Assignment 3 - CS 6650 | Fall 2022 | Northeastern University

Github URL: https://github.com/jasongautama/cs6650-Distributed-Systems/tree/main/assignment3

Overview

Assignment 3 is the continuation of the previous assignment. In this assignment, we added persistent storage that is sent by the client (refer here to assignment instruction). I decided to use Redis and the following document will discuss the design and result of the test.

Database Design

Redis is a memory database that stores the data in key-value pairs to achieve constant query time. My approach when trying to design how to store the data is: "How to store the data so the user can query the data easily?". Therefore, I worked backwards by answering a few of the questions that were raised in this assignment.

1. "For skier N, how many days have they skied this season?"
2. "For skier N, what are the vertical totals for each ski day?" (calculate vertical as liftID*10)
3. "For skier N, show me the lifts they rode on each ski day"
4. "How many unique skiers visited resort X on day N?"

Redis storage design:

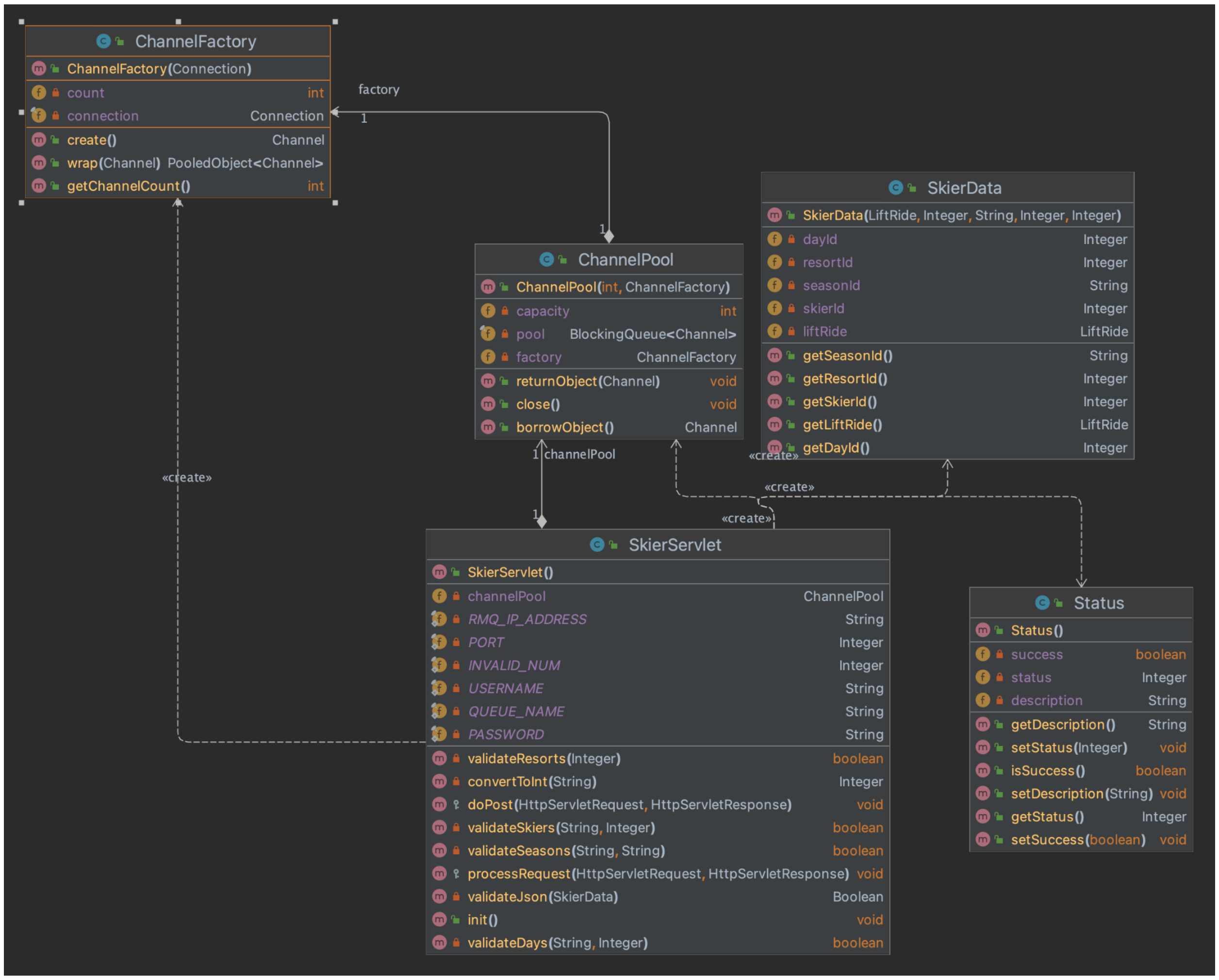
- Defined the key name in the convention *"keyType#valueType:someId"* for ease of search when we want to retrieve them. For the value, use a set list to store the data like dayId, liftId for the value. For instance,
    - `key: "skier_id#days:'skierID'", value:[set of dayId]`
    - `key: "day_id#vertical:'skierID'", value:[set of liftId]`
    - `key: "resort_id:'resortId'#days:'dayId", value:[set of skierId]`

Class Design

Design for Servlets (Server) - no changes from Assignment 2

- This class handles the POST request that is validated for the path & LiftRide body when the user sends requests to the EC2 instance.

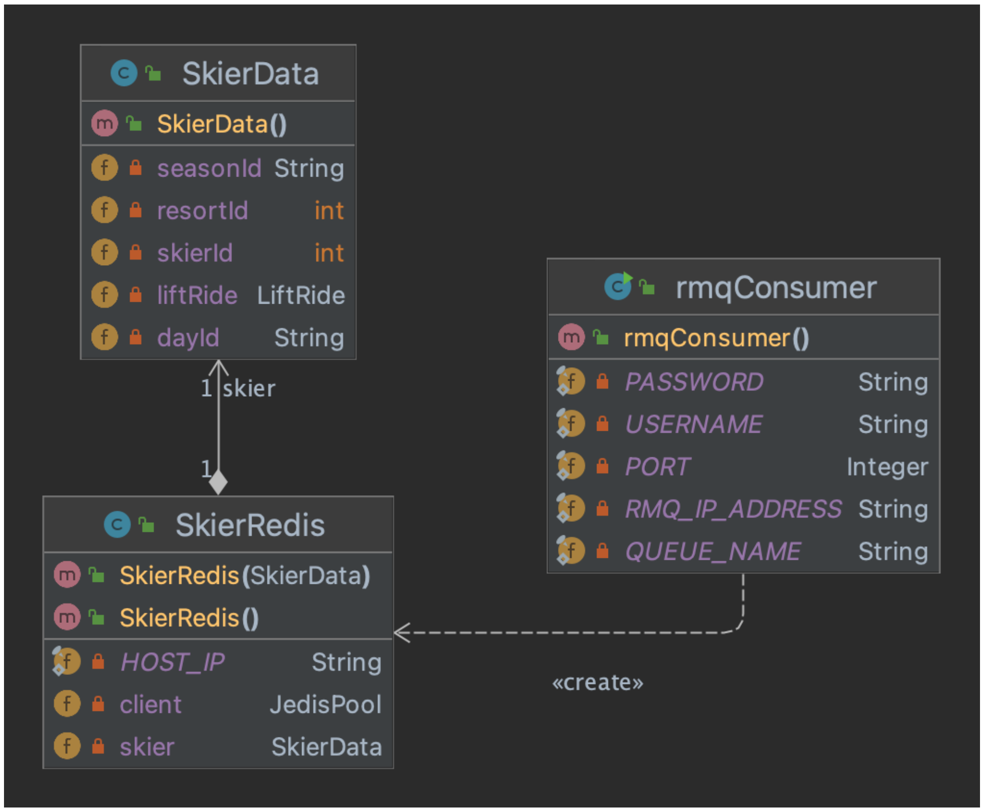
- Specify the private IP address of the RMQ and keep sending the payload of the **SkierData** to the RMQ
- Created a ***ChannelPool.java*** so we can reuse the thread whenever they have completed their work.



Design for RabbitMQ - no changes from Assignment 2

- Created skierQueue to be shared between producer-consumer by using t2.micro instance in AWS.
- Notes:
  - Created a separate Security Group that only allows inbound/outbound from consumer/producer private IPv4.

Design for Consumer - add Jedis/JedisPool to connect with Redis

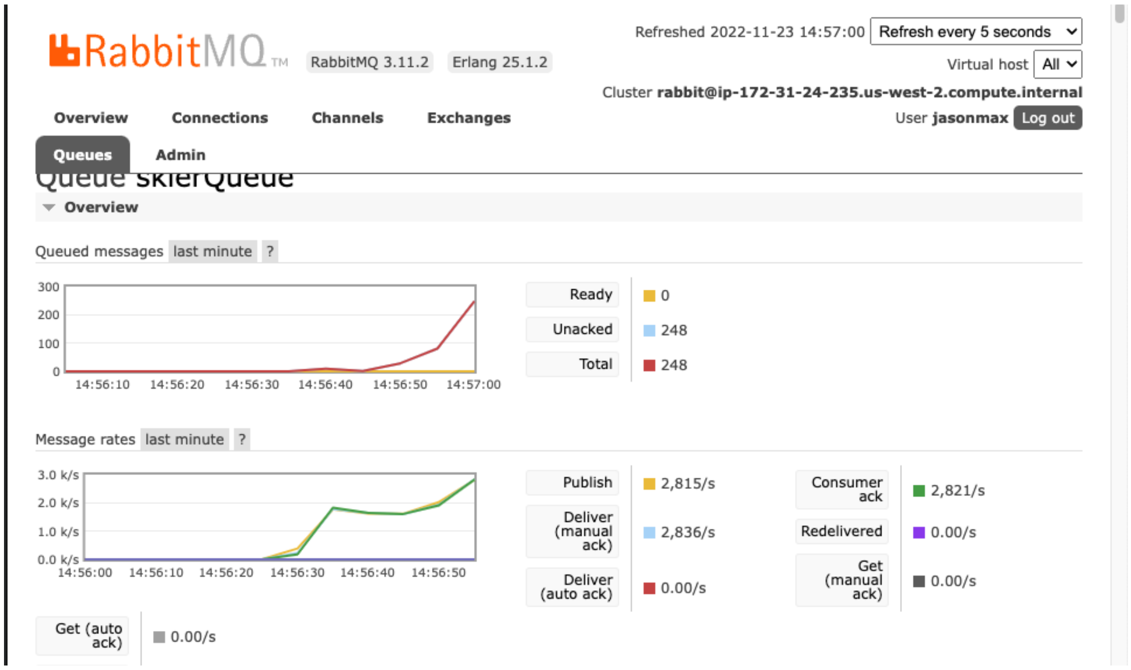


- Instead of writing it to a hashmap, The consumer here will retrieve the RabbitMQ *SkierQueue* and store the SkierData object into Redis by using the Jedis java library.
- In order to use Jedis to the fullest, I created SkierRedis class which initialized JedisPool, which is a resource pool to reuse the connection by other threads.

AWS Topology

Name	**Servlet**	**Rabbit MQ**	**Consumer**	**Redis**
EC2 type	t2.micro	t2.micro	t2.medium	t2.micro

Test Result



RabbitMQ™ RabbitMQ 3.11.2 Erlang 25.1.2

Refreshed 2022-11-23 14:57:00 Refresh every 5 seconds

Virtual host All

Cluster **rabbit@ip-172-31-24-235.us-west-2.compute.internal**

User **jasonmax** Log out

Overview Connections Channels Exchanges Queues Admin

Queue skierQueue

Overview

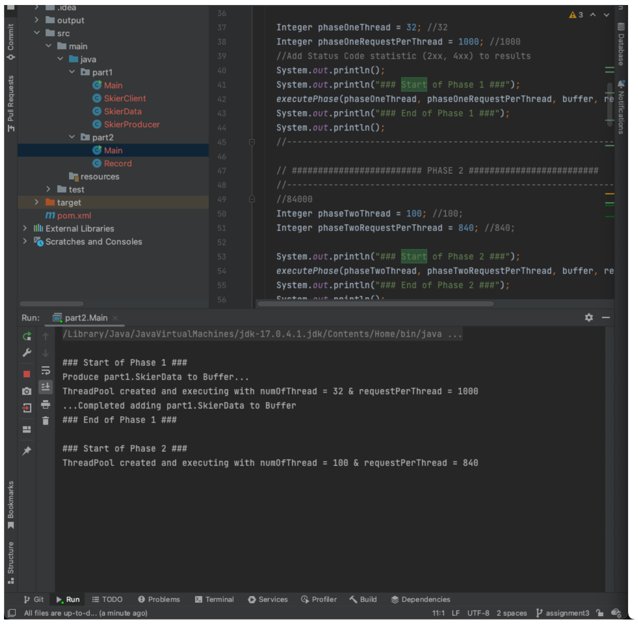
Queued messages last minute ?

Ready	🟨 0
Unacked	🟦 248
Total	🟥 248

Message rates last minute ?

Publish	🟨 2,815/s	Consumer ack	🟩 2,821/s
Deliver (manual ack)	🟦 2,836/s	Redelivered	🟪 0.00/s
Deliver (auto ack)	🟥 0.00/s	Get (manual ack)	⬛ 0.00/s
Get (auto ack)	⬜ 0.00/s		

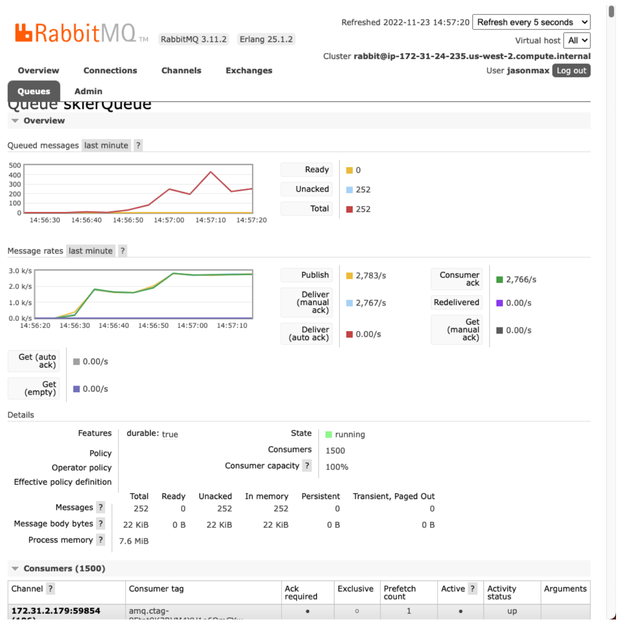
Rabbit MQ management console showing produced/consumed rate around 2,800/s. Client was in phase 2 where 840 message / 100 threads were executed



```java
Integer phaseOneThread = 32; //32
Integer phaseOneRequestPerThread = 1000; //1000
//Add Status Code statistic (2xx, 4xx) to results
System.out.println();
System.out.println("### Start of Phase 1 ###");
executePhase(phaseOneThread, phaseOneRequestPerThread, buffer, re
System.out.println("### End of Phase 1 ###");
System.out.println();
//-----------------------------------------------------------------

// ######################## PHASE 2 ########################
//-----------------------------------------------------------------
//84000
Integer phaseTwoThread = 100; //100;
Integer phaseTwoRequestPerThread = 840; //840;

System.out.println("### Start of Phase 2 ###");
executePhase(phaseTwoThread, phaseTwoRequestPerThread, buffer, re
System.out.println("### End of Phase 2 ###");
```

Run: part2.Main

```
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/java ...

### Start of Phase 1 ###
Produce part1.SkierData to Buffer...
ThreadPool created and executing with numOfThread = 32 & requestPerThread = 1000
...Completed adding part1.SkierData to Buffer
### End of Phase 1 ###

### Start of Phase 2 ###
ThreadPool created and executing with numOfThread = 100 & requestPerThread = 840
```



RabbitMQ™ RabbitMQ 3.11.2 Erlang 25.1.2

Refreshed 2022-11-23 14:57:20 Refresh every 5 seconds

Virtual host All

Cluster **rabbit@ip-172-31-24-235.us-west-2.compute.internal**

User **jasonmax** Log out

Overview Connections Channels Exchanges Queues Admin

Queue skierQueue

Overview

Queued messages last minute ?

Ready	🟨 0
Unacked	🟦 252
Total	🟥 252

Message rates last minute ?

Publish	🟨 2,783/s	Consumer ack	🟩 2,766/s
Deliver (manual ack)	🟦 2,767/s	Redelivered	🟪 0.00/s
Deliver (auto ack)	🟥 0.00/s	Get (manual ack)	⬛ 0.00/s
Get (auto ack)	⬜ 0.00/s		
Get (empty)	🟪 0.00/s		

Details

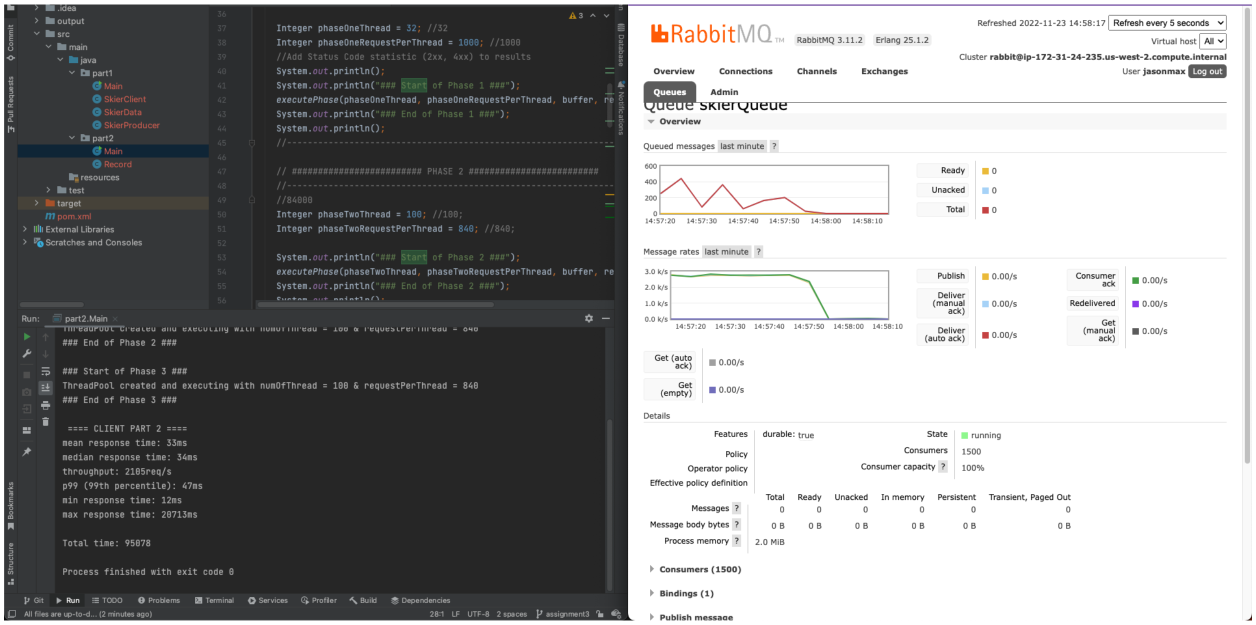
Features	durable: true	State	🟩 running
Policy		Consumers	1500
Operator policy		Consumer capacity ?	100%
Effective policy definition			

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	252	0	252	252	0	0
Message body bytes ?	22 KiB	0 B	22 KiB	22 KiB	0 B	0 B
Process memory ?	7.6 MiB					

Consumers (1500)

Channel ?	Consumer tag	Ack required	Exclusive	Prefetch count	Active ?	Activity status	Arguments
172.31.2.179:59854	amq.ctag-	●	○	1	●	up	

Image showing client is in phase 2 with queue under 500 messages and constant 3,000 message/s rates



```java
Integer phaseOneThread = 32; //32
Integer phaseOneRequestPerThread = 1000; //1000
//Add Status Code statistic (2xx, 4xx) to results
System.out.println();
System.out.println("### Start of Phase 1 ###");
executePhase(phaseOneThread, phaseOneRequestPerThread, buffer, re
System.out.println("### End of Phase 1 ###");
System.out.println();
//-----------------------------------------------------------------

// ######################## PHASE 2 ########################
//-----------------------------------------------------------------
//84000
Integer phaseTwoThread = 100; //100;
Integer phaseTwoRequestPerThread = 840; //840;

System.out.println("### Start of Phase 2 ###");
executePhase(phaseTwoThread, phaseTwoRequestPerThread, buffer, re
System.out.println("### End of Phase 2 ###");
```

```
ThreadPool created and executing with numOfThread = 100 & requestPerThread = 840
### End of Phase 2 ###

### Start of Phase 3 ###
ThreadPool created and executing with numOfThread = 100 & requestPerThread = 840
### End of Phase 3 ###

 ==== CLIENT PART 2 ====
mean response time: 33ms
median response time: 34ms
throughput: 2105req/s
p99 (99th percentile): 47ms
min response time: 12ms
max response time: 20713ms

Total time: 95078

Process finished with exit code 0
```

Refreshed 2022-11-23 14:58:17

Cluster rabbit@ip-172-31-24-235.us-west-2.compute.internal

Queue skierQueue

Overview

Queued messages last minute

Ready	0
Unacked	0
Total	0

Message rates last minute

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

Details

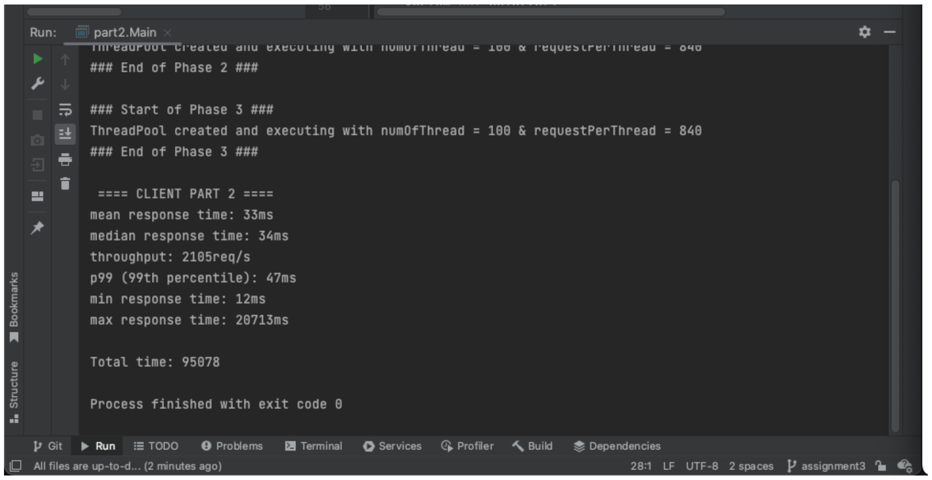
Features	durable: true	State	running
Policy		Consumers	1500
Operator policy		Consumer capacity	100%
Effective policy definition			

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages	0	0	0	0	0	0
Message body bytes	0 B	0 B	0 B	0 B	0 B	0 B
Process memory	2.0 MiB					

Consumers (1500)

Bindings (1)

Publish message



```
ThreadPool created and executing with numOfThread = 100 & requestPerThread = 840
### End of Phase 2 ###

### Start of Phase 3 ###
ThreadPool created and executing with numOfThread = 100 & requestPerThread = 840
### End of Phase 3 ###

 ==== CLIENT PART 2 ====
mean response time: 33ms
median response time: 34ms
throughput: 2105req/s
p99 (99th percentile): 47ms
min response time: 12ms
max response time: 20713ms

Total time: 95078

Process finished with exit code 0
```

Run: part2.Main
Git Run TODO Problems Terminal Services Profiler Build Dependencies
All files are up-to-d... (2 minutes ago)
28:1 LF UTF-8 2 spaces assignment3

Result: 200k requests took 95,078ms *(1m35s)* with throughput 2,105 req/s