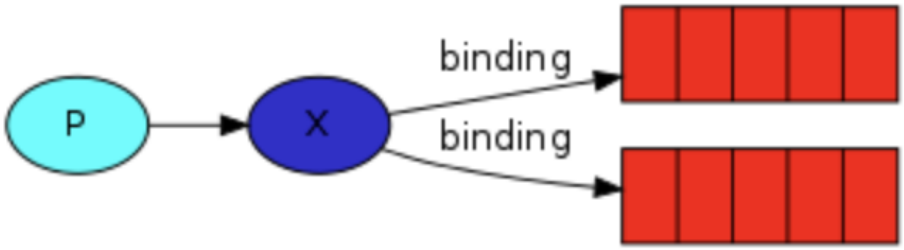
HW 3 Report - Linni Cai

Github URL:

https://github.com/linni-cai-lc/CS6650_Distributed_System/tree/main/hw3

Design:



In this assignment, since we have two consumers, one for skiers, one for resorts, but they share the same data from the server, so I chose RMQ publish/subscribe pattern, the server publishes, two consumers subscribe. The whole workflow is that client serves as producer, it sends plenty of posts to server, the server delivers results to consumers.

Database design is based on Redis key/value structure, each consumer stores the data in its instance's Redis storage.

- Skier Consumer: I created skierId + liftTime for skier consumer's uuid, since we know that a skier can only ski one time at the same liftTime, so this key combo will be unique, and doesn't overlap with other results.
- Resort Consumer: I created dayId + skierId + liftTime for resort consumer's uuid, similarly, the latter two can be unique, since we might need index later, dayId will be necessary to search as index keyword, so I add it into the key combo.
- The value for both is the same, it is a LiftRide JSON format string object, it can be easily converted back to the object for later usage.

Process:

- created 4 EC2 instance
    - 1 Linux instance running the server
        - provides with the skier API functionality
        - connect to load balancer
        - send messages to the queue
    - 2 Linux instance running the consumer

        - 1 for skier consumer
        - 1 for resort consumer
        - receive messages from the queue
    - 1 Ubuntu instance running the RabbitMQ server
        - owns the queue and store messages

Name	Instance ID
Linux (Server)	i-0e965884ba592ab77
Ubuntu (RabbitMQ)	i-066a6081de2958826
Linux (Consumer_Resort)	i-0c1485ce96d70d50a
Linux (Consumer_Skier)	i-0285065fef21ceb1c

Results:

The experiments are based on 20000 skiers, 40 lifts. Overall the queue size is below 100.

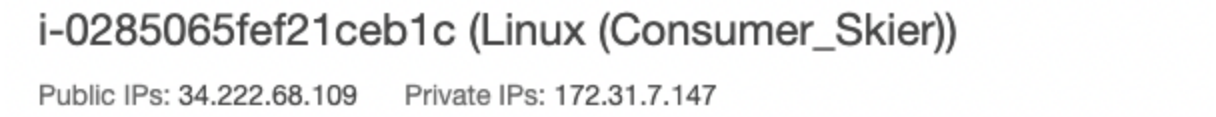
My mitigation strategy is to add a **circuit breaker**, I added it on the Client side, to limit the speed of POST generation. Specifically, it will hold POST when there is a specific number of POST sent to the server already, and restart to send POST when partial of previous POST finishes, finally sending the rest of required numbers of POST.

- Compared to the experiment **without circuit breaker,** the experiment with **circuit breaker** has a smaller queue size, approximately half of queue size.
- Compared to the experiment with **128 client threads**, the experiment with **256 client threads** has a smaller queue size.

Step 1 Skiers

- Command window, use **java -jar consumer_skier.jar**

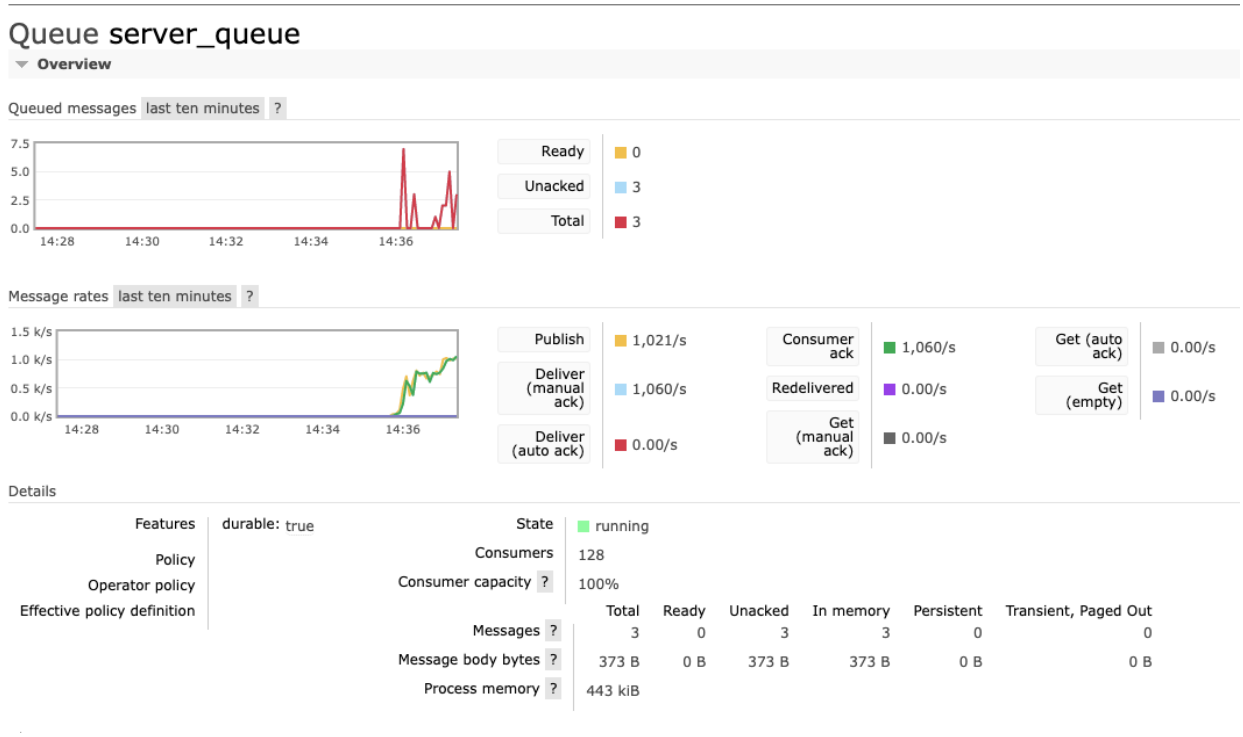
```
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
```



- RMQ management window for queue size
- 128 client threads

```
------  PART 1  ------
number of successful requests sent: 159977
number of unsuccessful requests: 0
the total run time for all phases to complete: 197158
the total throughput in requests per second: 0

------  PART 2  ------
mean response time (millisecs): 109
median response time (millisecs): 72
throughput: 0
p99 (99th percentile) response time: 393
min response time (millisecs): 11
max response time (millisecs): 7774
```

- The queue size range is 0 - 8, message rate is send/receive = 1021 / 1060 = 0.96



- 256 client threads

```
------  PART 1  ------
number of successful requests sent: 160420
number of unsuccessful requests: 0
```

```
the total run time for all phases to complete: 191333
the total throughput in requests per second: 0

------  PART 2  ------
mean response time (millisecs): 214
median response time (millisecs): 185
throughput: 0
p99 (99th percentile) response time: 859
min response time (millisecs): 11
max response time (millisecs): 2251
```

- The queue size range is 0 - 225, message rate is send/receive = 1035 / 1007 = 1.03

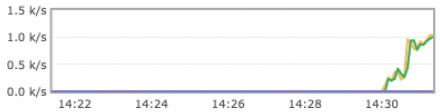
Queue server_queue

Overview

Queued messages last ten minutes ?

Ready	0
Unacked	105
Total	105

Message rates last ten minutes ?



Publish	1,035/s	Consumer ack	1,007/s	Get (auto ack)	0.00/s
Deliver (manual ack)	1,025/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

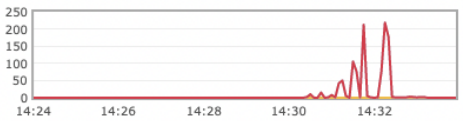
Features	durable: true	State	running
Policy		Consumers	128
Operator policy		Consumer capacity ?	100%
Effective policy definition			

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	105	0	105	105	0	0
Message body bytes ?	13 kiB	0 B	13 kiB	13 kiB	0 B	0 B
Process memory ?	443 kiB					

Queue server_queue

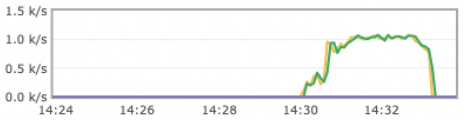
Overview

Queued messages last ten minutes ?



Ready	0
Unacked	0
Total	0

Message rates last ten minutes ?



Publish	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

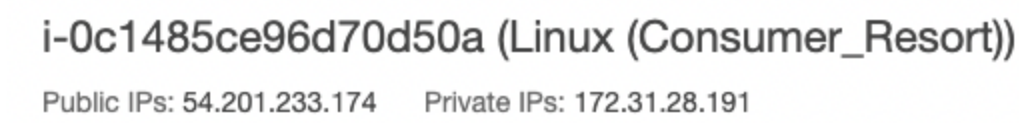
Features	durable: true	State	idle
Policy		Consumers	128
Operator policy		Consumer capacity ?	100%
Effective policy definition			

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	0	0	0	0	0	0
Message body bytes ?	0 B	0 B	0 B	0 B	0 B	0 B
Process memory ?	89 kiB					

Step 2 Resorts

- Command window, use **java -jar consumer_resort.jar**

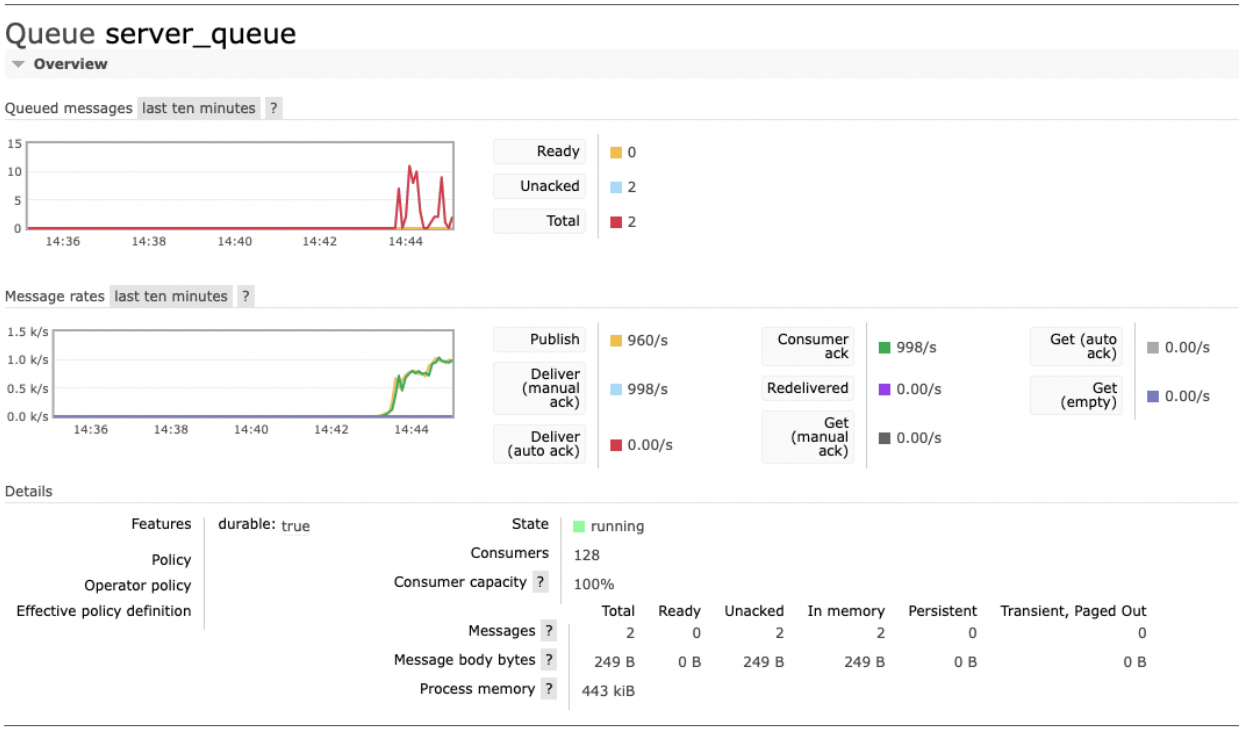
```
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
[*] Waiting for messages. To exit press CTRL+C
```

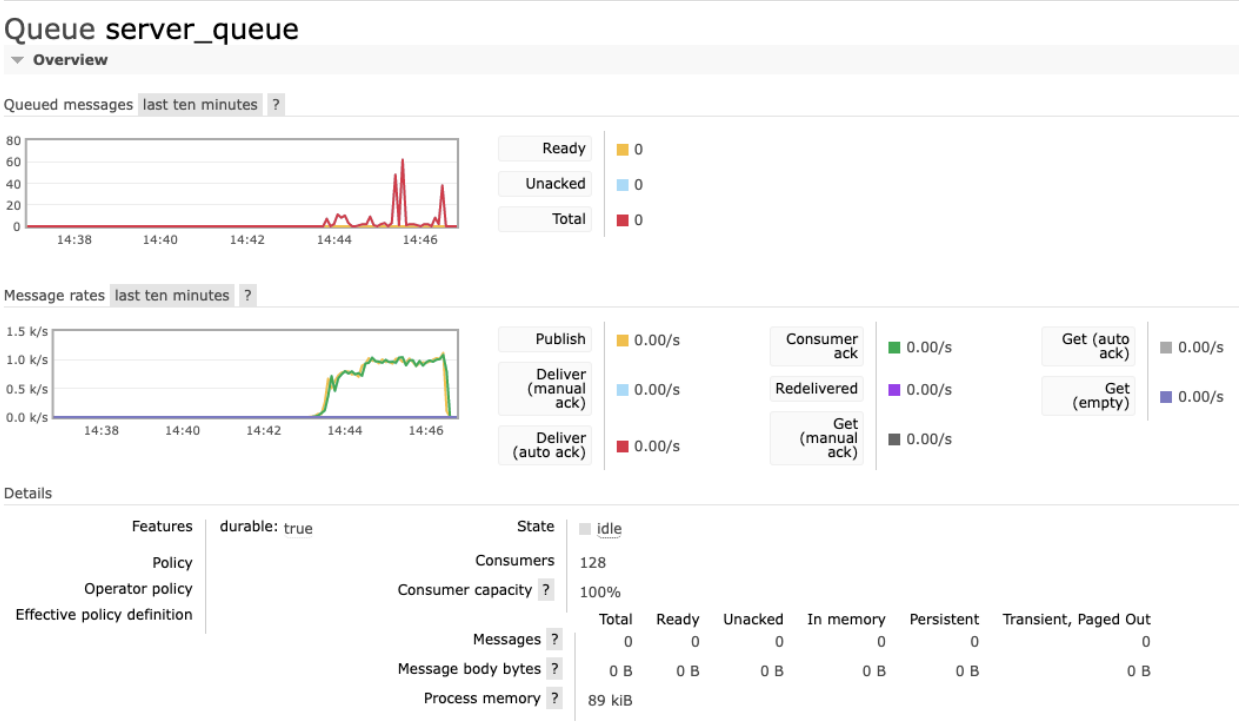


- RMQ management window for queue
- 128 client threads

```
------  PART 1  ------
number of successful requests sent: 159977
number of unsuccessful requests: 0
the total run time for all phases to complete: 204252
the total throughput in requests per second: 0

------  PART 2  ------
mean response time (millisecs): 114
median response time (millisecs): 76
throughput: 0
p99 (99th percentile) response time: 403
min response time (millisecs): 11
max response time (millisecs): 8234
```

- The queue size range is 0 - 65, message rate is send/receive = 960 / 998 = 0.96



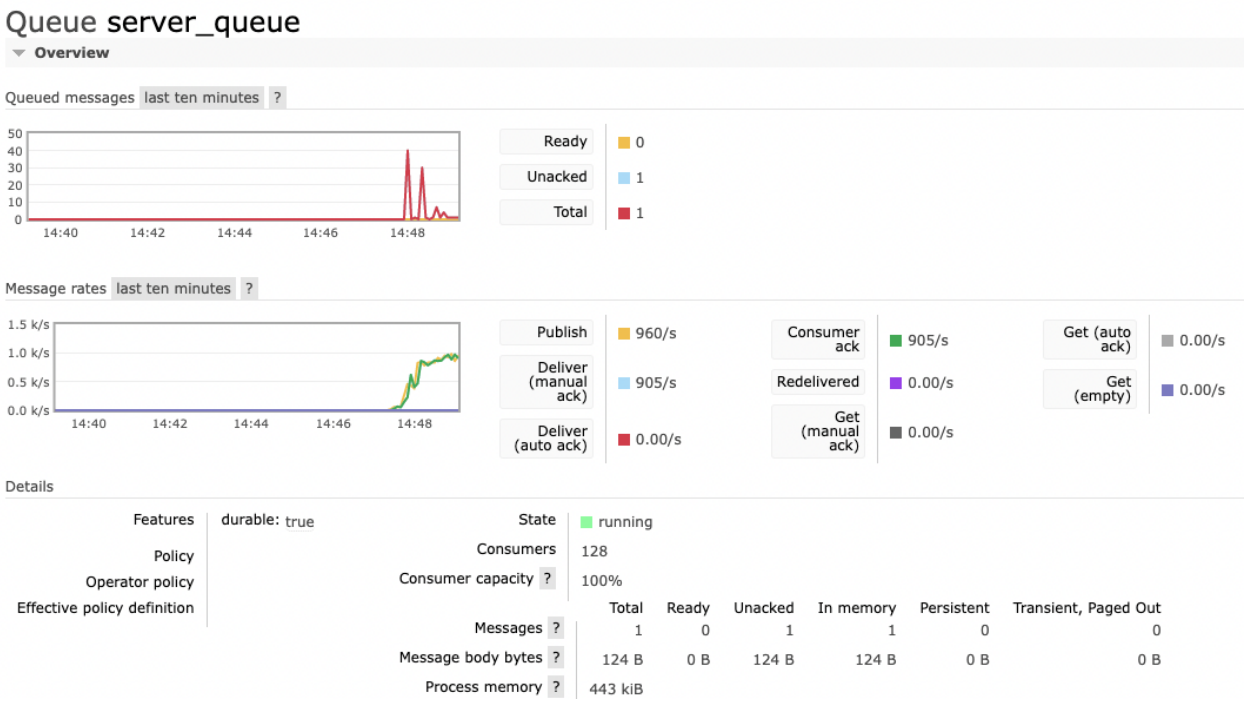


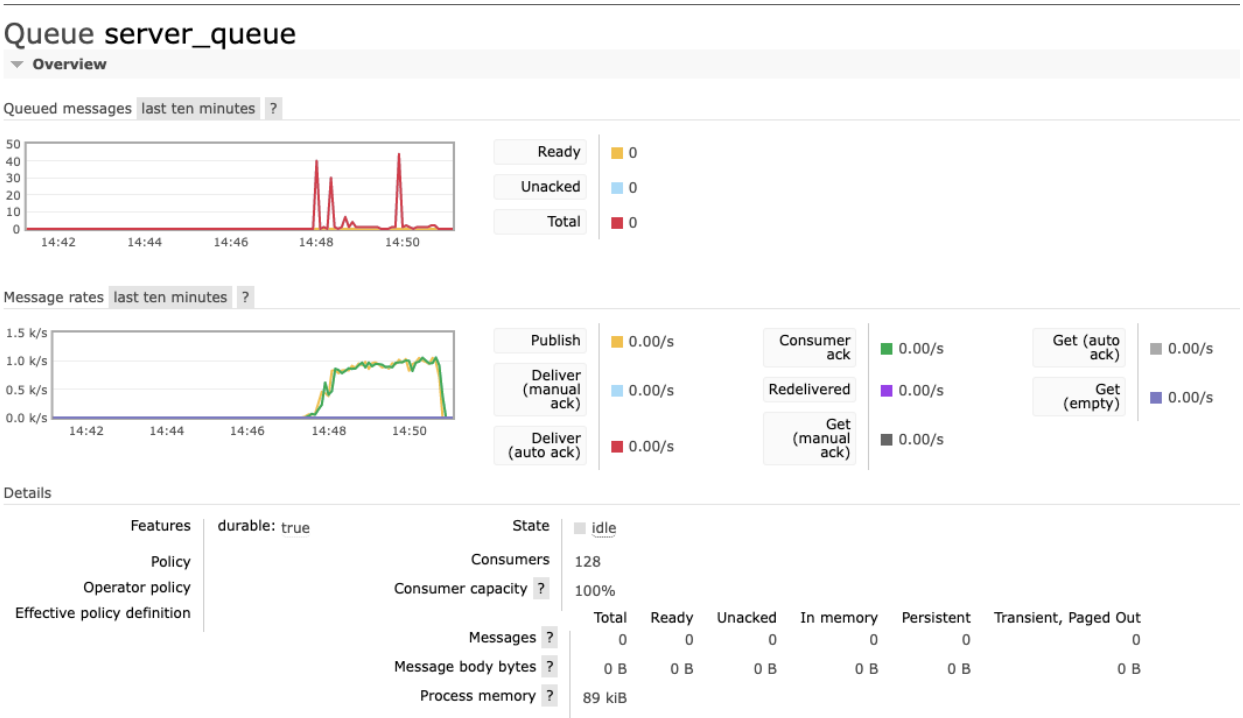
- 256 client threads

```
------  PART 1  ------
number of successful requests sent: 160420
number of unsuccessful requests: 0
the total run time for all phases to complete: 211749
```

```
the total throughput in requests per second: 0

------  PART 2  ------
mean response time (millisecs): 231
median response time (millisecs): 190
throughput: 0
p99 (99th percentile) response time: 1014
min response time (millisecs): 11
max response time (millisecs): 11390
```

- The queue size range is 0 - 45, message rate is send/receive = 960 / 905 = 1.06





Step 3 Both

In two EC2 instances, one for skier jar, one for resort jar, start together

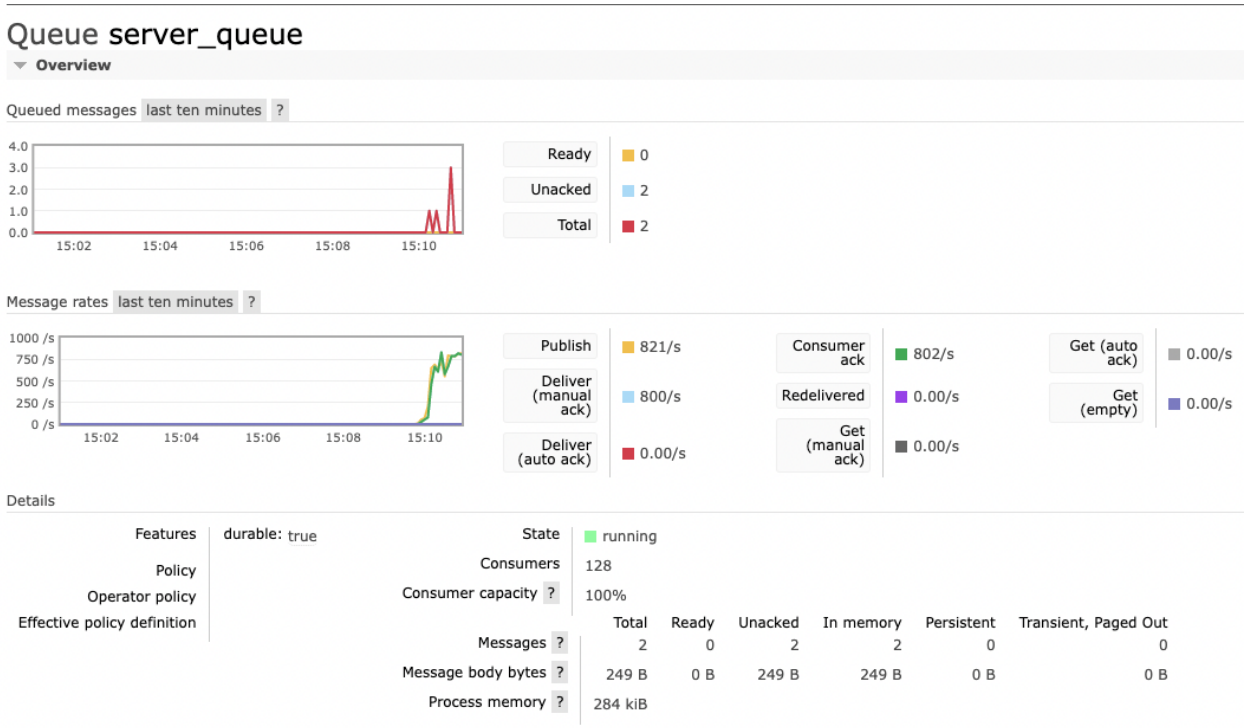
Without circuit breaker

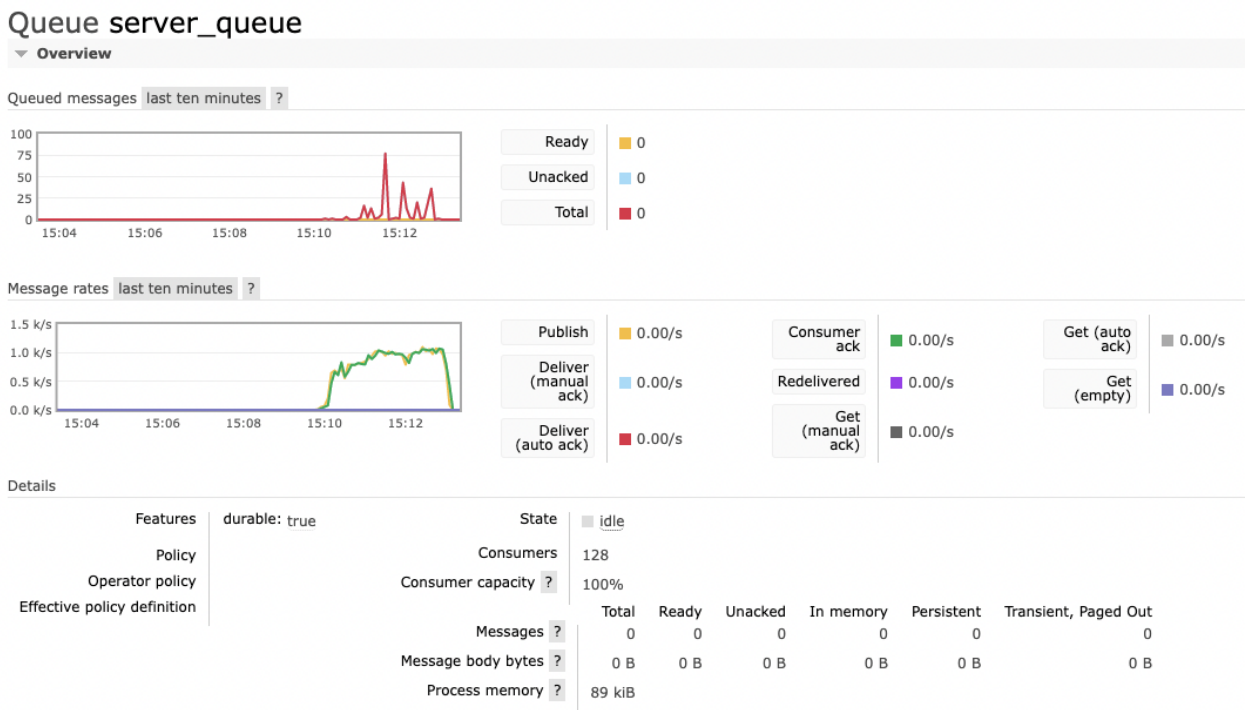
- run both consumers: skiers and resorts
- 128 client threads

```
------  PART 1  ------
number of successful requests sent: 159977
number of unsuccessful requests: 0
the total run time for all phases to complete: 198027
the total throughput in requests per second: 0
```

```
------  PART 2  ------
mean response time (millisecs): 113
median response time (millisecs): 73
throughput: 0
p99 (99th percentile) response time: 422
min response time (millisecs): 11
max response time (millisecs): 6556
```

- The queue size range is 0 - 80, message rate is send/receive = 821 / 802 = 1.02





- 256 client threads

```
------  PART 1  ------
number of successful requests sent: 160420
number of unsuccessful requests: 0
the total run time for all phases to complete: 215961
the total throughput in requests per second: 0

------  PART 2  ------
mean response time (millisecs): 228
median response time (millisecs): 184
throughput: 0
p99 (99th percentile) response time: 964
min response time (millisecs): 11
max response time (millisecs): 13540
```

- The queue size range is 0 - 15, message rate is send/receive = 979 / 947 = 1.03

Queue server_queue

Overview

Queued messages last ten minutes ?

Ready	0
Unacked	1
Total	1

Message rates last ten minutes ?

Publish	979/s	Consumer ack	947/s	Get (auto ack)	0.00/s
Deliver (manual ack)	949/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

Features	durable: true	State	running
Policy		Consumers	128
Operator policy		Consumer capacity ?	100%
Effective policy definition			

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	1	0	1	1	0	0
Message body bytes ?	125 B	0 B	125 B	125 B	0 B	0 B
Process memory ?	443 kiB					

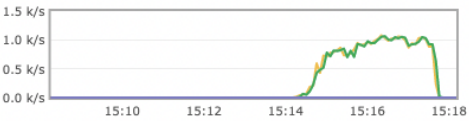
Queue server_queue

Overview

Queued messages last ten minutes ?

Ready	0
Unacked	0
Total	0

Message rates last ten minutes ?



Publish	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

Features	durable: true	State	idle
Policy		Consumers	128
Operator policy		Consumer capacity ?	100%
Effective policy definition			

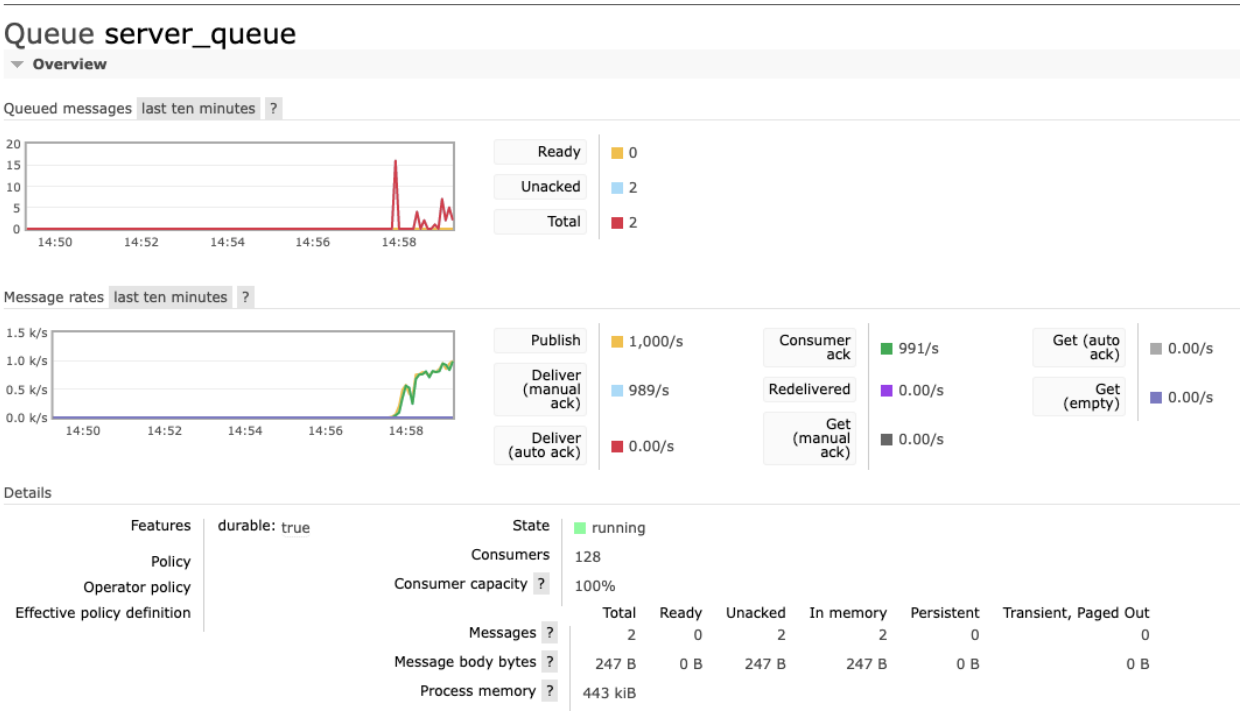
	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	0	0	0	0	0	0
Message body bytes ?	0 B	0 B	0 B	0 B	0 B	0 B
Process memory ?	89 kiB					

With circuit breaker

- RMQ management window for queue
- 128 client threads

```
------  PART 1  ------
number of successful requests sent: 159977
number of unsuccessful requests: 0
the total run time for all phases to complete: 204767
the total throughput in requests per second: 0

------  PART 2  ------
mean response time (millisecs): 117
median response time (millisecs): 73
throughput: 0
p99 (99th percentile) response time: 433
min response time (millisecs): 11
max response time (millisecs): 7962
```

- The queue size range is 0 - 40, message rate is send/receive = 1000 / 991 = 1.01



- 256 client threads

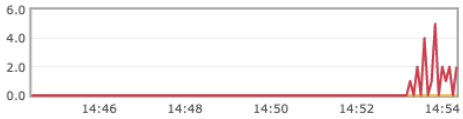
```
-----  PART 1  ------
number of successful requests sent: 160420
number of unsuccessful requests: 0
the total run time for all phases to complete: 214347
the total throughput in requests per second: 0

------  PART 2  ------
mean response time (millisecs): 239
median response time (millisecs): 195
throughput: 0
p99 (99th percentile) response time: 1041
min response time (millisecs): 11
max response time (millisecs): 7606
```

- The queue size range is 0 - 6, message rate is send/receive = 892 / 869 = 1.03

Queue server_queue

Overview

Queued messages last ten minutes ?



Ready	0
Unacked	2
Total	2

Message rates last ten minutes ?

Publish	892/s	Consumer ack	869/s	Get (auto ack)	0.00/s
Deliver (manual ack)	869/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

Features	durable: true
Policy	
Operator policy	
Effective policy definition	

State	running
Consumers	256
Consumer capacity ?	100%

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	2	0	2	2	0	0
Message body bytes ?	247 B	0 B	247 B	247 B	0 B	0 B
Process memory ?	725 kiB					

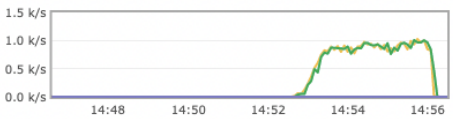
Queue server_queue

Overview

Queued messages last ten minutes ?

Ready	0
Unacked	0
Total	0

Message rates last ten minutes ?



Publish	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

Features	durable: true
Policy	
Operator policy	
Effective policy definition	

State	idle
Consumers	256
Consumer capacity ?	100%

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	0	0	0	0	0	0
Message body bytes ?	0 B	0 B	0 B	0 B	0 B	0 B
Process memory ?	162 kiB					