

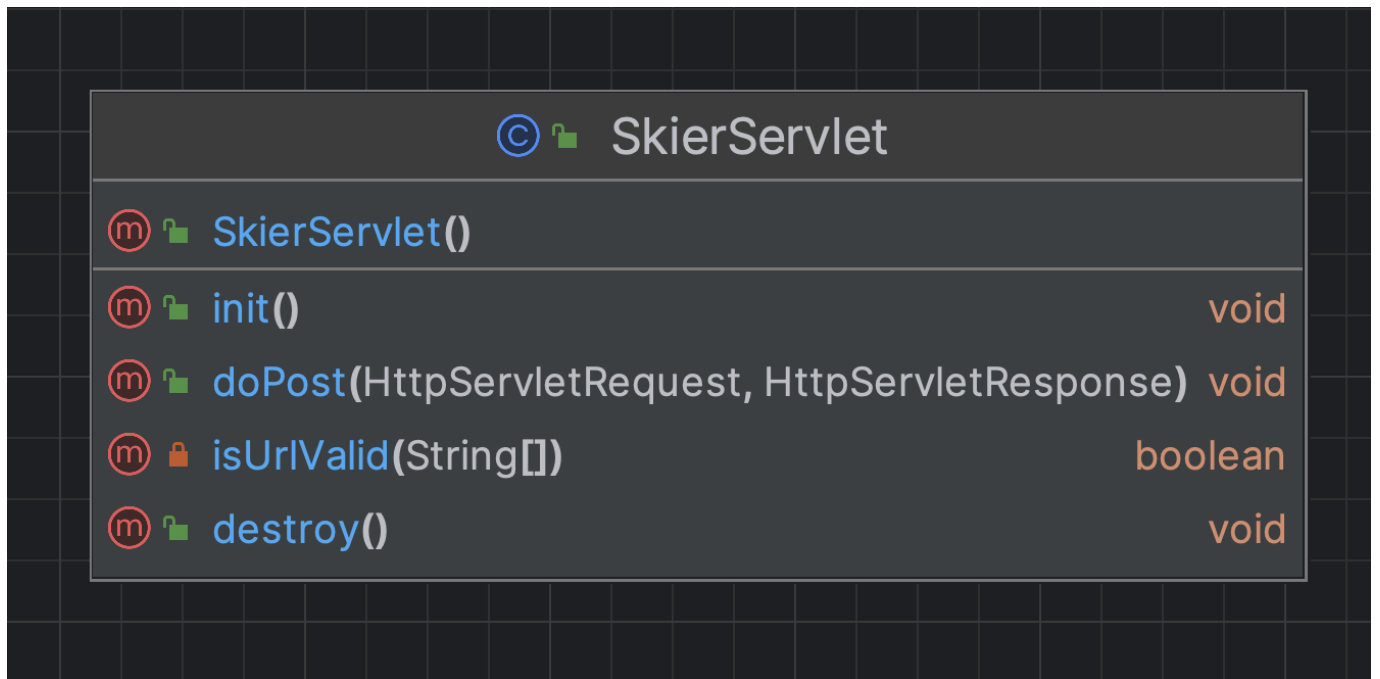
CS6650 Assignment2 Report

Student: Xunyan Zhang

Git Repo: <https://github.com/zhxunynn/CS6650Assignments/tree/main/Assignment2>

Design

Server



The server structure remains basically identical to the previous assignment, but I introduced configuration based method to this assignment. Server now can read configuration file and load it to start, which benefits a lot when we frequently restarting instances. An example configuration file is

```
erServlet.java  Server/.../rabbitmq.conf ×
gins supporting *.conf files found.
172.31.17.117
5672
server
cs6650hw2server
2💡
homework2|
```

After loading configuration file from disk, the server will use `ConnectionFactory` to establish a connection to RabbitMQ, and then ready to serve!

The server will firstly check if there is any wrong with the request, which is identical to last assignment. Then we parse it as `LiftRide` object, so that we can get the useful content like skierID, time, liftID, etc from the object and squeeze them into the message object.

Then we take a channel from the channel pool, and publish the message to the queue we created in RabbitMQ, return the 201 to clients.

```

LiftRide liftRide = gson.fromJson(body, LiftRide.class);
int skierID = Integer.parseInt(parts[7]);
JsonObject msg = new JsonObject();
msg.add("skierID", new JsonPrimitive(skierID));
msg.add("time", new JsonPrimitive(liftRide.getTime()));
msg.add("liftID", new JsonPrimitive(liftRide.getLiftID()));
Channel channel = null;
try {
    channel = channelPool.take();
    channel.basicPublish("", rabbitMQName, null,
msg.toString().getBytes());
    response.setStatus(HttpServletResponse.SC_CREATED);
    response.getWriter().write("{\"Message\": \"The URL is valid,
message sent to Rabbit MQ successfully!\"}");
} catch (InterruptedException e) {
    response.setStatus(HttpServletResponse.SC_EXPECTATION_FAILED);
    throw new ServletException("Failed to initialize RabbitMQ
connection", e);
} finally {
    if (channel != null)
        channelPool.add(channel);
}

```

Consumer

© ConsumerThread

Ⓜ ConsumerThread(String, Connection)

Ⓜ run() void

© Consumer

Ⓜ ◦ Consumer(String)

ⓕ numThreads Integer

ⓕ connection Connection

ⓕ rabbitMQName String

Ⓜ main(String[]) void

Ⓟ numThreads Integer

Ⓟ connection Connection

Ⓟ rabbitMQName String

After reading the reference docs in Addendum of Assignment2 Guidance, I decided to use `BlockingQueue` as the channel pool. I created the channel pool that shares 20~100 (based on

configuration) pre-created channels. Then used an executorPool to start multiple threads, pulling messages from RabbitMQ and acknowledge them.

Client

Since there is no requirement on changing the Client source code, I directly copied the one from last assignment, with the following parameters: `-nt 32 -nr 200000 --host cs6650loadbalancer-2138608119.us-west-2.elb.amazonaws.com:8080/Server`.

Exploration

Queued messages last ten minutes ?



- How many client threads are optimal after phase 1 to maximize throughput?
- How many queue consumers threads do I need to keep the queue size as close to zero as possible? Less than a 1000 max is a great target, but the main aim is to ensure the queue length doesn't continually grow, then shrink, giving a 'pointy' queue length profile, ie \wedge . An increase to a plateau is fine, ie $\neg \wedge$. If the plateau is less than a 1000, you are in great shape!

I understand these questions are not necessary to be answered, but I'm curious about them. So I made several different parameters, the left one is recorded when the consumer threads=10, the middle one is 20, the tall one is 50, and the one after the first top is 30, the toppest one is 24, and the right one is 20 again. So we can conclude from this that, the num of consumer threads doesn't have the linear relationship with queued messages.

Single Instance Result Analysis

First of all, the AWS instances when executing the single instance analysis are as below:

Instances (3) Info										
<input type="text" value="Find Instance by attribute or tag (case-sensitive)"/>				Any state ▾		< 1 > ⚙				
<input type="checkbox"/>	Name ↗	Instance ID	Instance state ▾	Instance type ▾	Status check	Alarm status	Availability Zone ▾	Public IPv4 DNS ▾	Public IPv4 ... ▾	Elastic IP
<input type="checkbox"/>	CS6650Server	i-0962dcd21baae1db	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2b	ec2-34-222-1-101.us-w...	34.222.1.101	-
<input type="checkbox"/>	Consumer	i-02cafc6e89febb052	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2b	ec2-35-89-19-146.us-w...	35.89.19.146	-
<input type="checkbox"/>	RabbitMQ	i-07aebdd5c0170531f	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2b	ec2-34-221-177-135.us...	34.221.177.135	-

The parameters is:

Name:
☐ Store as project file ⚙

Run on:

Local machine ▾
Manage targets...

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run Modify options ▾ ⌘M

java corretto-17 ▾
-cp Homework2Client1 ▾

ClientMain

⋮

-nt 32 -nr 200000 --host ec2-34-222-1-101.us-west-2.compute.amaz

⋮ ↗

And the result is:



```
/Users/xunyan/.sdkman/candidates/java/17.0.9-amzn/bin/java ...
```

```
Mar 14, 2024 6:17:14 AM ClientMain main
INFO: Both client and server are ready!
Mar 14, 2024 6:17:14 AM ClientMain main
INFO: Ready to run phases!
Mar 14, 2024 6:17:14 AM ClientMain doPhase
INFO: Startup is ready to start!
Mar 14, 2024 6:17:14 AM ClientMain doPhase
INFO: Startup phase is going to execute 32 threads with 1000 requests each.
Mar 14, 2024 6:17:38 AM ClientMain doPhase
INFO: Startup has already terminated 1 thread(s).
Mar 14, 2024 6:17:38 AM ClientMain doPhase
INFO: Catchup is ready to start!
Mar 14, 2024 6:17:38 AM ClientMain doPhase
INFO: Catchup phase is going to execute 96 threads with 1750 requests each.
Mar 14, 2024 6:18:22 AM ClientMain doPhase
INFO: Catchup has already terminated 96 thread(s).
```

```
=====Results=====
```

```
Successful Requests: 200000
```

```
Failed Requests: 0
```

```
Total run time: 68484 (ms)
```

```
Total Throughput in requests per second: 2920.390164125927
```

```
=====
```



RabbitMQ 3.9.13

Erlang 24.2.1

Refreshed 202

Cluster **rabbit@ip**

Overview

Connections

Channels

Exchanges

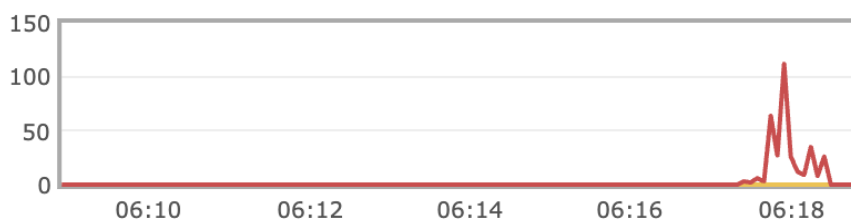
Queues

Admin

Overview

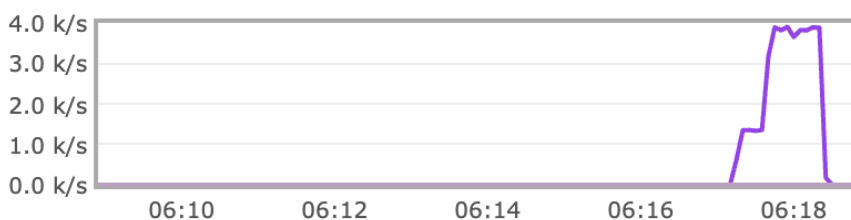
▼ Totals

Queued messages last ten minutes ?



Ready	0
Unacked	0
Total	0

Message rates last ten minutes ?



Publish	0.00/s
Publisher confirm	0.00/s
Deliver (manual ack)	0.00/s

From the above chart, we can notice that the maximum queued message is around 115, which is super good compared to the recommended good value - 1000 in assignment guidance.

The throughput is 2920, which is sort of similar to the assignment 1 (≤ 3000).

Load Balanced Results Analysis

I followed the [tutorial](#) to set up an auto scaling group with 4 instances load balanced, the steps are attached as below:

Instances (8) Info

Find Instance by attribute or tag (case-sensitive)

Running

Refresh

Connect

Instance state

Actions

Launch instances

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
<input type="checkbox"/>	CS6650ASG	i-09d33fd73edcc74f8	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2c	ec2-35-90-138-37.us-w...	35.90.138.37	-
<input type="checkbox"/>	CS6650ASG	i-07a9159412a7ea943	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2c	ec2-34-210-165-178.us...	34.210.165.178	-
<input type="checkbox"/>	CS6650ASG	i-0a10d9d336212c7ff	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2a	ec2-34-210-209-156.us...	34.210.209.156	-
<input type="checkbox"/>	CS6650ASG	i-00640efa43b3549d0	Running	t2.micro	Initializing	View alarms +	us-west-2a	ec2-18-236-121-174.us...	18.236.121.174	-
<input type="checkbox"/>	CS6650ASG	i-0b99d577fb958d8ff	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2b	ec2-54-213-20-222.us-...	54.213.20.222	-
<input type="checkbox"/>	Consumer	i-02cafc6e89febb052	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2b	ec2-35-89-205-231.us-...	35.89.205.231	-
<input type="checkbox"/>	CS6650Server-1	i-081e95a9ea6da9e65	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2b	ec2-35-91-165-236.us-...	35.91.165.236	-
<input type="checkbox"/>	RabbitMQ	i-07aebdd5c0170531f	Running	t2.micro	2/2 checks passed	View alarms +	us-west-2b	ec2-18-246-236-253.us...	18.246.236.253	-

Load balancers (1/1)

Filter load balancers

< 1 >

Refresh

Actions

Create load balancer

<input checked="" type="checkbox"/>	Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
<input checked="" type="checkbox"/>	CS6650LoadBalancer	CS6650LoadBalancer-213...	Active	vpc-08f5c30a5899037...	3 Availability Zones	application	March 14, 2024, 04:43 (UTC-07:00)

Load balancer: CS6650LoadBalancer

Details

Load balancer type Application	Status Active	VPC vpc-08f5c30a589903741	IP address type IPv4
Scheme Internet-facing	Hosted zone Z1H1FL5HABSF5	Availability Zones subnet-0d6a16e075df0adfc us-west-2b (usw2-az1) subnet-09559160515d6adcb us-west-2c (usw2-az3) subnet-047a16c218cc02cb9 us-west-2a (usw2-az2)	Date created March 14, 2024, 04:43 (UTC-07:00)
Load balancer ARN arn:aws:elasticloadbalancing:us-west-2:808111394274:loadbalancer/app/CS6650LoadBalancer/37d63386c1c8b3b4		DNS name CS6650LoadBalancer-2138608119.us-west-2.elb.amazonaws.com (A Record)	

Listeners and rules (2) Info

Filter listeners

< 1 >

Refresh

Manage rules

Manage listener

Add listener

<input type="checkbox"/>	Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate	mTLS
<input type="checkbox"/>	HTTP:80	Forward to target group • CS6650LoadBalancer 1 (100%) • Group-level stickiness: Off	1 rule	ARN	Not applicable	Not applicable	Not applicable
<input type="checkbox"/>	HTTP:8080	Forward to target group • CS6650LoadBalancer 1 (100%) • Group-level stickiness: Off	1 rule	ARN	Not applicable	Not applicable	Not applicable

Details

arn:aws:elasticloadbalancing:us-west-2:808111394274:targetgroup/CS6650LoadBalancer/5f82b94ba1f4c82a

Target type

Instance

Protocol : Port

HTTP: 80

Protocol version

HTTP1

VPC

[vpc-08f5c30a589903741](#)

IP address type

IPv4

Load balancer

[CS6650LoadBalancer](#)

4

Total targets

0

Healthy

0

Unhealthy

0

Unused

4

Initial

0

Draining

0

Anomalous

► Distribution of targets by Availability Zone (AZ)

Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets

Monitoring

Health checks

Attributes

Tags

Registered targets (4)

Info

Anomaly mitigation: Not applicable

Deregister

Register targets

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

Filter targets

	Instance ID	Name	Port	Zone	Health status	Health status details	Launch time	Anomaly detectio...
<input type="checkbox"/>	i-0ed05aa31b51092d6	CS6650Server...	8080	us-west-2b	Initial	Target registration is i...	March 14, 2024, 06:15 (UTC-07:00)	Normal
<input type="checkbox"/>	i-0d58c45fe1b222901	CS6650Server...	8080	us-west-2b	Initial	Target registration is i...	March 14, 2024, 06:11 (UTC-07:00)	Normal
<input type="checkbox"/>	i-042f6df490f4a7152	CS6650Server...	8080	us-west-2b	Initial	Target registration is i...	March 14, 2024, 06:10 (UTC-07:00)	Normal
<input type="checkbox"/>	i-081e95a9ea6da9e65	CS6650Server-...	8080	us-west-2b	Initial	Target registration is i...	March 14, 2024, 05:14 (UTC-07:00)	Normal

Then I used the following parameters to start the test on load balanced server.

Application

Consumer

Client1Local

Client2Local

Client1AWS

Client2AWS

Client1AWSLatency

Maven

Tomcat Server

Run/Debug Configurations

Name:

Client1AWS

Store as project file

Run on:

Local machine

Manage targets...

Run configurations may be executed locally or on a target; for example in a Docker Container or on a remote host using SSH.

Build and run

Modify options

java corretto-17

-cp Homework2Client1

ClientMain

-nt 32 -nr 200000 --host cs6650loadbalancer-2138608119.us-west-2

CLI arguments to your application.

Working directory:

/Users/xunyan/Documents/Northeastern/CS6650/Homev

Environment variables:

Environment variables or .env files

Separate variables with semicolon: VAR=value; VAR1=value1

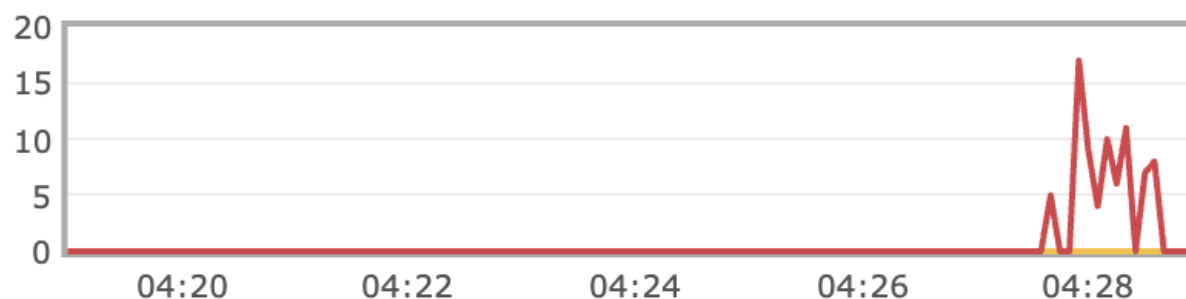
Open run/debug tool window when started

Overview
Connections
Channels
Exchanges

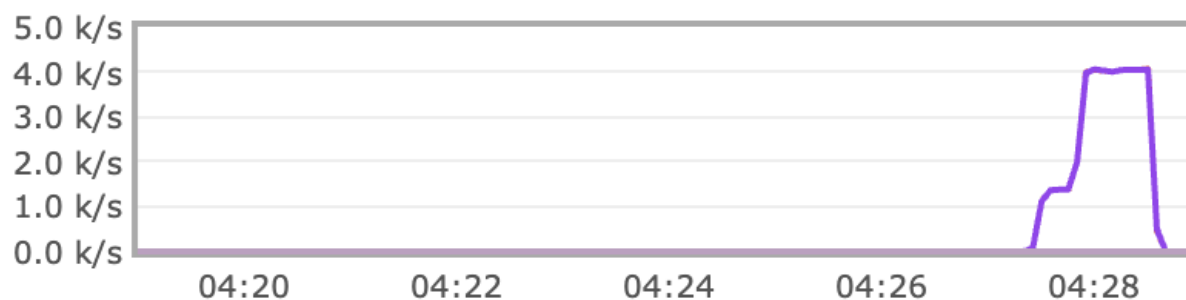
Overview

▼ **Totals**

Queued messages last ten minutes ?



Message rates last ten minutes ?



```
Client1AWS x
/Users/xunyan/.sdkman/candidates/java/17.0.9-amzn/bin/java ...
Mar 14, 2024 4:27:24 AM ClientMain main
INFO: Both client and server are ready!
Mar 14, 2024 4:27:24 AM ClientMain main
INFO: Ready to run phases!
Mar 14, 2024 4:27:24 AM ClientMain doPhase
INFO: Startup is ready to start!
Mar 14, 2024 4:27:24 AM ClientMain doPhase
INFO: Startup phase is going to execute 32 threads with 1000 requests each.
Mar 14, 2024 4:27:49 AM ClientMain doPhase
INFO: Startup has already terminated 1 thread(s).
Mar 14, 2024 4:27:49 AM ClientMain doPhase
INFO: Catchup is ready to start!
Mar 14, 2024 4:27:49 AM ClientMain doPhase
INFO: Catchup phase is going to execute 96 threads with 1750 requests each.
Mar 14, 2024 4:28:31 AM ClientMain doPhase
INFO: Catchup has already terminated 96 thread(s).

=====Results=====
Successful Requests: 200000
Failed Requests: 0
Total run time: 66515 (ms)
Total Throughput in requests per second: 3006.840562279185

=====

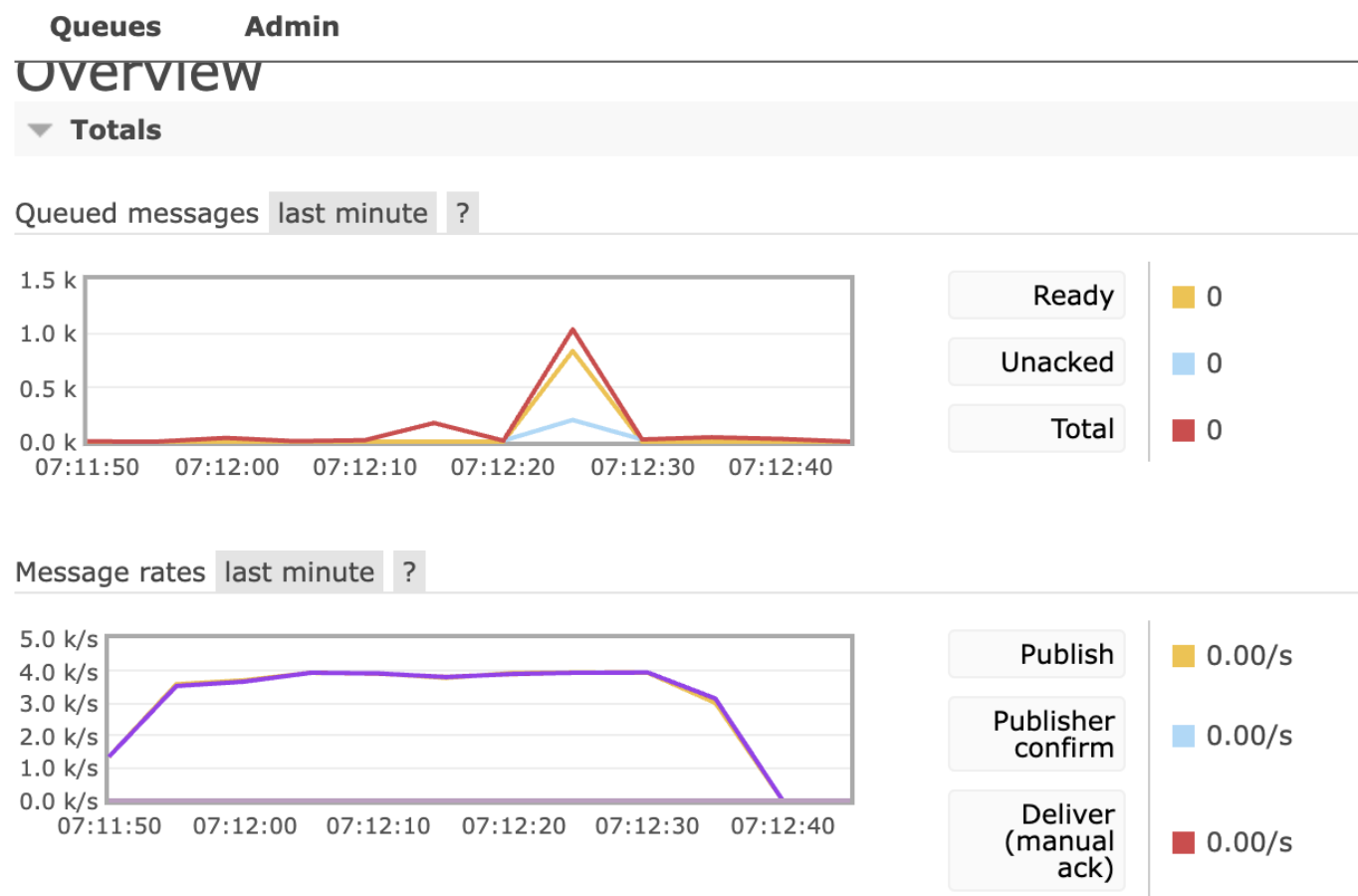
Process finished with exit code 0
```

As we can see, the result looks better than previous. The maximum queued message was only ~16, and the throughput is 3006, better than the last one without load balancing.

Bonus

After deep diving into the client & server, I found out the server is busy with both receiving messages from clients, and sending messages to message queue, while the most important thing is to respond to users as quick as possible. Based on this, I increased the client's threads,

and uses same amount of threads in server to send messages to RabbitMQ, and then get the following amazing results, although the queued messages increased but it's still under limit.



```
/Users/xunyan/.sdkman/candidates/java/17.0.9-amzn/bin/java ...
```

```
Mar 14, 2024 7:11:30 AM ClientMain main
```

```
INFO: Both client and server are ready!
```

```
Mar 14, 2024 7:11:30 AM ClientMain main
```

```
INFO: Ready to run phases!
```

```
Mar 14, 2024 7:11:30 AM ClientMain doPhase
```

```
INFO: Startup is ready to start!
```

```
Mar 14, 2024 7:11:30 AM ClientMain doPhase
```

```
INFO: Startup phase is going to execute 32 threads with 1000 requests each.
```

```
Mar 14, 2024 7:11:54 AM ClientMain doPhase
```

```
INFO: Startup has already terminated 1 thread(s).
```

```
Mar 14, 2024 7:11:54 AM ClientMain doPhase
```

```
INFO: Catchup is ready to start!
```

```
Mar 14, 2024 7:11:54 AM ClientMain doPhase
```

```
INFO: Catchup phase is going to execute 96 threads with 1750 requests each.
```

```
Mar 14, 2024 7:12:38 AM ClientMain doPhase
```

```
INFO: Catchup has already terminated 96 thread(s).
```

```
=====Results=====
```

```
Successful Requests: 200000
```

```
Failed Requests: 0
```

```
Total run time: 42980 (ms)
```

```
Total Throughput in requests per second: 4653.327128897162
```

```
=====
```

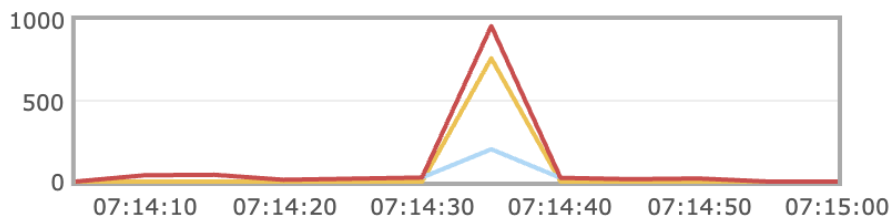
```
Process finished with exit code 0
```

```
|
```

Overview

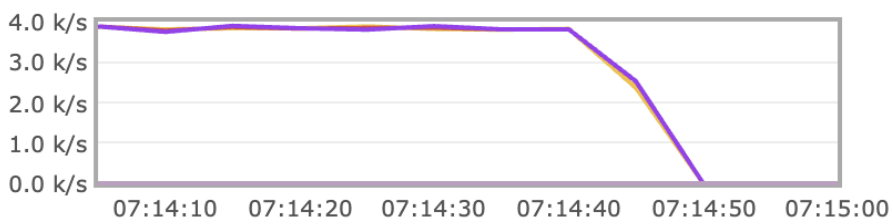
▼ Totals

Queued messages **last minute** ?



Ready	0
Unacked	0
Total	0

Message rates **last minute** ?



Publish	0.00/s
Publisher confirm	0.00/s
Deliver (manual ack)	0.00/s

```
/Users/xunyan/.sdkman/candidates/java/17.0.9-amzn/bin/java ...
```

Mar 14, 2024 7:13:39 AM ClientMain main

INFO: Both client and server are ready!

Mar 14, 2024 7:13:39 AM ClientMain main

INFO: Ready to run phases!

Mar 14, 2024 7:13:39 AM ClientMain doPhase

INFO: Startup is ready to start!

Mar 14, 2024 7:13:39 AM ClientMain doPhase

INFO: Startup phase is going to execute 32 threads with 1000 requests each.

Mar 14, 2024 7:14:03 AM ClientMain doPhase

INFO: Startup has already terminated 1 thread(s).

Mar 14, 2024 7:14:03 AM ClientMain doPhase

INFO: Catchup is ready to start!

Mar 14, 2024 7:14:03 AM ClientMain doPhase

INFO: Catchup phase is going to execute 96 threads with 1750 requests each.

Mar 14, 2024 7:14:47 AM ClientMain doPhase

INFO: Catchup has already terminated 96 thread(s).

=====Results=====

Successful Requests: 200000

Failed Requests: 0

Total run time: 40413 (ms)

Total Throughput in requests per second: 4948.902580852696

=====