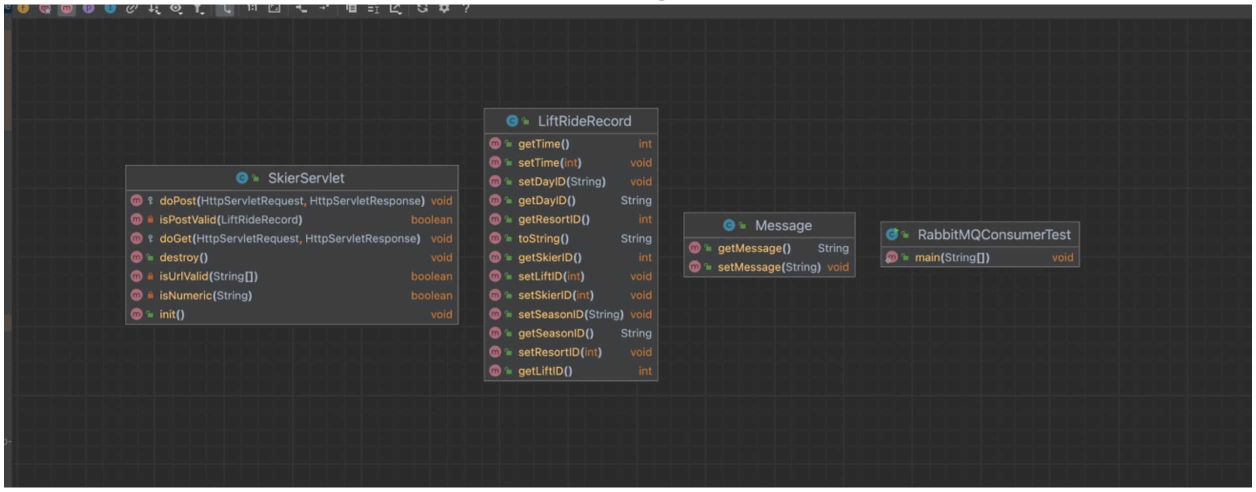
Assignment 3 Zegui Jiang

LIftServer Remain Same with Assignment 2



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

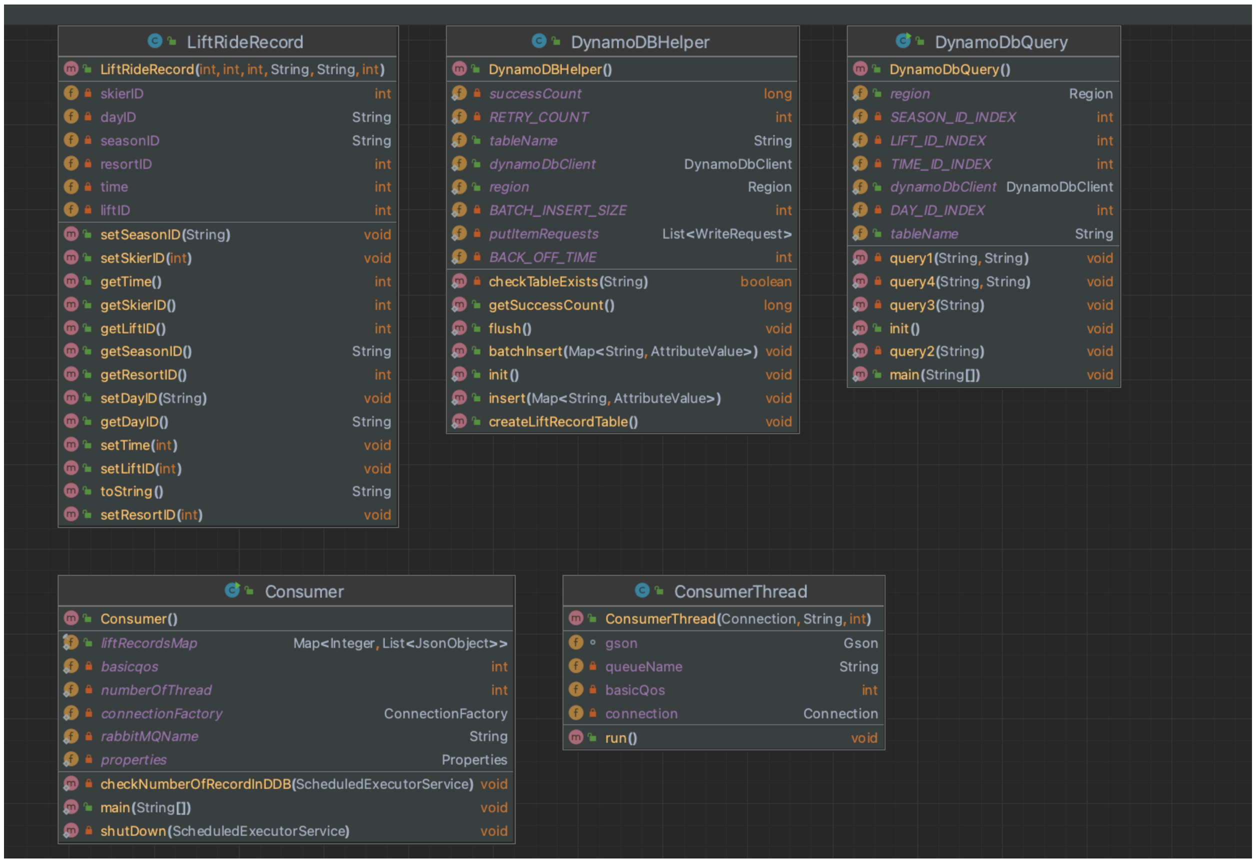
Init(): Reads and configures settings from the `rabbitmq.conf` configuration file found in resources. It extracts the RabbitMQ queue name and the number of channels from the properties file. All channels are stored in a channel pool, which is a Linked Blocking Queue.

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 Lift Record Consumer



LiftRideRecord	
LiftRideRecord(int, int, int, String, String, int)	
skierID	int
dayID	String
seasonID	String
resortID	int
time	int
liftID	int
setSeasonID(String)	void
setSkierID(int)	void
getTime()	int
getSkierID()	int
getLiftID()	int
getSeasonID()	String
getResortID()	int
setDayID(String)	void
getDayID()	String
setTime(int)	void
setLiftID(int)	void
toString()	String
setResortID(int)	void

DynamoDBHelper	
DynamoDBHelper()	
successCount	long
RETRY_COUNT	int
tableName	String
dynamoDbClient	DynamoDbClient
region	Region
BATCH_INSERT_SIZE	int
putItemRequests	List<WriteRequest>
BACK_OFF_TIME	int
checkTableExists(String)	boolean
getSuccessCount()	long
flush()	void
batchInsert(Map<String, AttributeValue>)	void
init()	void
insert(Map<String, AttributeValue>)	void
createLiftRecordTable()	void

DynamoDbQuery	
DynamoDbQuery()	
region	Region
SEASON_ID_INDEX	int
LIFT_ID_INDEX	int
TIME_ID_INDEX	int
dynamoDbClient	DynamoDbClient
DAY_ID_INDEX	int
tableName	String
query1(String, String)	void
query4(String, String)	void
query3(String)	void
init()	void
query2(String)	void
main(String[])	void

Consumer	
Consumer()	
liftRecordsMap	Map<Integer, List<JsonObject>>
basicqos	int
numberOfThread	int
connectionFactory	ConnectionFactory
rabbitMQName	String
properties	Properties
checkNumberOfRecordInDDB(ScheduledExecutorService)	void
main(String[])	void
shutDown(ScheduledExecutorService)	void

ConsumerThread	
ConsumerThread(Connection, String, int)	
gson	Gson
queueName	String
basicQos	int
connection	Connection
run()	void

The RabbitMQ consumer, building upon Assignment 2, has integrated DynamoDB as the database to store lift ride records.

DynamoDBHelper

Init: Sets up the DynamoDbClient with the specified AWS region (Region.US_WEST_2). If initialization fails, it prints an error message.

1 - Create table: Checks if the LiftRideRecordTable table exists. If not, it creates the table with a specific schema designed to support queries related to ski lift rides.

This schema includes: Primary key: **skierID** (HASH) and **liftInfo** (RANGE) for identifying records.
skierID: skier id is partition key because most of query want to check attributes with different skier id
LiftInfo: This is little design in this table. I want to carry most info in to the table. Liftinfo is the concatenate String with symbol ":", we can decouple when we want these information

Global Secondary Index (GSI): ResortAndDay with resortID (HASH) and dayID (RANGE) to support queries on unique skiers visiting a resort on a specific day. This index includes seasonID

and skierID as non-key attributes. The table and index are provisioned with specific read and write capacity units.
Reason to enable GSI with ResortID and DayID, because the query "How many unique skiers visited resort X on day N?" required scan all over resort id, this will result fast scan.

2 - Inserting Records:

Single inserts (insert method): Inserts a single item into the LiftRideRecordTable. If the insertion fails, it prints an error message along with the item details.

Batch inserts (batch Insert and flush methods): Accumulates insert requests and performs a batch insert when the number of accumulated requests reaches a threshold (BATCH_INSERT_SIZE). The flush method processes the batch insert request; handling retries with exponential backoff (BACK_OFF_TIME) in case of unprocessed items. It keeps track of successfully processed items in success Count.

DynamoDbQuery

Since I could not find a good way to use query in AWS console, I choose use java code to implement these four queries:

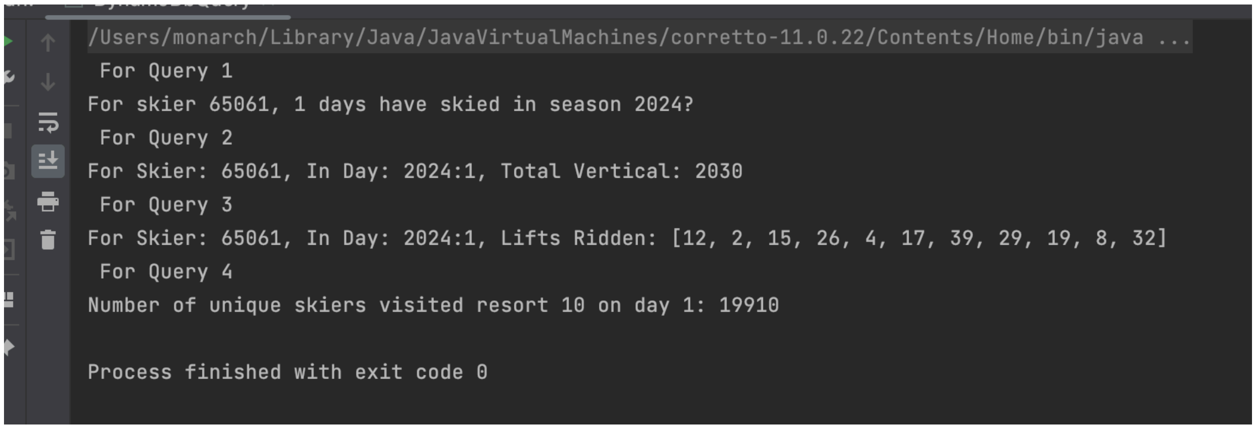
"For skier N, how many days have they skied this season?"

"For skier N, what are the vertical totals for each ski day?" (calculate vertical as liftID*10)

"For skier N, show me the lifts they rode on each ski day"

"How many unique skiers visited resort X on day N?"

The result is:



```
/Users/monarch/Library/Java/JavaVirtualMachines/corretto-11.0.22/Contents/Home/bin/java ...
 For Query 1
For skier 65061, 1 days have skied in season 2024?
 For Query 2
For Skier: 65061, In Day: 2024:1, Total Vertical: 2030
 For Query 3
For Skier: 65061, In Day: 2024:1, Lifts Ridden: [12, 2, 15, 26, 4, 17, 39, 29, 19, 8, 32]
 For Query 4
Number of unique skiers visited resort 10 on day 1: 19910

Process finished with exit code 0
```

Challenge on Dynamo DB deployment and problem solving.

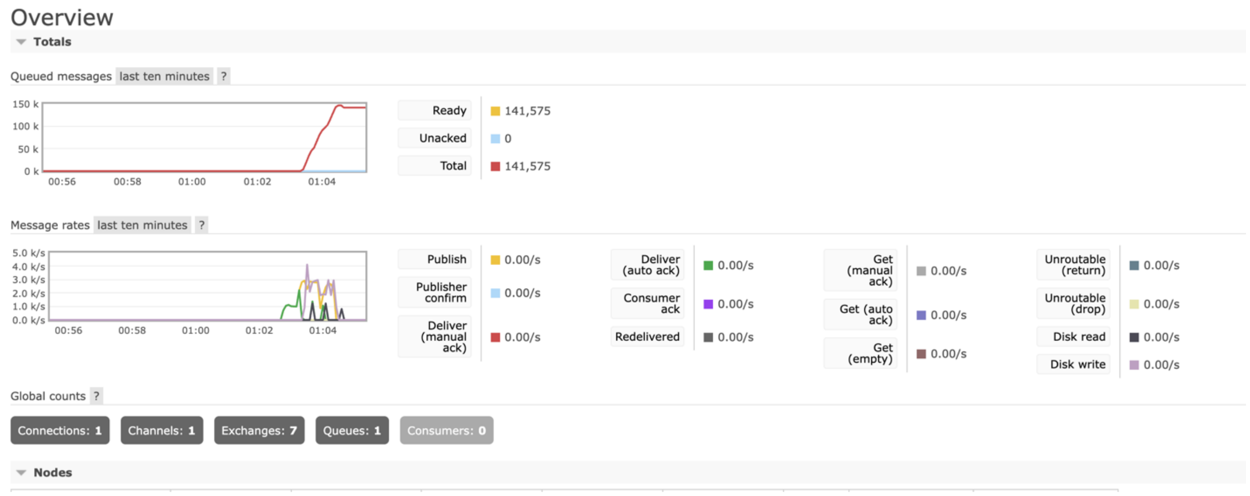
Single insert and batch insert:

When I try using single insert, the speed of writing to the database becomes very slow, and the latency is very high. Therefore, I used batch write logic, which allows writing multiple records at the same time. Moreover, I implemented retry and back-off time logic to enhance stability.

WCU&RCU:

When setting the writeCapacityUnits and readCapacityUnits, I tried many combinations, but none worked well. There would be a lot of backlog in RabbitMQ, and the writing of consumers would also be blocked. Later, I adjusted the WCU to on-demand. DynamoDB will automatically adjust the size and speed of write and read, preventing blockages. RabbitMQ will also be healthier.

Before:



Overview

Totals

Queued messages last ten minutes ?

Ready	🟨 141,575
Unacked	🟦 0
Total	🟥 141,575

Message rates last ten minutes ?

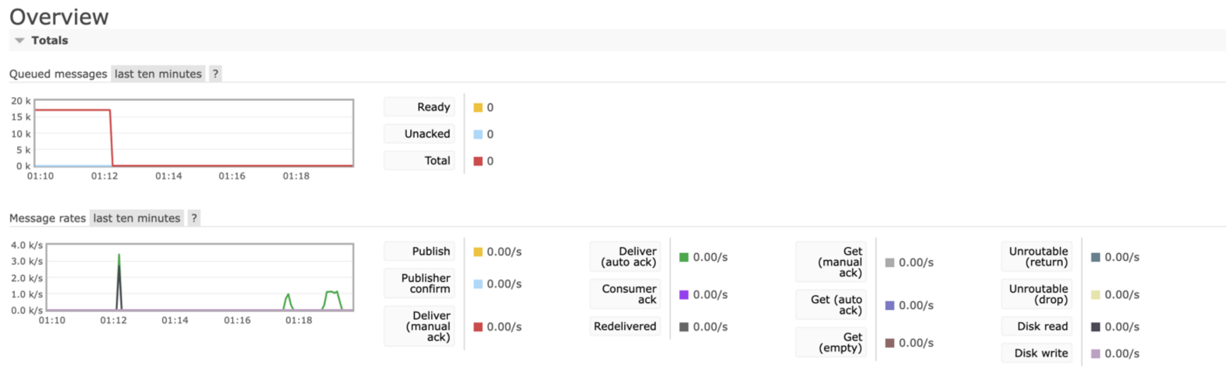
Publish	🟨 0.00/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	⬜ 0.00/s	Unroutable (return)	⬛ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 0.00/s	Get (auto ack)	🟪 0.00/s	Unroutable (drop)	🟨 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	⬛ 0.00/s	Get (empty)	🟫 0.00/s	Disk read	⬛ 0.00/s
						Disk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

Global counts ?

Connections: 1 Channels: 1 Exchanges: 7 Queues: 1 Consumers: 0

Nodes

After:



Overview

Totals

Queued messages last ten minutes ?

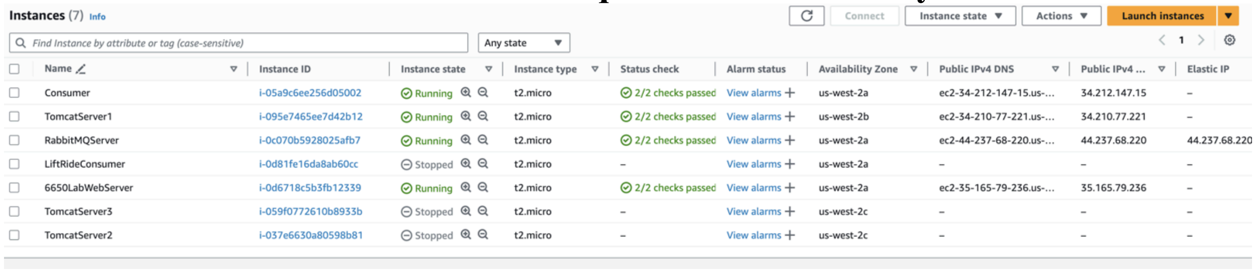
Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last ten minutes ?

Publish	🟨 0.00/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	🟪 0.00/s	Get (auto ack)	🟪 0.00/s	Unroutable (drop)	🟨 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	⬛ 0.00/s	Get (empty)	🟫 0.00/s	Disk read	⬛ 0.00/s
						Disk write	🟪 0.00/s

Deployment Topology:

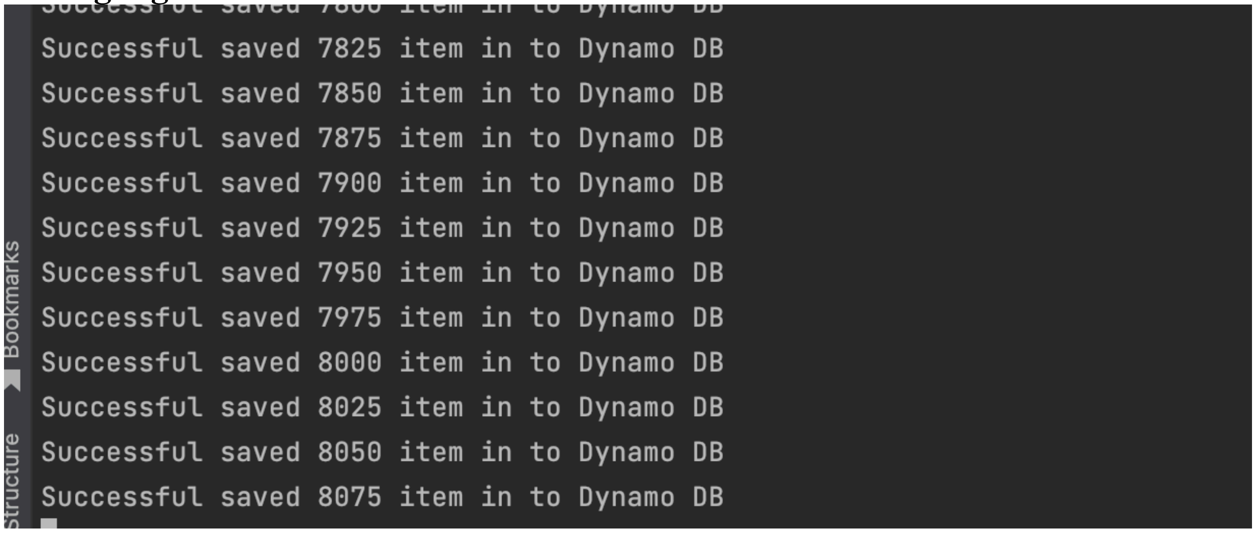
Client -> TomcatServer -> RabbitMq <- Consumer -> DynamoDB



Instances (7)

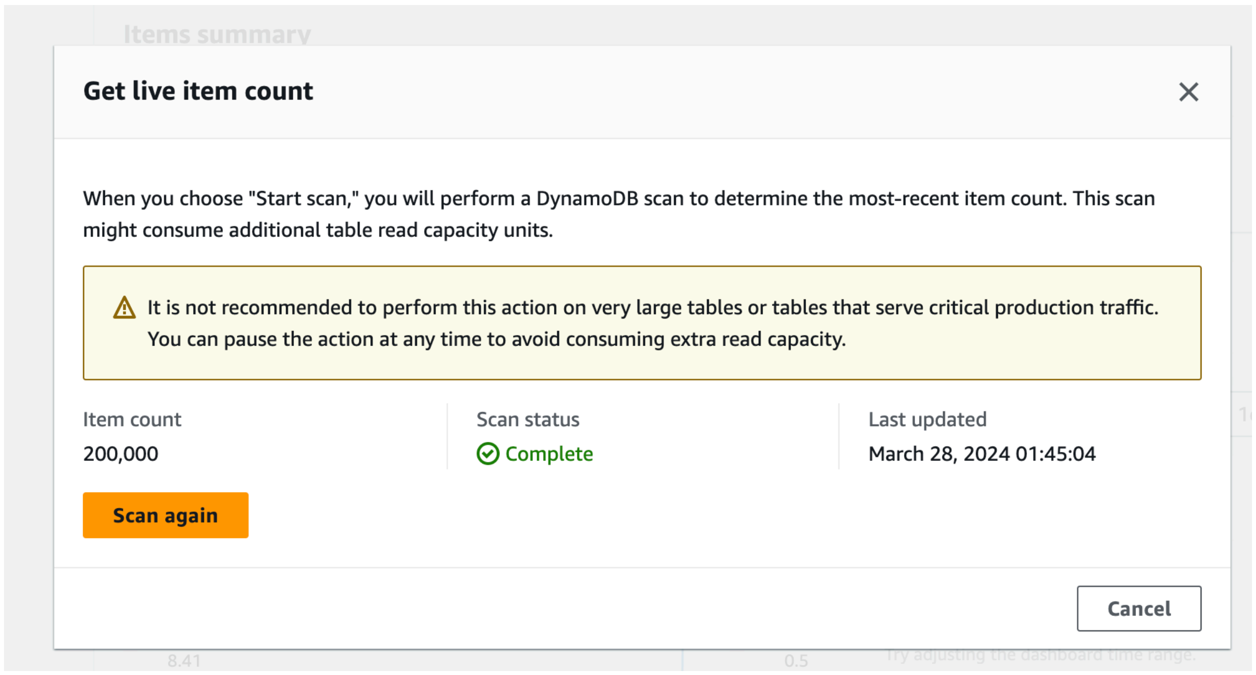
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
Consumer	i-05a9c6ee256d05002	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2a	ec2-34-212-147-15.us-...	34.212.147.15	–
TomcatServer1	i-095e7465ee7d42b12	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2b	ec2-34-210-77-221.us-...	34.210.77.221	–
RabbitMQServer	i-0c070b5928025afb7	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2a	ec2-44-237-68-220.us-...	44.237.68.220	44.237.68.220
LiftRideConsumer	i-0d81fe16da8ab60cc	Stopped	t2.micro	–	View alarms +	us-west-2a	–	–	–
6650LabWebServer	i-0d6718c5b3fb12339	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2a	ec2-35-165-79-236.us-...	35.165.79.236	–
TomcatServer3	i-059f0772610b8933b	Stopped	t2.micro	–	View alarms +	us-west-2c	–	–	–
TomcatServer2	i-037e6630a80598b81	Stopped	t2.micro	–	View alarms +	us-west-2c	–	–	–

Saving log:



```
Successful saved 7825 item in to Dynamo DB
Successful saved 7850 item in to Dynamo DB
Successful saved 7875 item in to Dynamo DB
Successful saved 7900 item in to Dynamo DB
Successful saved 7925 item in to Dynamo DB
Successful saved 7950 item in to Dynamo DB
Successful saved 7975 item in to Dynamo DB
Successful saved 8000 item in to Dynamo DB
Successful saved 8025 item in to Dynamo DB
Successful saved 8050 item in to Dynamo DB
Successful saved 8075 item in to Dynamo DB
```

Saving result:

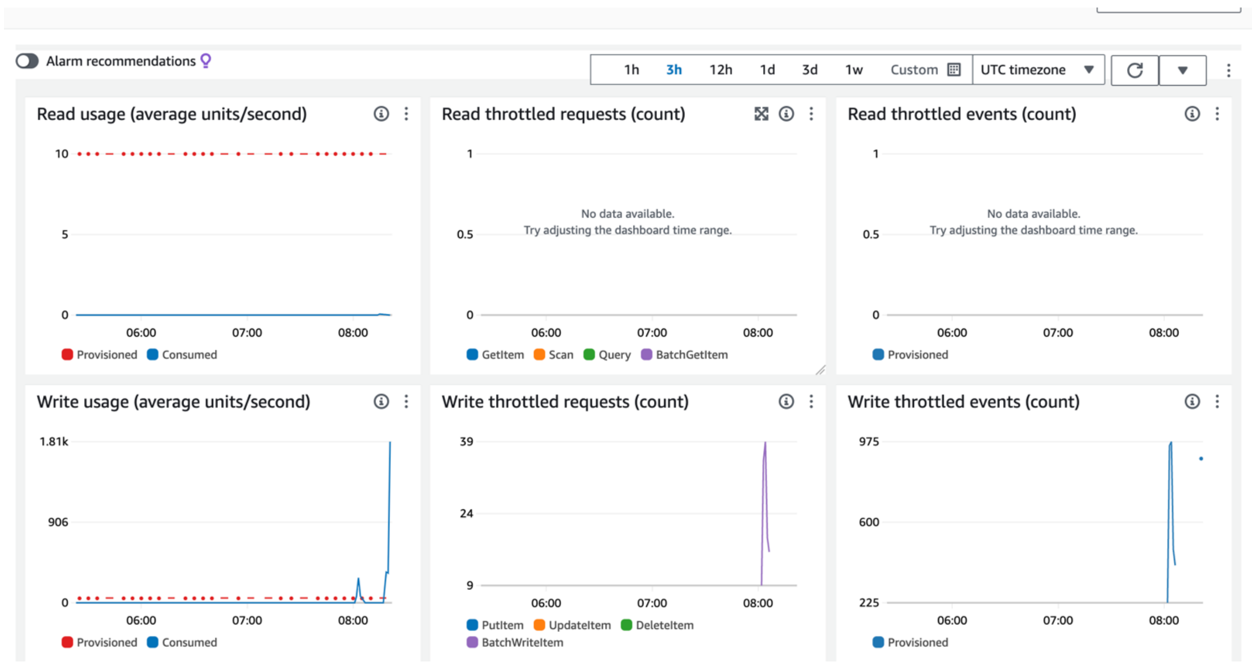


Get live item count

When you choose "Start scan," you will perform a DynamoDB scan to determine the most-recent item count. This scan might consume additional table read capacity units.

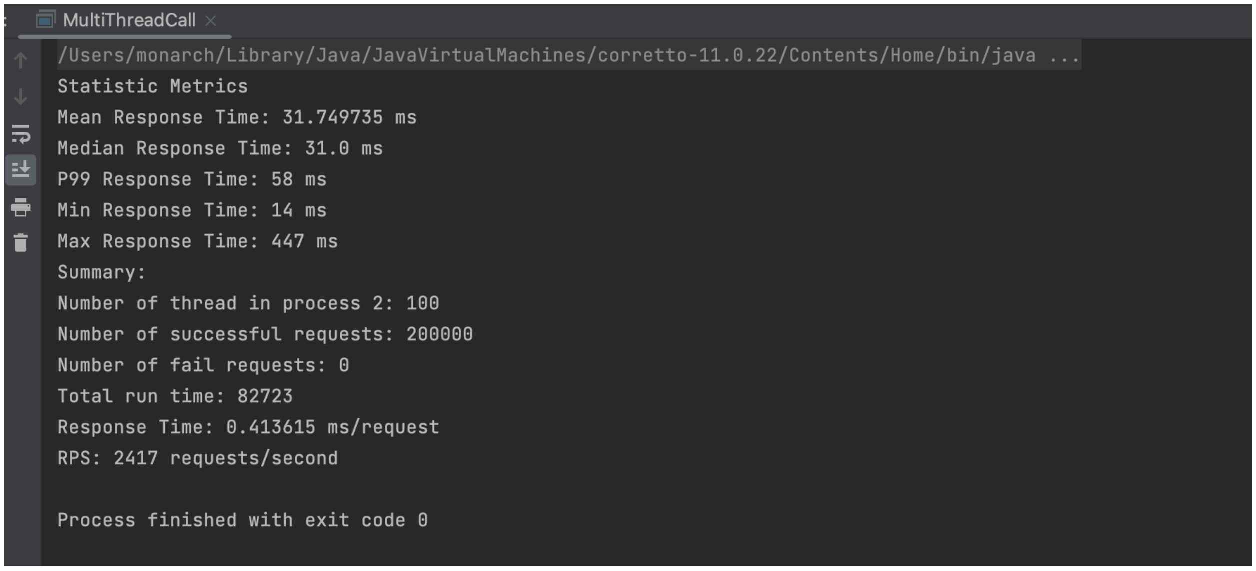
It is not recommended to perform this action on very large tables or tables that serve critical production traffic. You can pause the action at any time to avoid consuming extra read capacity.

Item count	Scan status	Last updated
200,000	Complete	March 28, 2024 01:45:04


Alarm recommendations
1h
3h
12h
1d
3d
1w
Custom
UTC timezone
Read usage (average units/second)
Provisioned
Consumed
Read throttled requests (count)
No data available.
Try adjusting the dashboard time range.
GetItem
Scan
Query
BatchGetItem
Read throttled events (count)
No data available.
Try adjusting the dashboard time range.
Provisioned
Write usage (average units/second)
1.81k
906
Provisioned
Consumed
Write throttled requests (count)
39
24
9
PutItem
UpdateItem
DeleteItem
BatchWriteItem
Write throttled events (count)
975
600
225
Provisioned
06:00
07:00
08:00

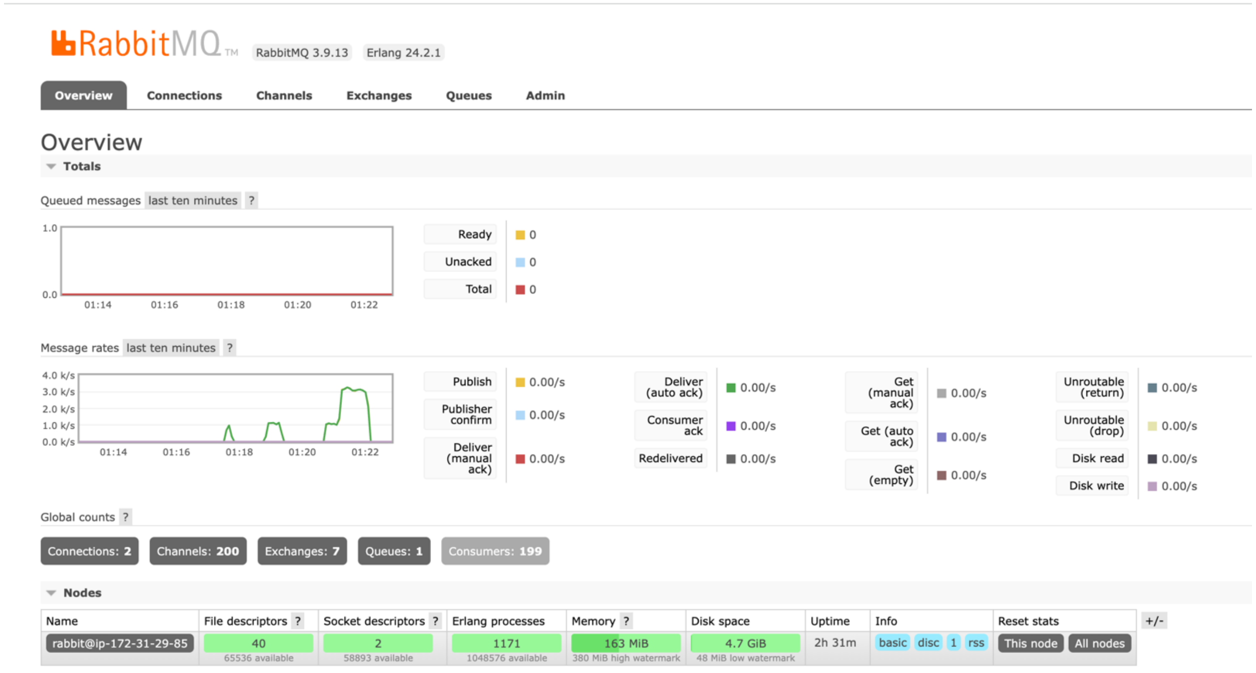
Client test run and RMQ console screen shots showing your best throughput with ideally short, stable queue lengths.

Number of Thread in Client mode 2: 100
Number of Thread in Consumer: 400



```
MultiThreadCall
/Users/monarch/Library/Java/JavaVirtualMachines/corretto-11.0.22/Contents/Home/bin/java ...
Statistic Metrics
Mean Response Time: 31.749735 ms
Median Response Time: 31.0 ms
P99 Response Time: 58 ms
Min Response Time: 14 ms
Max Response Time: 447 ms
Summary:
Number of thread in process 2: 100
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 82723
Response Time: 0.413615 ms/request
RPS: 2417 requests/second

Process finished with exit code 0
```



Overview

Totals

Queued messages last ten minutes ?

Ready	0
Unacked	0
Total	0

Message rates last ten minutes ?

Publish	0.00/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	0.00/s	Get (auto ack)	0.00/s	Unroutable (drop)	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Get (empty)	0.00/s	Disk read	0.00/s
						Disk write	0.00/s

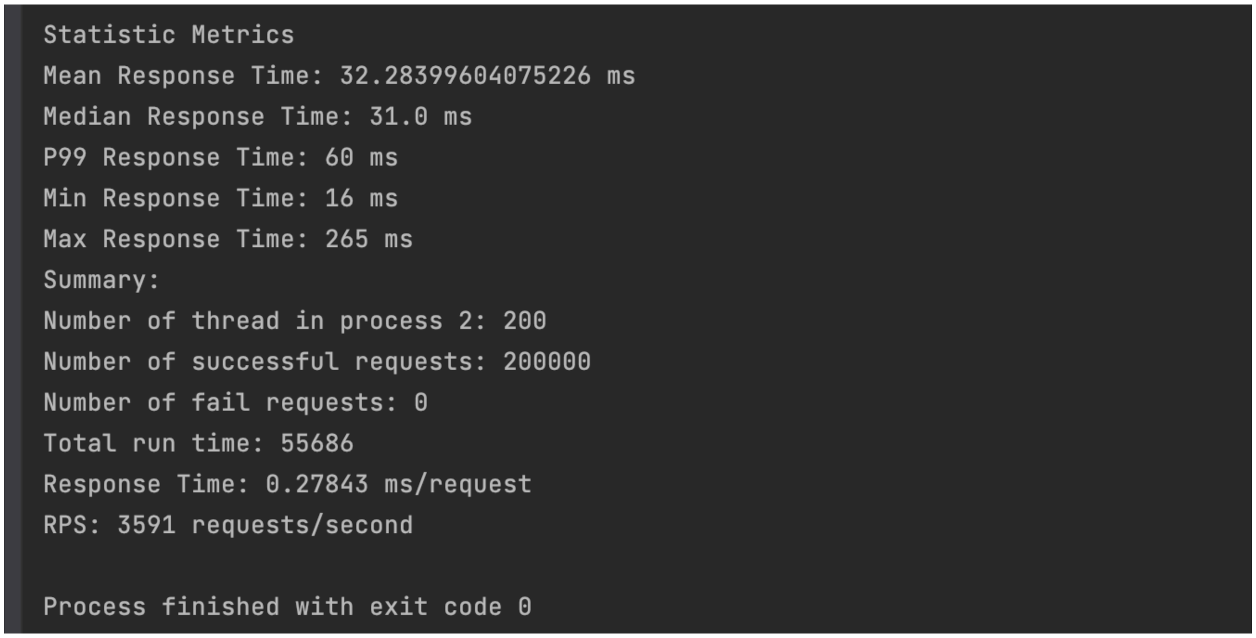
Global counts ?

Connections: 2 | Channels: 200 | Exchanges: 7 | Queues: 1 | Consumers: 199

Nodes

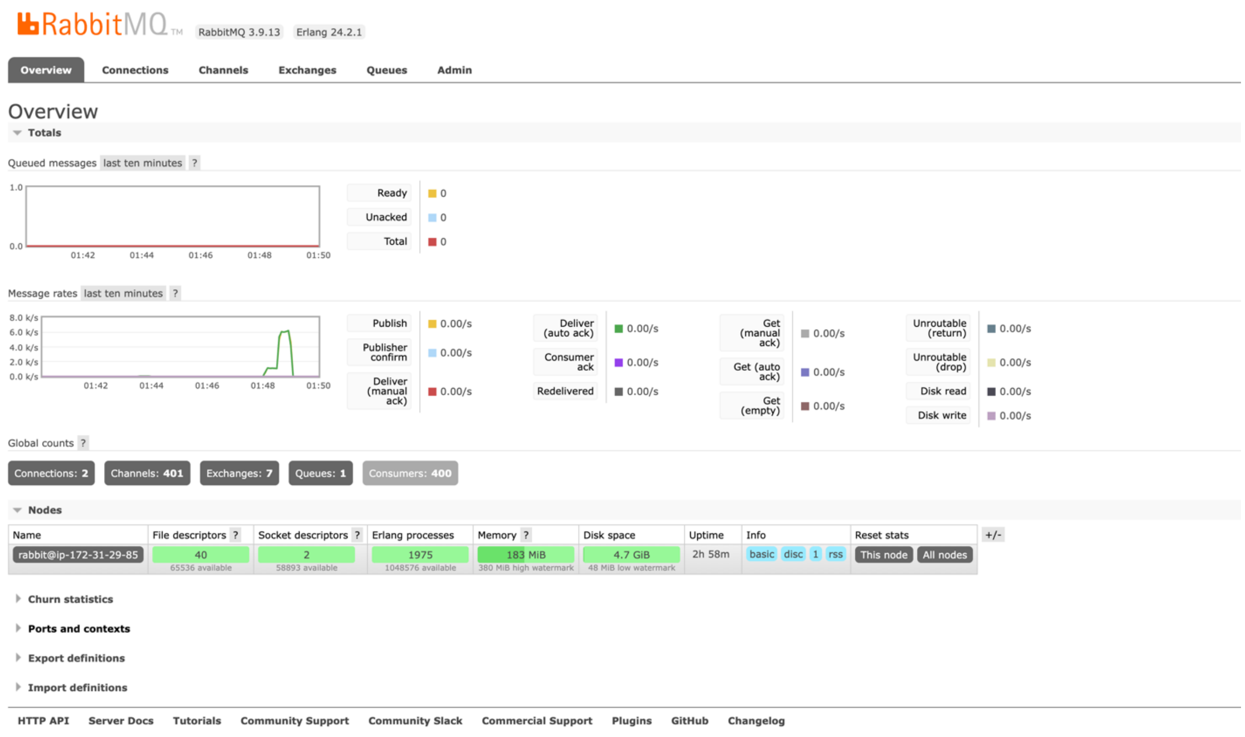
Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	+/-
rabbit@ip-172-31-29-85	40 65536 available	2 58893 available	1171 1048576 available	163 MiB 380 MiB high watermark	4.7 GiB 48 MiB low watermark	2h 31m	basic disc 1 rss	This node All nodes	

Number of Thread in Client mode 2: 200
Number of Thread in Consumer: 400



```
Statistic Metrics
Mean Response Time: 32.28399604075226 ms
Median Response Time: 31.0 ms
P99 Response Time: 60 ms
Min Response Time: 16 ms
Max Response Time: 265 ms
Summary:
Number of thread in process 2: 200
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 55686
Response Time: 0.27843 ms/request
RPS: 3591 requests/second

Process finished with exit code 0
```



Overview

Totals

Queued messages last ten minutes ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last ten minutes ?

Publish	🟨 0.00/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	🟪 0.00/s	Get (auto ack)	🟦 0.00/s	Unroutable (drop)	🟨 0.00/s
Deliver (manual ack)	🟥 0.00/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: 401 Exchanges: 7 Queues: 1 Consumers: 400

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@ip-172-31-29-85	40 65536 available	2 58893 available	1975 1048576 available	183 MiB 380 MiB high watermark	4.7 GiB 48 MiB low watermark	2h 58m	basic disc 1 rss	This node All nodes

Churn statistics

Ports and contexts

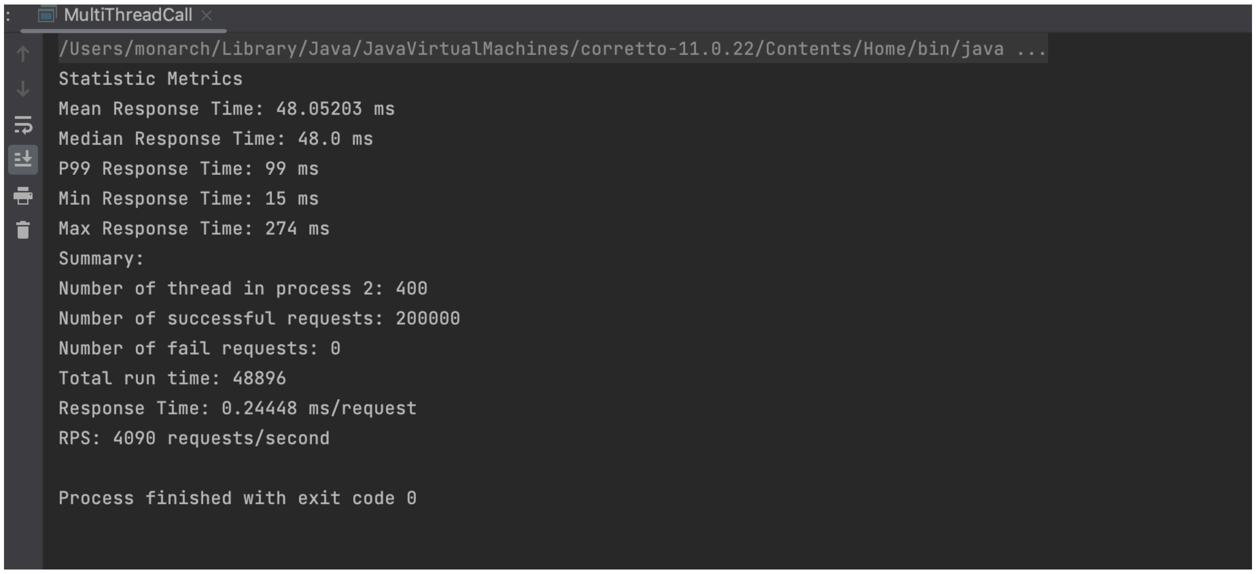
Export definitions

Import definitions

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

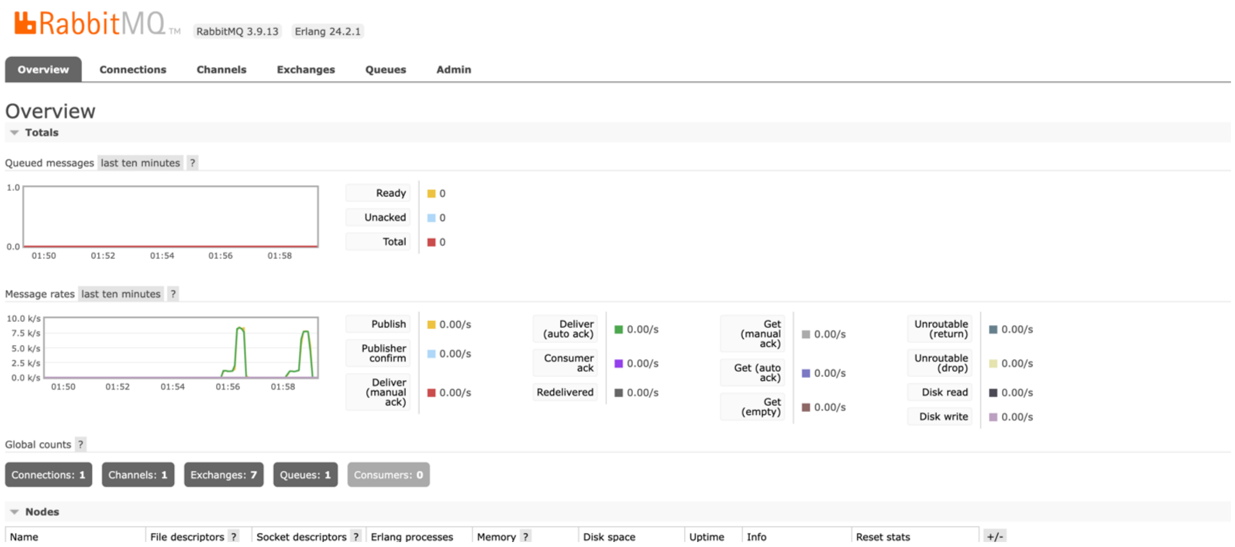
Number of Thread in Client mode 2: 400

Number of Thread in Consumer: 400



```
/Users/monarch/Library/Java/JavaVirtualMachines/corretto-11.0.22/Contents/Home/bin/java ...
Statistic Metrics
Mean Response Time: 48.05203 ms
Median Response Time: 48.0 ms
P99 Response Time: 99 ms
Min Response Time: 15 ms
Max Response Time: 274 ms
Summary:
Number of thread in process 2: 400
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 48896
Response Time: 0.24448 ms/request
RPS: 4090 requests/second

Process finished with exit code 0
```



Overview

Totals

Queued messages last ten minutes ?

Ready	0
Unacked	0
Total	0

Message rates last ten minutes ?

Publish	0.00/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	0.00/s	Get (auto ack)	0.00/s	Unroutable (drop)	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Get (empty)	0.00/s	Disk read	0.00/s
						Disk write	0.00/s

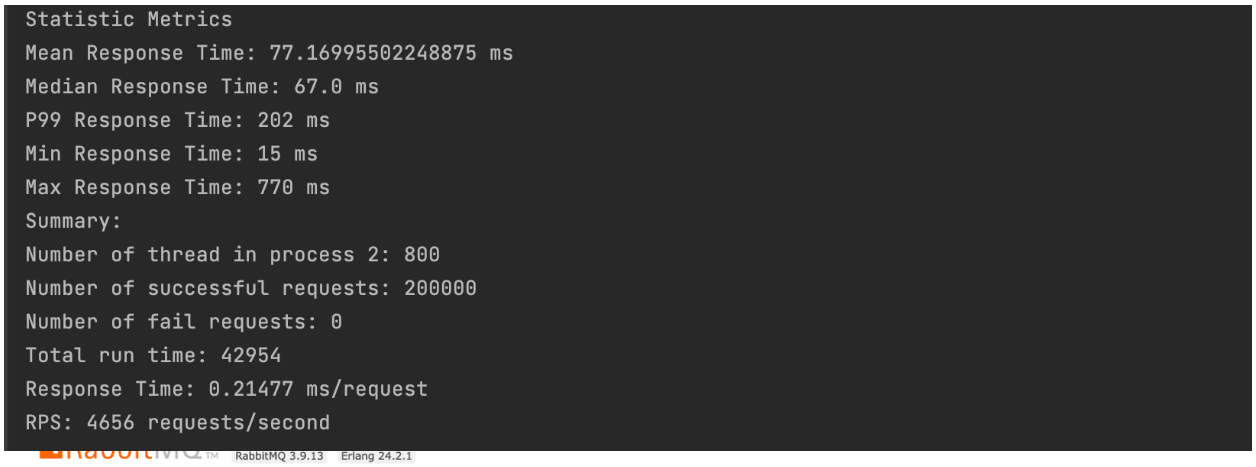
Global counts ?

Connections: 1 Channels: 1 Exchanges: 7 Queues: 1 Consumers: 0

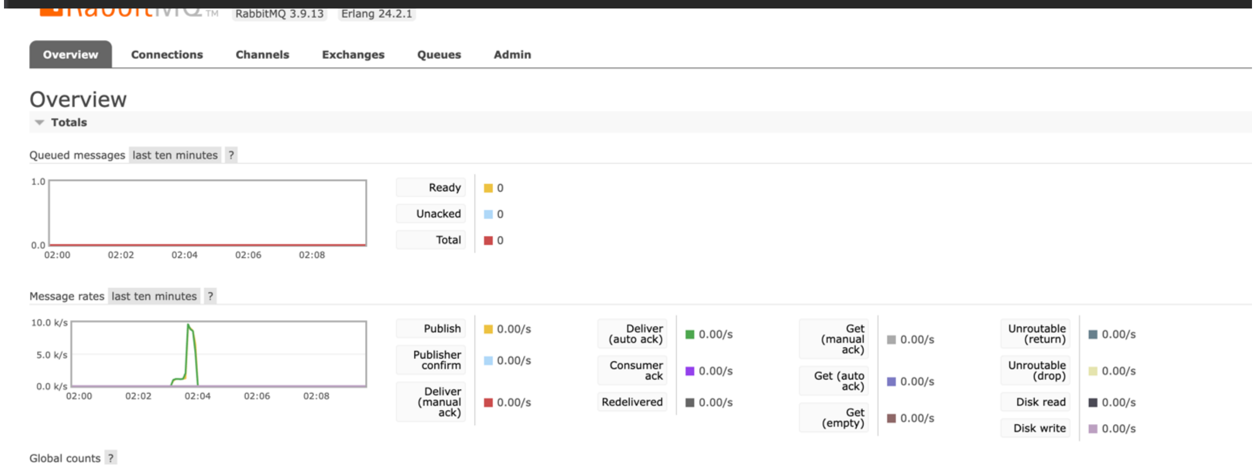
Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	+/-

Number of Thread in Client mode 2: 800
Number of Thread in Consumer: 400



```
Statistic Metrics
Mean Response Time: 77.16995502248875 ms
Median Response Time: 67.0 ms
P99 Response Time: 202 ms
Min Response Time: 15 ms
Max Response Time: 770 ms
Summary:
Number of thread in process 2: 800
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 42954
Response Time: 0.21477 ms/request
RPS: 4656 requests/second
```



Overview

Totals

Queued messages last ten minutes ?

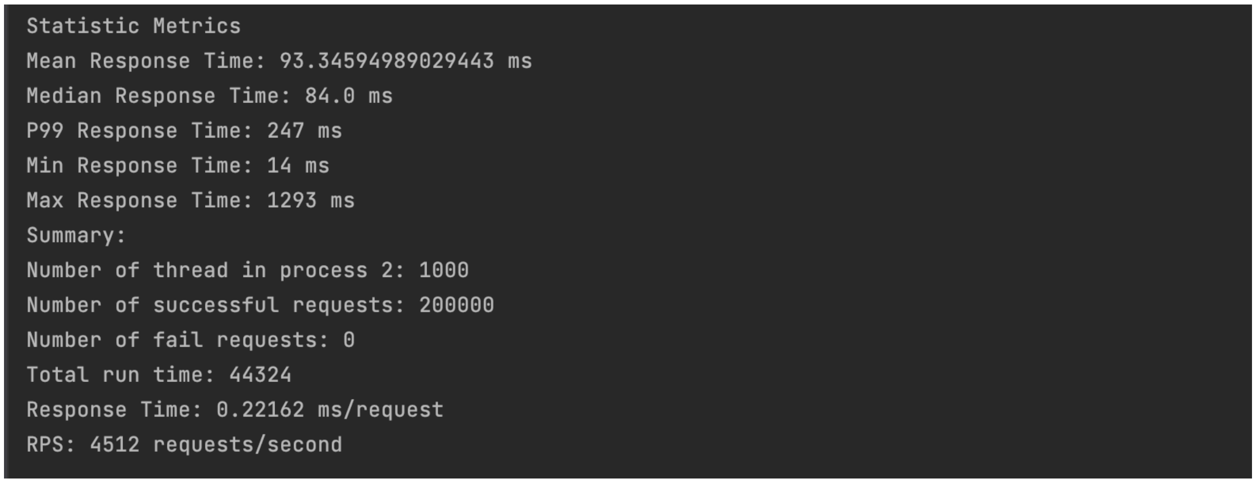
Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last ten minutes ?

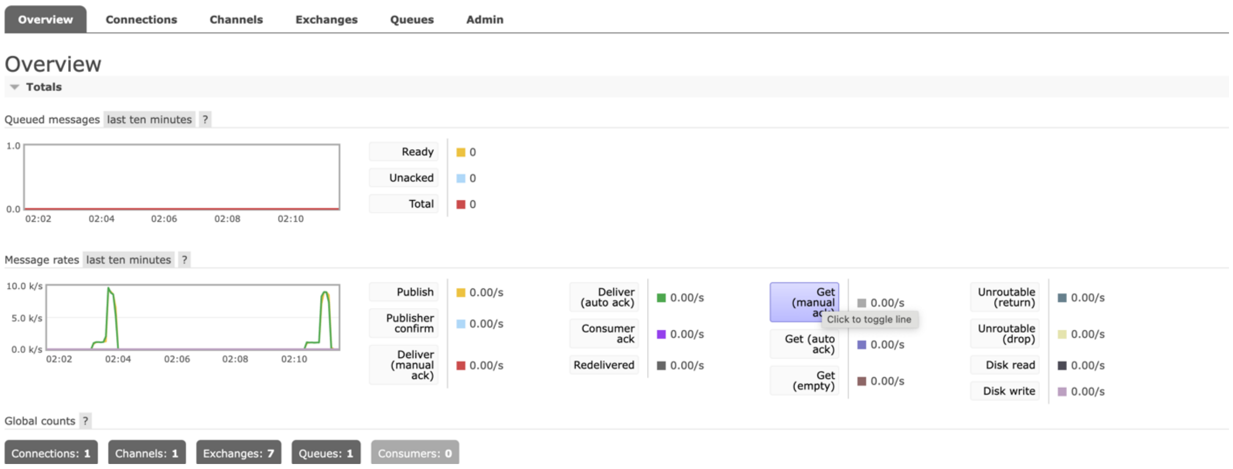
Publish	🟨 0.00/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	■ 0.00/s	Unroutable (return)	■ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 0.00/s	Get (auto ack)	🟦 0.00/s	Unroutable (drop)	🟨 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	■ 0.00/s	Get (empty)	🟫 0.00/s	Disk read	■ 0.00/s
						Disk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

Global counts ?

Number of Thread in Client mode 2: 1000
Number of Thread in Consumer: 400



```
Statistic Metrics
Mean Response Time: 93.34594989029443 ms
Median Response Time: 84.0 ms
P99 Response Time: 247 ms
Min Response Time: 14 ms
Max Response Time: 1293 ms
Summary:
Number of thread in process 2: 1000
Number of successful requests: 200000
Number of fail requests: 0
Total run time: 44324
Response Time: 0.22162 ms/request
RPS: 4512 requests/second
```



Overview

Totals

Queued messages last ten minutes ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last ten minutes ?

Publish	🟨 0.00/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	🟪 0.00/s	Get (auto ack)	🟦 0.00/s	Unroutable (drop)	🟨 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	⬛ 0.00/s	Get (empty)	🟫 0.00/s	Disk read	⬛ 0.00/s
						Disk write	🟪 0.00/s

Global counts ?

Connections: **1** Channels: **1** Exchanges: **7** Queues: **1** Consumers: **0**