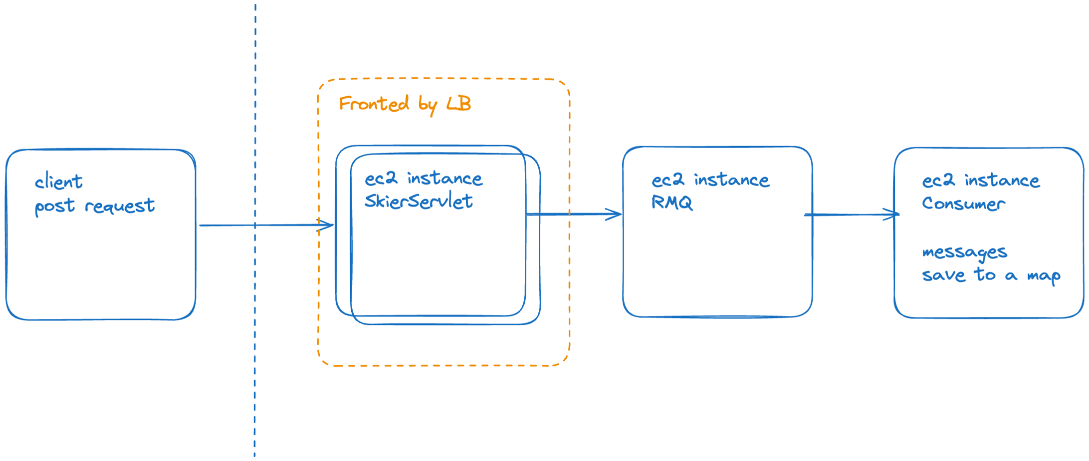
A2 Report- Yifei Dong

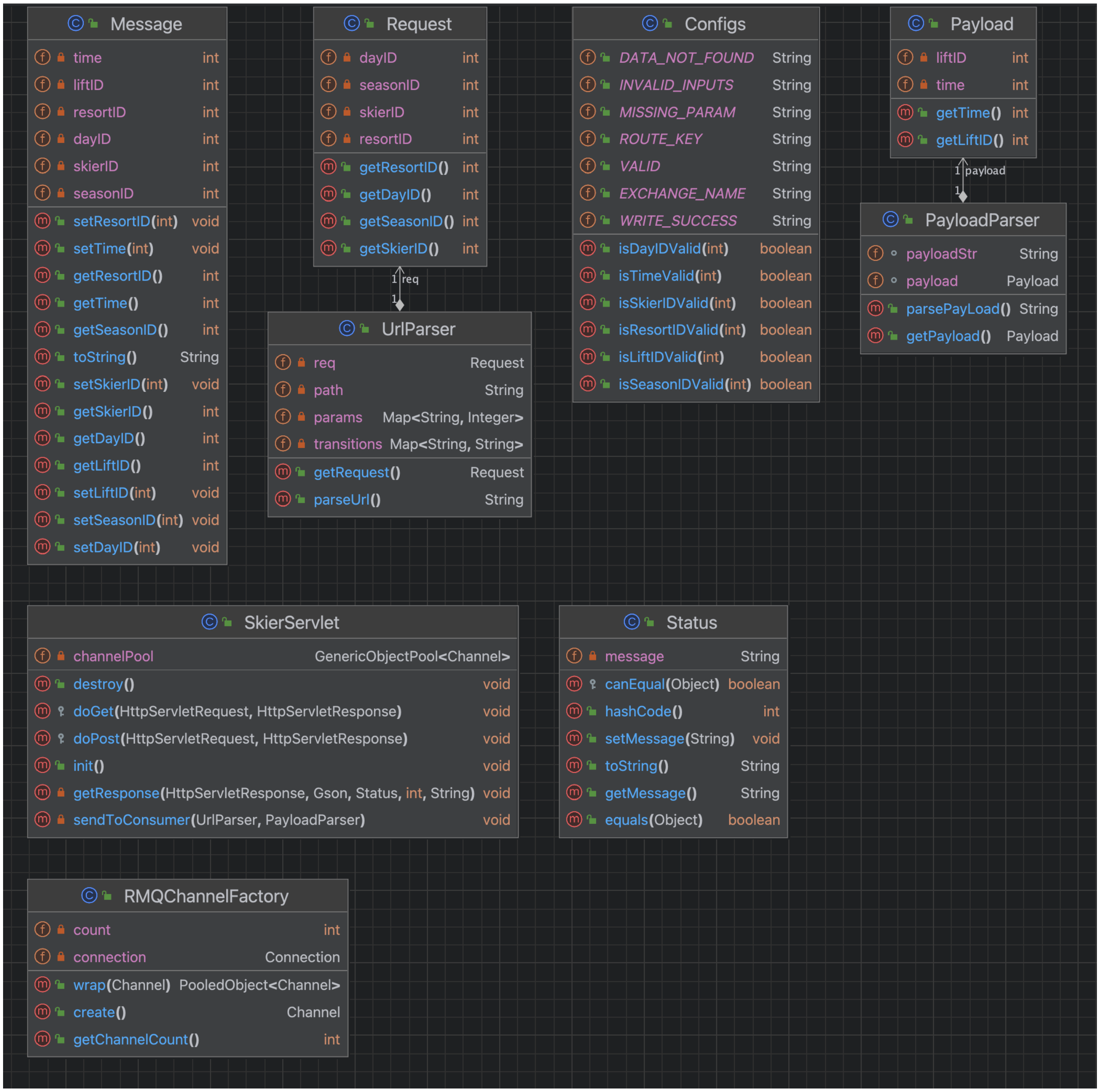
High Level Architecture

Clients send post requests to SkierServlet sitting on EC2 instance and validate the requests. After validation, the information in request url path and request body will be stored as String message ready for further processing. The validated message will be sent over to a RabbitMQ on another EC2 instance and eventually be pulled out of the queue by our consumer. Consumer gets the message and parses it into structured information and finally stored into a map using skierID as key.



As we recall in A1, client launches 32 threads and sends each 1000 POST requests to servlet. After that a total number of 20k requests will be sent to the servlet. And servlet pooled channels to send validated requests as messages to RabbitMQ and finally pull out by consumers in a multi-threaded fashion.

Servlet java image



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

Request	
dayID	int
seasonID	int
skierID	int
resortID	int
getResortID()	int
getDayID()	int
getSeasonID()	int
getSkierID()	int

Configs	
DATA_NOT_FOUND	String
INVALID_INPUTS	String
MISSING_PARAM	String
ROUTE_KEY	String
VALID	String
EXCHANGE_NAME	String
WRITE_SUCCESS	String
isDayIDValid(int)	boolean
isTimeValid(int)	boolean
isSkierIDValid(int)	boolean
isResortIDValid(int)	boolean
isLiftIDValid(int)	boolean
isSeasonIDValid(int)	boolean

Payload	
liftID	int
time	int
getTime()	int
getLiftID()	int

PayloadParser	
payloadStr	String
payload	Payload
parsePayLoad()	String
getPayload()	Payload

UrlParser	
req	Request
path	String
params	Map<String, Integer>
transitions	Map<String, String>
getRequest()	Request
parseUrl()	String

SkierServlet	
channelPool	GenericObjectPool<Channel>
destroy()	void
doGet(HttpServletRequest, HttpServletResponse)	void
doPost(HttpServletRequest, HttpServletResponse)	void
init()	void
getResponse(HttpServletResponse, Gson, Status, int, String)	void
sendToConsumer(UrlParser, PayloadParser)	void

Status	
message	String
canEqual(Object)	boolean
hashCode()	int
setMessage(String)	void
toString()	String
getMessage()	String
equals(Object)	boolean

RMQChannelFactory	
count	int
connection	Connection
wrap(Channel)	PooledObject<Channel>
create()	Channel
getChannelCount()	int

POJO class: Message, Status, Payload, Request.
Message: POJO to include validated data in its fields from POST request url and POST request body.
Payload: POJO to include validated data in POST request body.
Request: POJO to include validated data in POST request url.
Status: POJO for printing response message purpose.

Configs: To include configurations and static methods for data validation.

Parser class: PayloadParser, UrlParser
PayloadParser: To parse and validate payload data, and convert it into a string. And it has a method to get validated payload in structured data.
UrlParser: To parse and validate url and its data in the path.

SkierServlet: Major runner of the servlet.
RMQChannelFactory: A simple RabbitMQ channel factory based on the Apache pooling libraries.

More on URL Parsing and validation

Url format:
IP:8080/<application>/skiers/{resortID}/seasons/{seasonID}/days/{dayID}/skiers/{skierID}

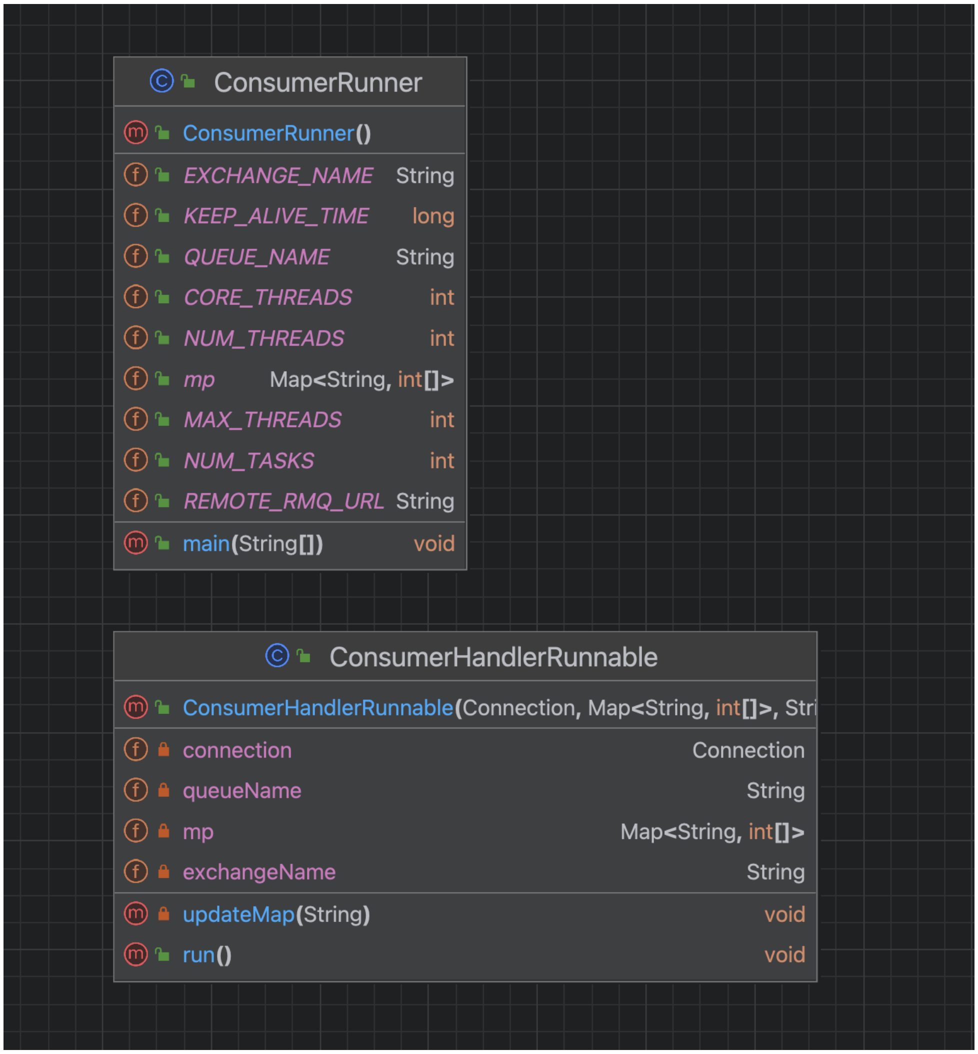
UrlParser processing steps:
Step 1: Analyze and validate url

Url left empty like "IP:8080/<application>/skiers" gives 400, invalid inputs
Url missing param as "IP:8080/<application>/skiers/seasons/days/skiers"
gives 400, missing param
Url not aligning the format as misspelling or other errors
"IP:8080/<application>/skiers/{resortID}/<error_input>/"
gives 400, invalid inputs
Valid url path format + path param in valid range = url is valid

Step 2: Validate req body
Or url valid but with liftID/time missing
Invalid liftID/time format as misspelling or other errors gives 400 invalid inputs
Valid lifetID/time format but missing lifeID/time data gives 400 data not found
Valid req body format + req body data in valid range = req body is valid

Step 3:
Pack the validated data, send it to a queue and return 201 to client

Consumer java image

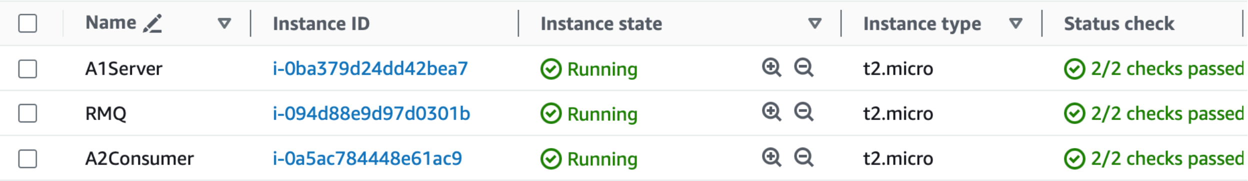


ConsumerRunner	
ConsumerRunner()	
EXCHANGE_NAME	String
KEEP_ALIVE_TIME	long
QUEUE_NAME	String
CORE_THREADS	int
NUM_THREADS	int
mp	Map<String, int[]>
MAX_THREADS	int
NUM_TASKS	int
REMOTE_RMQ_URL	String
main(String[])	void

ConsumerHandlerRunnable	
ConsumerHandlerRunnable(Connection, Map<String, int[]>, Stri	
connection	Connection
queueName	String
mp	Map<String, int[]>
exchangeName	String
updateMap(String)	void
run()	void

ConsumerRunner utilizes a self-defined thread pool to consume messages pulled from RabbitMQ. ConsumerHandlerRunnable defines a specific task submitted to the thread pool, in which updateMap method is invoked to parse the message into a key-value object – skierID as key, and an array like [time, liftID, resortID, seasonID, dayID, skierID] stores the remaining data left in the message.

Single instance tests without load balancers



□	Name	Instance ID	Instance state	Instance type	Status check
□	A1Server	i-0ba379d24dd42bea7	✓ Running	t2.micro	✓ 2/2 checks passed
□	RMQ	i-094d88e9d97d0301b	✓ Running	t2.micro	✓ 2/2 checks passed
□	A2Consumer	i-0a5ac784448e61ac9	✓ Running	t2.micro	✓ 2/2 checks passed

We will start with test max throughputs with a single servlet and no load balancers. The Servlet is setup on A1Server instance and other instances sitting on RMQ and A2Consumer respectively.

Summary

No. Client threads	No. Consumer threads	Queue length	Throughputs	Wall time
50	50	2~5	1098 req/s	182 s
50	100	1~11	1092 req/s	183 s
100	100	1~4	1020 req/s	196 s
100	200	1~4	1156 req/s	173 s
200	200	1~3	1351 req/s	148 s
200	300	2~5	1360 req/s	148 s

Answers to questions:

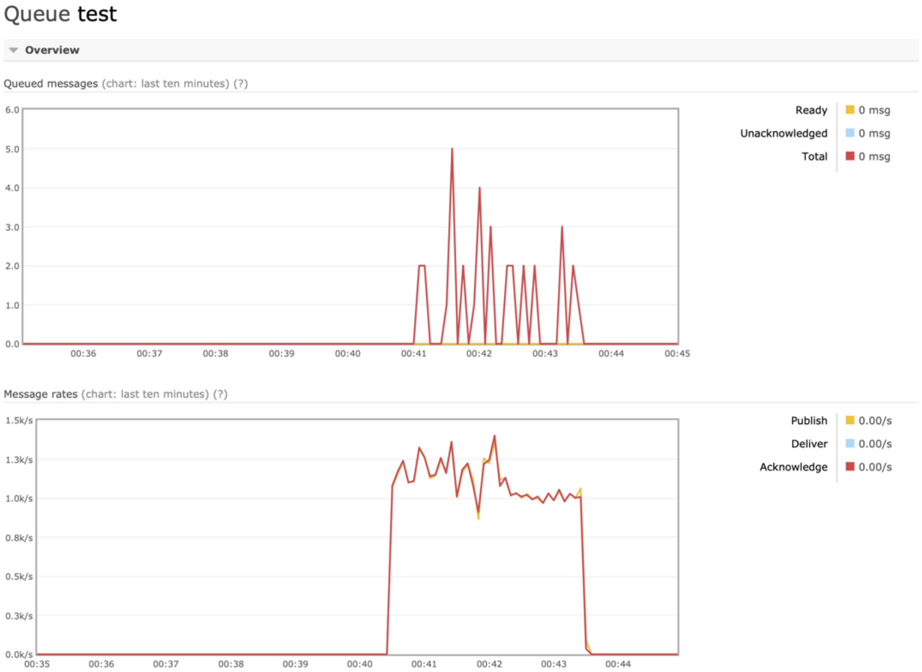
□ How many client threads are optimal after phase 1 to maximize throughput?

200 client threads seems to be optimal in my case as it performs with more throughput and the smallest queue length. Although not included here, a greater number of client threads like 300 against 300 consumer threads will not significantly improve the throughput, but in reverse might cause request failure (i.e. retry > 5) due to consumers competing to pull message and update on a synchronized map.

□ How many queue consumers threads do I need to keep the queue size as close to zero as possible?

200 consumer threads is needed to keep the queue size as small as possible.

50 client threads && 50 consumer threads


Queue test

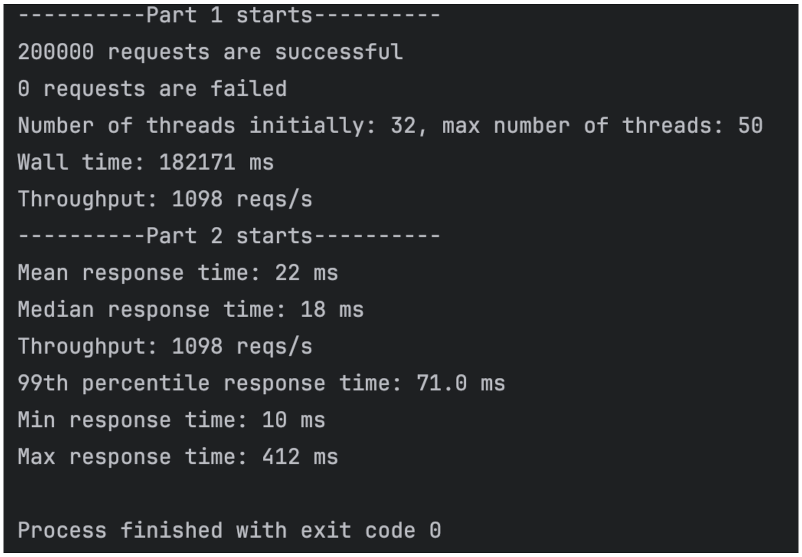
Overview

Queued messages (chart: last ten minutes) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

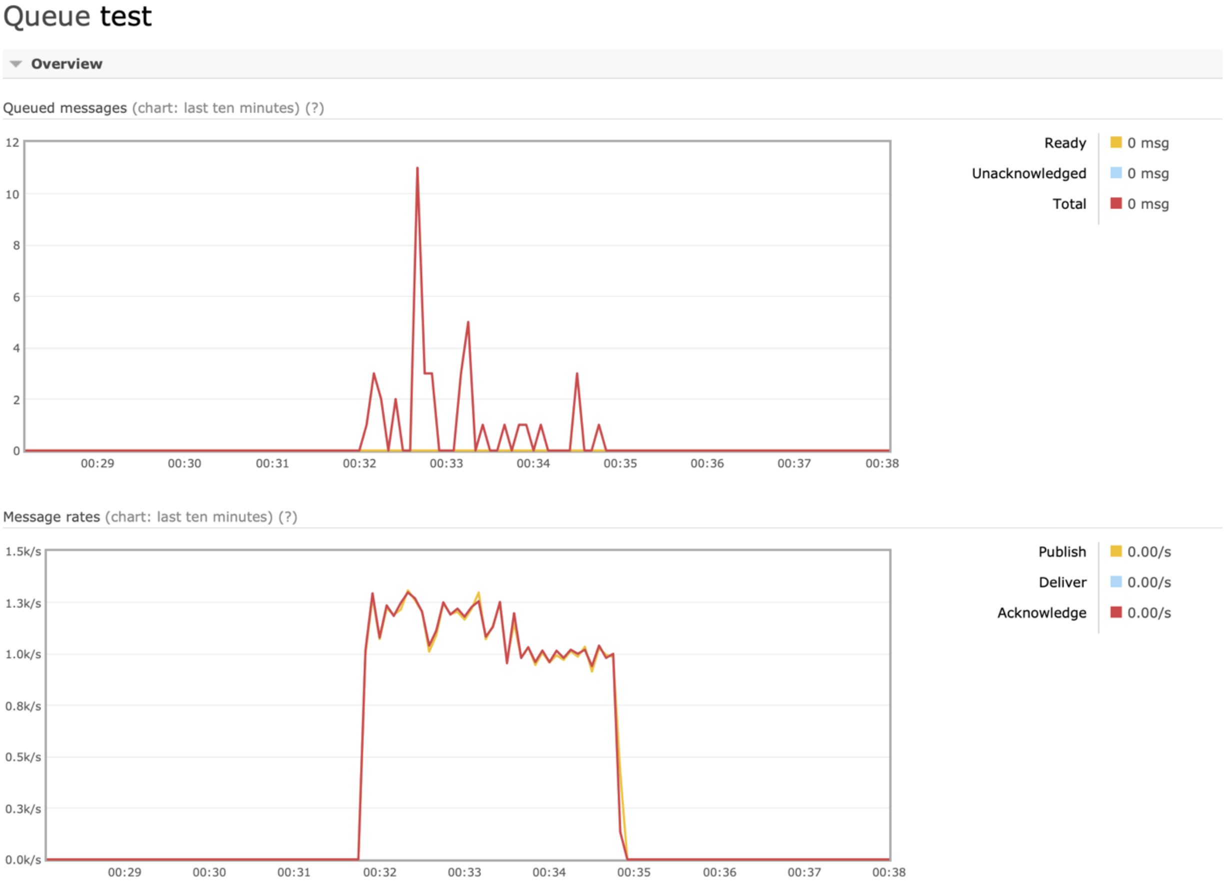
Message rates (chart: last ten minutes) (?)

Publish	🟨 0.00/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s


```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 50
Wall time: 182171 ms
Throughput: 1098 reqs/s
----------Part 2 starts----------
Mean response time: 22 ms
Median response time: 18 ms
Throughput: 1098 reqs/s
99th percentile response time: 71.0 ms
Min response time: 10 ms
Max response time: 412 ms

Process finished with exit code 0
```

50 client threads && 100 consumer threads



Queue test

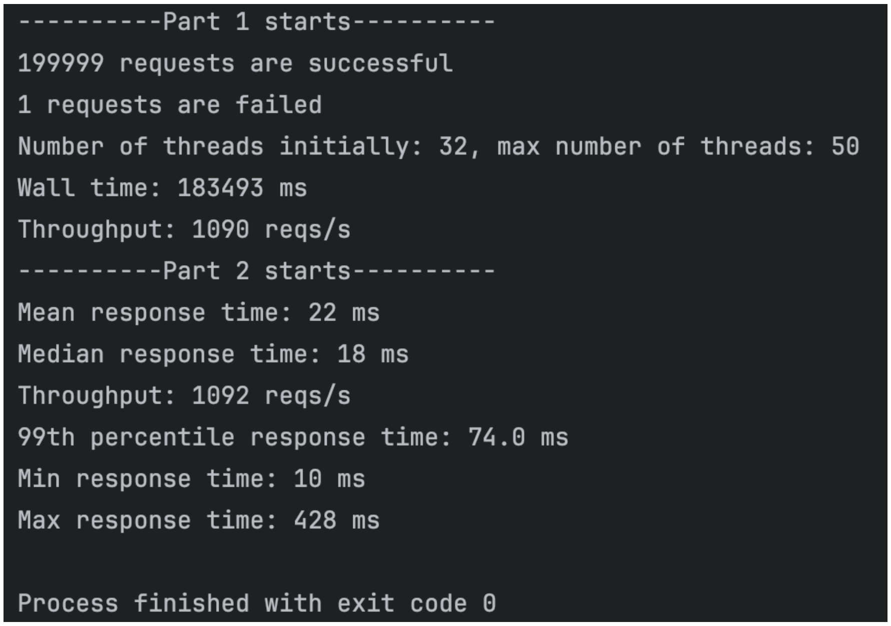
Overview

Queued messages (chart: last ten minutes) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

Message rates (chart: last ten minutes) (?)

Publish	🟨 0.00/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s

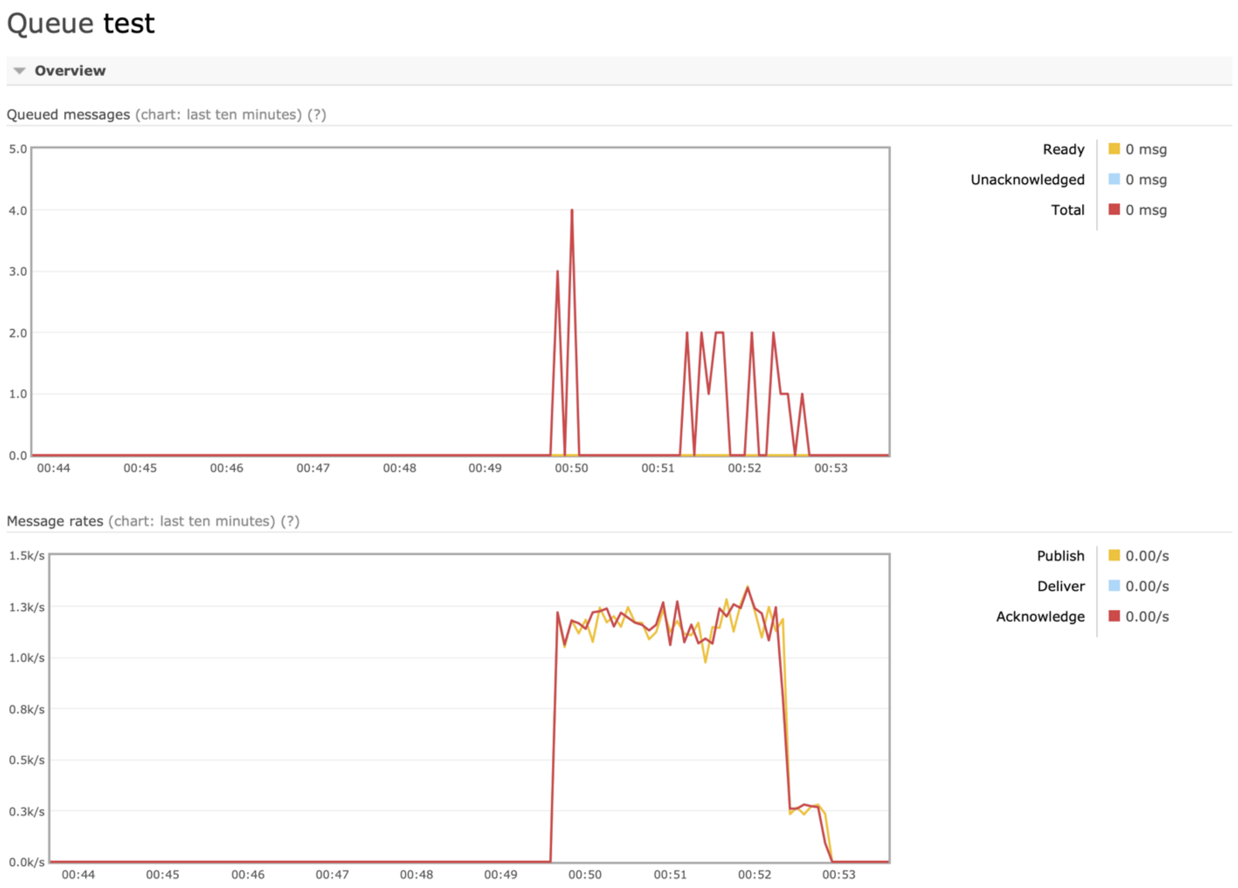


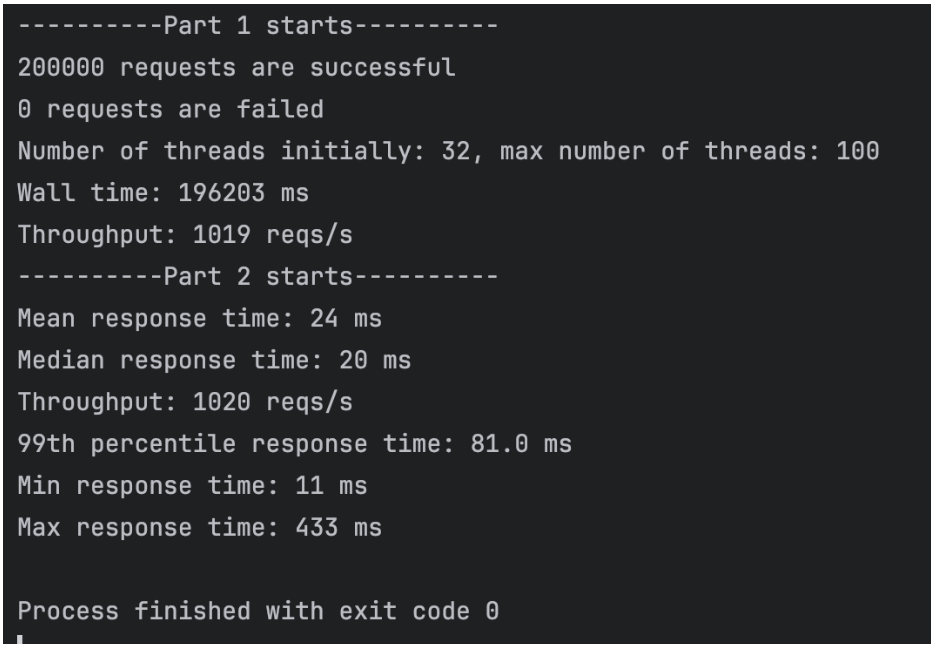
```
----------Part 1 starts----------
199999 requests are successful
1 requests are failed
Number of threads initially: 32, max number of threads: 50
Wall time: 183493 ms
Throughput: 1090 reqs/s
----------Part 2 starts----------
Mean response time: 22 ms
Median response time: 18 ms
Throughput: 1092 reqs/s
99th percentile response time: 74.0 ms
Min response time: 10 ms
Max response time: 428 ms

Process finished with exit code 0
```

There is a spike in queue length that might be caused by network jitter. Consequently, we can inspect a loss of request in the console.

100 client threads && 100 consumer threads

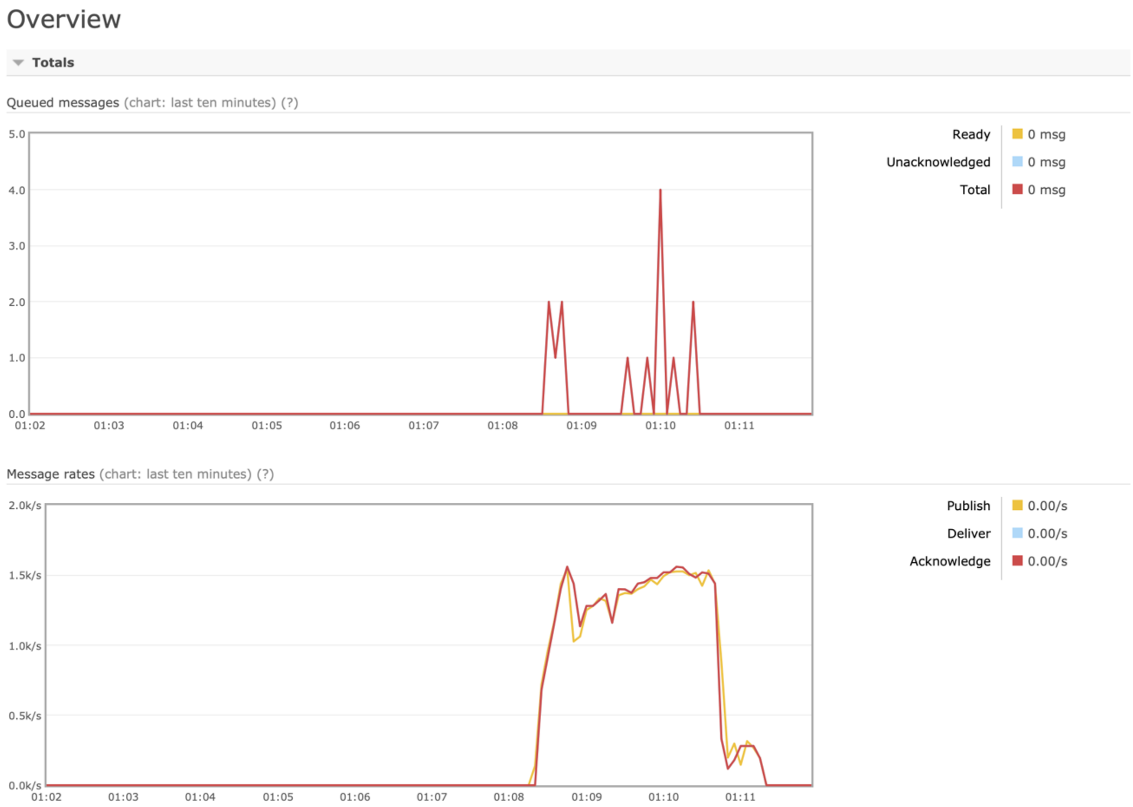




```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 100
Wall time: 196203 ms
Throughput: 1019 reqs/s
----------Part 2 starts----------
Mean response time: 24 ms
Median response time: 20 ms
Throughput: 1020 reqs/s
99th percentile response time: 81.0 ms
Min response time: 11 ms
Max response time: 433 ms

Process finished with exit code 0
```

100 client threads && 200 consumer threads


Overview

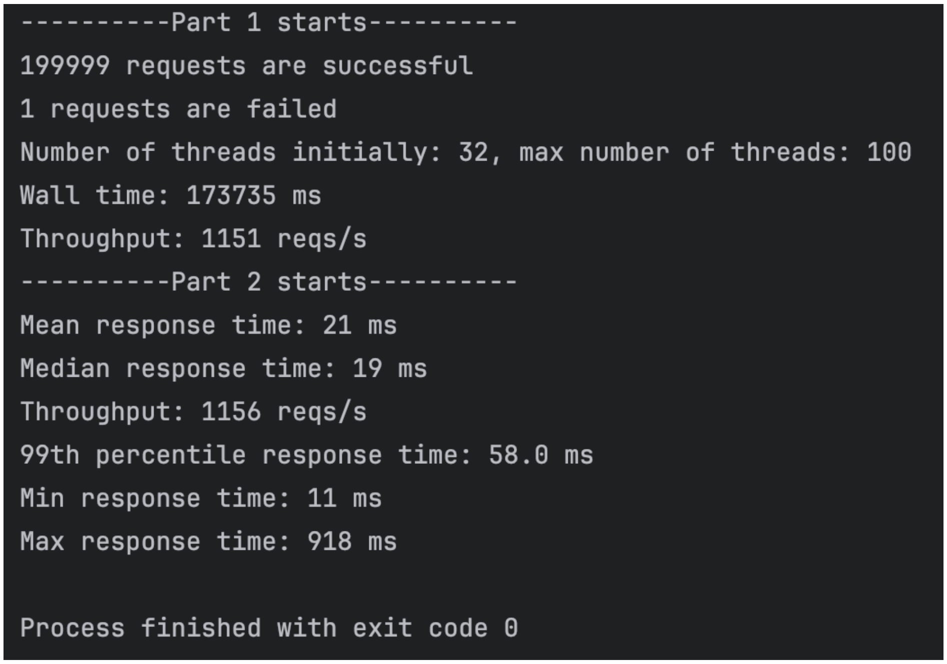
Totals

Queued messages (chart: last ten minutes) (?)

Ready 🟨 0 msg
Unacknowledged 🟦 0 msg
Total 🟥 0 msg

Message rates (chart: last ten minutes) (?)

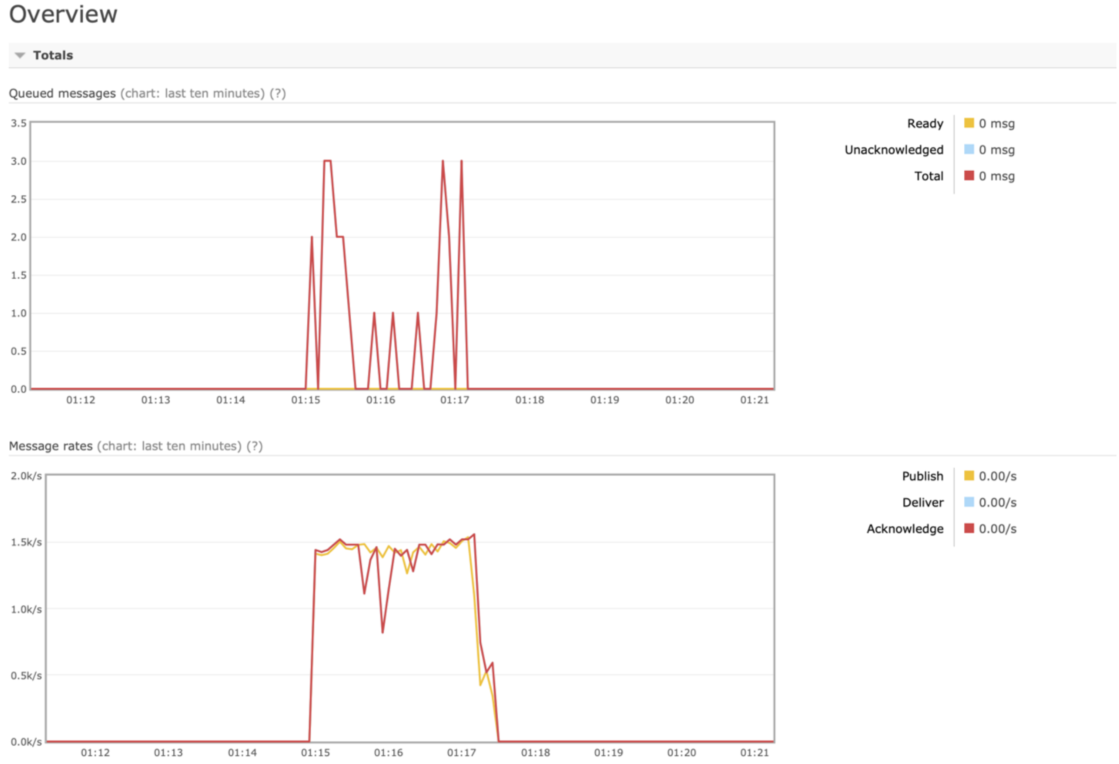
Publish 🟨 0.00/s
Deliver 🟦 0.00/s
Acknowledge 🟥 0.00/s



```
----------Part 1 starts----------
199999 requests are successful
1 requests are failed
Number of threads initially: 32, max number of threads: 100
Wall time: 173735 ms
Throughput: 1151 reqs/s
----------Part 2 starts----------
Mean response time: 21 ms
Median response time: 19 ms
Throughput: 1156 reqs/s
99th percentile response time: 58.0 ms
Min response time: 11 ms
Max response time: 918 ms

Process finished with exit code 0
```

200 client threads && 200 consumer threads



Overview

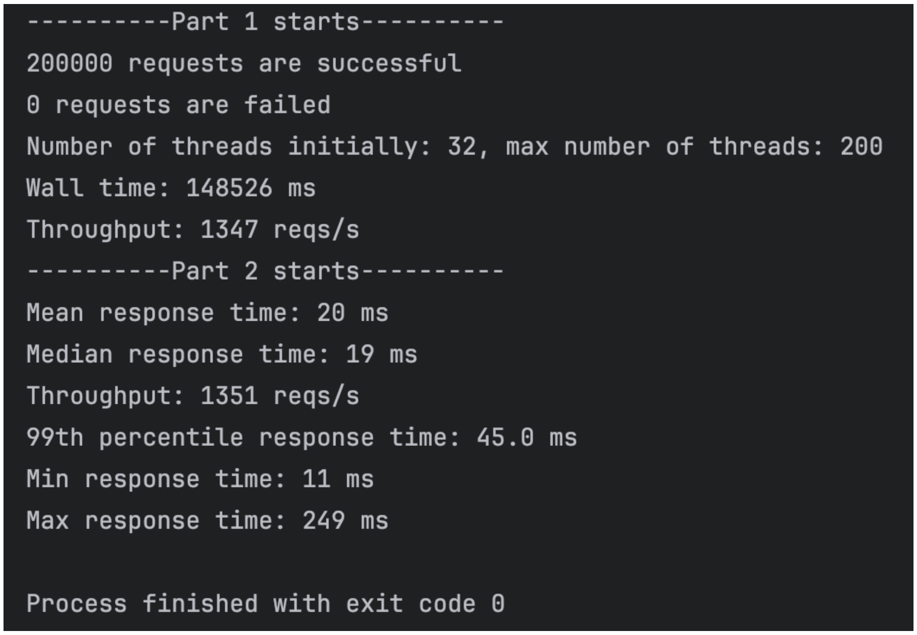
Totals

Queued messages (chart: last ten minutes) (?)

Ready 🟨 0 msg
Unacknowledged 🟦 0 msg
Total 🟥 0 msg

Message rates (chart: last ten minutes) (?)

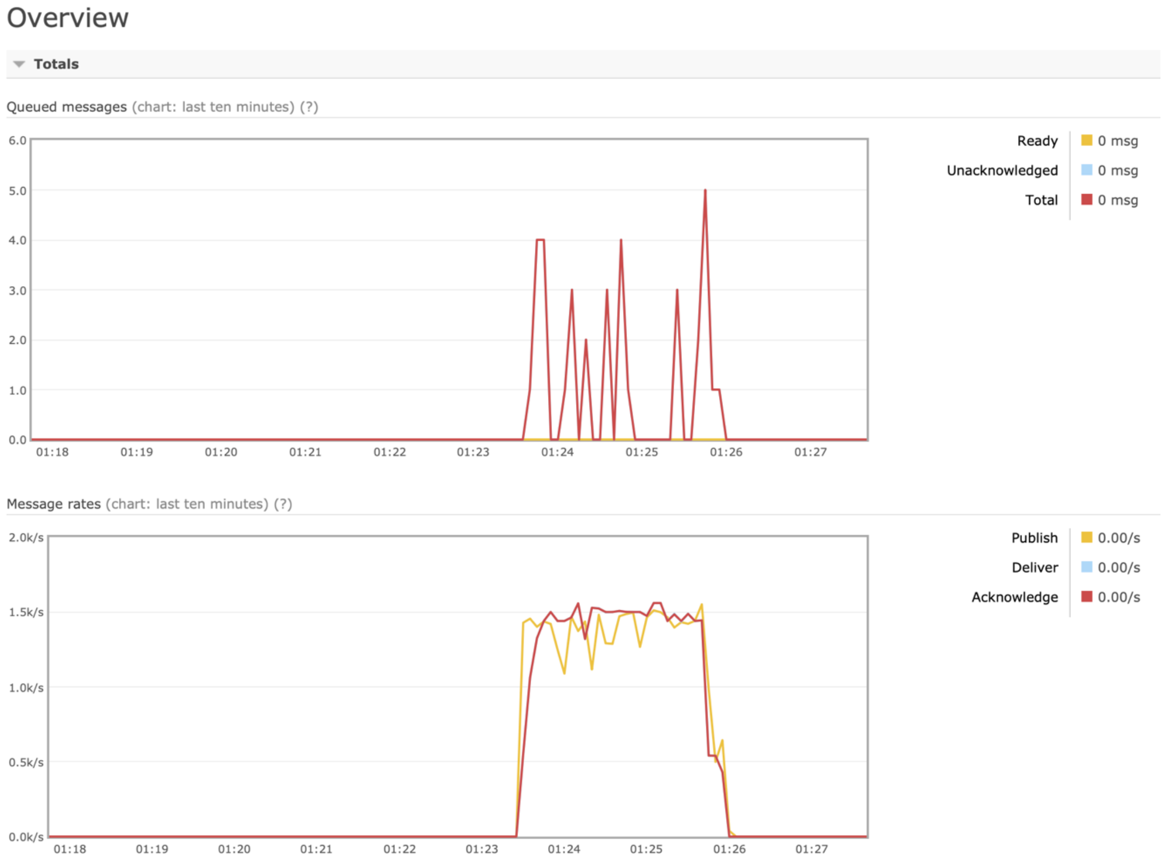
Publish 🟨 0.00/s
Deliver 🟦 0.00/s
Acknowledge 🟥 0.00/s



```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 200
Wall time: 148526 ms
Throughput: 1347 reqs/s
----------Part 2 starts----------
Mean response time: 20 ms
Median response time: 19 ms
Throughput: 1351 reqs/s
99th percentile response time: 45.0 ms
Min response time: 11 ms
Max response time: 249 ms

Process finished with exit code 0
```

200 client threads && 300 consumer threads



Overview

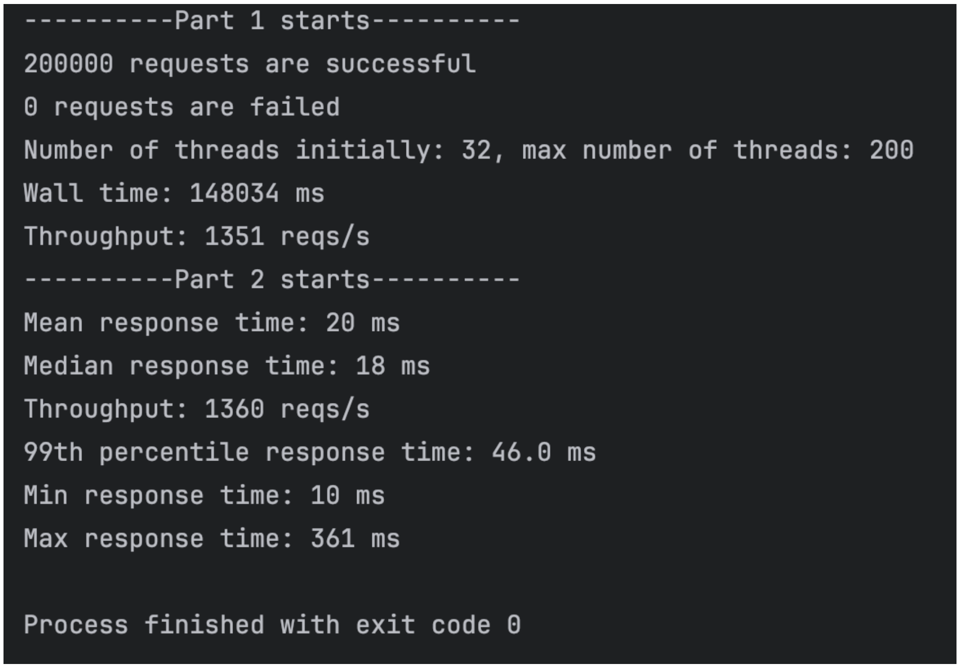
Totals

Queued messages (chart: last ten minutes) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

Message rates (chart: last ten minutes) (?)

Publish	🟨 0.00/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s

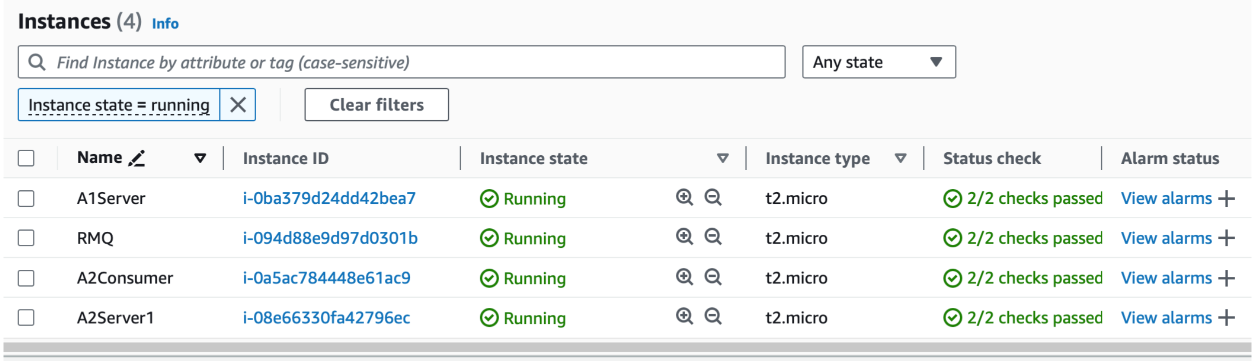


```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 200
Wall time: 148034 ms
Throughput: 1351 reqs/s
----------Part 2 starts----------
Mean response time: 20 ms
Median response time: 18 ms
Throughput: 1360 reqs/s
99th percentile response time: 46.0 ms
Min response time: 10 ms
Max response time: 361 ms

Process finished with exit code 0
```

Load balanced 2 instance tests

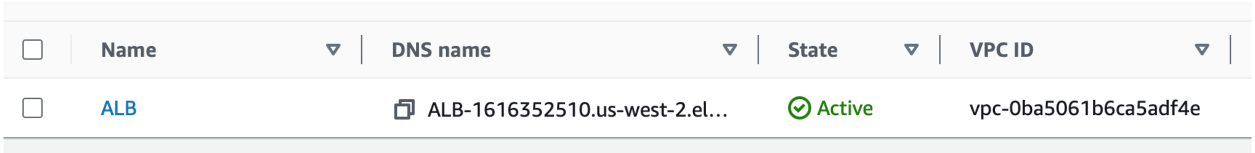
Instances



Instances (4) Info

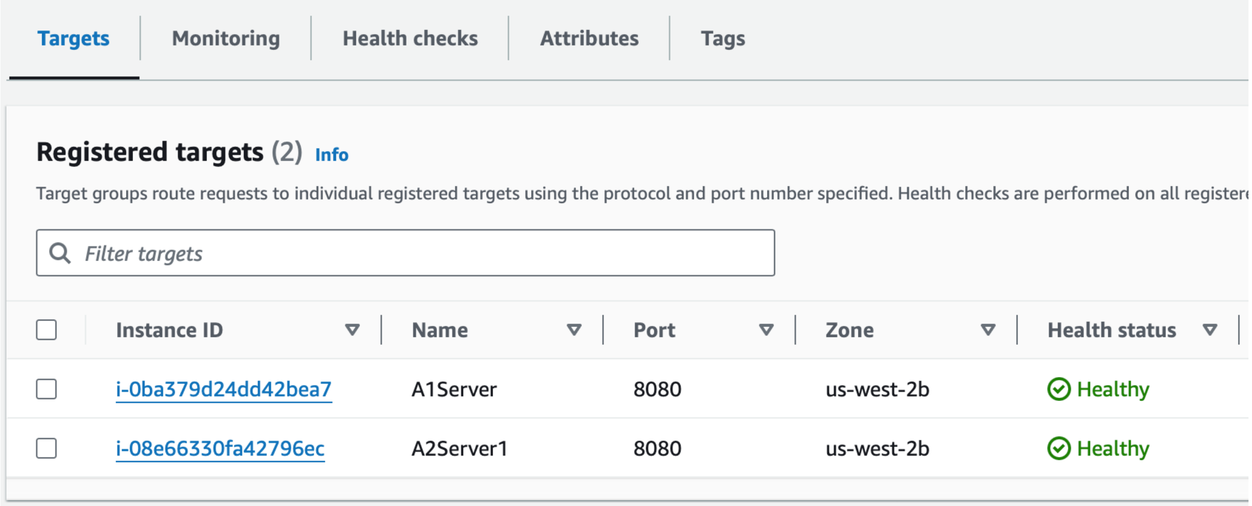
□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status
□	A1Server	i-0ba379d24dd42bea7	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +
□	RMQ	i-094d88e9d97d0301b	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +
□	A2Consumer	i-0a5ac784448e61ac9	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +
□	A2Server1	i-08e66330fa42796ec	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +

ALB



□	Name	DNS name	State	VPC ID
□	ALB	ALB-1616352510.us-west-2.el…	✓ Active	vpc-0ba5061b6ca5adf4e

Target Group of 2 instances

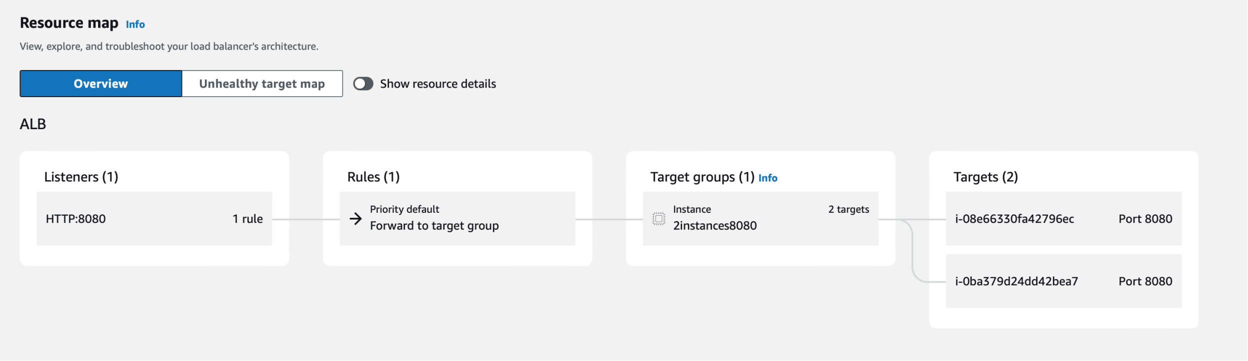


Registered targets (2) Info

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registere

□	Instance ID	Name	Port	Zone	Health status
□	i-0ba379d24dd42bea7	A1Server	8080	us-west-2b	✓ Healthy
□	i-08e66330fa42796ec	A2Server1	8080	us-west-2b	✓ Healthy

Resource map



Resource map Info

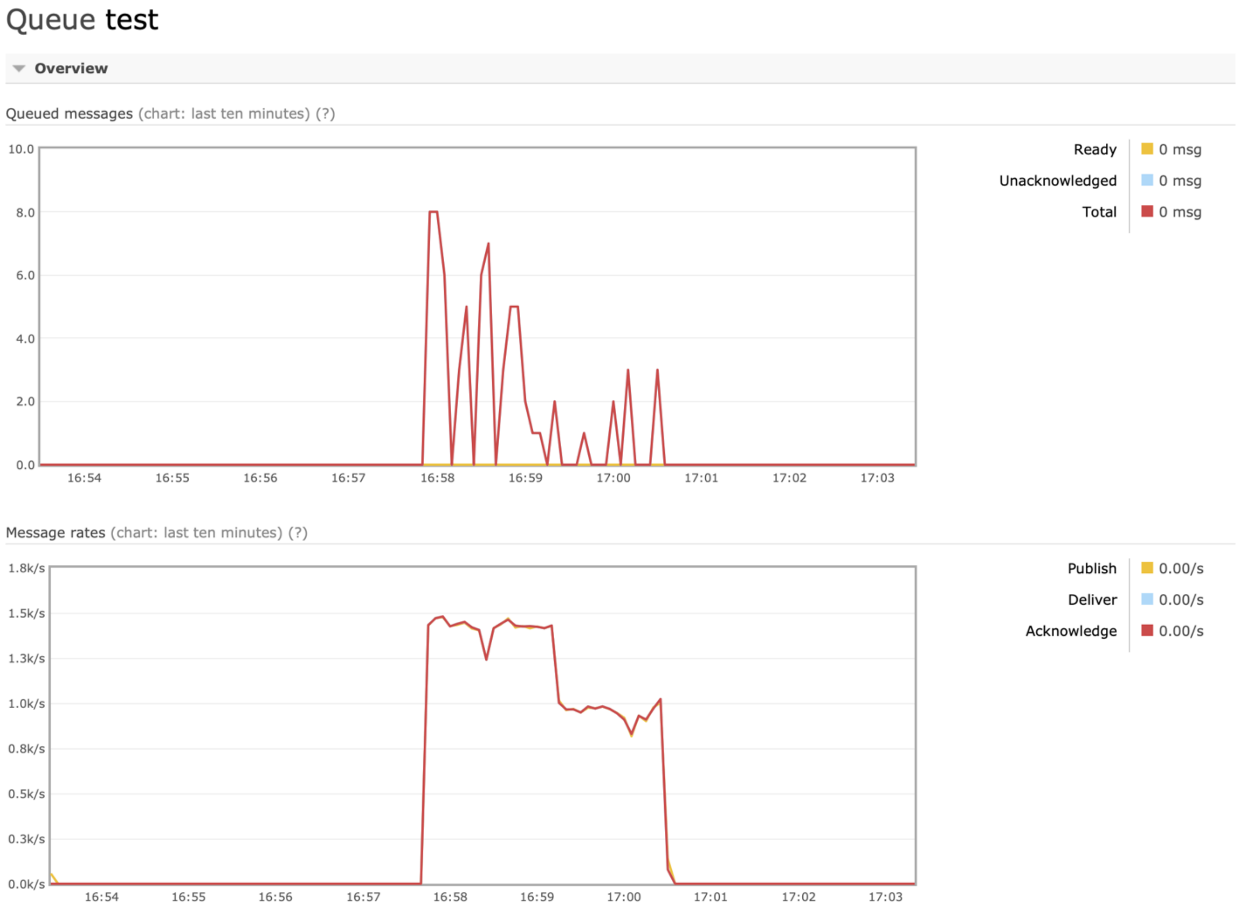
View, explore, and troubleshoot your load balancer's architecture.

Summary

Single instance					Load balanced 2 instances				
No. Client threads	No. Consumer threads	Queue length	Throughputs	Wall time	No. Client threads	No. Consumer threads each	Queue length	Throughputs	Wall time
50	50	2~5	1098 req/s	182 s	50	50	1~8	1197 req/s	167 s
50	100	1~11	1092 req/s	183 s					
100	100	1~4	1020 req/s	196 s	100	100	1~12	1129 req/s	177 s
100	200	1~4	1156 req/s	173 s					
200	200	1~3	1351 req/s	148 s	200	200	2~12	1242 req/s	162 s
200	300	2~5	1360 req/s	148 s					

Performance of load balanced instances is highly improved comparing to test case without load balancer. The highlighted part shows that load balanced test case is more comparable to the performance of single instance test with twice as much consumer threads.
We can expect load balanced instance to have greater performance in terms of throughputs, because single instance test is taken at around 1AM which should be in better traffic condition.
Regarding to reaching the minimum size of queue length, load balanced test does not show significant advantage.
In the single instance test, I would say 200 client threads against 200 consumer threads has the best performance and 50 client threads against 50 consumer threads might more likely to be optimal as it shows competent throughputs as the top player and a minimized queue length in addition to it and thus resulting in 0 request loss during the message pulling process.

50 client threads && 50 consumer threads each instance with ALB



Queue test

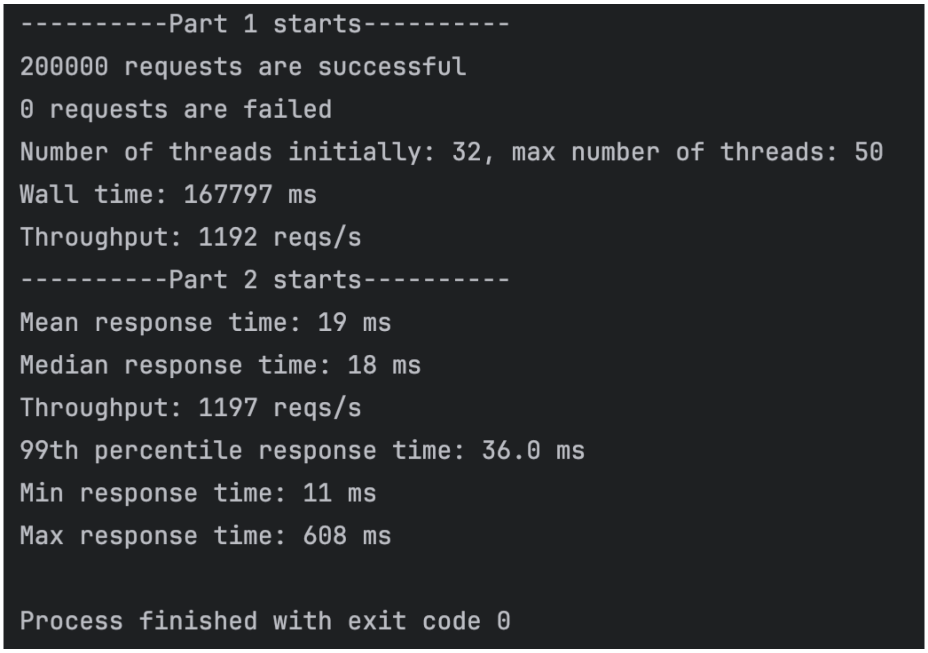
Overview

Queued messages (chart: last ten minutes) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

Message rates (chart: last ten minutes) (?)

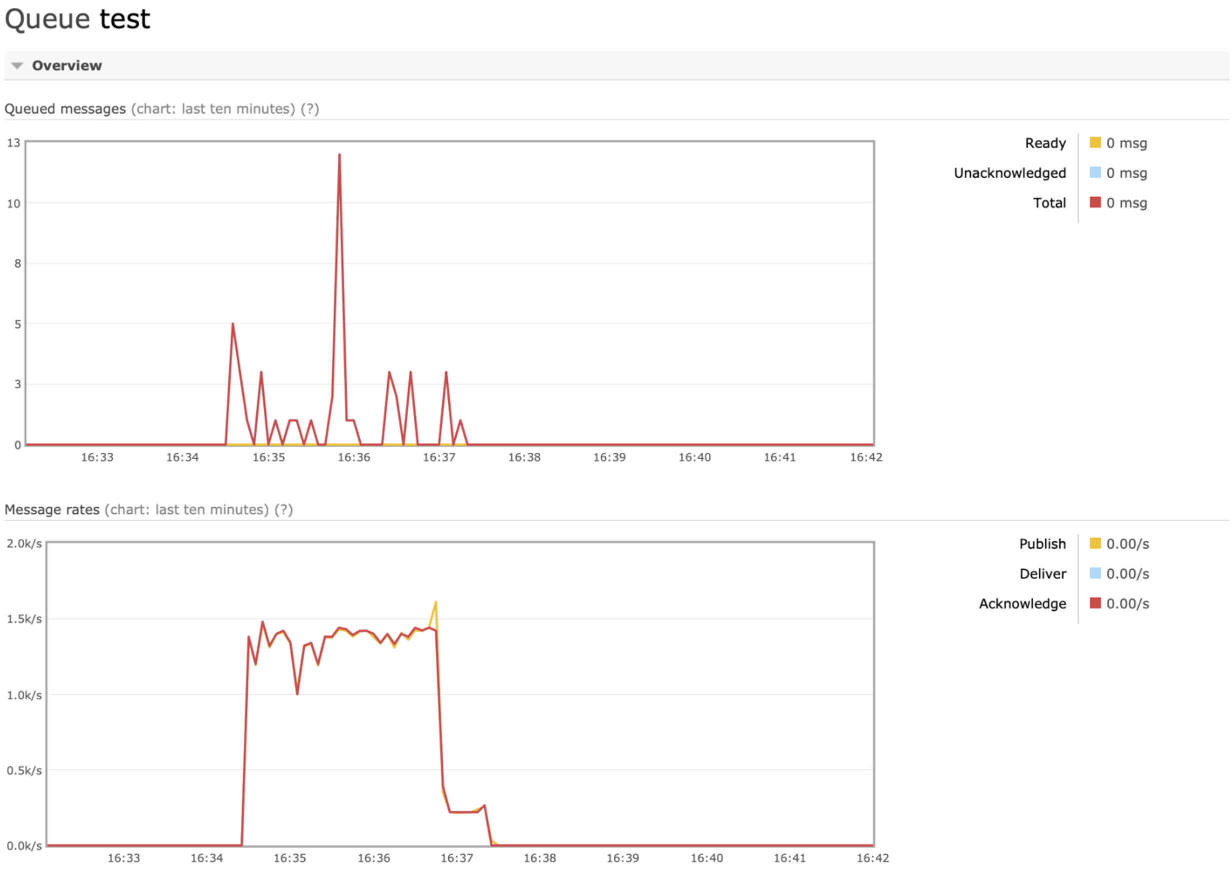
Publish	🟨 0.00/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s



```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 50
Wall time: 167797 ms
Throughput: 1192 reqs/s
----------Part 2 starts----------
Mean response time: 19 ms
Median response time: 18 ms
Throughput: 1197 reqs/s
99th percentile response time: 36.0 ms
Min response time: 11 ms
Max response time: 608 ms

Process finished with exit code 0
```

100 client threads && 100 consumer threads each instance with ALB



Queue test

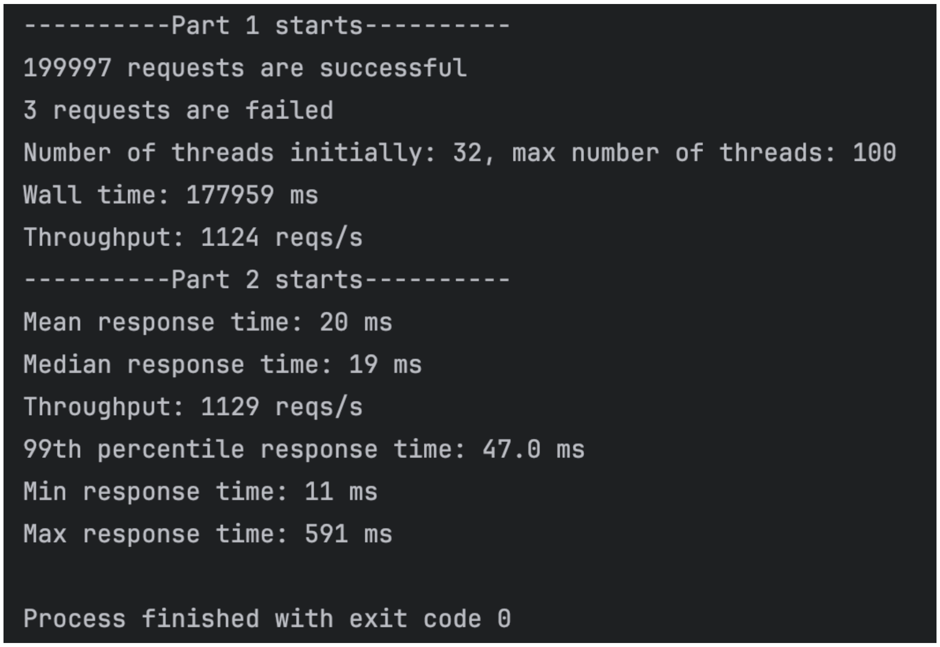
Overview

Queued messages (chart: last ten minutes) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

Message rates (chart: last ten minutes) (?)

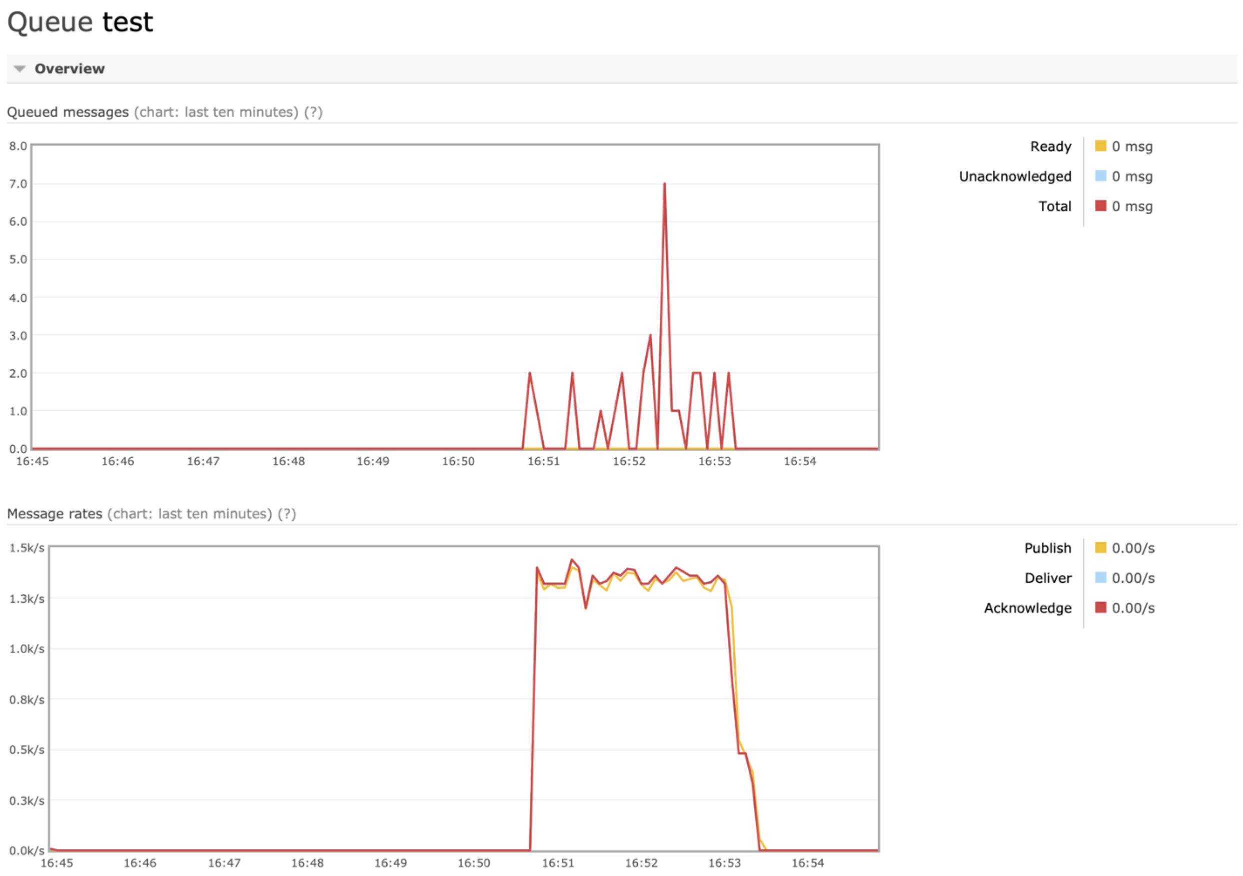
Publish	🟨 0.00/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s



```
----------Part 1 starts----------
199997 requests are successful
3 requests are failed
Number of threads initially: 32, max number of threads: 100
Wall time: 177959 ms
Throughput: 1124 reqs/s
----------Part 2 starts----------
Mean response time: 20 ms
Median response time: 19 ms
Throughput: 1129 reqs/s
99th percentile response time: 47.0 ms
Min response time: 11 ms
Max response time: 591 ms

Process finished with exit code 0
```

200 client threads && 200 consumer threads each instance with ALB



Queue test

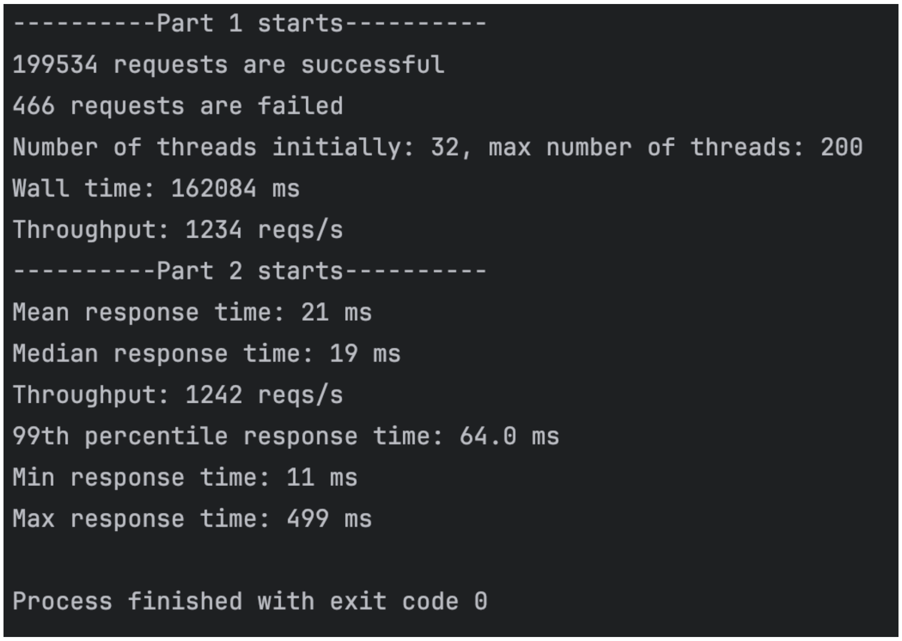
Overview

Queued messages (chart: last ten minutes) (?)

Ready 🟨 0 msg
Unacknowledged 🟦 0 msg
Total 🟥 0 msg

Message rates (chart: last ten minutes) (?)

Publish 🟨 0.00/s
Deliver 🟦 0.00/s
Acknowledge 🟥 0.00/s

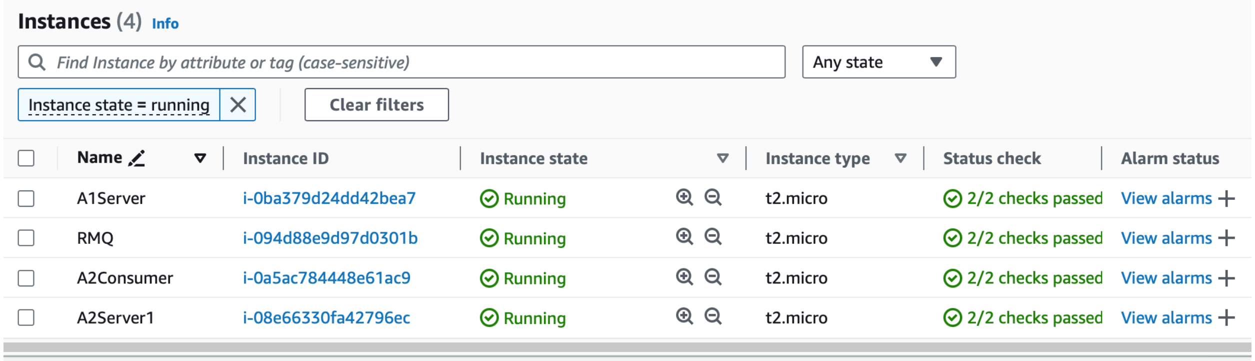


```
----------Part 1 starts----------
199534 requests are successful
466 requests are failed
Number of threads initially: 32, max number of threads: 200
Wall time: 162084 ms
Throughput: 1234 reqs/s
----------Part 2 starts----------
Mean response time: 21 ms
Median response time: 19 ms
Throughput: 1242 reqs/s
99th percentile response time: 64.0 ms
Min response time: 11 ms
Max response time: 499 ms

Process finished with exit code 0
```

Does NLB improve the performance?

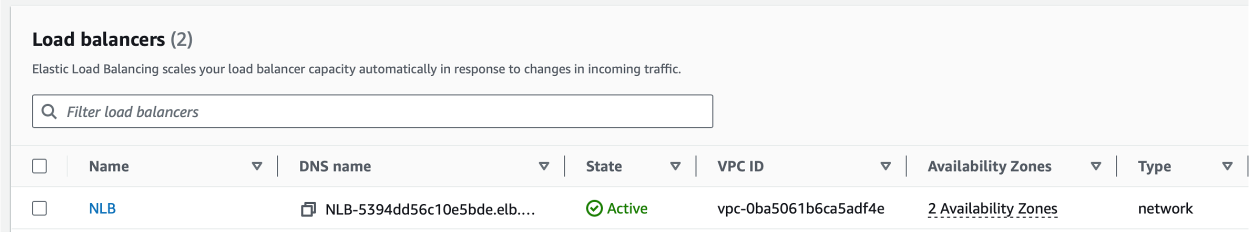
Instances



Instances (4) Info

□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status
□	A1Server	i-0ba379d24dd42bea7	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +
□	RMQ	i-094d88e9d97d0301b	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +
□	A2Consumer	i-0a5ac784448e61ac9	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +
□	A2Server1	i-08e66330fa42796ec	✓ Running	t2.micro	✓ 2/2 checks passed	View alarms +

NLB

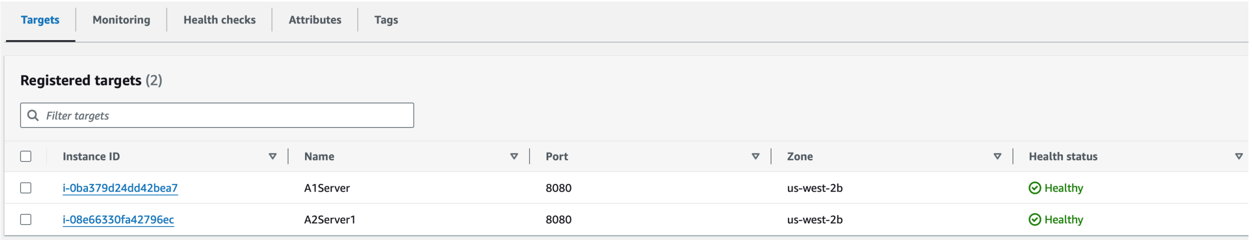


Load balancers (2)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

□	Name	DNS name	State	VPC ID	Availability Zones	Type
□	NLB	NLB-5394dd56c10e5bde.elb....	✓ Active	vpc-0ba5061b6ca5adf4e	2 Availability Zones	network

Target Group of 2 instances



Registered targets (2)

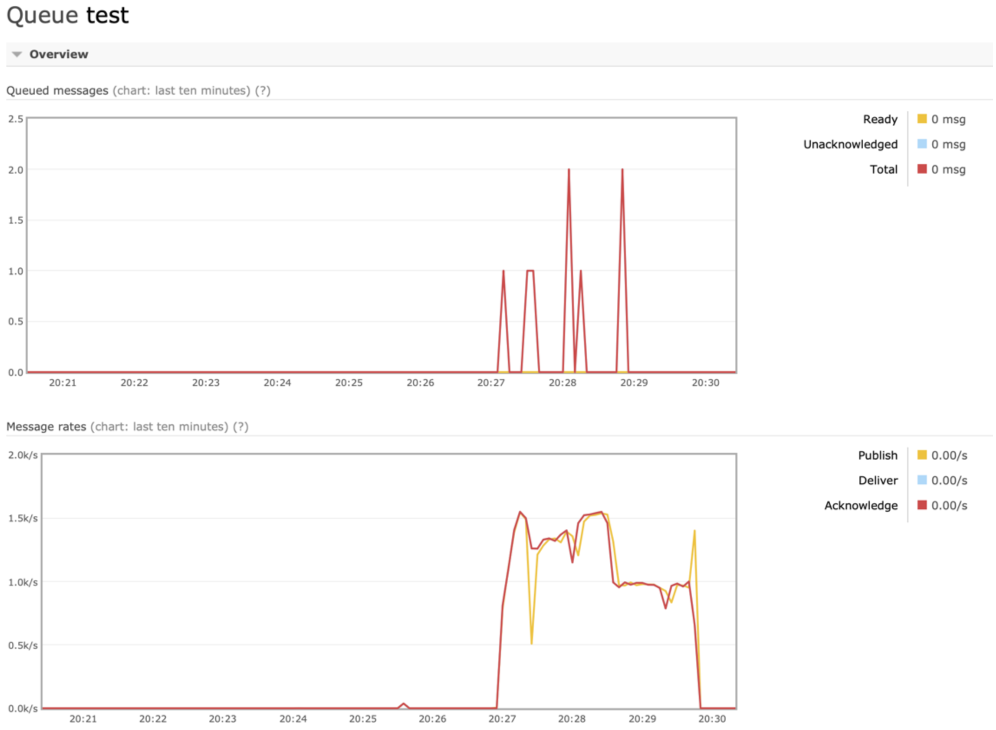
□	Instance ID	Name	Port	Zone	Health status
□	i-0ba379d24dd42bea7	A1Server	8080	us-west-2b	✓ Healthy
□	i-08e66330fa42796ec	A2Server1	8080	us-west-2b	✓ Healthy

Summary

Single instance					Load balanced 2 instances ALB				
No. Client threads	No. Consumer threads	Queue length	Throughputs	Wall time	No. Client threads	No. Consumer threads each	Queue length	Throughputs	Wall time
50	50	2~5	1098 req/s	182 s	50	50	1~8	1197 req/s	167 s
50	100	1~11	1092 req/s	183 s					
100	100	1~4	1020 req/s	196 s	100	100	1~12	1129 req/s	177 s
100	200	1~4	1156 req/s	173 s					
200	200	1~3	1351 req/s	148 s	200	200	2~12	1242 req/s	162 s
200	300	2~5	1360 req/s	148 s					
					Load balanced 2 instances NLB				
					50	50	1~2	1183 req/s	170 s
					100	100	1~5	1123 req/s	178 s
					200	200	1~5	1190 req/s	168 s

As is highlighted, once again 200 client threads against 200 consumer threads has the best throughput and minimized wall time. But I would say in the load balanced test with NLB, 50 client threads against 50 consumer threads has comparable throughput and has the minimized queue length which resulting in 0 request loss during the data transmission.
In my test cases, NLB beats ALB in terms of minimizing queue length.

50 client threads && 50 consumer threads each instance with NLB



Queue test

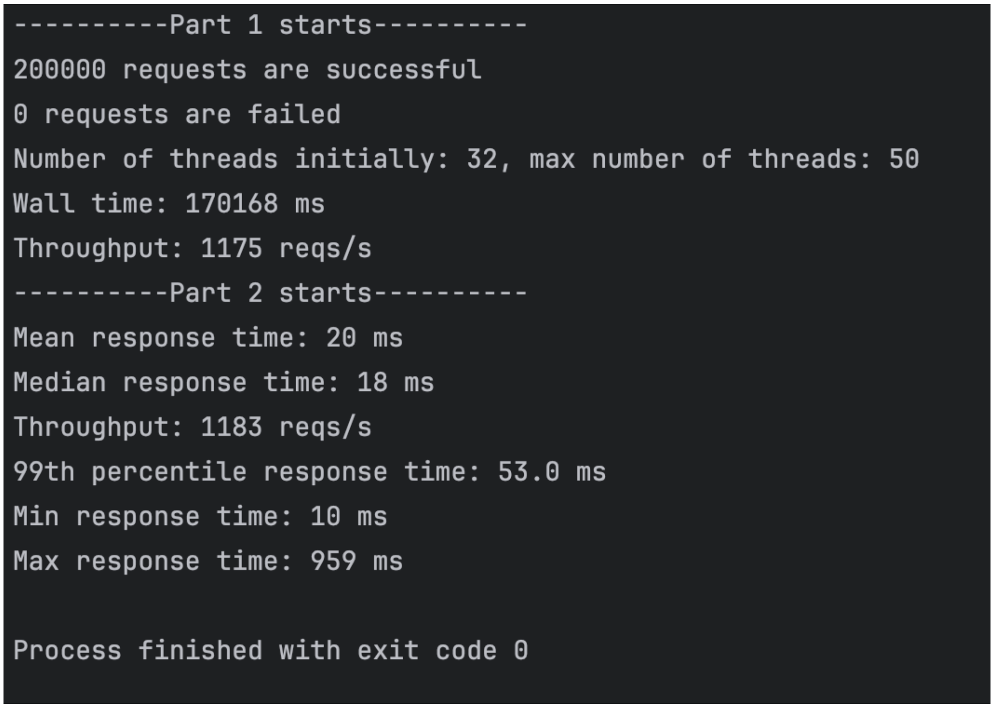
Overview

Queued messages (chart: last ten minutes) (?)

Ready 🟨 0 msg
Unacknowledged 🟦 0 msg
Total 🟥 0 msg

Message rates (chart: last ten minutes) (?)

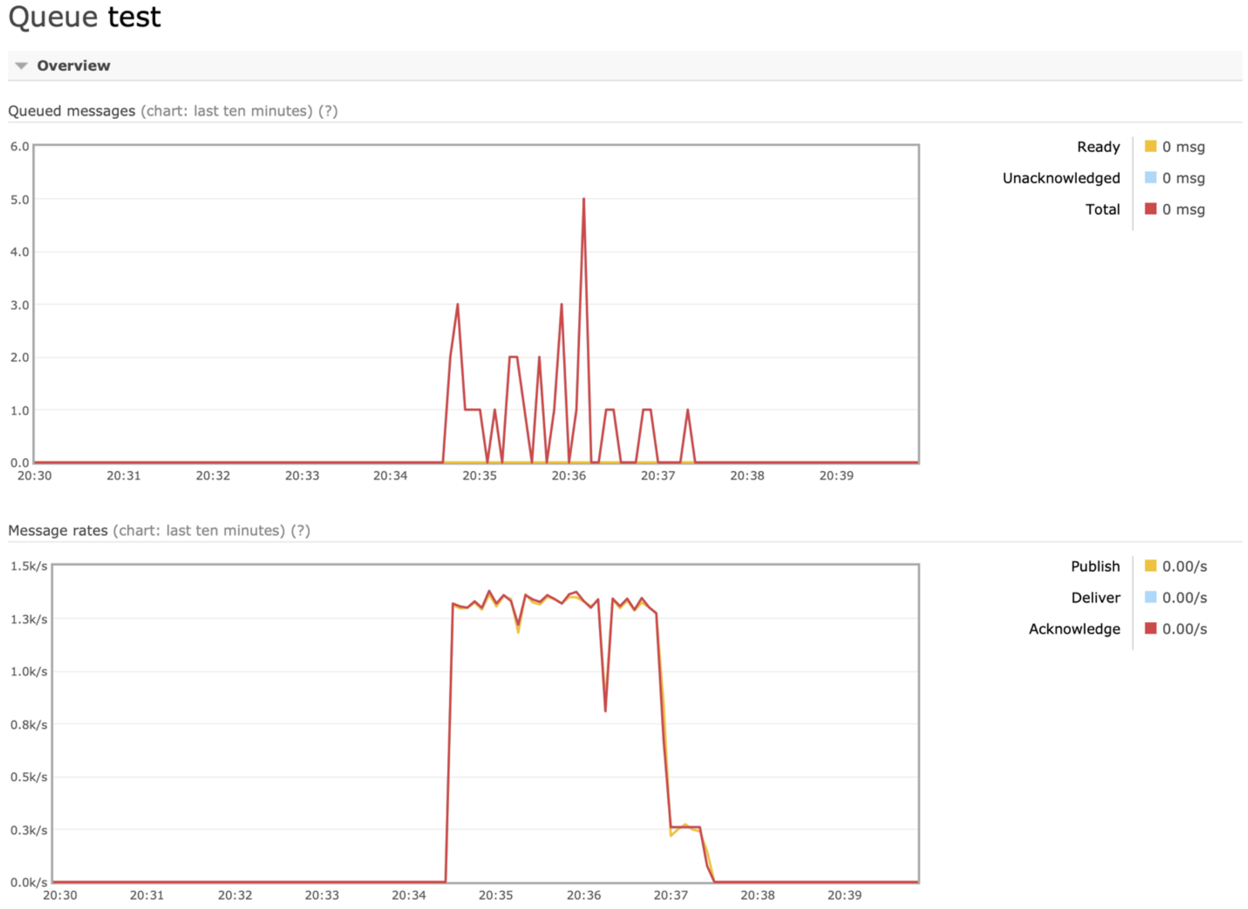
Publish 🟨 0.00/s
Deliver 🟦 0.00/s
Acknowledge 🟥 0.00/s



```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 50
Wall time: 170168 ms
Throughput: 1175 reqs/s
----------Part 2 starts----------
Mean response time: 20 ms
Median response time: 18 ms
Throughput: 1183 reqs/s
99th percentile response time: 53.0 ms
Min response time: 10 ms
Max response time: 959 ms

Process finished with exit code 0
```

100 client threads && 100 consumer threads each instance with NLB



Queue **test**

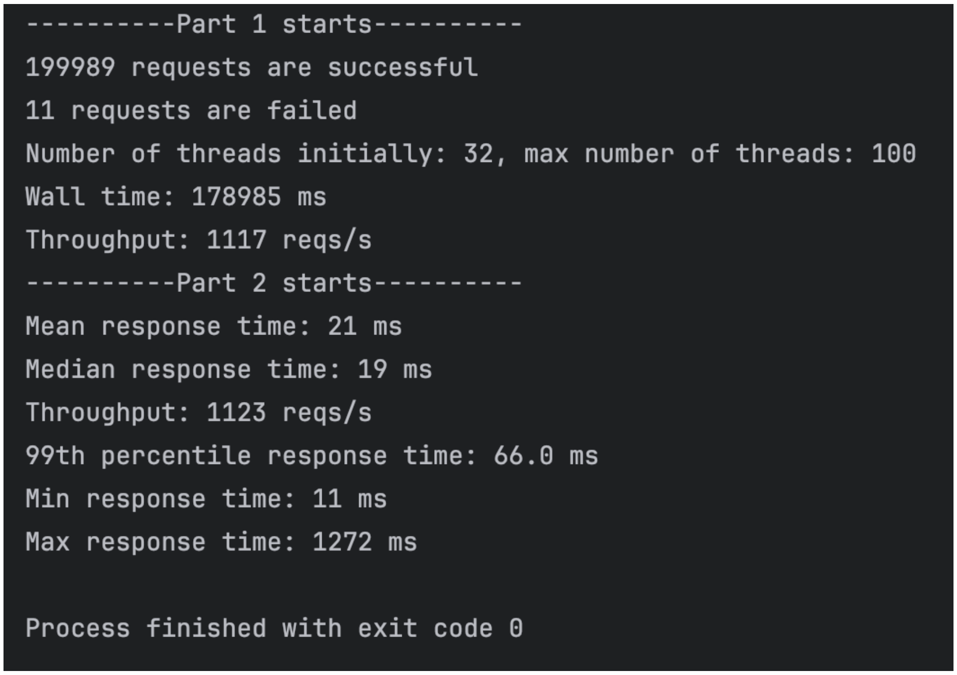
▼ **Overview**

Queued messages (chart: last ten minutes) (?)

Ready 🟨 0 msg
Unacknowledged 🟦 0 msg
Total 🟥 0 msg

Message rates (chart: last ten minutes) (?)

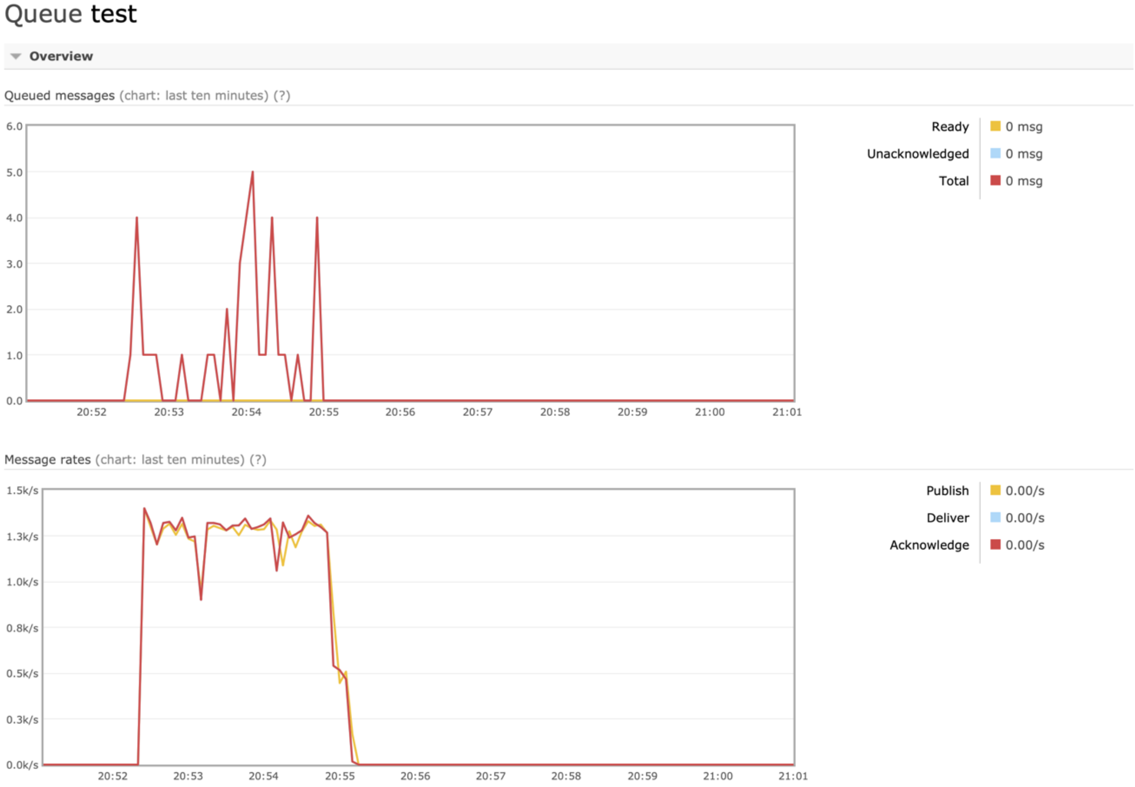
Publish 🟨 0.00/s
Deliver 🟦 0.00/s
Acknowledge 🟥 0.00/s



```
----------Part 1 starts----------
199989 requests are successful
11 requests are failed
Number of threads initially: 32, max number of threads: 100
Wall time: 178985 ms
Throughput: 1117 reqs/s
----------Part 2 starts----------
Mean response time: 21 ms
Median response time: 19 ms
Throughput: 1123 reqs/s
99th percentile response time: 66.0 ms
Min response time: 11 ms
Max response time: 1272 ms

Process finished with exit code 0
```

200 client threads && 200 consumer threads each instance with NLB



Queue test

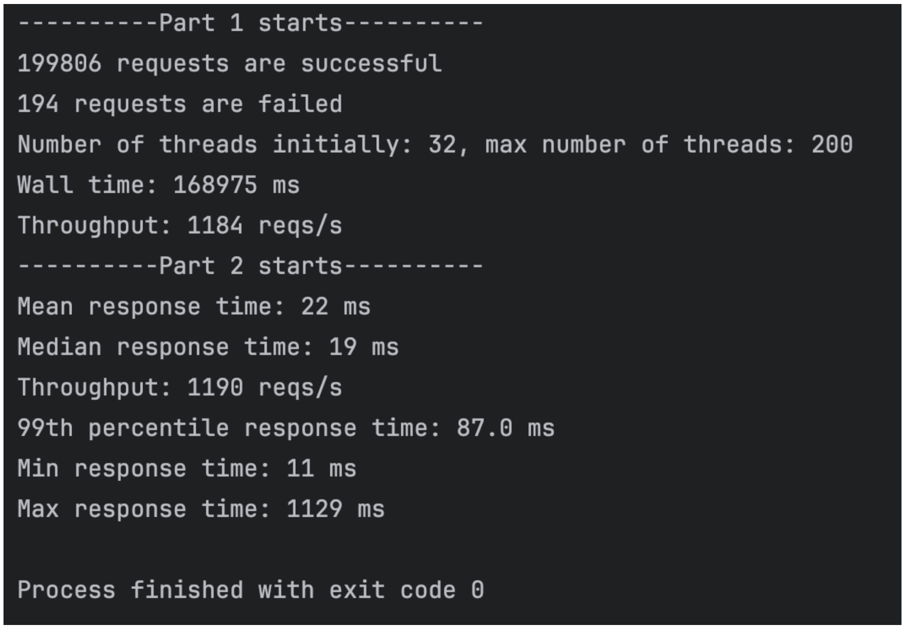
Overview

Queued messages (chart: last ten minutes) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

Message rates (chart: last ten minutes) (?)

Publish	🟨 0.00/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s



```
----------Part 1 starts----------
199806 requests are successful
194 requests are failed
Number of threads initially: 32, max number of threads: 200
Wall time: 168975 ms
Throughput: 1184 reqs/s
----------Part 2 starts----------
Mean response time: 22 ms
Median response time: 19 ms
Throughput: 1190 reqs/s
99th percentile response time: 87.0 ms
Min response time: 11 ms
Max response time: 1129 ms

Process finished with exit code 0
```