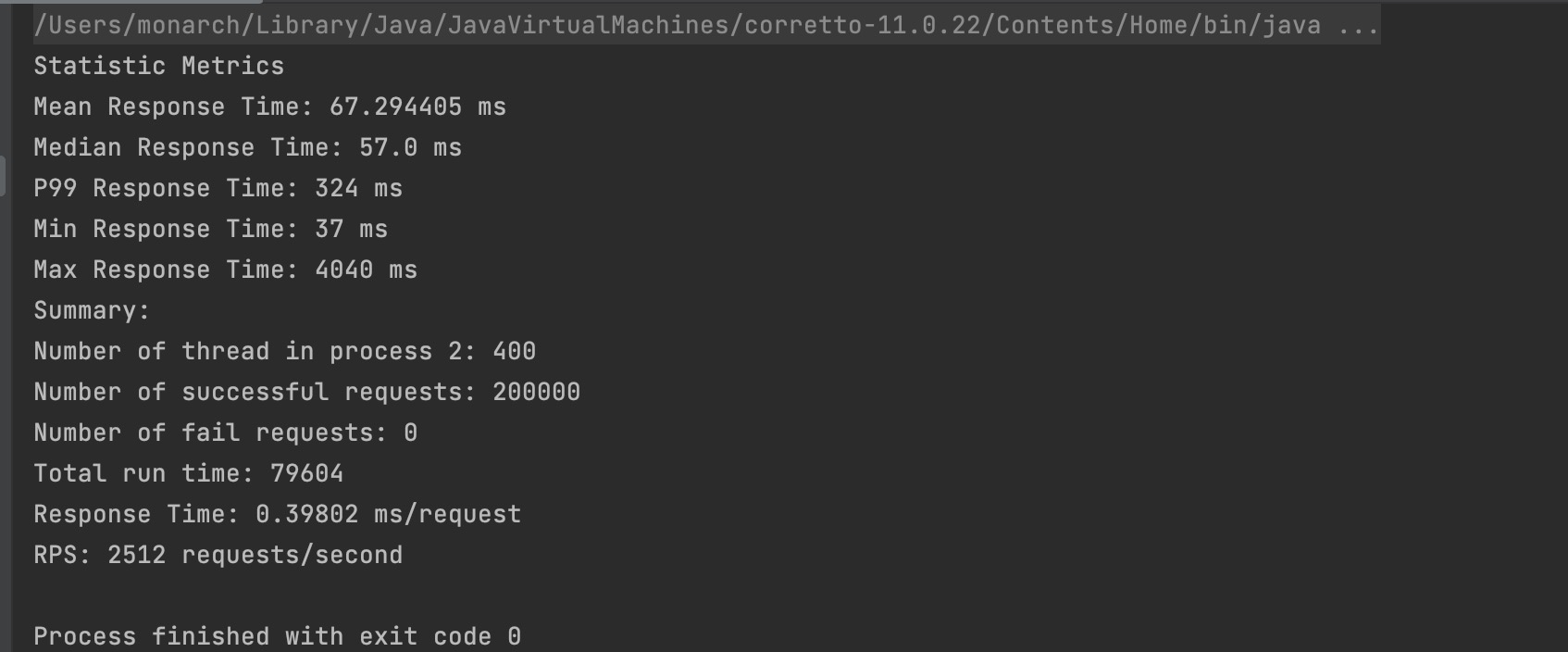
****

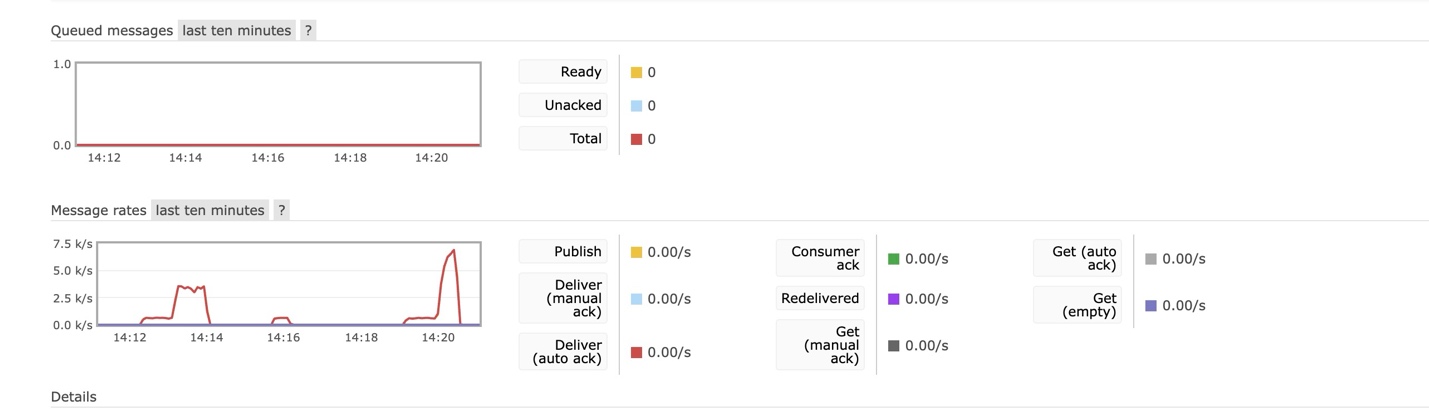
****

**Number of Thread in Client: 400**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

****

****

**Number of Thread in Client: 800**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

**Note: Performance greater than single server**

****

**Number of Thread in Client: 1200**

**Number of Thread in Consumer: 24**

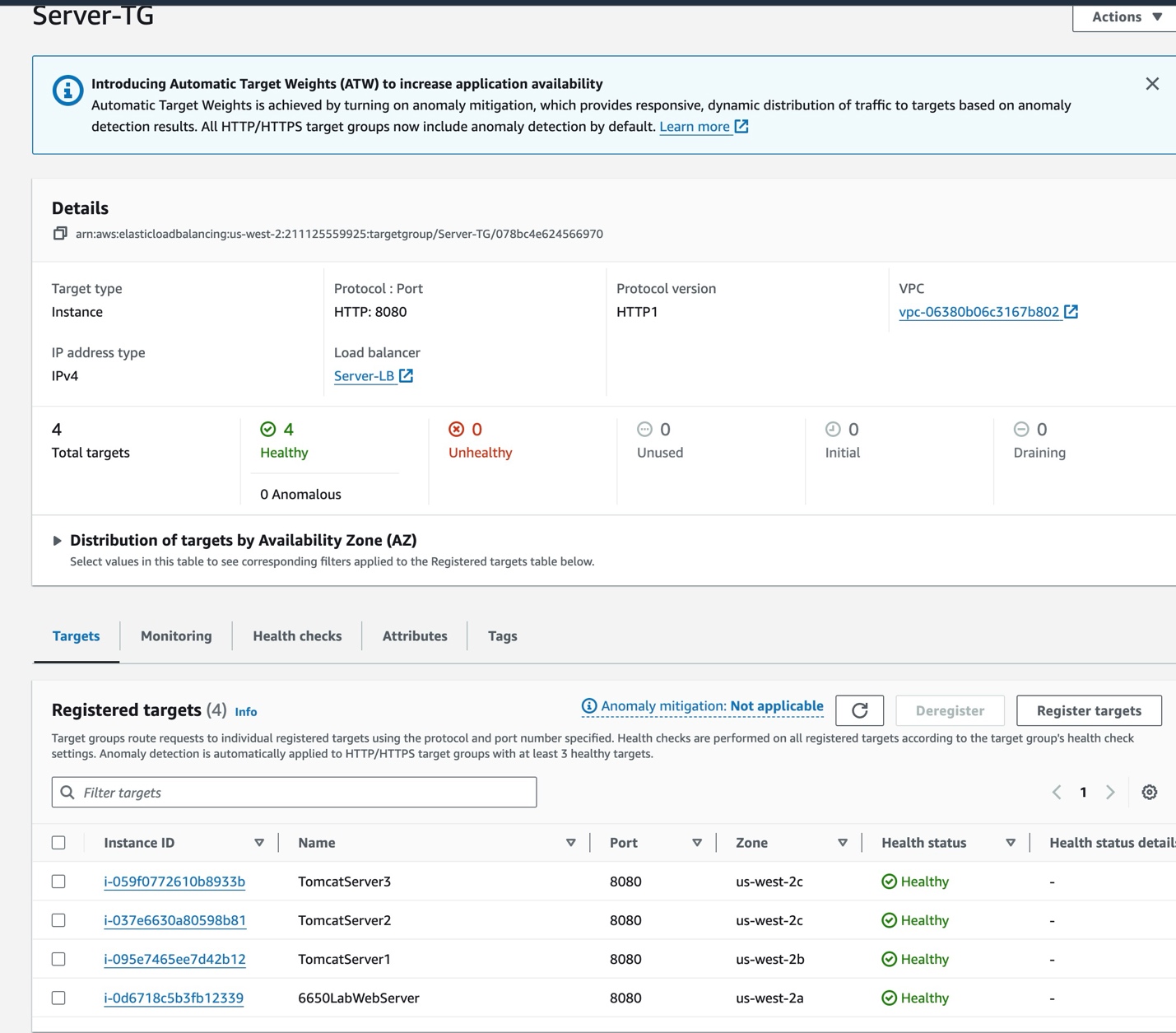
**Number of basicqos: 1**

**Note: Starting unstable**

****

****

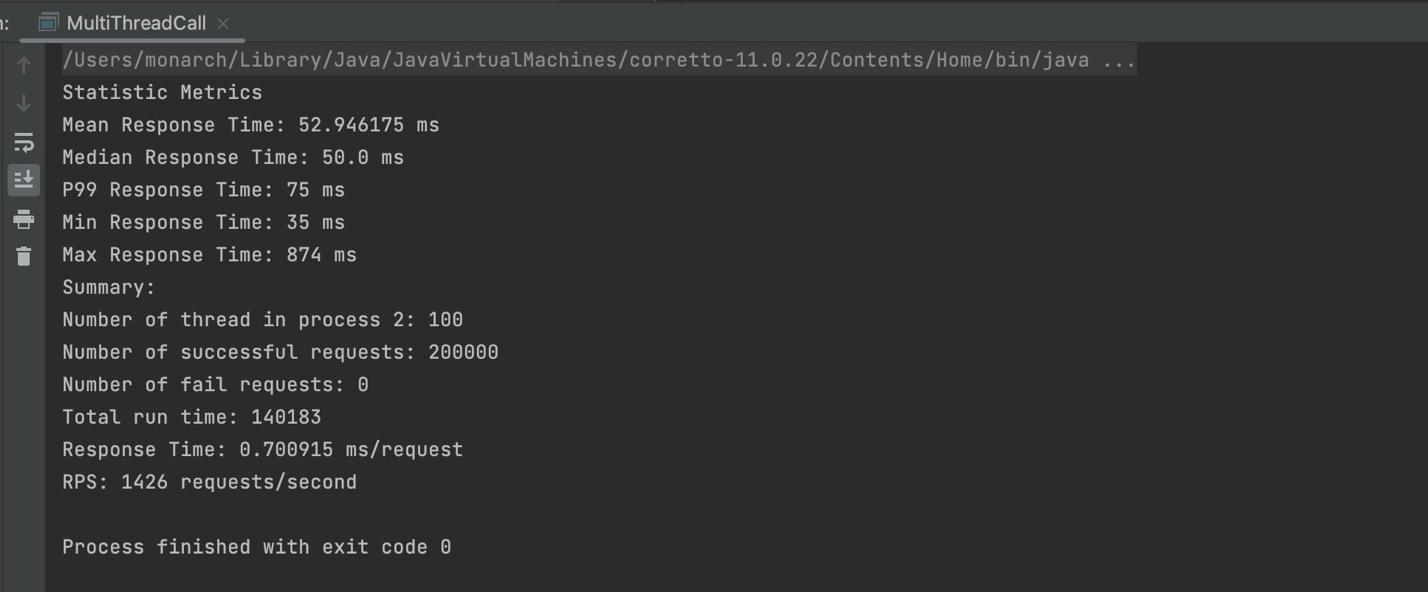
**With Four Server**

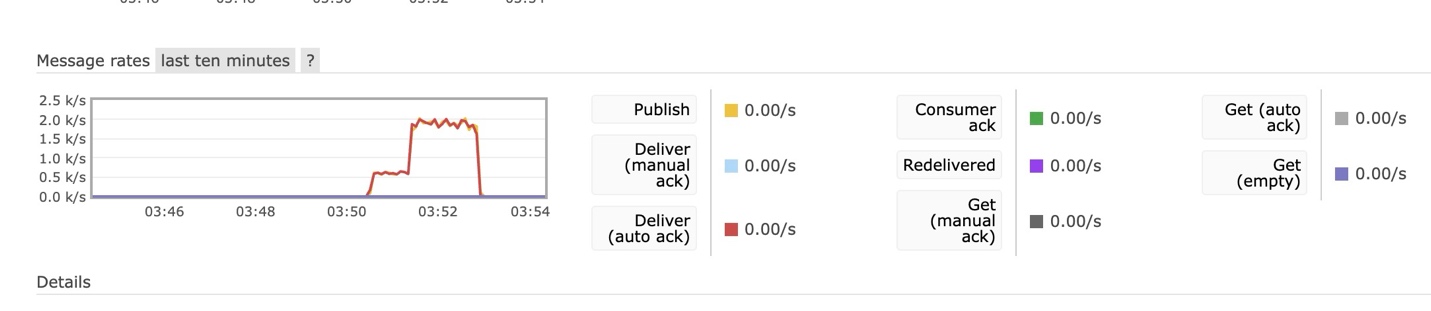
****

**Number of Thread in Client: 100**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

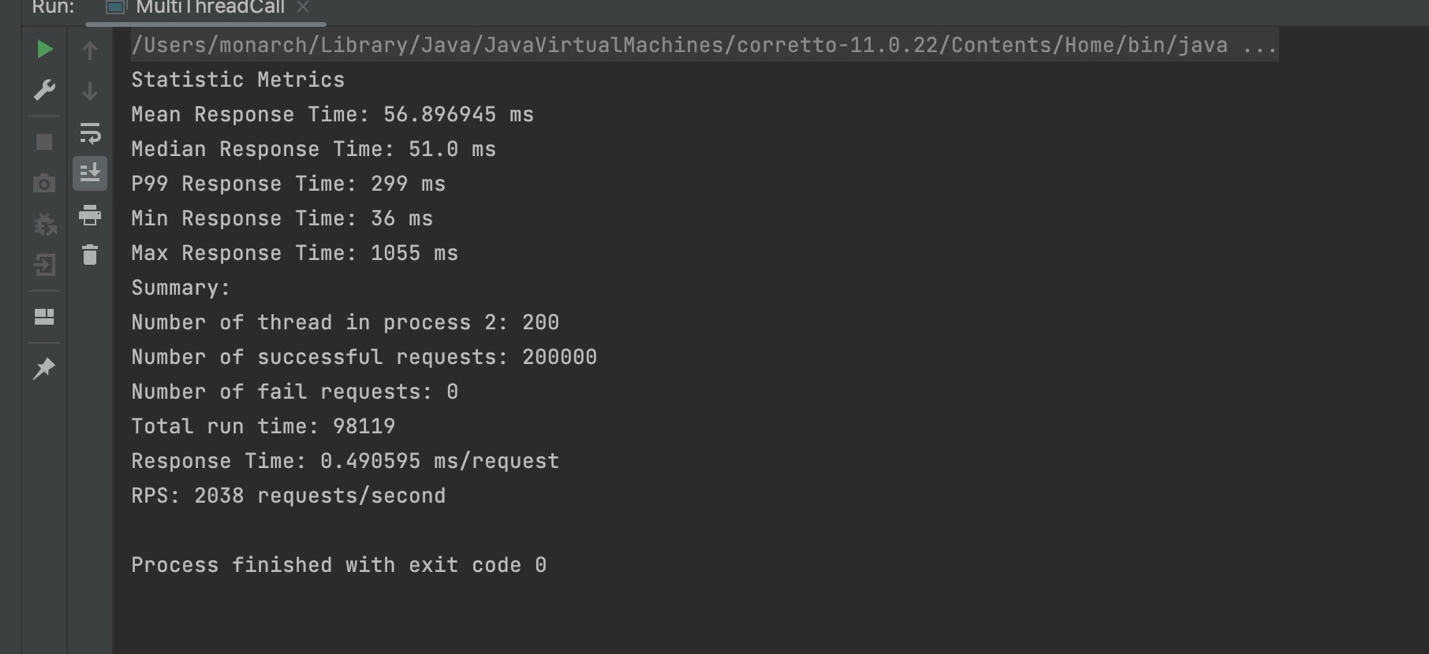
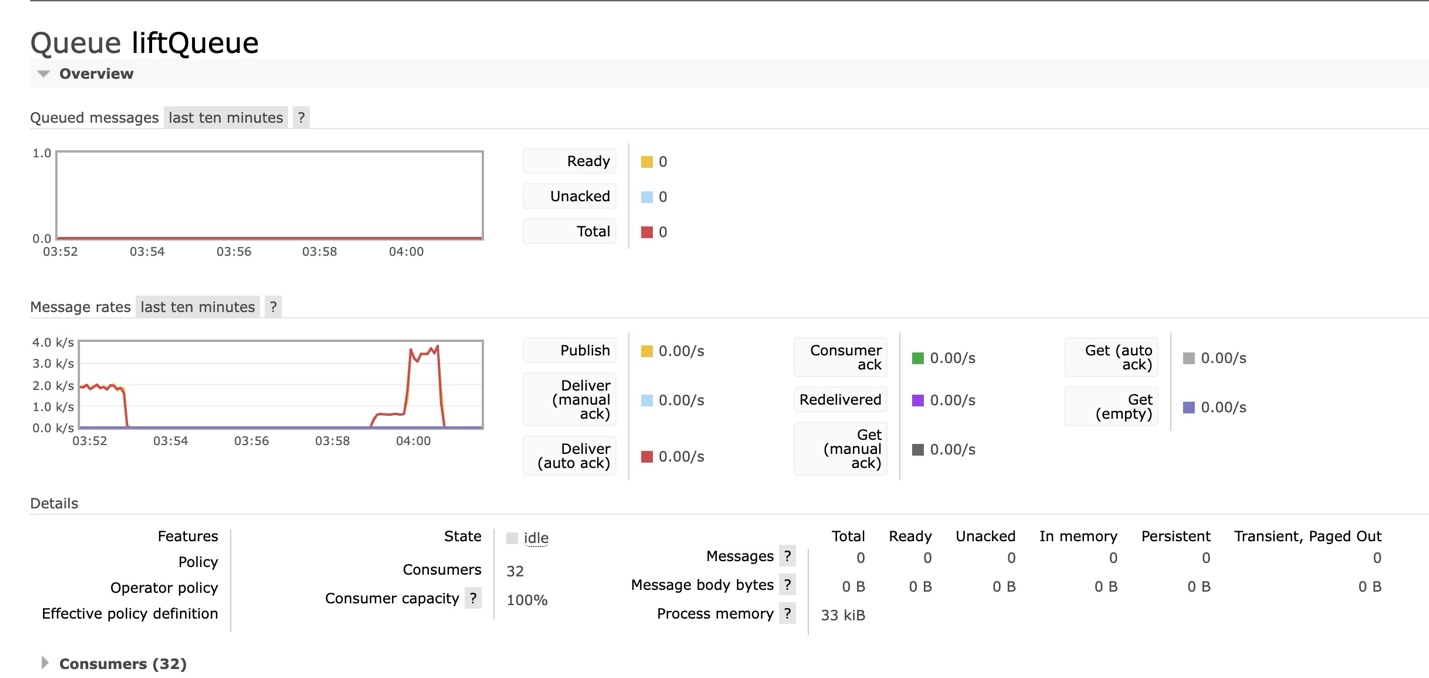
****

****

**Number of Thread in Client: 200**

**Number of Thread in Consumer: 24**

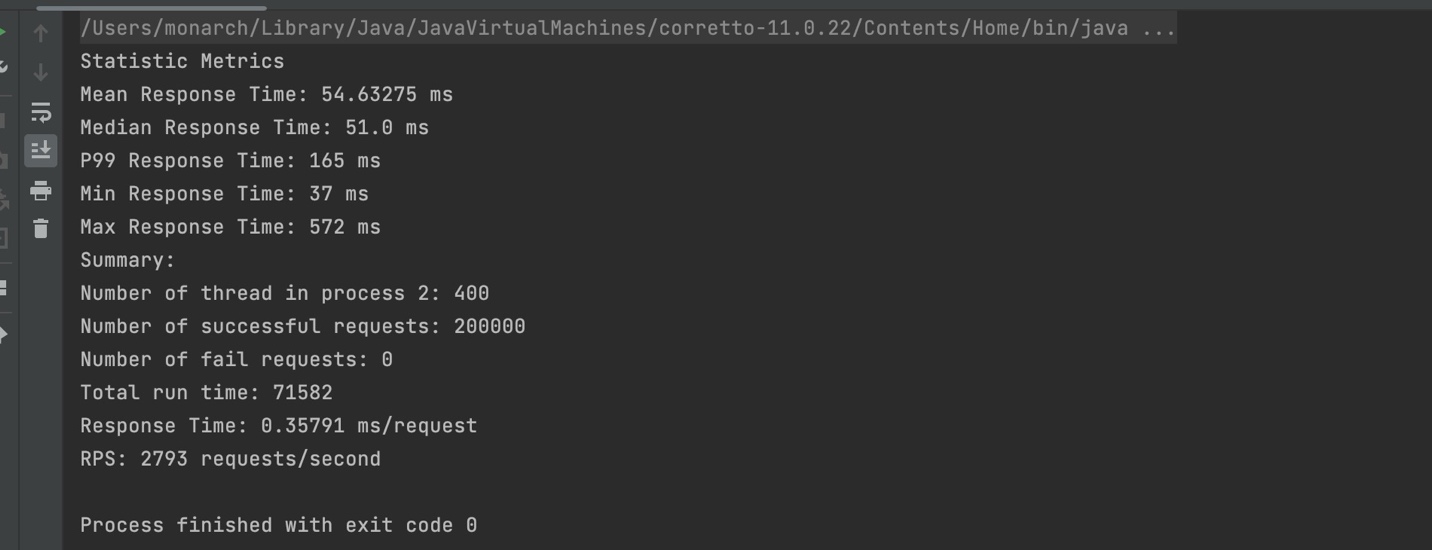
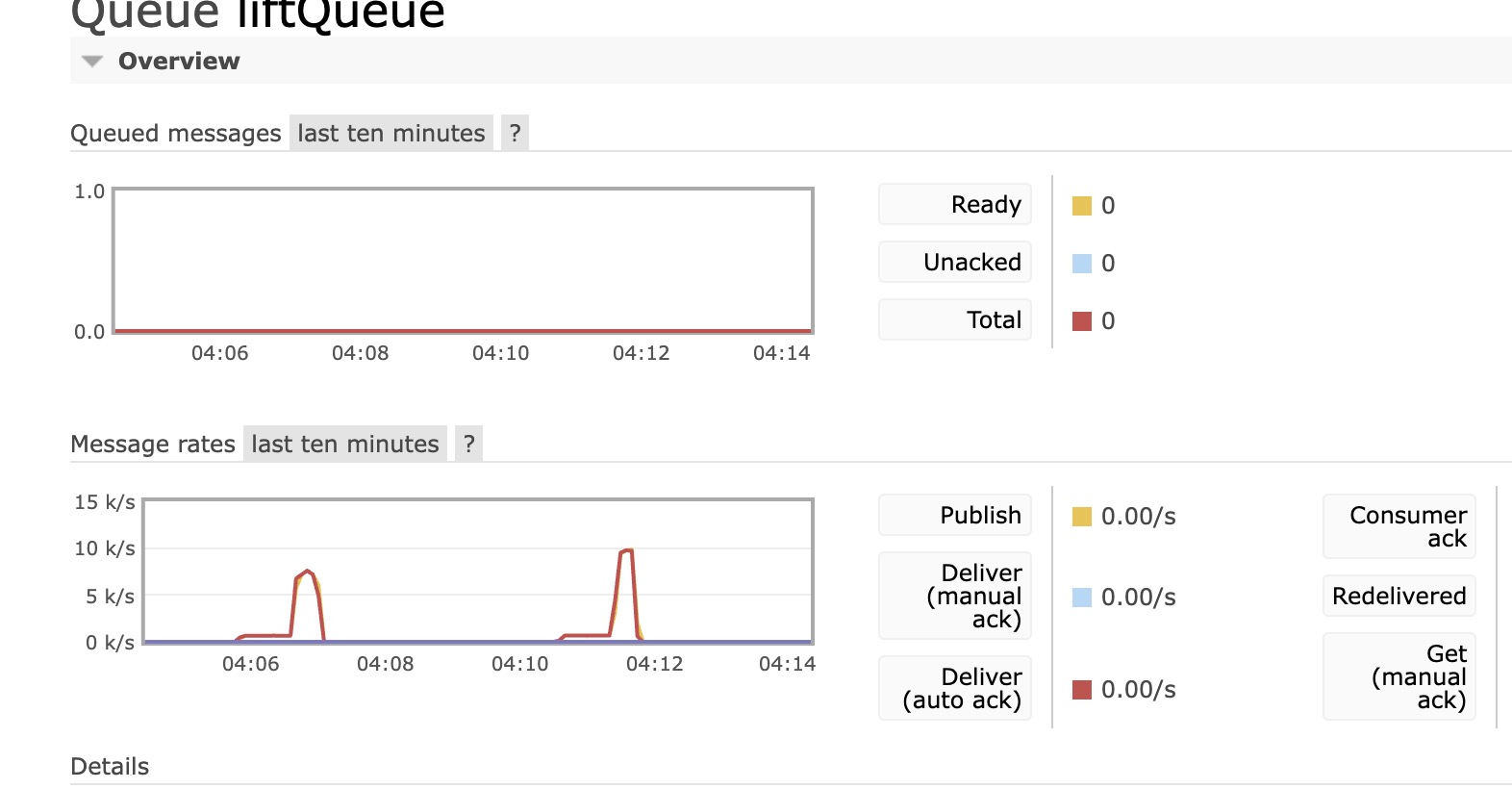
**Number of basicqos: 1**

****

**Number of Thread in Client: 400**

**Number of Thread in Consumer: 24**

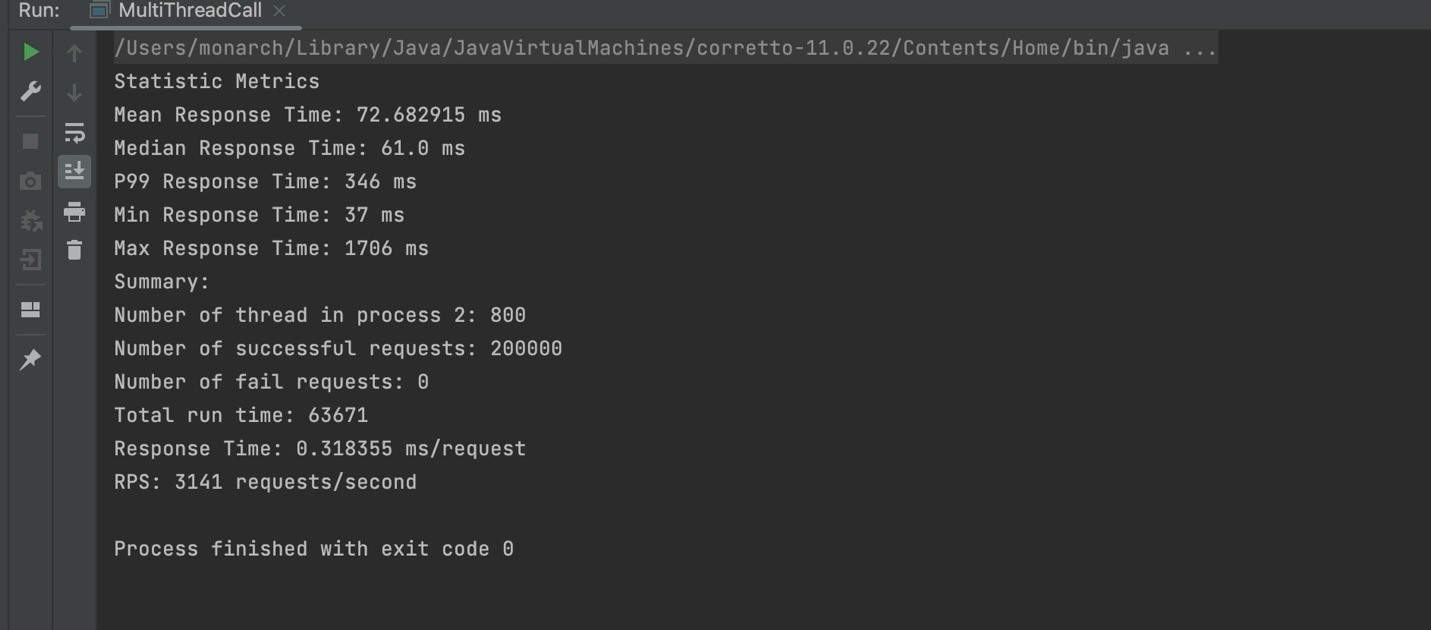
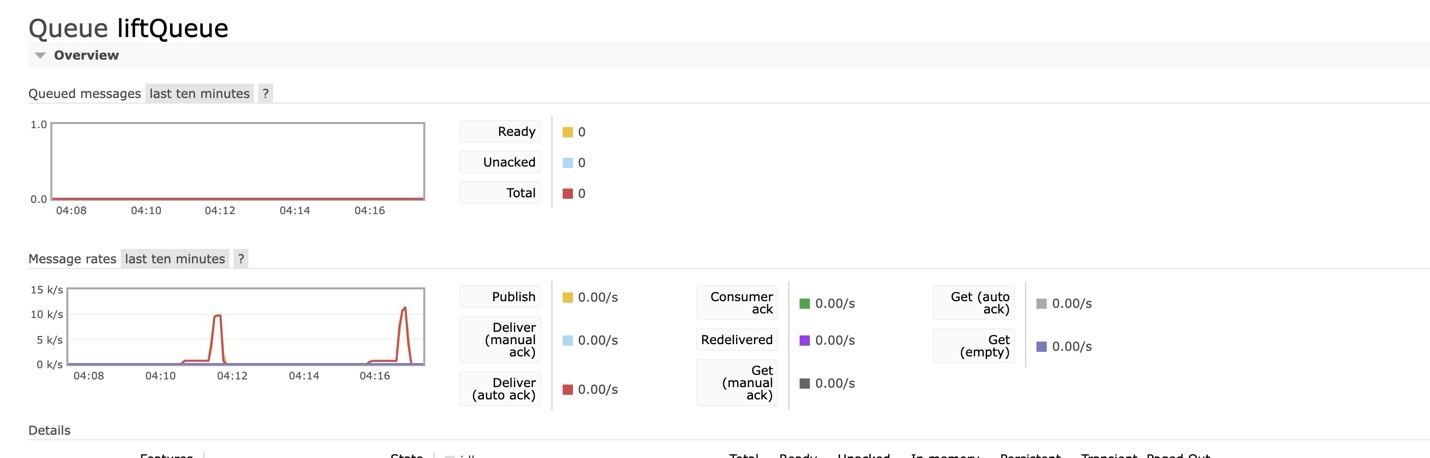
**Number of basicqos: 1**

****

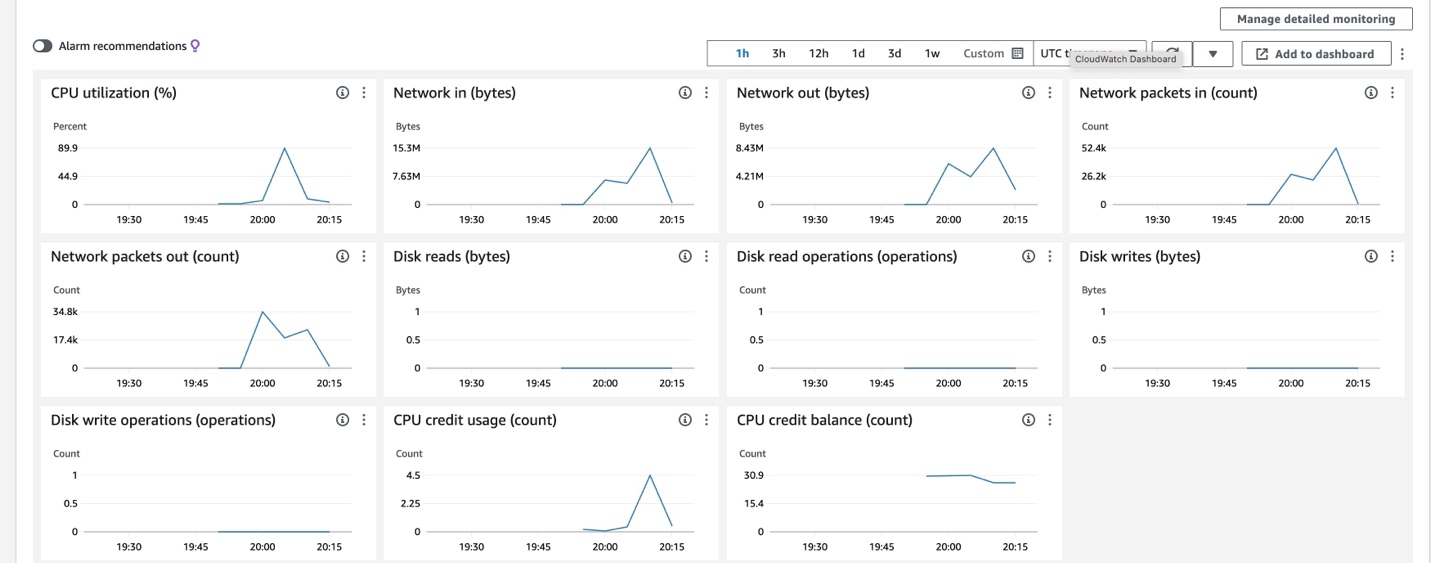
**Number of Thread in Client: 800**

**Number of Thread in Consumer: 24**

**Number of basicqos: 1**

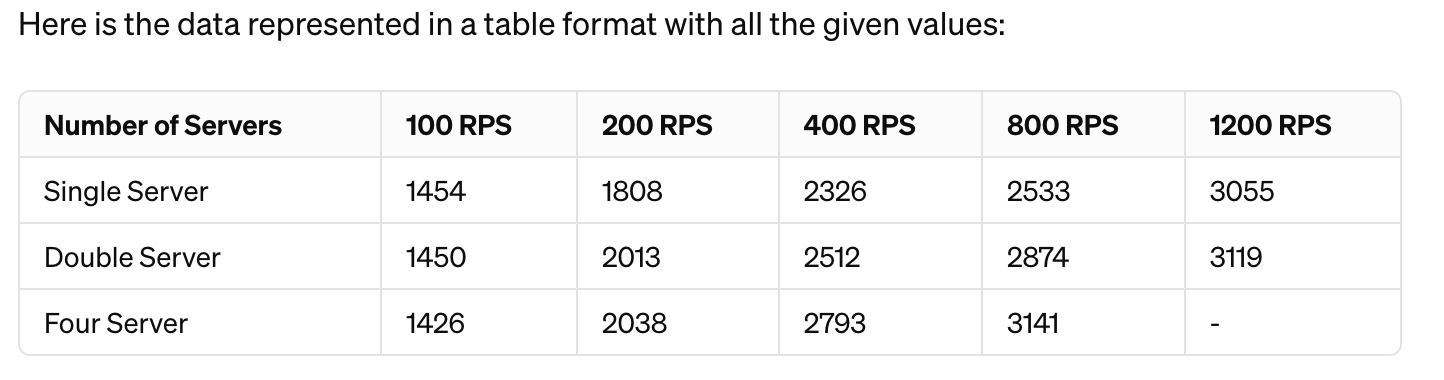
****

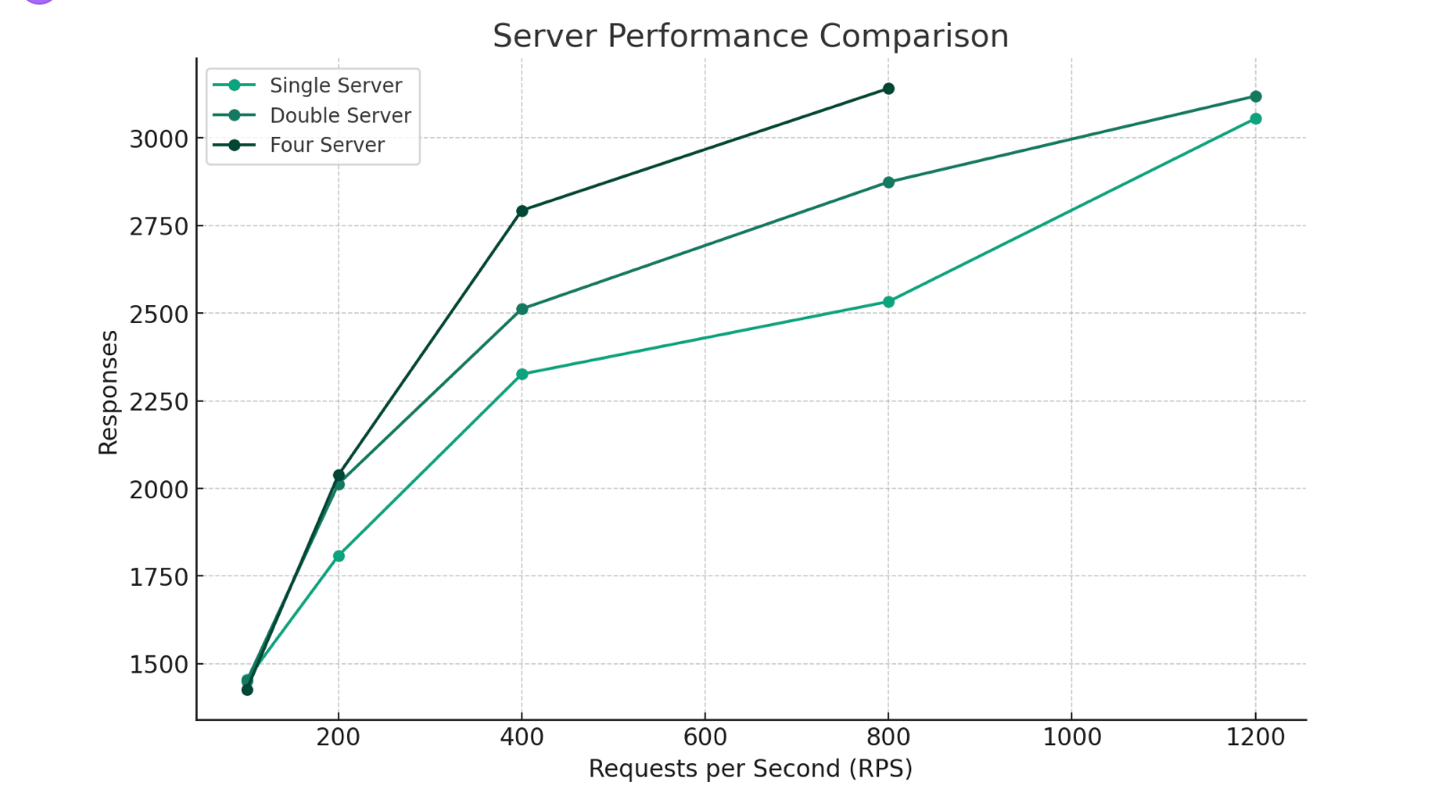
**CPU analyse:**



There is a significant peak where utilization going up to 89.9. And following this peak, CPU utilization drops sharply back to a lower level. I think this is healthy.

**RPS Plot with Different server:**





Conclusion:  
Here's the plot showing the performance comparison between single, double, and four servers across different Thread number. Each additional server in the configuration leads to an increase in the Speed of PRS, showing that adding more servers can improve the system's ability to handle higher loads. The four-server setup, in particular, appears to offer the best performance, the data for 1200 thread is missing because the connection become unstable. However, we can see, under 800 threads, four servers have greater performance than single server, which increased 24%. [(3141 - 2533) / 2533 = 0.24 ].  
  
Thanks  
Zegui Jiang