Tiffany Lastimosa

Assignment 3 Write-up

GitHub Link

https://github.com/tifflastimosa/a3-distributed-systems-data-layer

Extension

Please know I directly messaged Dr. Ian at the beginning of Thanksgiving break requesting an extension because of a personal matter. Kindly note it was granted. Many thanks for the granted extension.

Screenshots of RabbitMQ, Redis, Throughput

Test Screenshots Section (link)

I. Database Decision

For this assignment, three databases were considered: MySQL, MongoDB, DynamoDB, and Redis. MySQL was considered because it is robust, available, and open sourced. However, it lacks flexibility. MongoDB was then considered because it is document based and it is flexible should there be any changes to a data model. It can also handle complex queries and it is scalable. DynamoDB was also considered because it was fast.

On the other hand, Redis is an in-memory database and it much faster in comparison to MySQL and MongoDB. Moreover, it is easy to set-up, deploy on an ec2 instance, and it offers the Jedis dependency to create connections and write a Java program to write to the Redis database. Thus, I chose Redis to try a new database and observe how fast it was.

II. Redis Database Schemas

When designing the database schema for Redis, it became apparent that there many trade-offs because it is a key-value data store.

a. Potential Schema 1 - Sets

key	value
skierId:12389342- UUID:1234-5678-9012-3456	{ “dayId”: 1, “seasonId”: 123, “resortId”: 123, “time”: 12, “lift”: 20, “vertical” 200 }

Trade-off: In this solution, it applies the idea of a document database. One benefit of this approach, is that data can be quickly written to the Redis database and prevent data from being overwritten. It would allow the proposed queries, but the values are nested. To query with this schema, the user would have to get the desired skierId and then query each value of the skier. In the worst-case scenario, there would be a plethora of the desired skierId (i.e.100,000 entries) and the user would have to query each key. The GET requests may take much longer in comparison to POST requests and would not be ideal for a web application.

b. Potential Schema 2 - Sets

key	**value**
skierId:12389342- UUID:1234-5678-9012-3456	{ “dayId”: 1, “seasonId”: 123, “resortId”: 123, “time”: [12, 35, 21] “lift”: [20, 10, 20] “vertical” [200, 100, 200] }

Trade-off: This solution also applies the idea of a document database. The difference is that a PUT operation would have to be implemented in the Java Consumer program. Thus, the Java program would have a POST operation if the skierId did not exist, but a PUT operation if the skiderId did exist. The skierId would have to be found and access the nested times, lifts, and verticals, which are stored in arrays. It could potentially handle the queries, but it would be much more complicated because the user would have to query the nested arrays.

c. Potential Schema 3 - Hashes

key	**attribute**	**value**
skierId:12389342-UUID:1234-5678- 9012-3456	“dayId”	1
skierId:12389342-UUID:1234-5678- 9012-3456	“seasonId”	123
skierId:12389342-UUID:1234-5678- 9012-3456	“resortId”	123
skierId:12389342-UUID:1234-5678- 9012-3456	“time”	12

skierId:12389342-UUID:1234-5678-9012-3456	“lift”	20
skierId:12389342-UUID:1234-5678-9012-3456	“vertical”	200

Trade-off: This schema uses the hashes data type in Redis. This would not slow down the POST operation, however, the user is presented with complex queries because they would have to get the desired skierId and then grab each attribute. It is very similar to the previous options because it works with nested data to perform the queries.

d. Schema 4 - Sets

key	**value**
skierId:12389342-resortId:123-seasonId:123-dayId:1-time:12-lift:27-vertical:270-UUID:1234-5678-9012-3456	{ “vertical”:270 }

Trade-off: Schema 4 uses sets. Moreover, rather than storing all the data as a value, the data is stored in the key. This makes it easy to do a POST operation and would make it easier to query by using a delimiter. The caveat to this schema is that the user would have to define each query they desire. It is up to the developer to anticipate of what would be queried or to work closely with their team to determine what queries would be needed.

For the schema, I chose to go with option d) Schema 4 because it would be easier to query since the data to be queried will not be nested Moreover, it will be faster to write to the Redis database.

III. Deployment Topologies | Instance Types

Please see the following Deployment Topologies | Instance Types (link) section of this document for the different deployment topologies. If you would like to view the Google Sheet of the topologies and instance types, please see the following Google Sheet link: A3 Data (link). When reviewing the report, please refer to the color legend for additional information regarding the topologies and instance types.

IV. Observations – *to reference the tests, see* A3 Data (link)

Assignment 2 Revisited

In assignment 2, after implanting the message queue and the load balancer, the client exhibited low throughput and low/stable message queue size due to network latency. The message queue size never breached 1000 messages.

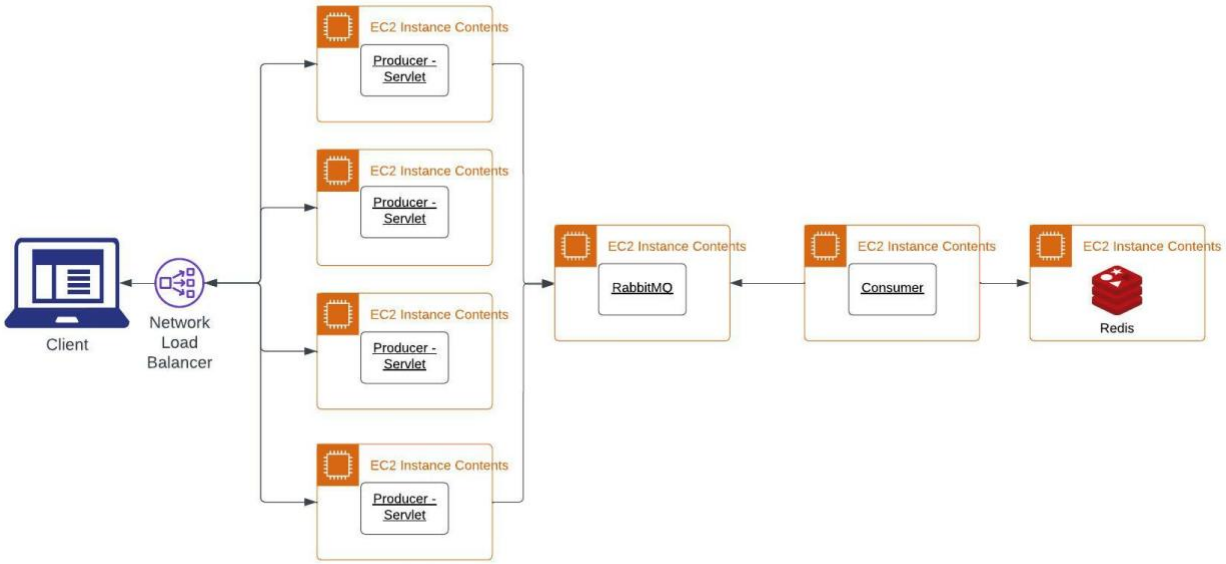
Assignment 3 Network

For assignment 3, the client did not exhibit much network latency and rarely had socket exceptions. The networkQuality results showed that the responsiveness in the network was higher than the responsiveness in assignment 2.

```
==== SUMMARY ====
Uplink capacity: 20.247 Mbps
Downlink capacity: 436.545 Mbps
Responsiveness: Medium (337 RPM)
Idle Latency: 24.833 milli-seconds
```

Figure 1 Network for Tests 1 - 17

Step 1



Test 1 – 17 with Load Balancers

In tests 1 through 17, the client experienced higher throughput and very few socket exceptions in comparison to assignment 2. One key observation I made was that the message queue would get large. With the fast responsiveness and the added load balancer, the client was able to

send POST requests faster to the servlets and the servlets were publishing messages faster to the message queue. Thus, the message queue got large very fast when the client hit the second phase of 168 threads and 1000 threads.

As I increased the number of connections to the message queue in the Consumer java program, the message queue got smaller because it was consuming more messages from the queue. For example, in tests 12 to 14, the max messages went from approximately 6,000 to 3,000.

Test 22 – 23 with Load Balancers

In tests 22 and 23, the message queue was significantly lower. This could be potentially due to the responsiveness being lower (see *Figure 2 Network for Tests 22 and 23*) than the responsiveness in *Figure 1 Network for Tests 1 – 17.*

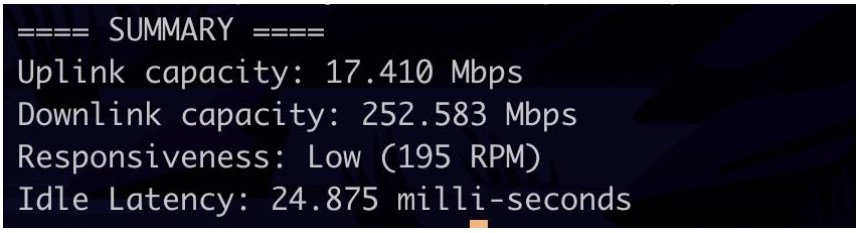


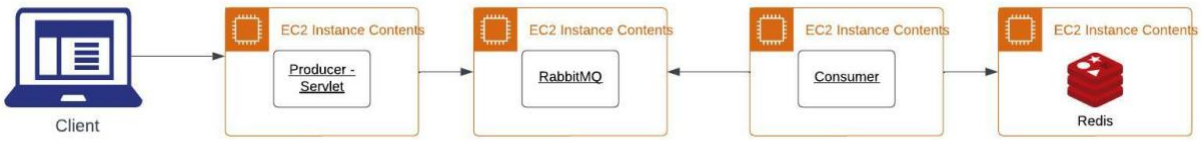
Figure 2 Network for Tests 22 and 23

Additionally, I changed the phase proportion rom 32 threads:168 threads with 1000 requests to 100 threads:100 threads with 1000. The proportion may have helped by creating a bottleneck to slow the number of messages published to the message queue.

Step 2 – No Load Balancer, Increased Capacity

Choice:

An alternative to circuit breaker for step 2, I deployed a second Java Consumer program on another ec2 instance and tested it on different instance types. I also removed the load balancer, changed the sizes of the instances for the servlet, RabbitMQ, consumer, and Redis.



Test 18 – 19 – medium instances

In tests 18 and 19, the throughput was approximately the same, but the queue was much smaller. It appears the combination of the removal of the load balancer and decrease in

responsiveness kept the message queue smaller. Per Redis Insight, increased from 2145 to 2349 commands per second when increasing the number of consumers. The commands per second shows how many commands per second (in this case POST operations) were being sent to Redis.

Test 20 – 21 – large instances

For tests 20 and 21, like tests 18 and 19, the throughput remained approximately the same and the message queue was small. It appears that the message queue was somewhat smaller than tests 18 and 19. In Redis Insight, the commands were higher than 18 and 19. This could be due to the increased size in the instance size.

Test 24 – 25 – large instances, client in t3.medium instance

In these tests, I wanted to observe what would happen if the client making the POST requests was on an its own ec2 instance. Once I ran the program, there was a significant increase in the message queue. Rather than the message queue holding around 200 messages, it held approximately 15,000 requests. This shows that the removal of the network latency significantly increased the message queue size and created a backlog. The client rapidly was sending requests and the consumer could not keep up. Lastly, the throughput increased and was the fastest throughput amongst all the other tests.

However, when adding a second consumer, the message queue size significant decreased. It went from 15,000 to 1,500 messages in the queue. If another consumer was added, then the message queue may be much smaller than 1,500 messages. Another key observation in the addition of the second consumer was the Redis commands per second. Compared to around 2,000 commands per second, it increased to 3,970 commands per second and was quickly writing to the database.

V. Conclusion

Overall, should this web application be deployed to the public, Redis is not the ideal database. MongoDB or Firestore might be more optimal for an application that stores information about a skier and their lifts because it is fast and flexible.Moreover, increasing the number of consumers will help with keeping the message queue low/stable. This was observed in tests 24 and 25. By adding an additional consumer, we could potentially see close to zero messages in the queue since going from one to two consumers had significant effect. The caveat to this would be

that too many consumers connecting to the RabbitMQ may increase the memory capacity of the RabbitMQ and could have either an adverse or no effect to the size of the message queue.

III. Deployment Topologies | Instance Types

Color Legend	**Number of Connections to RabbitMQ**					
	Deployment Topologies | AWS EC2 Types					
	Indicates change to AWS EC2					
STEP 1 - with loadbalancer						
Test 1 - 32 threads - 1000 reqs, 168 threads reqs						
	Client	**Servlet (load balanced x4)**	**RabbitMQ**	**Consumer**	**RedisDB**	**Notes**
Configuration	local	t2.micro	t3.medium	t3.medium	t3.medium	- From A2, observed that memory consumed quickly, thus on the 1st test, will increase from t3.micro to t3.medium
Rabbit MQ Channels		800	~ 1000	200		- Compared to A2, experienced faster network speed, which could potentially explain why the queue was flooded with messages
Redis Connections				200 connections to Redis		- Consumer not pulling enough messages
Max Messages	12,000					
Throughput	2048.97					
Test 2 - 32 threads - 1000 reqs, 168 threads reqs						
	Client	**Servlet (load balanced x4)**	**RabbitMQ**	**Consumer**	**RedisDB**	**Notes**
Configuration	local	t2.micro	t3.medium	t3.medium	t3.medium	- Will the rabbit mq be able to handle the amount of connections? Yes. Saw some improvement.
Rabbit MQ Channels		800	~ 1300	500		- Queue size decreased with the increased number of consumers
Redis Connections				500 connections to Redis		- Since the consumer was able to pull off more messages, throughput also improved, but very minimal improvement. This might be due to
Max Messages	7,500					
Throughput	2171.481					
Test 3						
	Client	**Servlet (load balanced x4)**	**RabbitMQ**	**Consumer**	**RedisDB**	**Notes**
Configuration	local	t2.micro	t3.medium	t3.medium	t3.medium	- Will the rabbit mq be able to handle the amount of connections? Yes, saw smaller queue, but slightly lower throughput
Rabbit MQ Channels		800	~ 1600	800		- Slightly slower network when client was ran
Redis Connections				800 connections to Redis		
Max Messages	4,000					
Throughput	1907.614					
Test 4						
	Client	**Servlet (load balanced x4)**	**RabbitMQ**	**Consumer**	**RedisDB**	**Notes**
Configuration	local	t2.micro	t3.medium	t3.medium	t3.medium	
Rabbit MQ Channels		800	~ 2000	1200		
Redis Connections				1200		
Max Messages	4,000					
Throughput	2166.072					

Test 5						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.medium	t3.medium	t3.medium	
Rabbit MQ Channels		800	~ 2800	2000		
Redis Connections				2000		
Max Messages	<4,000					
Throughput	2005.836					
Test 6 - all 200,000 requests at once						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.medium	t3.medium	t3.medium	
Rabbit MQ Channels		800	~ 2800	2000		
Redis Connections				2000		
Max Messages	3,000					
Throughput	2582.011					
Test 7 - client-metrics on ec2 instance; 32 threads - 1000 reqs, 168 threads reqs						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	t3.medium - deployed to ec2 instance	t2.micro	t3.medium	t3.medium	t3.medium	- Message queue got really large, it could be that because it had better network speed, it was able to push requests faster than the rabbit mq and consumer was to process the messages and push to the database
Rabbit MQ Channels		800	~ 2800	2000		
Redis Connections				2000		
Max Messages	60,000					
Throughput	1773.521					
Test 8 - consumer has 4000 channels, redis connections						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.medium	t3.medium	t3.medium	Observations: Created 4000 consumer channels to retrieve messages, however, only 2047 created
Rabbit MQ Channels		800	~ 2800	2000		- RabbitMQ might not allow for that much channels
Redis Connections				2000		- Also could be that Consumer java program can only allow for so many threads
Max Messages	< 3,000					
Throughput	1901.807					
RabbitMQ Instance size Increase from t3.medium to t3.large						
Test 9						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.medium	t3.medium	
Rabbit MQ Channels		800	~1600	800		
Redis Connections				800		
Max Messages	< 4,000					
Throughput	2104.886					

Test 10						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.medium	t3.medium	
Rabbit MQ Channels		800	~2000	1200		
Redis Connections				1200		
Max Messages	< 6,000					
Throughput	2166.8					
Test 11						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.medium	t3.medium	
Rabbit MQ Channels		800	~ 2800	2000		
Redis Connections				2000		
Max Messages	< 3,000					
Throughput	1924.946					
Extra Test - No Screenshots						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.medium	t3.medium	- Observed that only created 2047 consumers
Rabbit MQ Channels		800	~ 2047	3000		https://support.huaweicloud.com/intl/en-us/ae-ad-1-usermanual-rabbitmq/rabbitmq-faq-0001.html
Redis Connections				3000		- Did not take screenshots because it would be similar to Test 11
Consumer Instance size increase from t3.medium to t3.large						
Test 12						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.large	t3.medium	
Rabbit MQ Channels		800	~ 1600	800		
Redis Connections				800		
Max Messages	< 6,000					
Throughput	2126.234					
Test 13						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.large	t3.medium	
Rabbit MQ Channels		800	~ 2000	1200		
Redis Connections				1200		
Max Messages	< 5,000					
Throughput	2147.904					

Test 14						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.large	t3.medium	
Rabbit MQ Channels		800	~ 2800	2000		
Redis Connections				2000		
Max Messages	< 3,000					
Throughput	1896.615					
Redis Instance size increase from t3.medium to t3.large						
Test 15						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.large	t3.large	
Rabbit MQ Channels		800	~ 1600	800		
Redis Connections				800		
Max Messages	< 6,000					
Throughput	2101.458					
Test 16						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.large	t3.large	
Rabbit MQ Channels		800	~ 2000	1200		
Redis Connections				1200		
Max Messages	< 4,000					
Throughput	2055.963					
Test 17						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.large	t3.large	
Rabbit MQ Channels		800	~ 2800	2000		
Redis Connections				2000		
Max Messages	< 2,500					
Throughput	1988.15					
STEP 1 REVISITED - with load balancer						
Test 22						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.large	t3.large	Test 22 - load balanced re-applied; phase changed to 100 threads, and 1000 requests 2x phase RabbitMQ large, consumer large, redis large 1200 consumers
Rabbit MQ Channels		800	~ 2000	1200		
Redis Connections				1200		
Max Messages	< 200					
Throughput	1959.286					

Test 23						
	Client	Servlet (load balanced x4)	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t2.micro	t3.large	t3.large	t3.large	Test 23 - load balanced re-applied; phase changed to 100 threads, and 1000 requests 2x phase RabbitMQ large, 2x consumer large, redis large 1200 consumers
Rabbit MQ Channels		800	~ 2000	1200		
Redis Connections				1200		
Max Messages	< 200					
Throughput	1925.965					
STEP 2 - no load balancer						
Test 18 - medium instances for servlet, rabbit-mq, consumer, redis db - 1200						
	Client	Servlet	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t3.medium	t3.medium	t3.medium	t3.medium	- Observed Redis speeds 2145
Rabbit MQ Channels		~ 200	~ 1400	1200		
Redis Connections				1200		
Max Messages	< 300					
Throughput	2107.525					
Test 19 - medium instances for servlet, rabbit-mq, 2xconsumer, redis db						
	Client	Servlet	RabbitMQ	Consumer (2x)	RedisDB	Notes
Configuration	local	t3.medium	t3.medium	t3.medium	t3.medium	- 2 consumers
Rabbit MQ Channels		~ 200	~ 1400	2400		- Observed Redis speeds - maxed observed was 2349
Redis Connections				2400		
Max Messages	< 200					
Throughput	1829.123					
All instances change to large						
Test 20 - large instances for servlet, rabbit-mq, consumer, redis db - 1200						
	Client	Servlet	RabbitMQ	Consumer	RedisDB	Notes
Configuration	local	t3.large	t3.large	t3.large	t3.large	- Observed Redis speeds - maxed observed was 2615
Rabbit MQ Channels		~ 200	~ 1400	1200		
Redis Connections				1200		
Max Messages	< 250					
Throughput	2144.979					
Test 21 - large instances for servlet, rabbit-mq, 2xconsumer, redis db - 1200						
	Client	Servlet	RabbitMQ	Consumer (2x)	RedisDB	Notes
Configuration	local	t3.large	t3.large	t3.large	t3.large	- 2 consumers
Rabbit MQ Channels		~ 200	~ 1400	1200		- Observed Redis speeds - maxed observed was 2615
Redis Connections				1200		
Max Messages	< 200					
Throughput	1912.794					

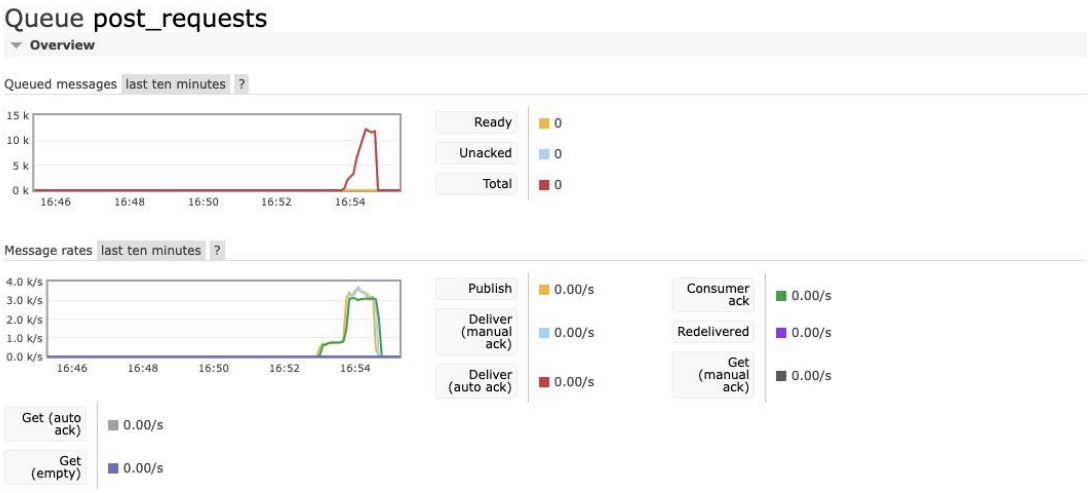
Test 24 - large instances for servlet, rabbit-mq, consumer, redis db - 1200; 100 threads w/ 1000 requests in 2 phases						
	Client	**Servlet**	**RabbitMQ**	**Consumer**	**RedisDB**	**Notes**
Configuration	t3.medium - deployed to ec2 instance	t3.large	t3.large	t3.large	t3.large	
Rabbit MQ Channels		~ 200	~ 1400	1200		
Redis Connections				1200		
Max Messages	< 15,000					
Throughput	2379.79					

Test 25 - large instances for servlet, rabbit-mq, 2xconsumer, redis db - 1200						
	Client	**Servlet**	**RabbitMQ**	**Consumer (2x)**	**RedisDB**	**Notes**
Configuration	t3.medium - deployed to ec2 instance	t3.large	t3.large	t3.large	t3.large	- 2 consumers
Rabbit MQ Channels		~ 200	~ 1400	1200		- Observed Redis speeds - max observed was 3970
Redis Connections				1200		
Max Messages	< 1,500					
Throughput	1875.679					

Test Screenshots

Test 1

Rabbit MQ



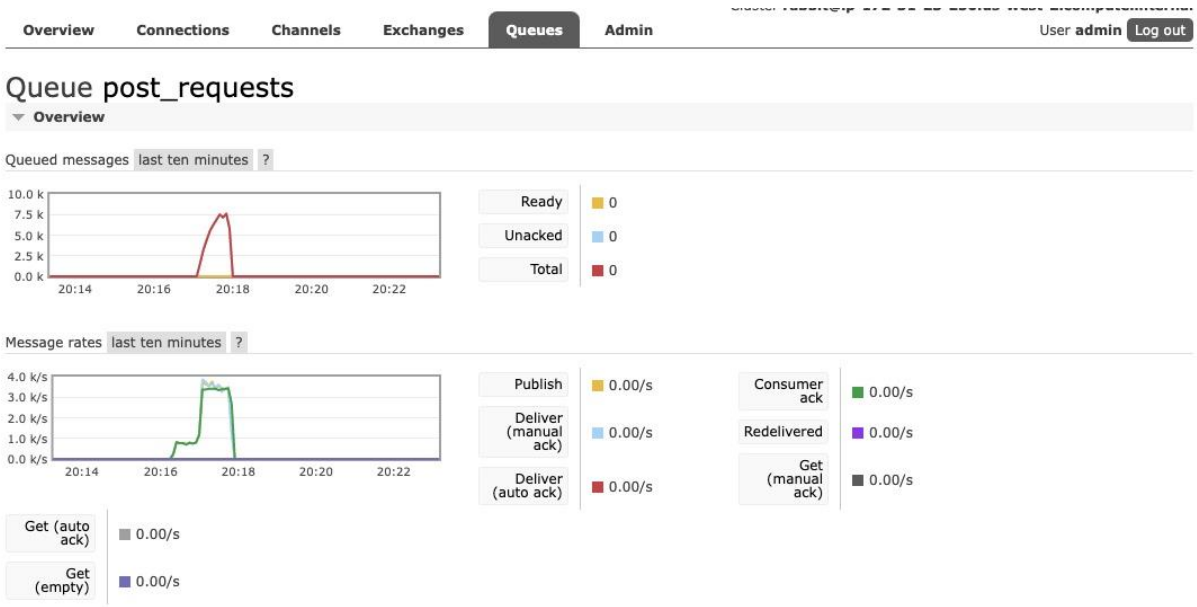
Redis

Throughput

```
--------------------------Configuration--------------------------
Number of Threads: 200
Total Requests: 200000
----------------------------Calculations---------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 97610 ms
Throughput: 2048.97039237783 requests per second
-------------------------------Stats-------------------------------
Mean: 50 ms
Median: 37 ms
Min Response Time: 16 ms
Max Response Time: 1035 ms
99th Percentile: 22 ms
```

Test 2

Rabbit MQ



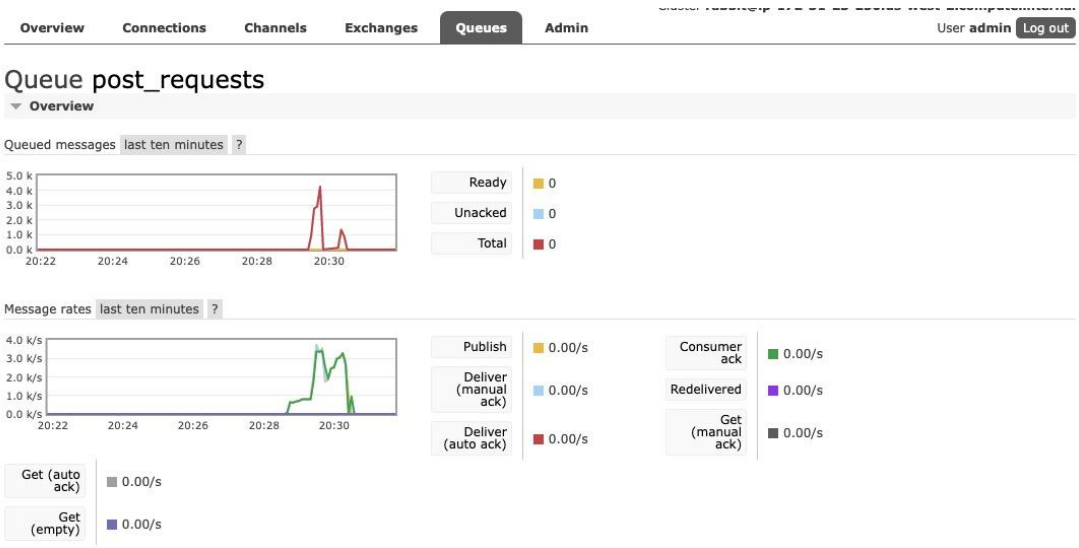
Redis

Throughput

```
----------------------------Configuration----------------------------
Number of Threads: 200
Total Requests: 200000
----------------------------Calculations----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 92103 ms
Throughput: 2171.4819278416553 requests per second
-------------------------------Stats-------------------------------
Mean: 47 ms
Median: 35 ms
Min Response Time: 14 ms
Max Response Time: 1089 ms
99th Percentile: 21 ms
```

Test 3

Rabbit MQ

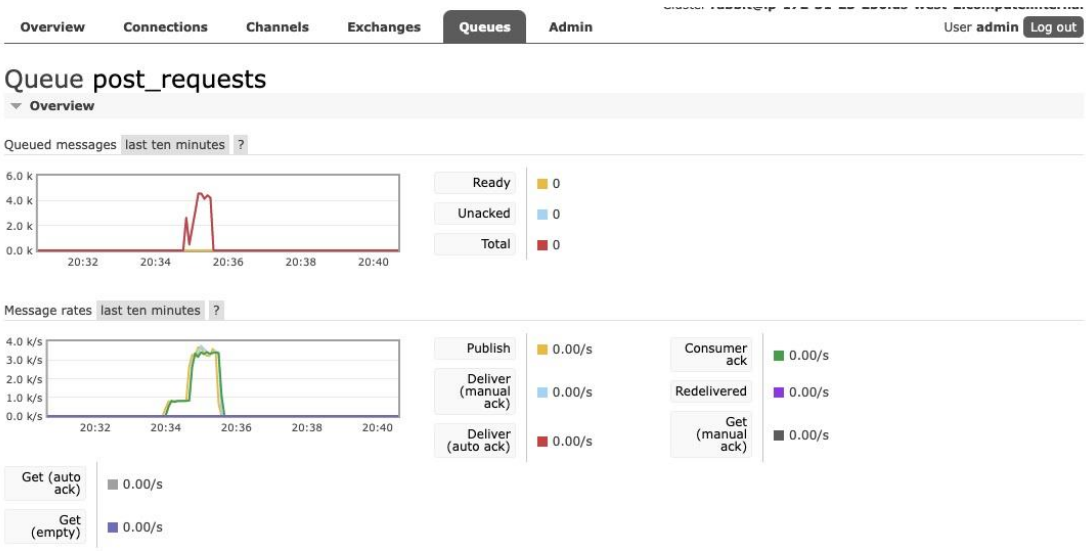


Redis

Throughput

```
----------------------------Configuration----------------------------
Number of Threads: 200
Total Requests: 200000
----------------------------Calculations----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 104843 ms
Throughput: 1907.6142422479325 requests per second
---------------------------------Stats----------------------------------
Mean: 56 ms
Median: 38 ms
Min Response Time: 19 ms
Max Response Time: 1522 ms
99th Percentile: 21 ms
```

Rabbit MQ



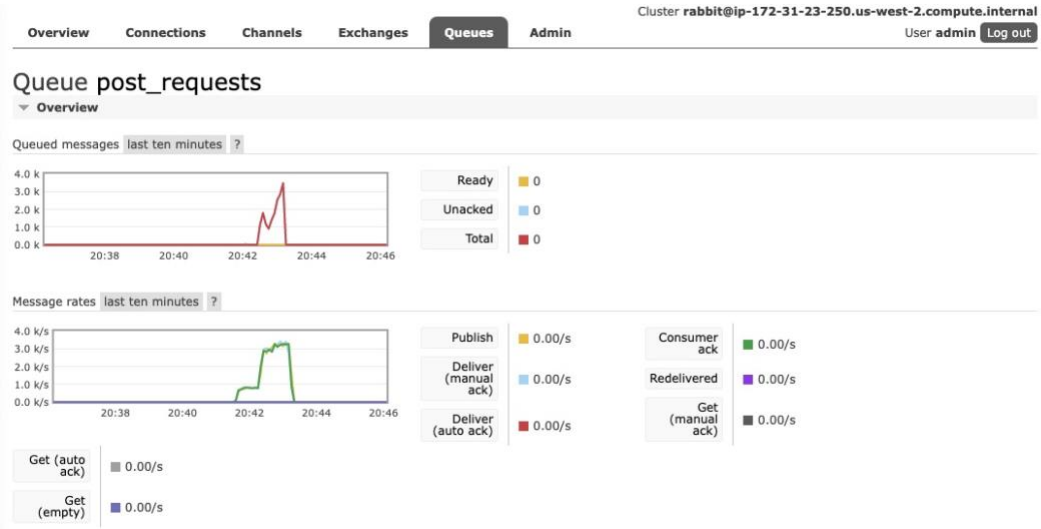
Redis

Throughput

```
---------------------------Configuration---------------------------
Number of Threads: 200
Total Requests: 200000
---------------------------Calculations---------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 92333 ms
Throughput: 2166.0728017068654 requests per second
-------------------------------Stats-------------------------------
Mean: 48 ms
Median: 35 ms
Min Response Time: 19 ms
Max Response Time: 963 ms
99th Percentile: 21 ms
```

Test 5

Rabbit MQ



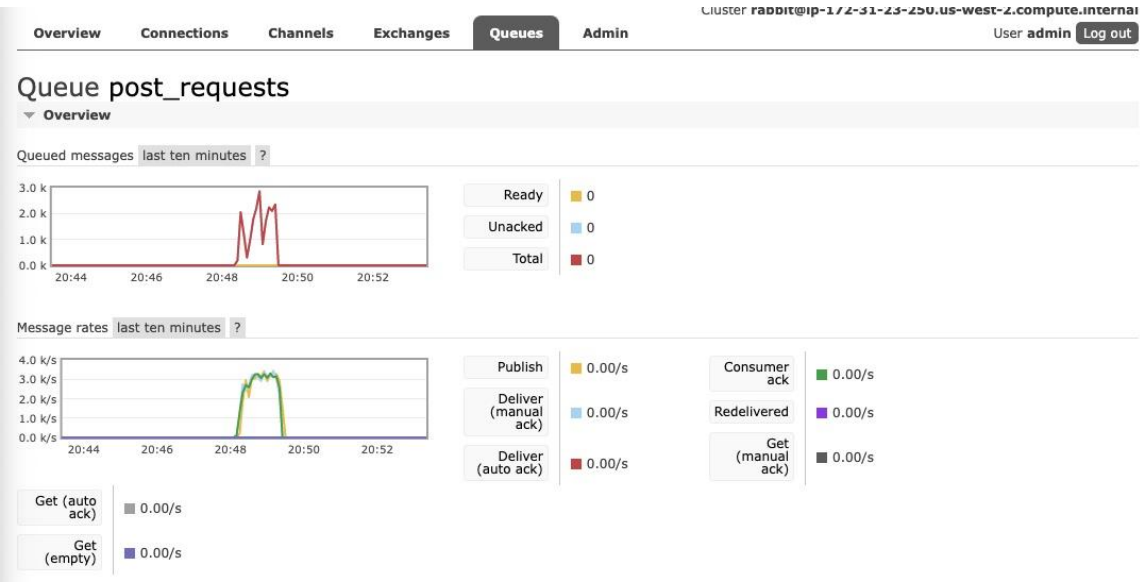
Redis

Throughput

```
----------------------------Configuration----------------------------
Number of Threads: 200
Total Requests: 200000
----------------------------Calculations----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 99709 ms
Throughput: 2005.836985628178 requests per second
-------------------------------Stats-------------------------------
Mean: 53 ms
Median: 39 ms
Min Response Time: 19 ms
Max Response Time: 1146 ms
99th Percentile: 21 ms
```

Test 6

Rabbit MQ



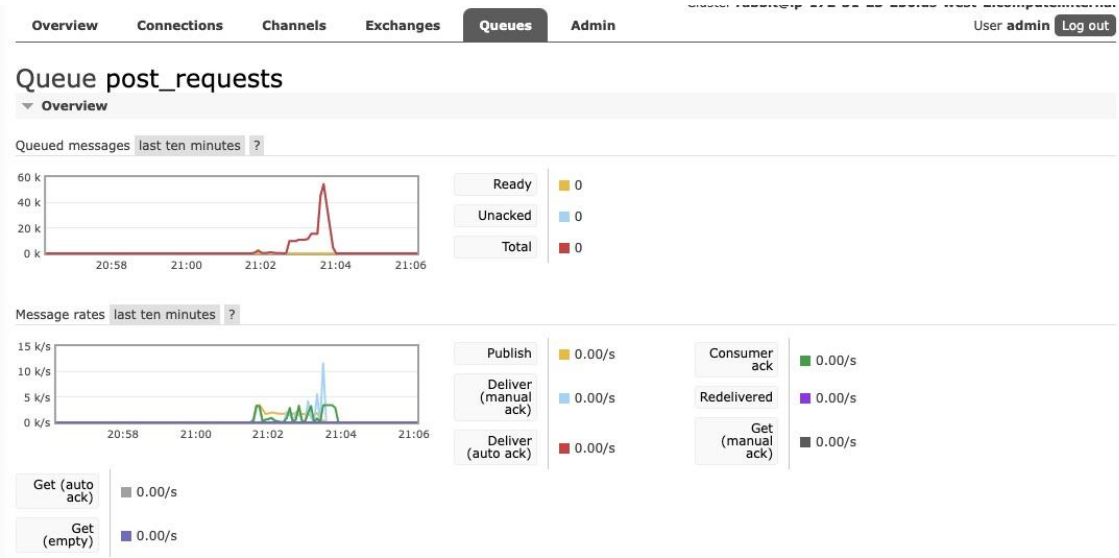
Redis

Throughput

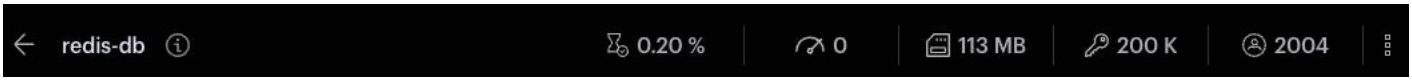
```
----------Configuration----------
Number of Threads: 200
Total Requests: 200000
----------Calculations----------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 77459 ms
Throughput: 2582.0111284679638 requests per second
----------Stats----------
Mean: 72 ms
Median: 48 ms
Min Response Time: 19 ms
Max Response Time: 2292 ms
99th Percentile: 27 ms
```

Test 7

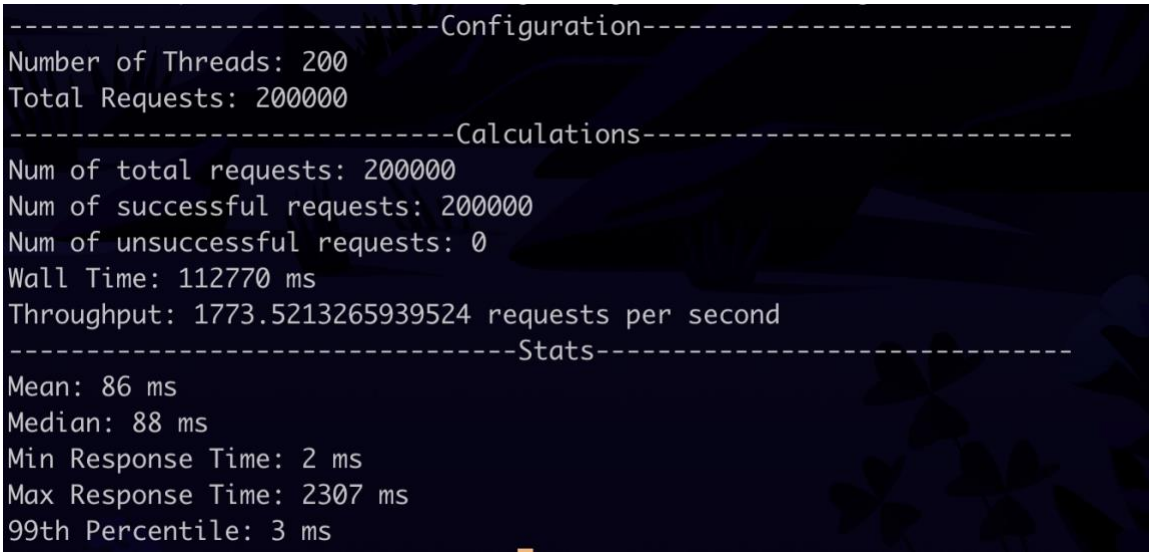
Rabbit MQ



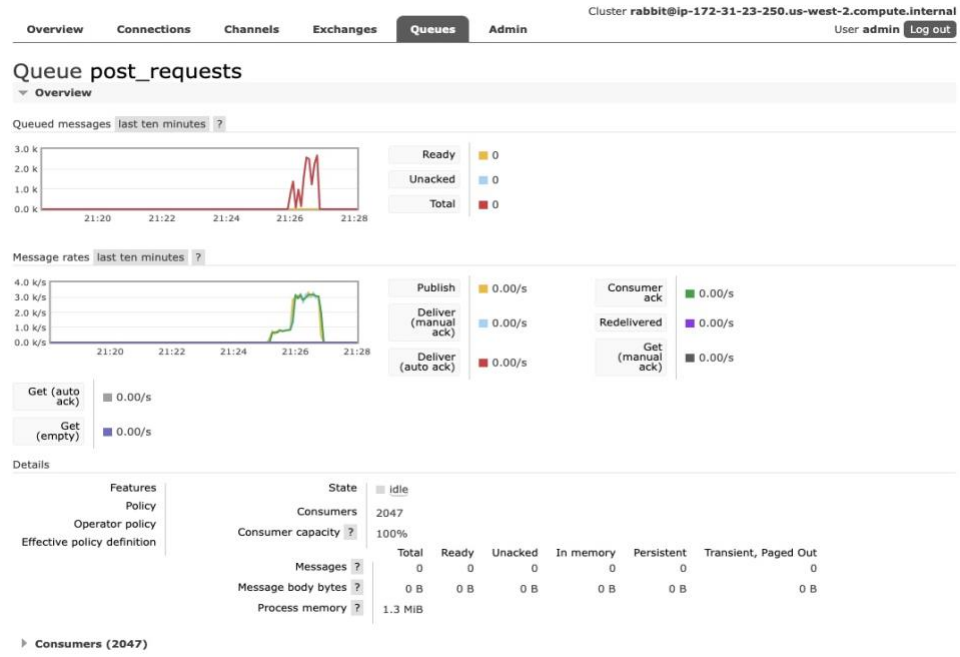
Redis



Throughput



Rabbit MQ



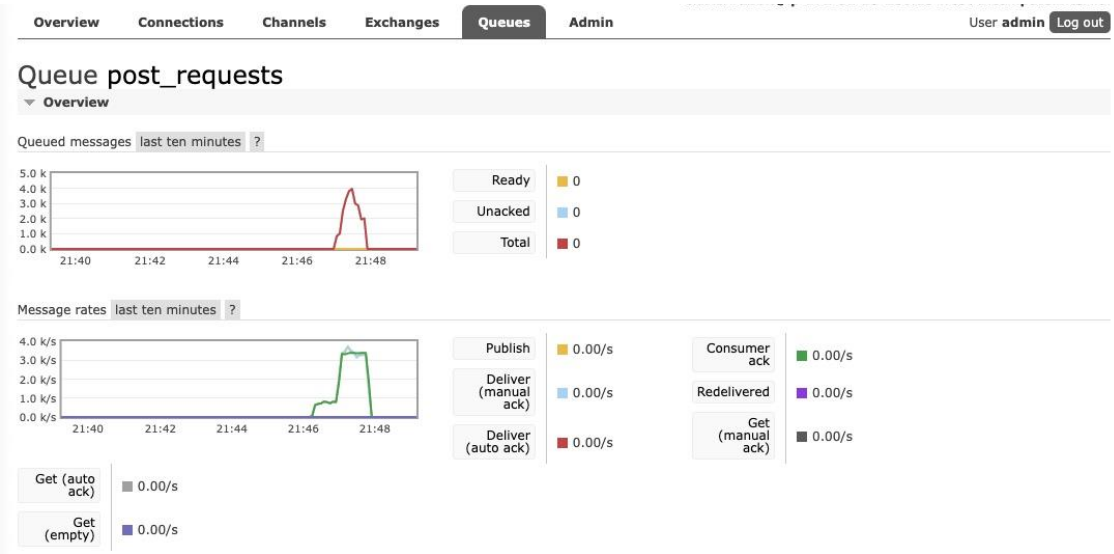
Redis

Throughput

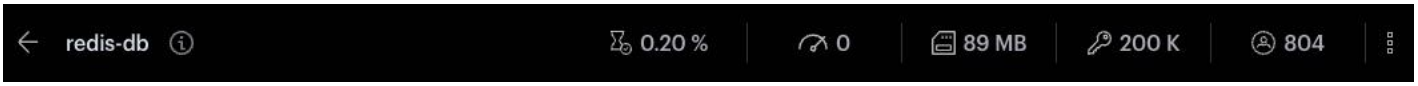
```
----------------------------Configuration----------------------------
Number of Threads: 200
Total Requests: 200000
-----------------------------Calculations----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 100918 ms
Throughput: 1981.8070116332071 requests per second
--------------------------------Stats--------------------------------
Mean: 53 ms
Median: 39 ms
Min Response Time: 15 ms
Max Response Time: 1293 ms
99th Percentile: 20 ms
```

Test 9

Rabbit MQ



Redis

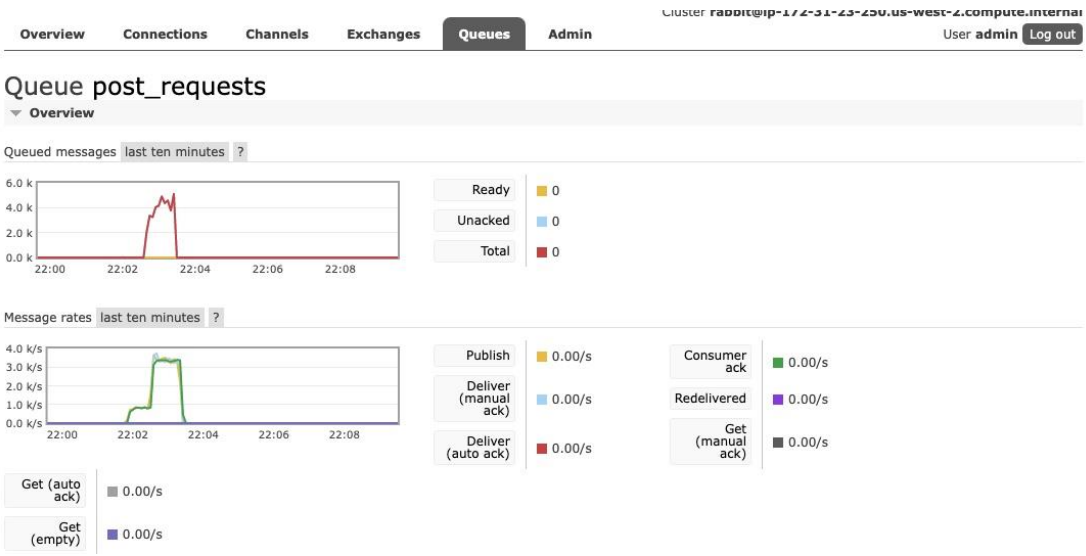


Throughput

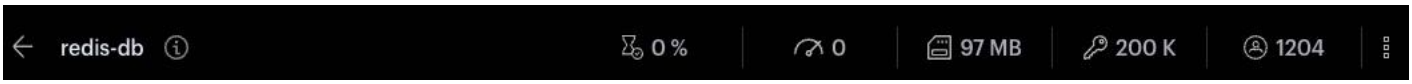
```
-------------------------Configuration-------------------------
Number of Threads: 200
Total Requests: 200000
-----------------------------Calculations-------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 95017 ms
Throughput: 2104.886493995811 requests per second
-------------------------------Stats-----------------------------
Mean: 48 ms
Median: 36 ms
Min Response Time: 19 ms
Max Response Time: 1061 ms
99th Percentile: 21 ms
```

Test 10

Rabbit MQ



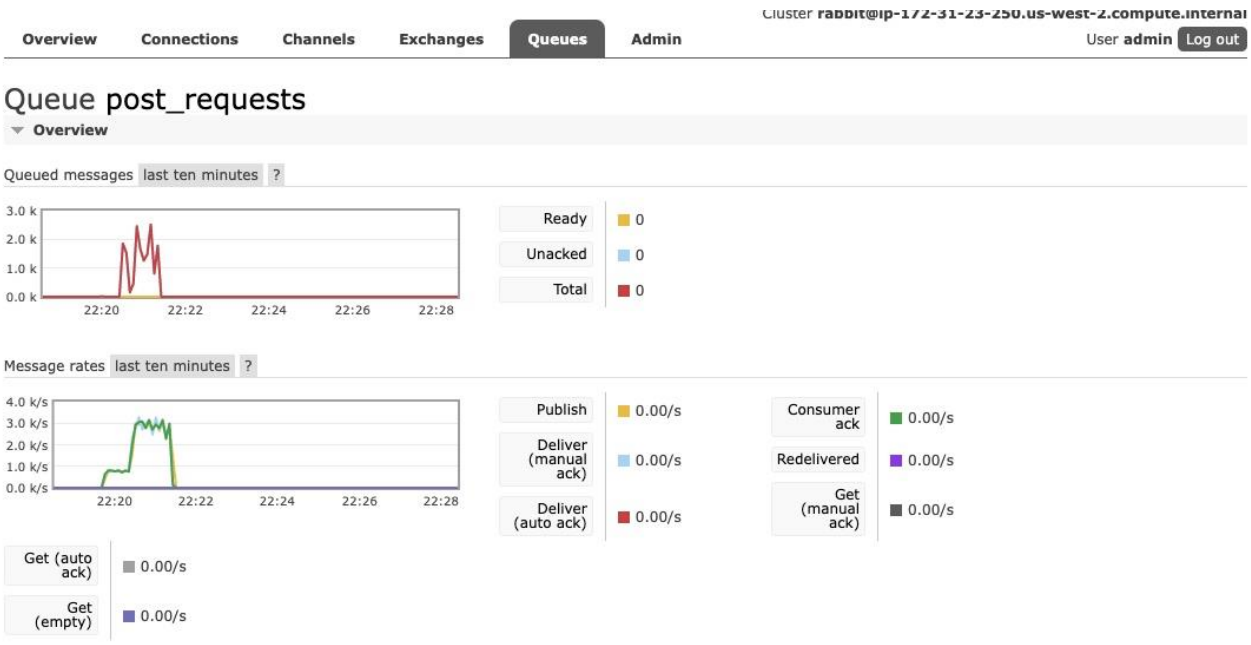
Redis



Throughput

```
-------------------------Configuration-------------------------
Number of Threads: 200
Total Requests: 200000
-------------------------Calculations-------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 92302 ms
Throughput: 2166.8002860176375 requests per second
-------------------------------Stats-------------------------------
Mean: 48 ms
Median: 35 ms
Min Response Time: 19 ms
Max Response Time: 1045 ms
99th Percentile: 21 ms
```

Rabbit MQ

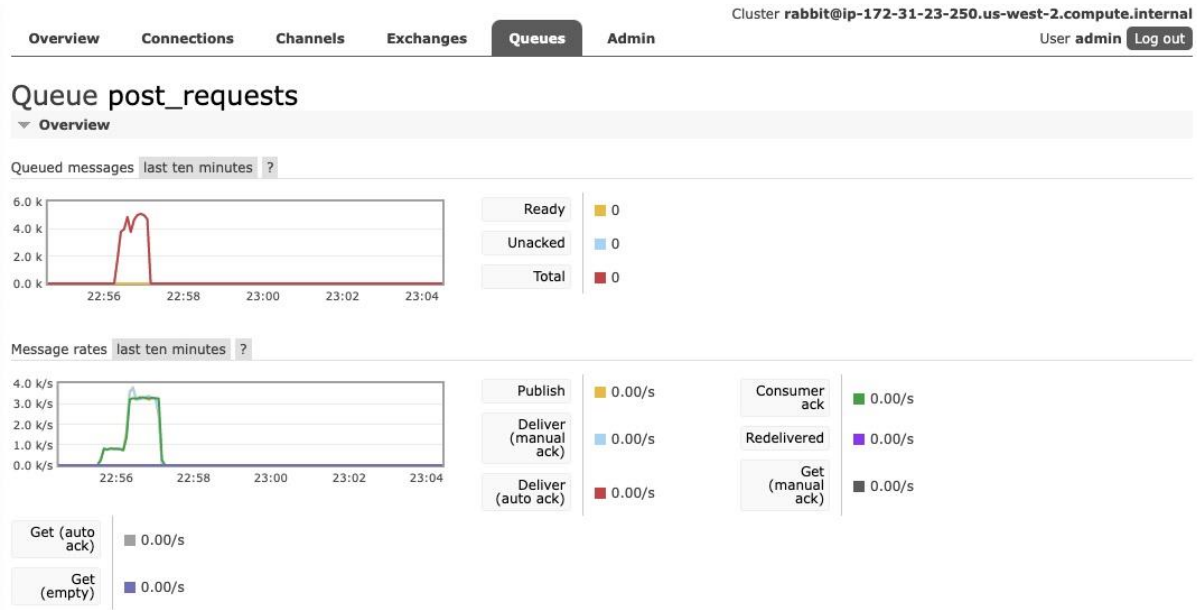


Redis

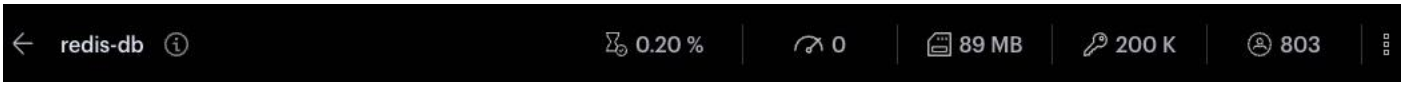
Throughput

```
-------------------------Configuration-------------------------
Number of Threads: 200
Total Requests: 200000
-------------------------Calculations-------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 103899 ms
Throughput: 1924.9463421207133 requests per second
-----------------------------Stats-----------+-----------------
Mean: 56 ms
Median: 41 ms
Min Response Time: 19 ms
Max Response Time: 1058 ms
99th Percentile: 21 ms
```

Rabbit MQ



Redis

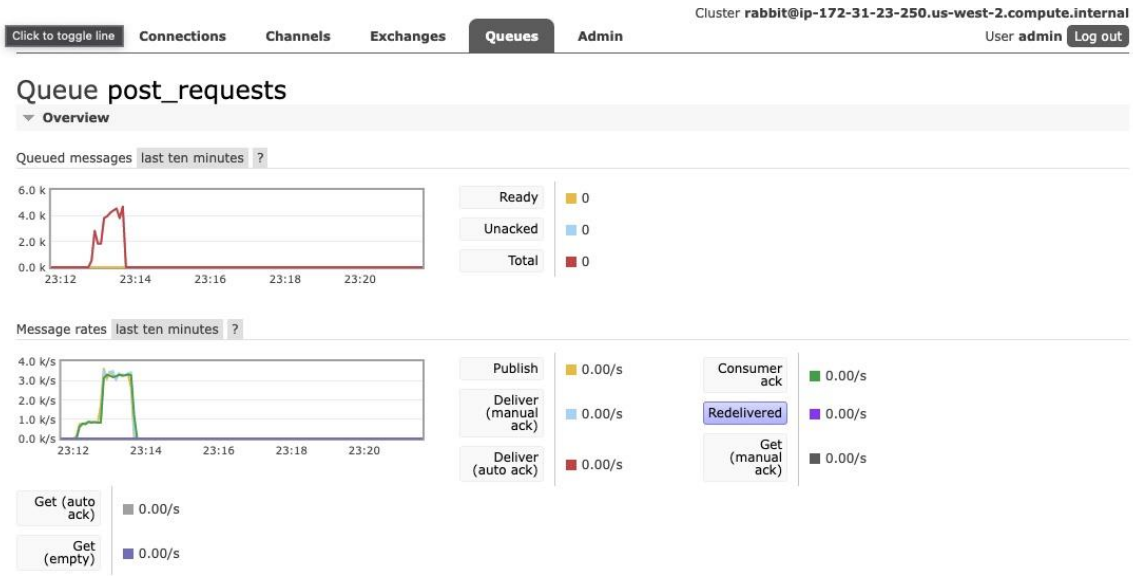


Throughput

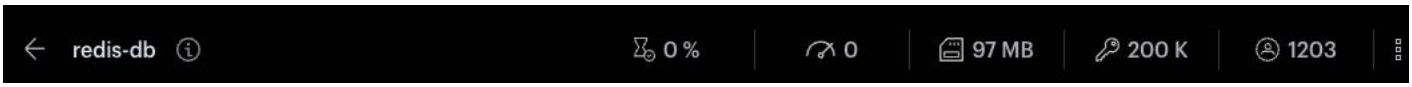
```
----------------------------Configuration----------------------------
Number of Threads: 200
Total Requests: 200000
----------------------------Calculations----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 94063 ms
Throughput: 2126.2345449326513 requests per second
-------------------------------Stats-------------------------------
Mean: 49 ms
Median: 37 ms
Min Response Time: 18 ms
Max Response Time: 1053 ms
99th Percentile: 20 ms
```

Test 13

Rabbit MQ



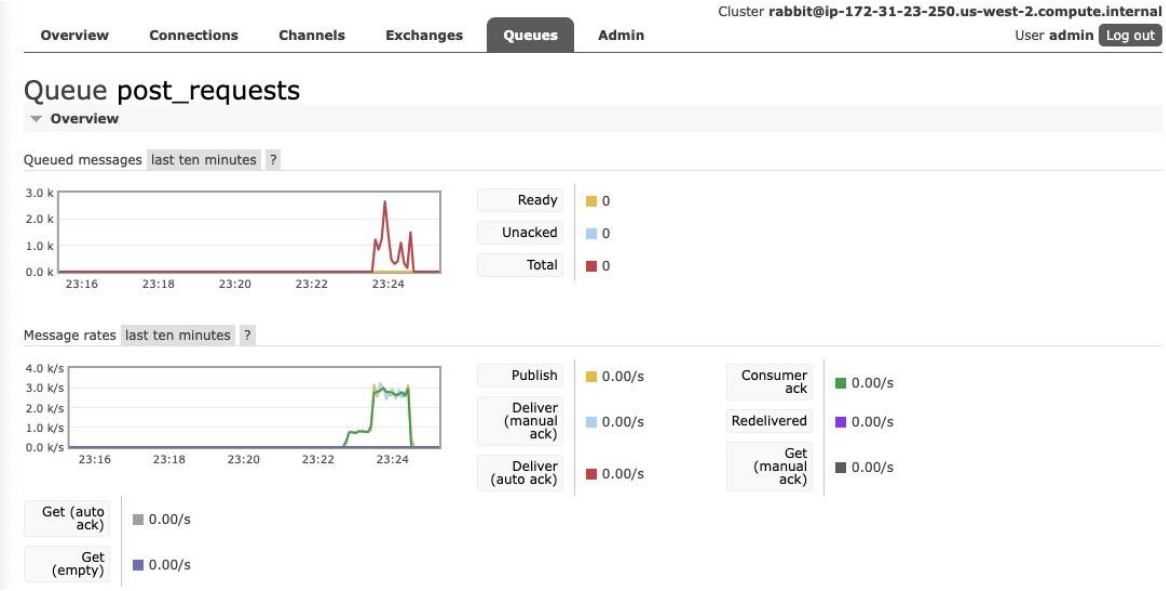
Redis



Throughput

```
----------Configuration----------
Number of Threads: 200
Total Requests: 200000
----------Calculations----------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 93114 ms
Throughput: 2147.9847189466673 requests per second
----------Stats----------
Mean: 49 ms
Median: 35 ms
Min Response Time: 17 ms
Max Response Time: 902 ms
99th Percentile: 21 ms
```

Rabbit MQ



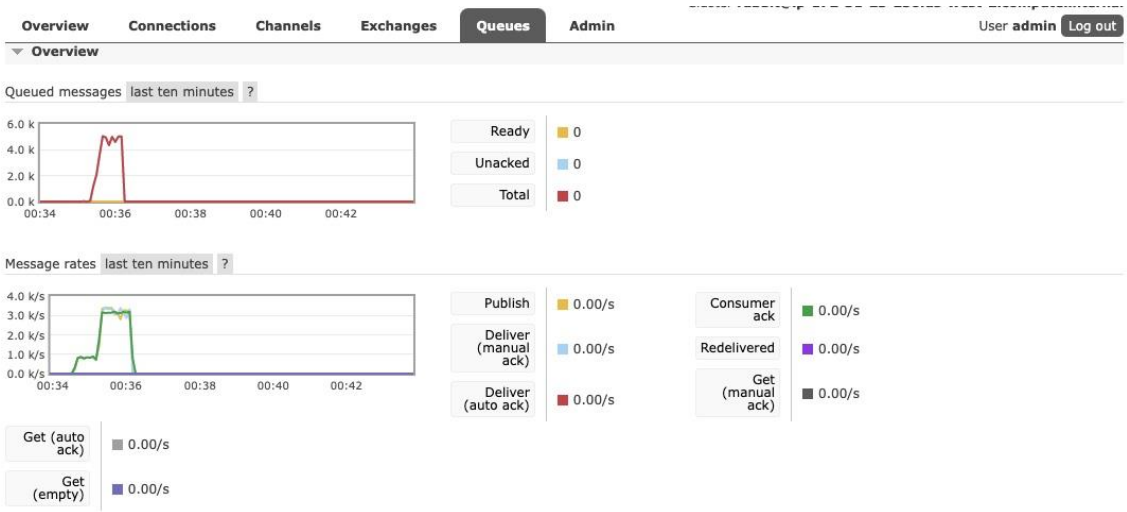
Redis

Throughput

```
----------Configuration----------
Number of Threads: 200
Total Requests: 200000
----------Calculations----------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 105451 ms
Throughput: 1896.6154896587038 requests per second
----------Stats----------
Mean: 58 ms
Median: 41 ms
Min Response Time: 19 ms
Max Response Time: 1034 ms
99th Percentile: 21 ms
```

Test 15

Rabbit MQ



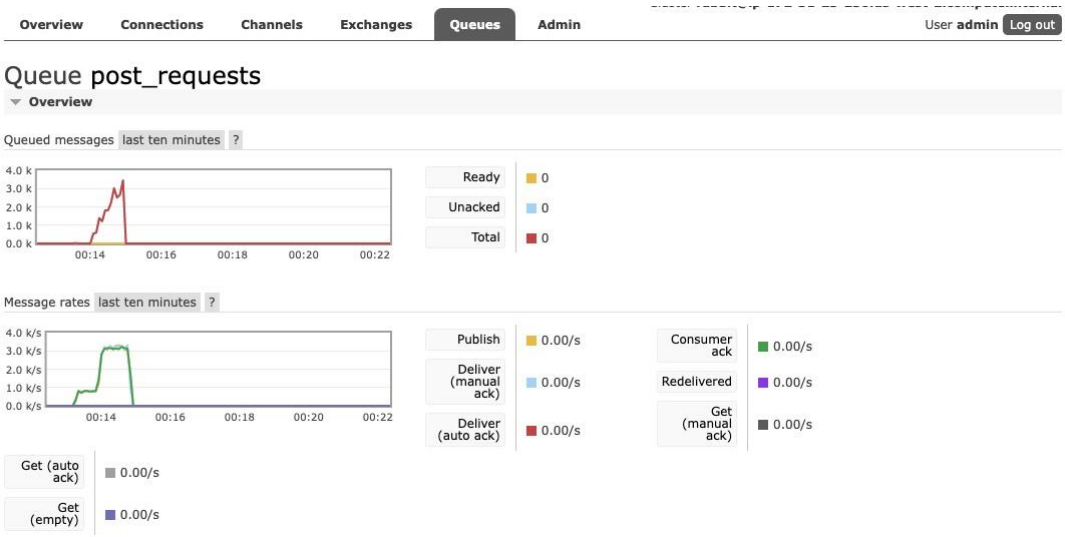
Redis

Throughput

```
----------------------------Configuration----------------------------
Number of Threads: 200
Total Requests: 200000
-----------------------------Calculations----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 95172 ms
Throughput: 2101.458412138024 requests per second
---------------------------------Stats---------------------------------
Mean: 51 ms
Median: 38 ms
Min Response Time: 18 ms
Max Response Time: 932 ms
99th Percentile: 20 ms
```

Test 16

Rabbit MQ



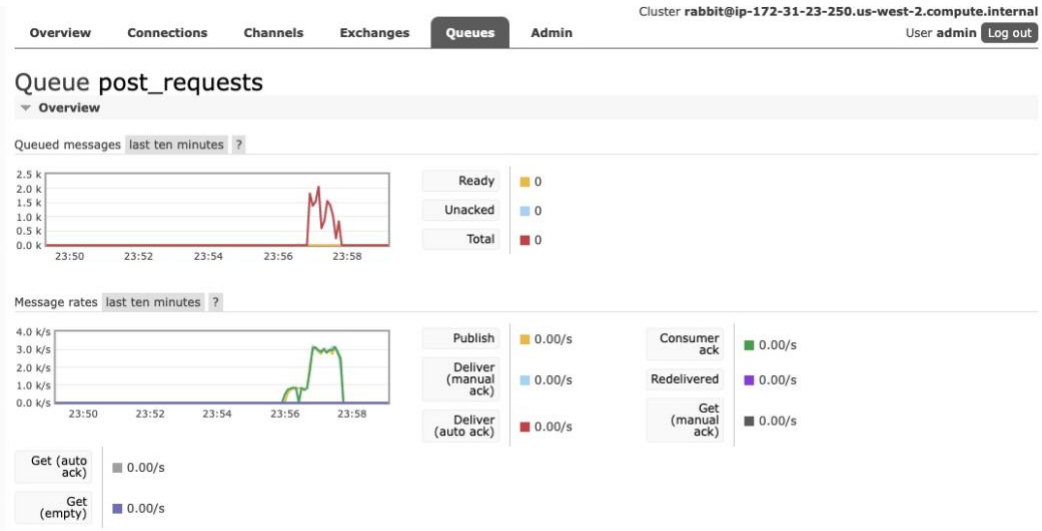
Redis

Throughput

```
----------------------------Configuration----------------------------
Number of Threads: 200
Total Requests: 200000
-----------------------------Calculations----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 97278 ms
Throughput: 2055.9633216143425 requests per second
--------------------------------Stats--------------------------------
Mean: 52 ms
Median: 39 ms
Min Response Time: 18 ms
Max Response Time: 1092 ms
99th Percentile: 21 ms
```

Test 17

Rabbit MQ

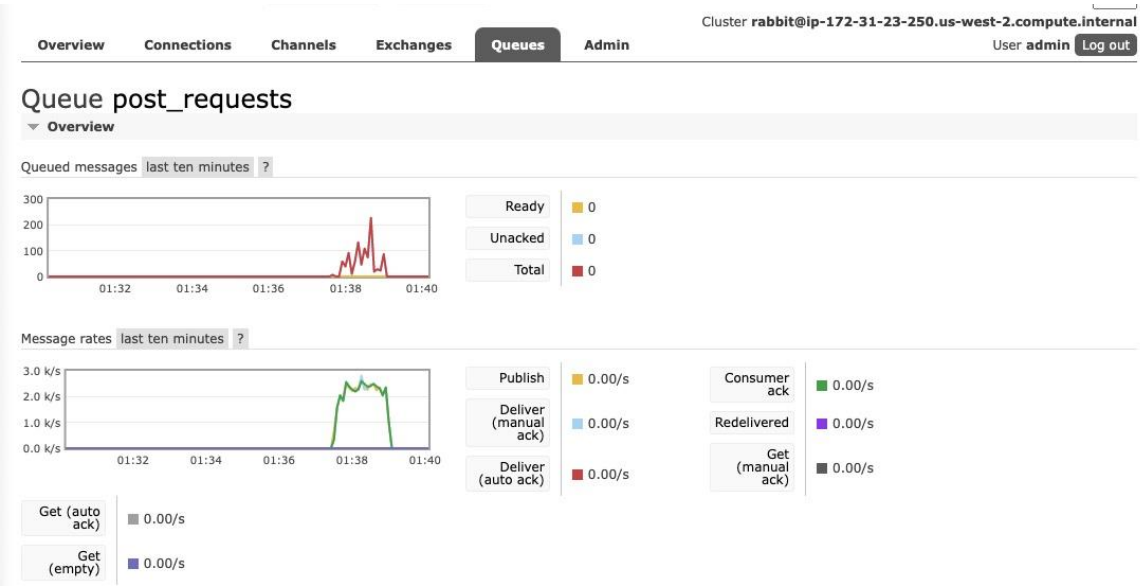


Redis

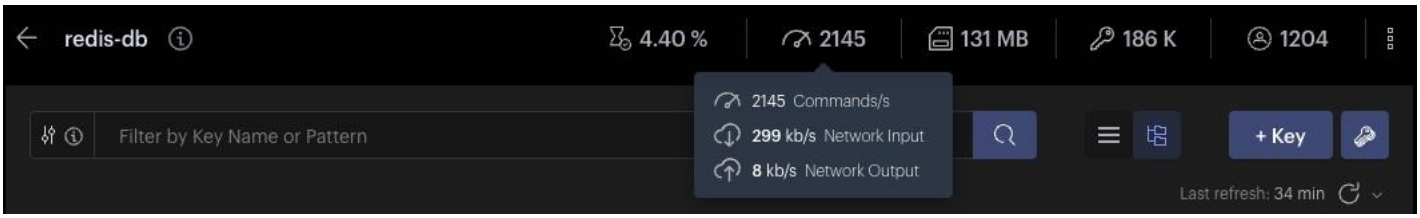
Throughput

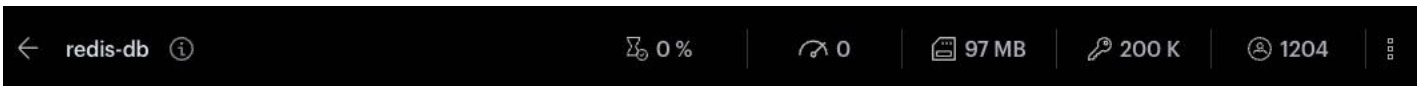
```
----------Configuration----------
Number of Threads: 200
Total Requests: 200000
----------Calculations----------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 100596 ms
Throughput: 1988.1506222911446 requests per second
----------Stats----------
Mean: 55 ms
Median: 41 ms
Min Response Time: 18 ms
Max Response Time: 1106 ms
99th Percentile: 21 ms
```

Rabbit MQ



Redis



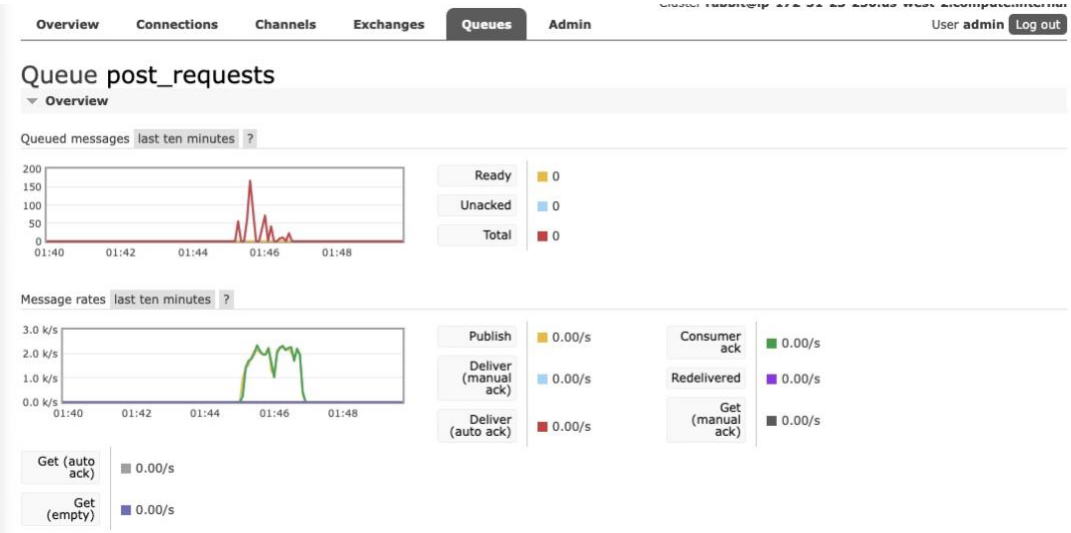


Throughput

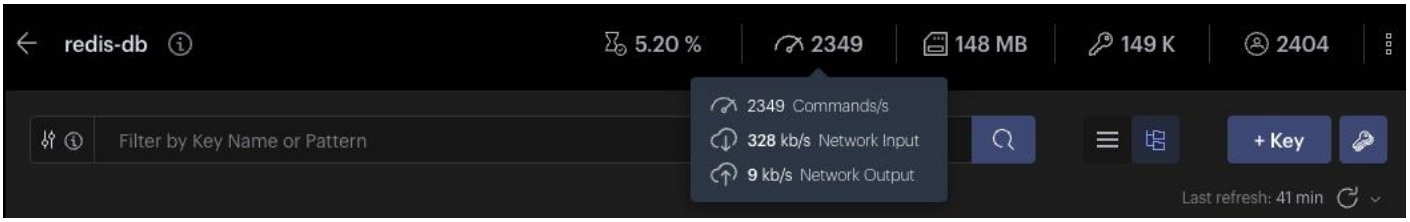
```
---------------------------Configuration-----------------------------
Number of Threads: 200
Total Requests: 200000
---------------------------Calculations-----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 94898 ms
Throughput: 2107.5259752576453 requests per second
--------------------------------Stats--------------------------------
Mean: 46 ms
Median: 33 ms
Min Response Time: 17 ms
Max Response Time: 1544 ms
99th Percentile: 21 ms
```

Test 19

Rabbit MQ



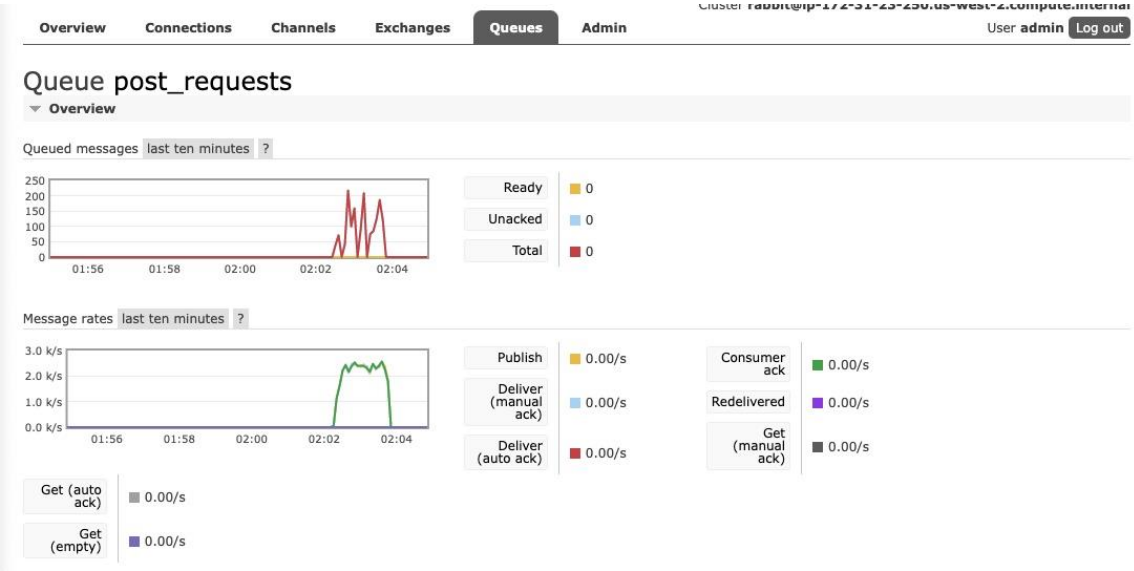
Redis



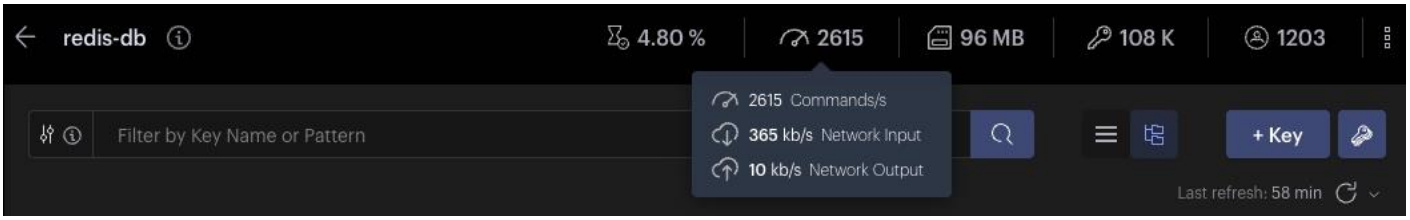
Throughput

```
---------------------------Configuration---------------------------
Number of Threads: 200
Total Requests: 200000
---------------------------Calculations---------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 109342 ms
Throughput: 1829.123301201734 requests per second
-------------------------------Stats-------------------------------
Mean: 55 ms
Median: 36 ms
Min Response Time: 19 ms
Max Response Time: 1481 ms
99th Percentile: 22 ms
```

Rabbit MQ



Redis

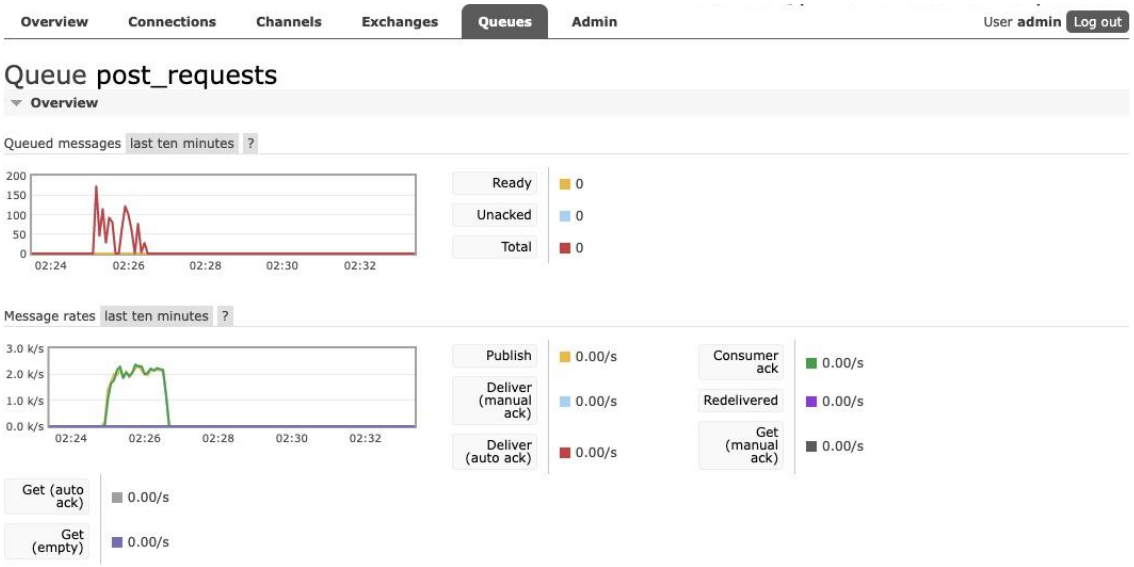


Throughput

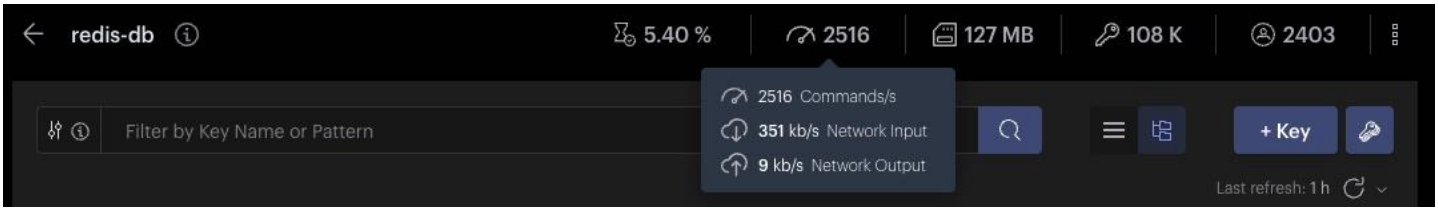
```
---------------------------Configuration---------------------------
Number of Threads: 200
Total Requests: 200000
---------------------------Calculations---------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 93241 ms
Throughput: 2144.9791400778627 requests per second
-------------------------------Stats-------------------------------
Mean: 46 ms
Median: 33 ms
Min Response Time: 17 ms
Max Response Time: 2321 ms
99th Percentile: 21 ms
```

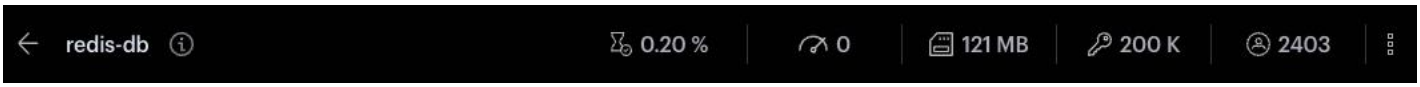
Test 21

Rabbit MQ



Redis





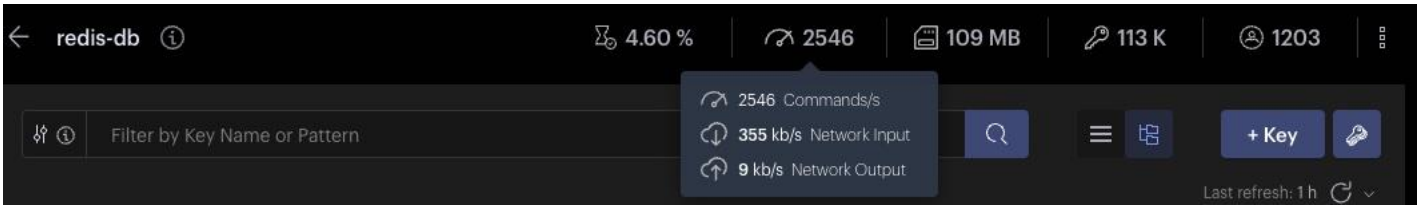
Throughput

```
--------------------------Configuration--------------------------
Number of Threads: 200
Total Requests: 200000
---------------------------Calculations--------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 104564 ms
Throughput: 1912.70418117134 requests per second
-------------------------------Stats-------------------------------
Mean: 51 ms
Median: 36 ms
Min Response Time: 17 ms
Max Response Time: 1610 ms
99th Percentile: 22 ms
```

Test 22

Rabbit MQ

Redis

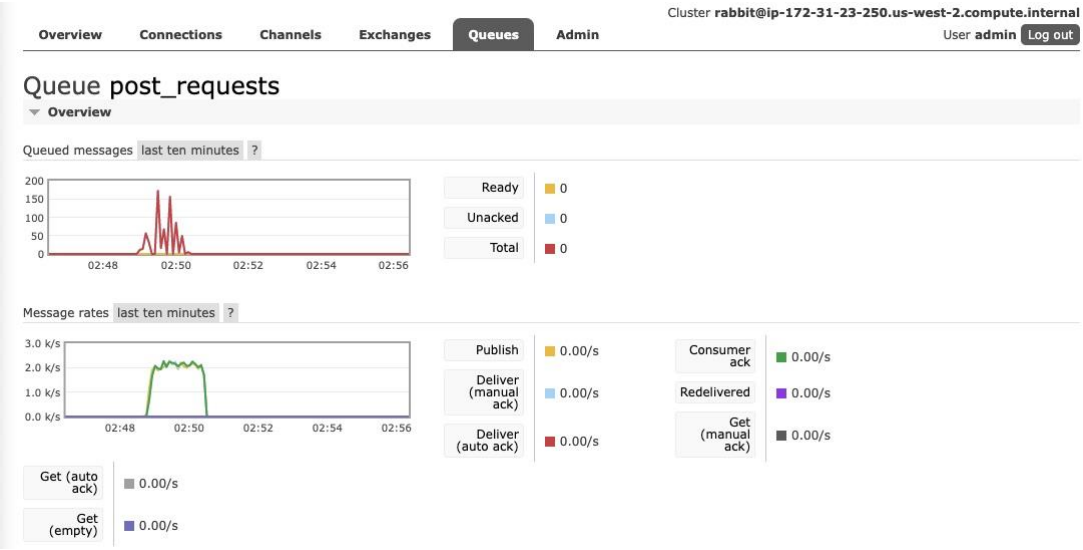


Throughput

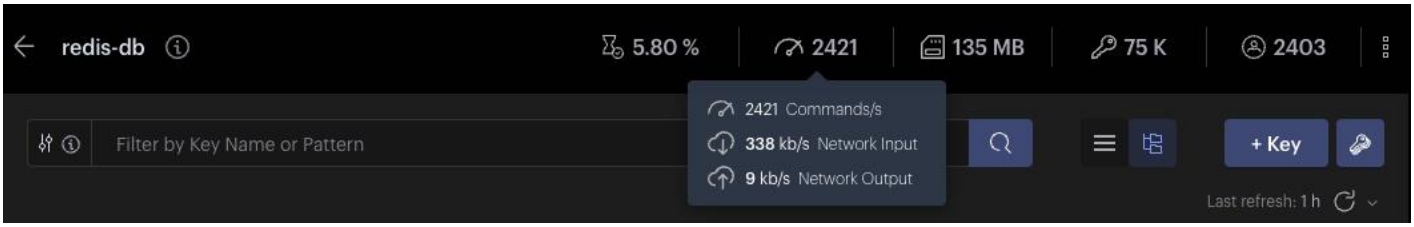
```
----------Configuration----------
Number of Threads: 200
Total Requests: 200000
----------Calculations----------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 102078 ms
Throughput: 1959.2860361684202 requests per second
----------Stats----------
Mean: 50 ms
Median: 35 ms
Min Response Time: 19 ms
Max Response Time: 2484 ms
99th Percentile: 22 ms
```

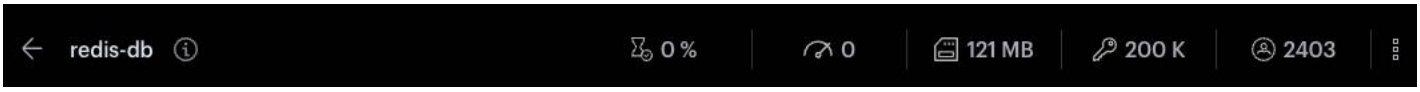
Test 23

Rabbit MQ



Redis

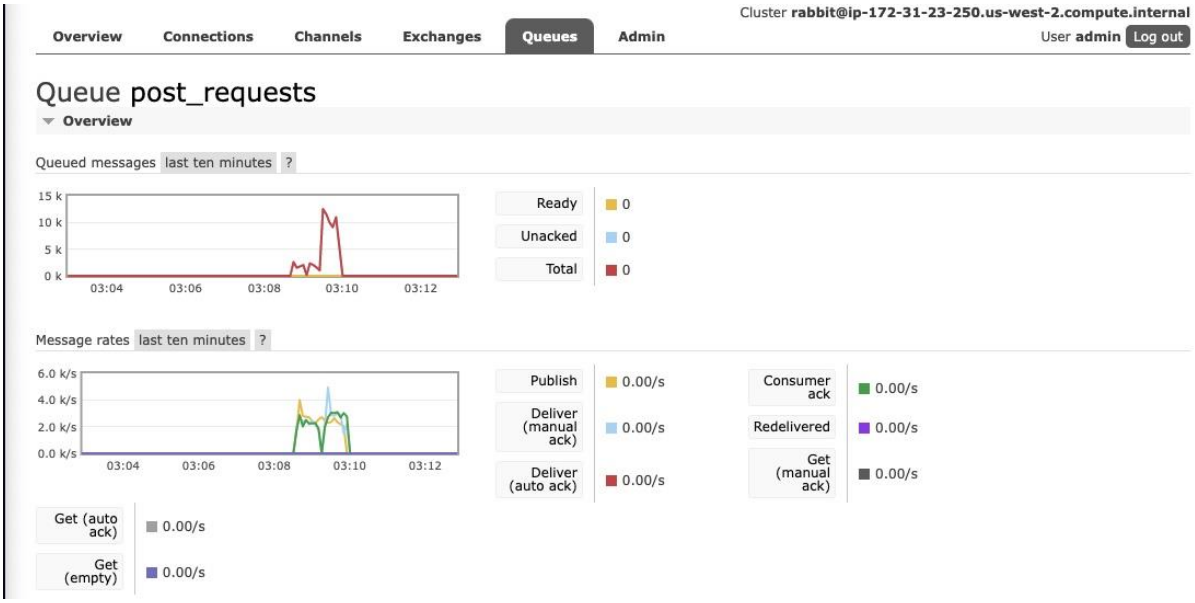




Throughput

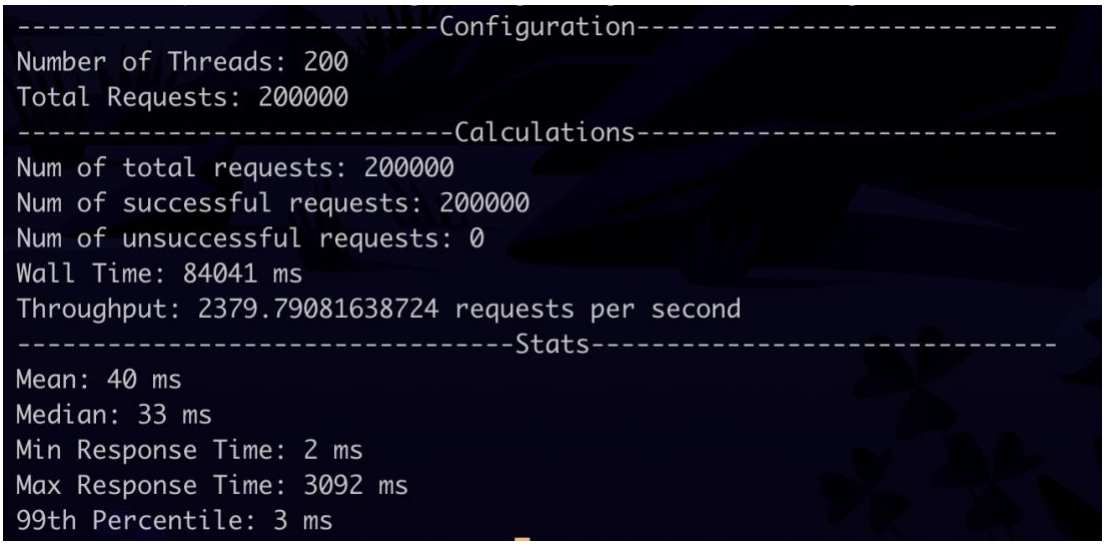
```
-------------------------Configuration-------------------------
Number of Threads: 200
Total Requests: 200000
-------------------------Calculations-------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 103844 ms
Throughput: 1925.96587188475 requests per second
-----------------------------Stats-----------------------------
Mean: 51 ms
Median: 37 ms
Min Response Time: 18 ms
Max Response Time: 1497 ms
99th Percentile: 23 ms
```

Rabbit MQ



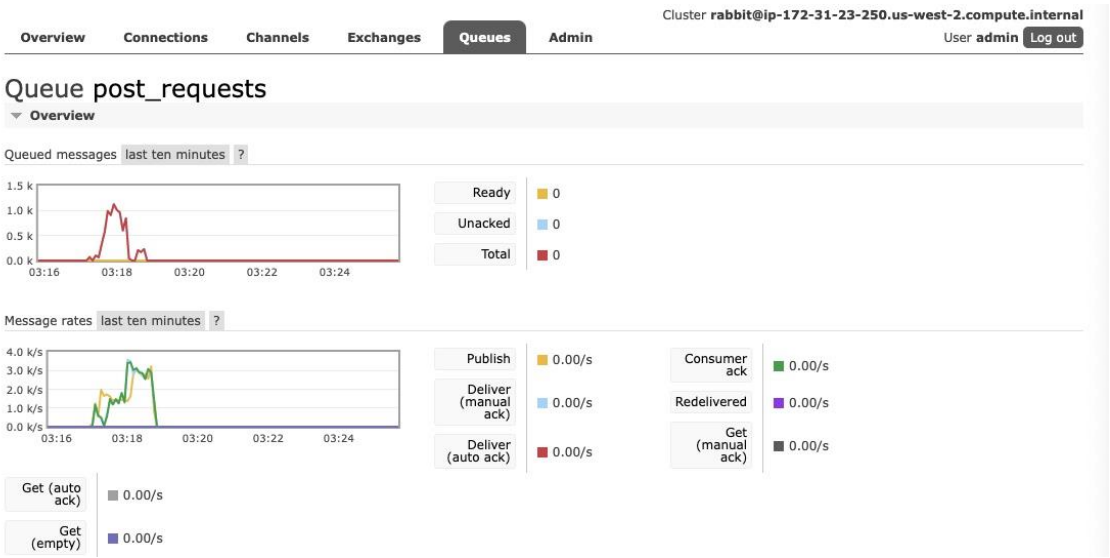
Redis

Throughput

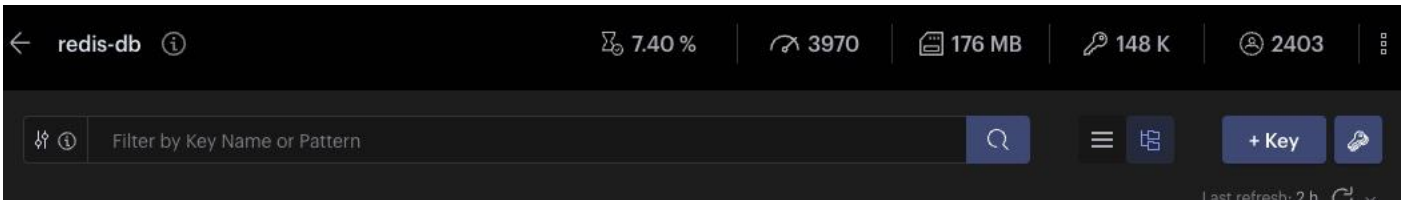


Test 25

Rabbit MQ



Redis



Throughput

```
----------------------------Configuration-----------------------------
Number of Threads: 200
Total Requests: 200000
----------------------------Calculations-----------------------------
Num of total requests: 200000
Num of successful requests: 200000
Num of unsuccessful requests: 0
Wall Time: 106628 ms
Throughput: 1875.6799339760664 requests per second
---------------------------------Stats---------------------------------
Mean: 48 ms
Median: 33 ms
Min Response Time: 2 ms
Max Response Time: 1272 ms
99th Percentile: 3 ms
```