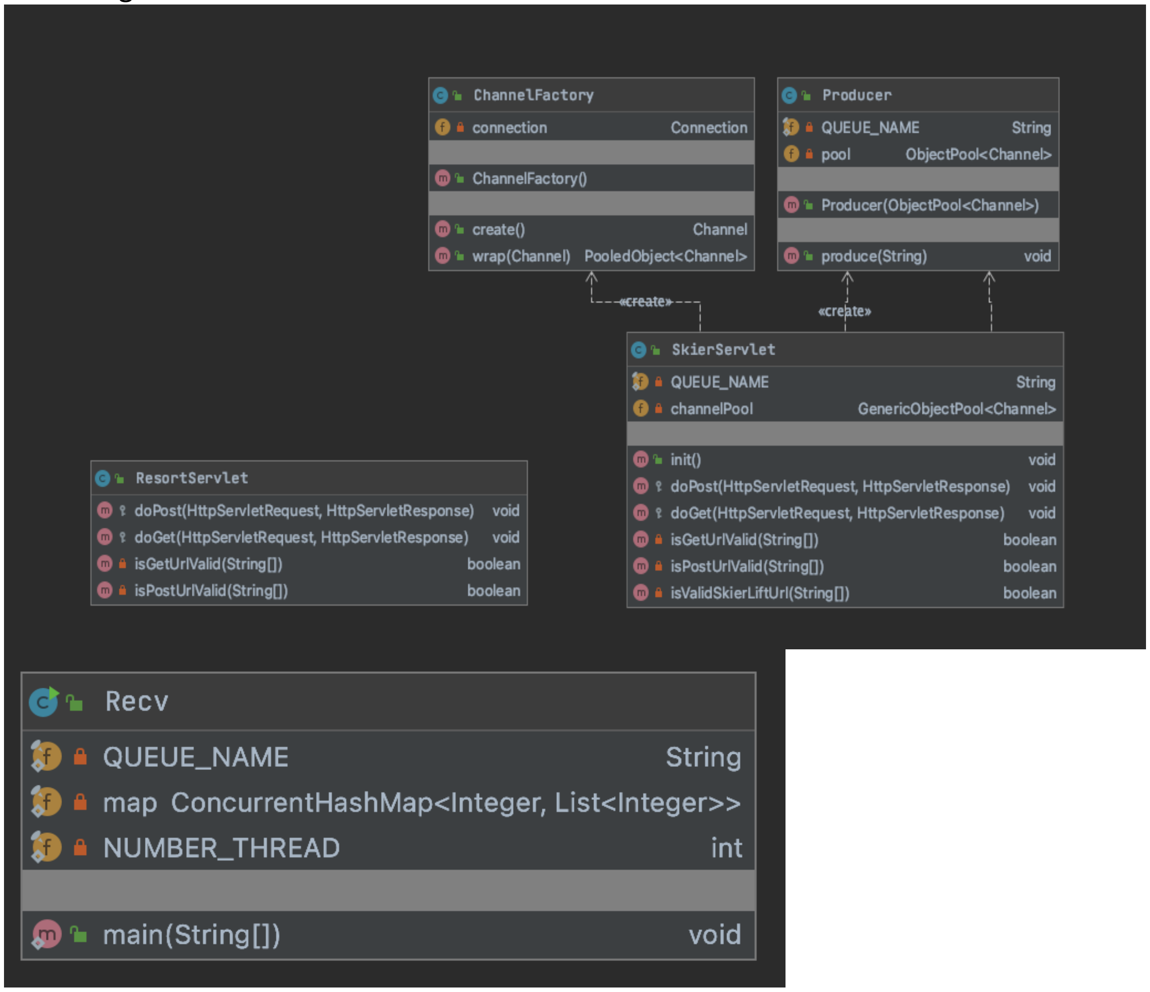
Assignment 2 – CS6650

Yuchen Tian

1. URL for Github Repo: https://github.com/tyuchn/cs6650-hw2-ytian
2. Server Design
   a. Class Diagram



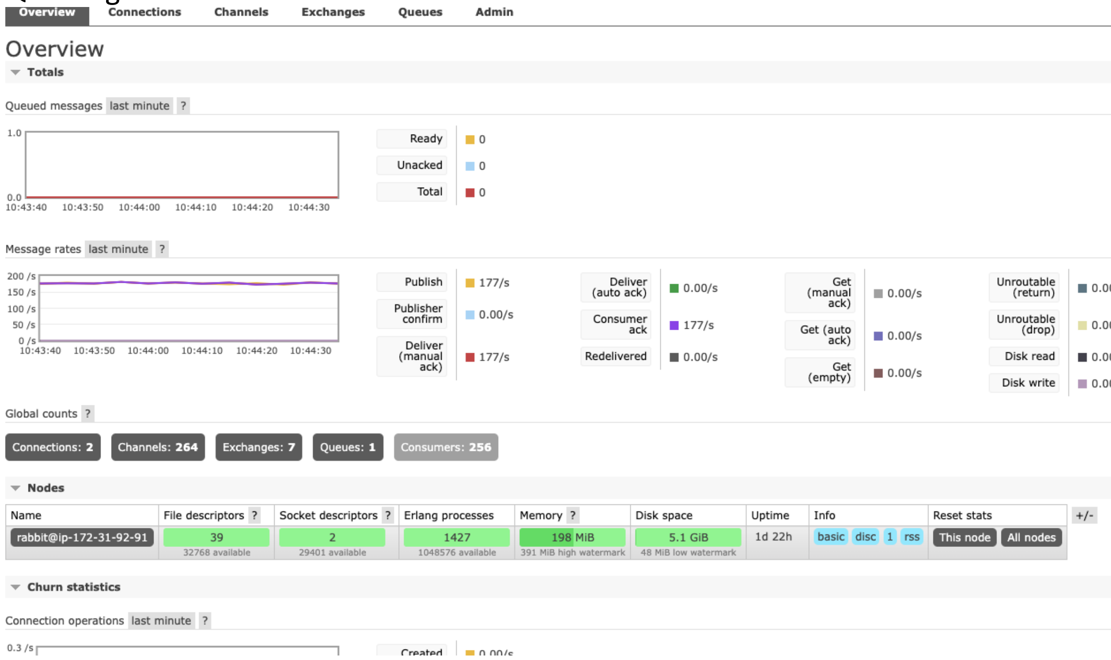
ChannelFactory
connection Connection
ChannelFactory()
create() Channel
wrap(Channel) PooledObject<Channel>
Producer
QUEUE_NAME String
pool ObjectPool<Channel>
Producer(ObjectPool<Channel>)
produce(String) void
«create»
«create»
SkierServlet
QUEUE_NAME String
channelPool GenericObjectPool<Channel>
init() void
doPost(HttpServletRequest, HttpServletResponse) void
doGet(HttpServletRequest, HttpServletResponse) void
isGetUrlValid(String[]) boolean
isPostUrlValid(String[]) boolean
isValidSkierLiftUrl(String[]) boolean
ResortServlet
doPost(HttpServletRequest, HttpServletResponse) void
doGet(HttpServletRequest, HttpServletResponse) void
isGetUrlValid(String[]) boolean
isPostUrlValid(String[]) boolean
Recv
QUEUE_NAME String
map ConcurrentHashMap<Integer, List<Integer>>
NUMBER_THREAD int
main(String[]) void

   b. Mechanisms

The server consists of a tomcat servlet (including a producer), a RabbitMQ message queue and a consumer. For the producer, since we mainly focus on the skier servlet. There are three main classes in the skier servlet: *SkierServlet*, *ChannelFactory* and *Producer*. The SkierServlet is refactored from the previous assignment, it has an instance of GenericObjectPool (in the init() method), which is initiated by a ChannelFactory class. This pool will serve as a channel pool that shares pre-created channels with the producers. The ChannelFactory class is a factory for channel creation, it is implemented on top of the apache pool2 package. Producer class is the producer for sending message to RabbitMQ, it is instantiated by a channel pool, and in the produce() method, it will borrow a channel object from the channel pool, then publish the message to the message queue, at last, it will return the borrowed channel to the channel pool for future use. Recv is the consumer class, it was a 256-thread program, each thread creates a channel, get the message and writes to a concurrent hashmap. It is always implemented on the rabbitmq client package.

c. The life cycle of a message
   When a post request was sent to the server, the skier servlet will first validate it, if the validation was passed, it will instantiate a producer object, and assign a channel from the channel pool to the producer, then the producer will publish the message to the message queue. On the consumer side, there is a 256-thread program, each thread will have a channel connecting with the message queue and consumer the message. At the same time, I use a concurrent map to store the record for each liftID.

3. Test Results
   a. 64 Threads
      i. Commands lines:
         -numThreads 64 -numSkiers 100000 -address 54.82.5.13:8080
      ii. RMQ management console:



Overview Connections Channels Exchanges Queues Admin

Overview

Totals

Queued messages last minute ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last minute ?

Publish	🟨 177/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	■ 0.00/s	Unroutable (return)	■ 0.0
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 177/s	Get (auto ack)	■ 0.00/s	Unroutable (drop)	■ 0.0
Deliver (manual ack)	🟥 177/s	Redelivered	■ 0.00/s	Get (empty)	■ 0.00/s	Disk read	■ 0.0
						Disk write	■ 0.0

Global counts ?

Connections: 2 Channels: 264 Exchanges: 7 Queues: 1 Consumers: 256

Nodes

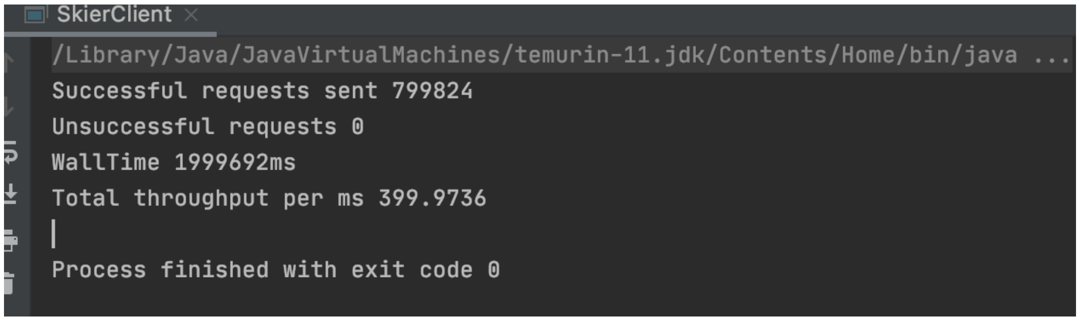
Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@ip-172-31-92-91	39 32768 available	2 29401 available	1427 1048576 available	198 MiB 391 MiB high watermark	5.1 GiB 48 MiB low watermark	1d 22h	basic disc 1 rss	This node All nodes

Churn statistics

Connection operations last minute ?

Created 🟨 0.00/s

      iii. Result:



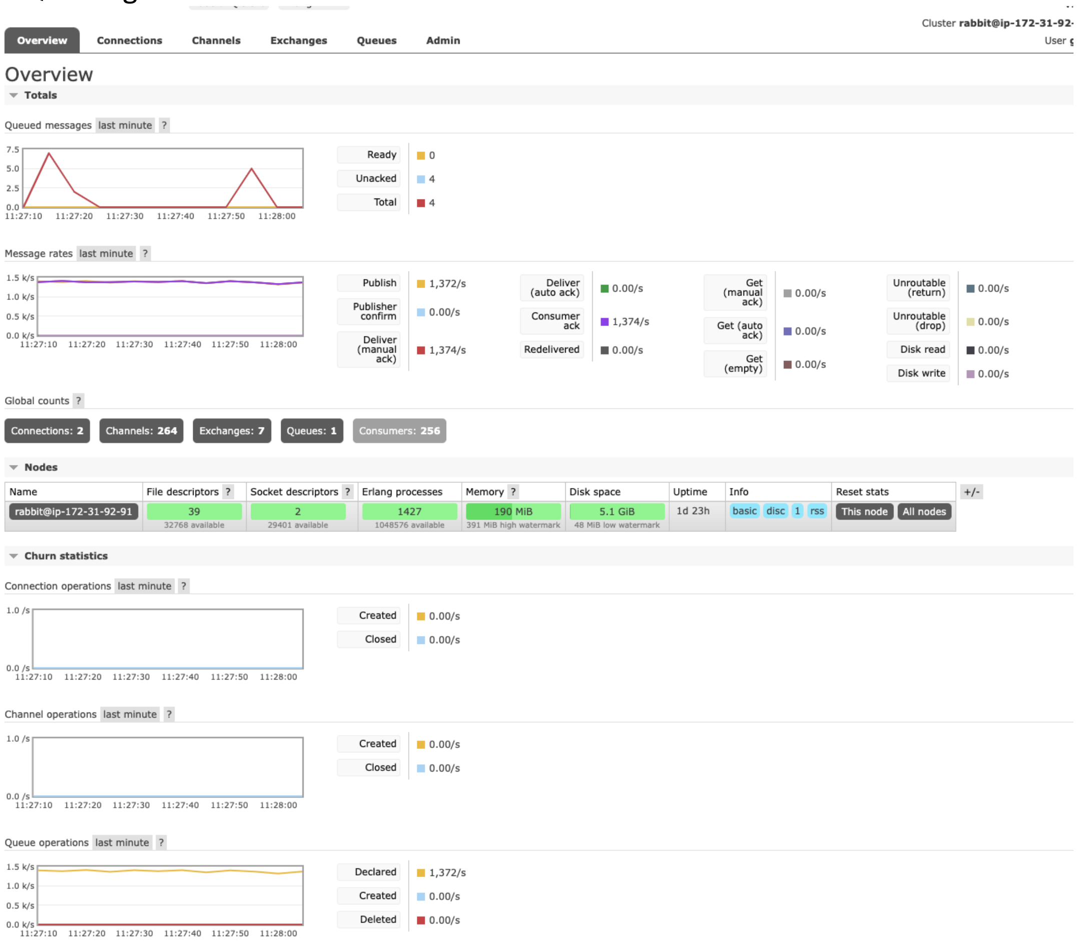
```
SkierClient
/Library/Java/JavaVirtualMachines/temurin-11.jdk/Contents/Home/bin/java ...
Successful requests sent 799824
Unsuccessful requests 0
WallTime 1999692ms
Total throughput per ms 399.9736

Process finished with exit code 0
```

b. 128 Threads

   i. Commands lines:

      -numThreads 128 -numSkiers 100000 -address 54.82.5.13:8080

   ii. RMQ management console:



Overview

Totals

Queued messages last minute ?

Ready	0
Unacked	4
Total	4

Message rates last minute ?

Publish	1,372/s	Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s	Unroutable (return)	0.00/s
Publisher confirm	0.00/s	Consumer ack	1,374/s	Get (auto ack)	0.00/s	Unroutable (drop)	0.00/s
Deliver (manual ack)	1,374/s	Redelivered	0.00/s	Get (empty)	0.00/s	Disk read	0.00/s
						Disk write	0.00/s

Global counts ?

Connections: 2 Channels: 264 Exchanges: 7 Queues: 1 Consumers: 256

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@ip-172-31-92-91	39 32768 available	2 29401 available	1427 1048576 available	190 MiB 391 MiB high watermark	5.1 GiB 48 MiB low watermark	1d 23h	basic disc 1 rss	This node All nodes

Churn statistics

Connection operations last minute ?

Created	0.00/s
Closed	0.00/s

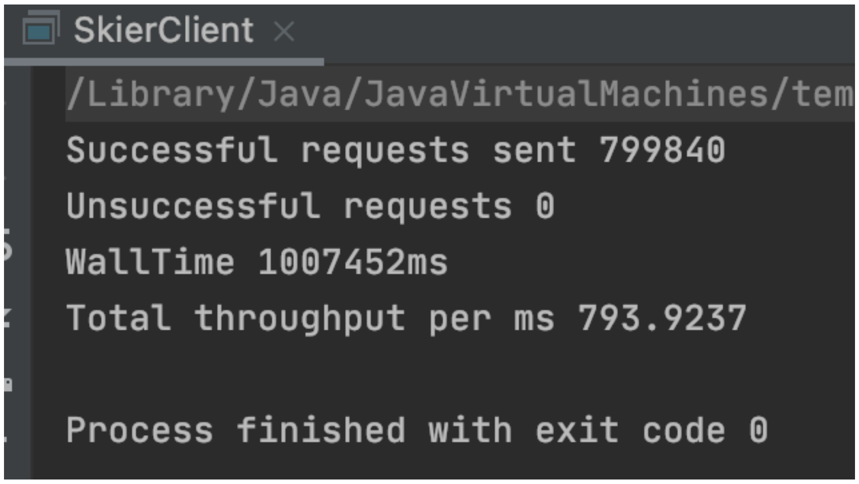
Channel operations last minute ?

Created	0.00/s
Closed	0.00/s

Queue operations last minute ?

Declared	1,372/s
Created	0.00/s
Deleted	0.00/s

   iii. Result:



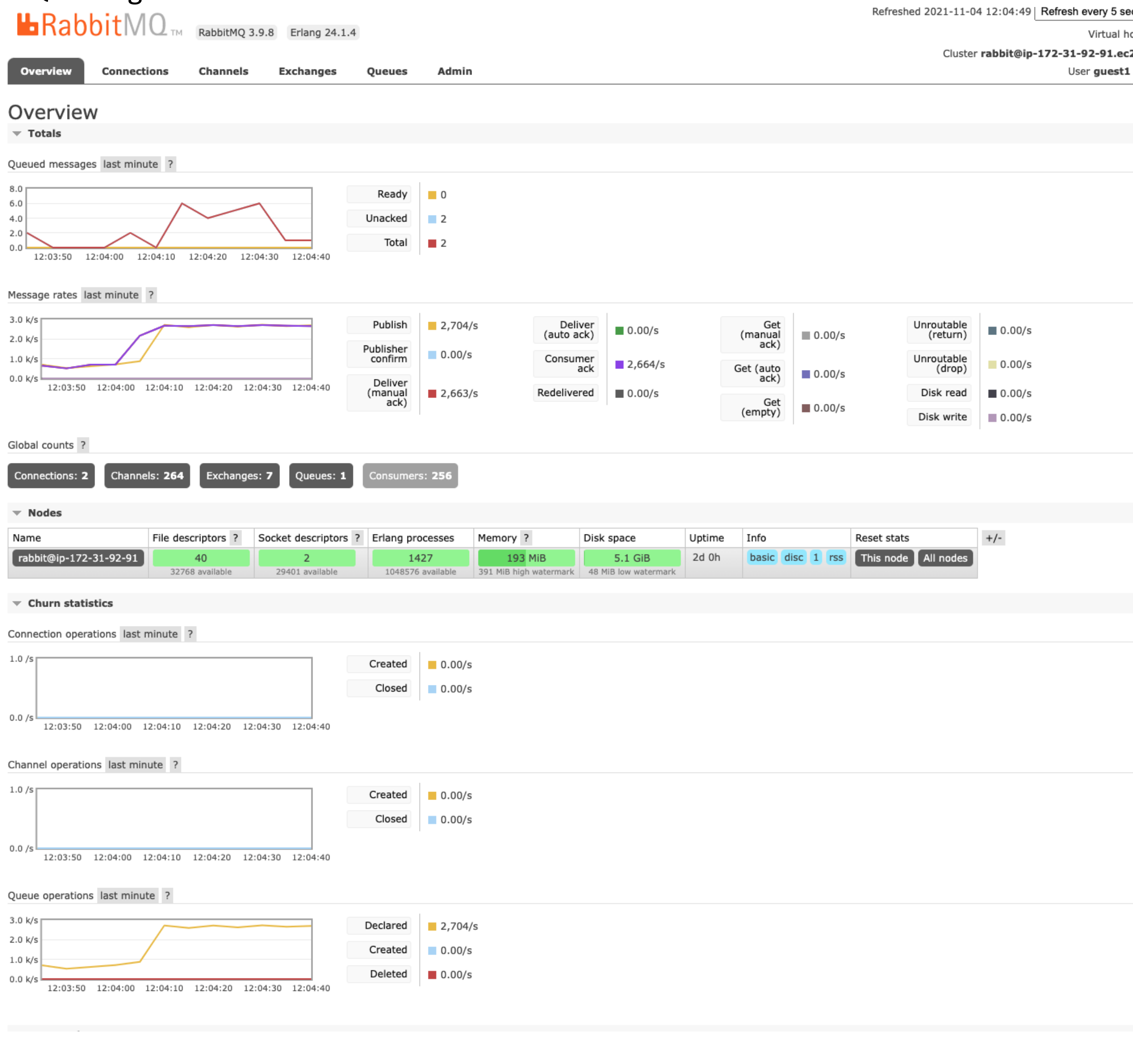
```
/Library/Java/JavaVirtualMachines/tem
Successful requests sent 799840
Unsuccessful requests 0
WallTime 1007452ms
Total throughput per ms 793.9237

Process finished with exit code 0
```

c. 256 Threads

   i. Commands lines:

-numThreads 256 -numSkiers 100000 -address 54.82.5.13:8080

ii. RMQ management console:



RabbitMQ 3.9.8 Erlang 24.1.4

Refreshed 2021-11-04 12:04:49 | Refresh every 5 se

Virtual ho

Cluster **rabbit@ip-172-31-92-91.ec2**

User **guest1**

Overview | Connections | Channels | Exchanges | Queues | Admin

Overview

Totals

Queued messages last minute ?

Ready	0
Unacked	2
Total	2

Message rates last minute ?

Publish	2,704/s	Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s	Unroutable (return)	0.00/s
Publisher confirm	0.00/s	Consumer ack	2,664/s	Get (auto ack)	0.00/s	Unroutable (drop)	0.00/s
Deliver (manual ack)	2,663/s	Redelivered	0.00/s	Get (empty)	0.00/s	Disk read	0.00/s
						Disk write	0.00/s

Global counts ?

Connections: **2** Channels: **264** Exchanges: **7** Queues: **1** Consumers: **256**

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@ip-172-31-92-91	40 32768 available	2 29401 available	1427 1048576 available	193 MiB 391 MiB high watermark	5.1 GiB 48 MiB low watermark	2d 0h	basic disc 1 rss	This node All nodes

Churn statistics

Connection operations last minute ?

Created	0.00/s
Closed	0.00/s

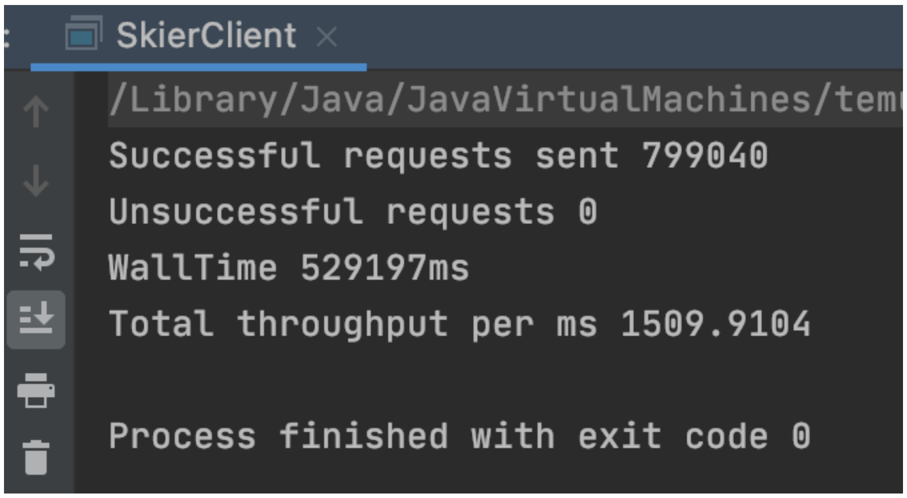
Channel operations last minute ?

Created	0.00/s
Closed	0.00/s

Queue operations last minute ?

Declared	2,704/s
Created	0.00/s
Deleted	0.00/s

iii. Result:



SkierClient

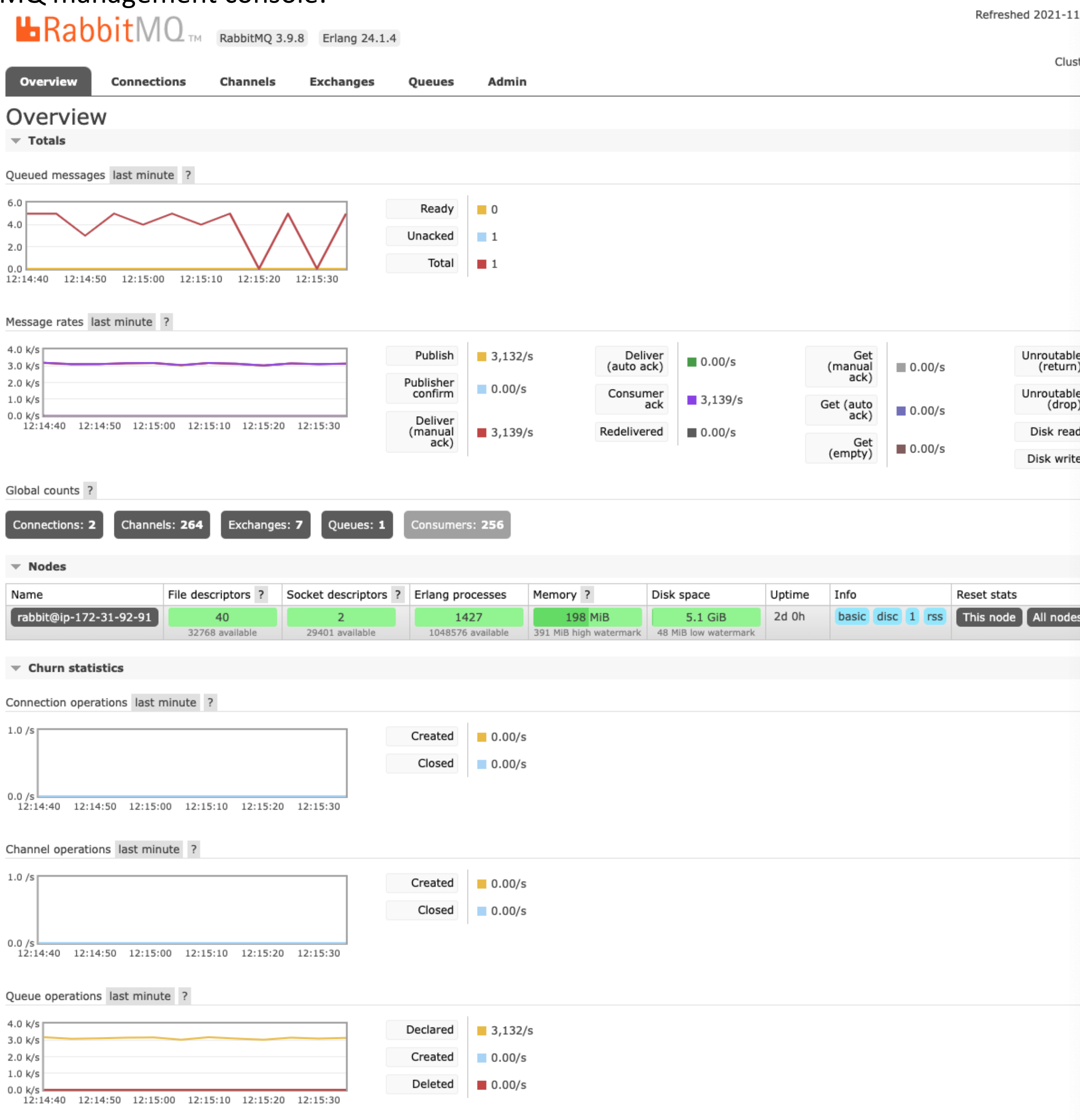
```
/Library/Java/JavaVirtualMachines/tem
Successful requests sent 799040
Unsuccessful requests 0
WallTime 529197ms
Total throughput per ms 1509.9104

Process finished with exit code 0
```

d. 512 Threads

i. Commands lines:

-numThreads 512 -numSkiers 100000 -address 54.82.5.13:8080

ii. RMQ management console:



RabbitMQ TM RabbitMQ 3.9.8 Erlang 24.1.4

Refreshed 2021-11

Clust

Overview Connections Channels Exchanges Queues Admin

Overview

Totals

Queued messages last minute ?

Ready	🟨 0
Unacked	🟦 1
Total	🟥 1

Message rates last minute ?

Publish	🟨 3,132/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	⬜ 0.00/s	Unroutable (return)	
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 3,139/s	Get (auto ack)	🟪 0.00/s	Unroutable (drop)	
Deliver (manual ack)	🟥 3,139/s	Redelivered	⬛ 0.00/s	Get (empty)	🟫 0.00/s	Disk read	
						Disk write	

Global counts ?

Connections: **2** Channels: **264** Exchanges: **7** Queues: **1** Consumers: **256**

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@ip-172-31-92-91	40 32768 available	2 29401 available	1427 1048576 available	198 MiB 391 MiB high watermark	5.1 GiB 48 MiB low watermark	2d 0h	basic disc 1 rss	This node All nodes

Churn statistics

Connection operations last minute ?

Created	🟨 0.00/s
Closed	🟦 0.00/s

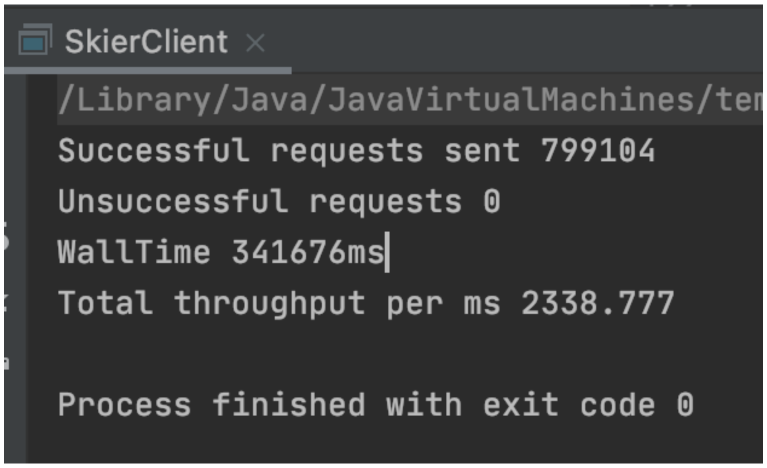
Channel operations last minute ?

Created	🟨 0.00/s
Closed	🟦 0.00/s

Queue operations last minute ?

Declared	🟨 3,132/s
Created	🟦 0.00/s
Deleted	🟥 0.00/s

iii. Result:



```
SkierClient
/Library/Java/JavaVirtualMachines/tem
Successful requests sent 799104
Unsuccessful requests 0
WallTime 341676ms
Total throughput per ms 2338.777

Process finished with exit code 0
```

4. Result Analysis:
   Based on the result, in most of the time, the message queue size is close to zero. Which means our 256-thread consumer is capable to consumer all the message sent by 64, 128, 256 and 512 threads producer, there's barely any message remains in the message queue. When a message was sent by the client, the producer will immediately send it to the queue, and there must be a consumer thread taking the message from its own channel at almost the same time. The testing result also indicates our design is efficient enough.