

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:

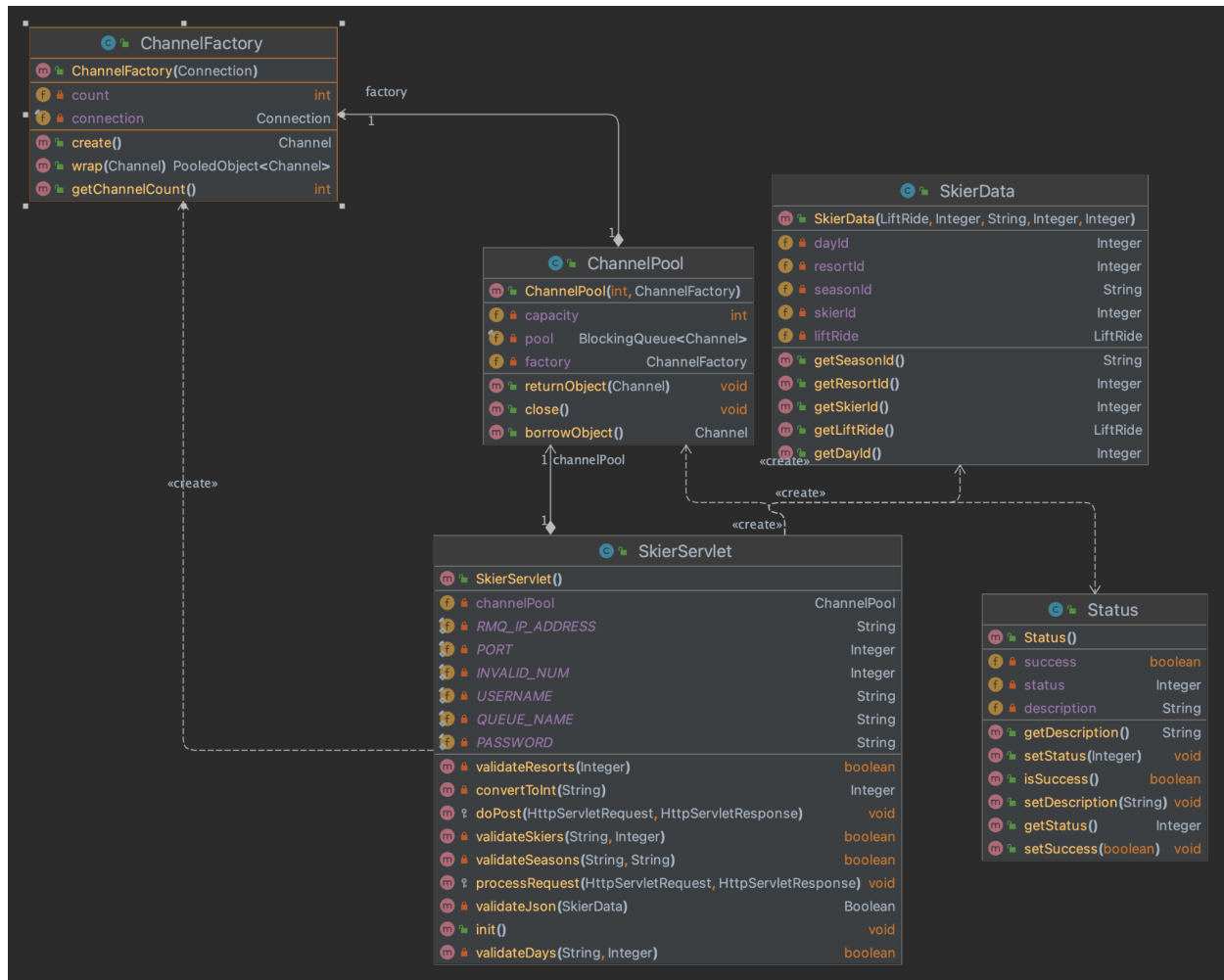
- Defined the key name in the convention "*keyType#valueType:someId*" for ease of search when we want to retrieve them. For the value, use a set list to store the data like dayId, liftId for the value. For instance,
 - *key: "skier_id#days:'skierID'", value:[set of dayId]*
 - *key: "day_id#vertical:'skierID'",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

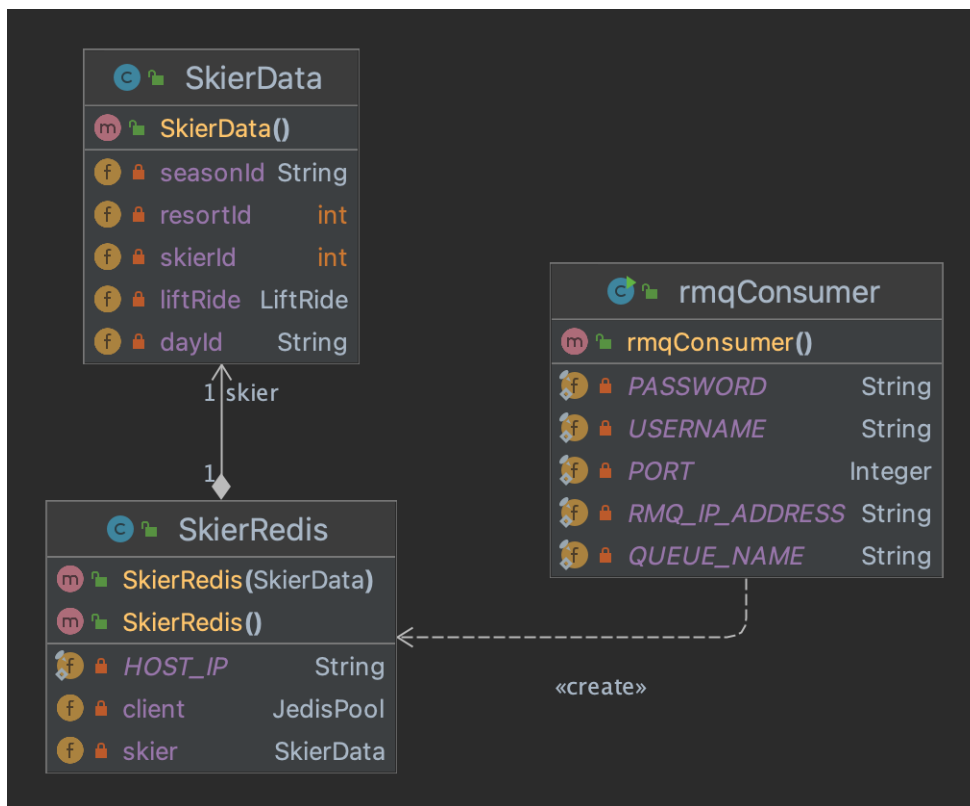
- Specify the private IP address of the RMQ and keep sending the payload of the **SkierData** to the RMQ
- Created a **ChannelPool.java** so we can reuse the thread whenever they have completed their work.



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

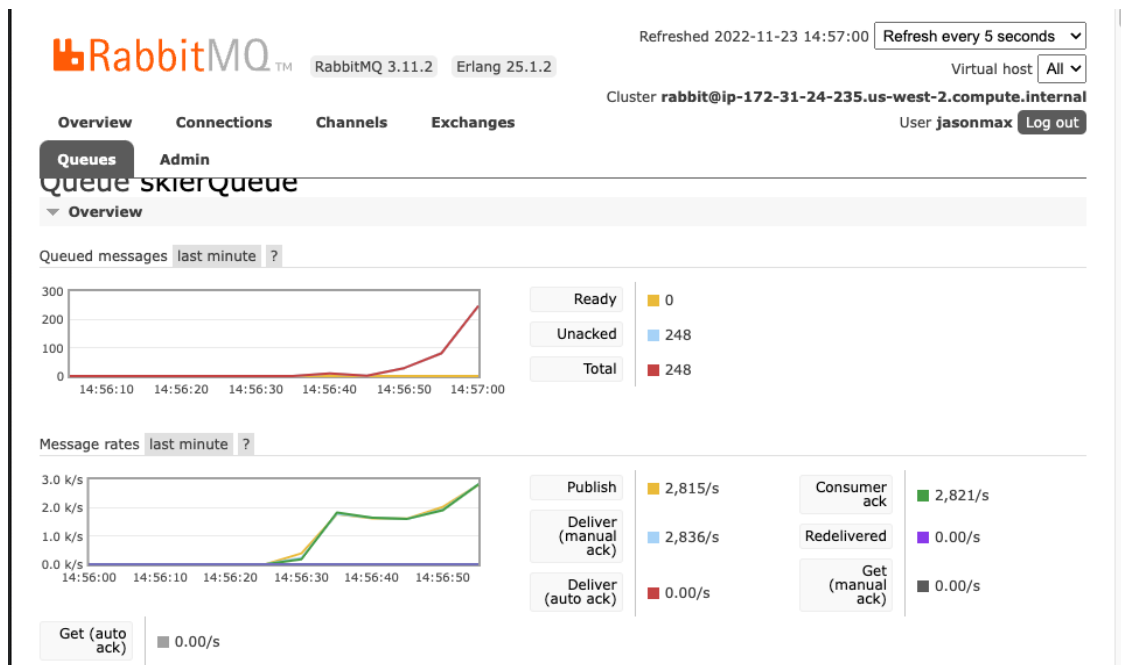


- Instead of writing it to a hashmap, The consumer here will retrieve the RabbitMQ *SkierQueue* and store the *SkierData* object into Redis by using the Jedis java library.
- In order to use Jedis to the fullest, I created *SkierRedis* class which initialized *JedisPool*, which is a resource pool to reuse the connection by other threads.
- Run 1,500 threads to consume from the RMQ

AWS Topology

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

Test Result



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

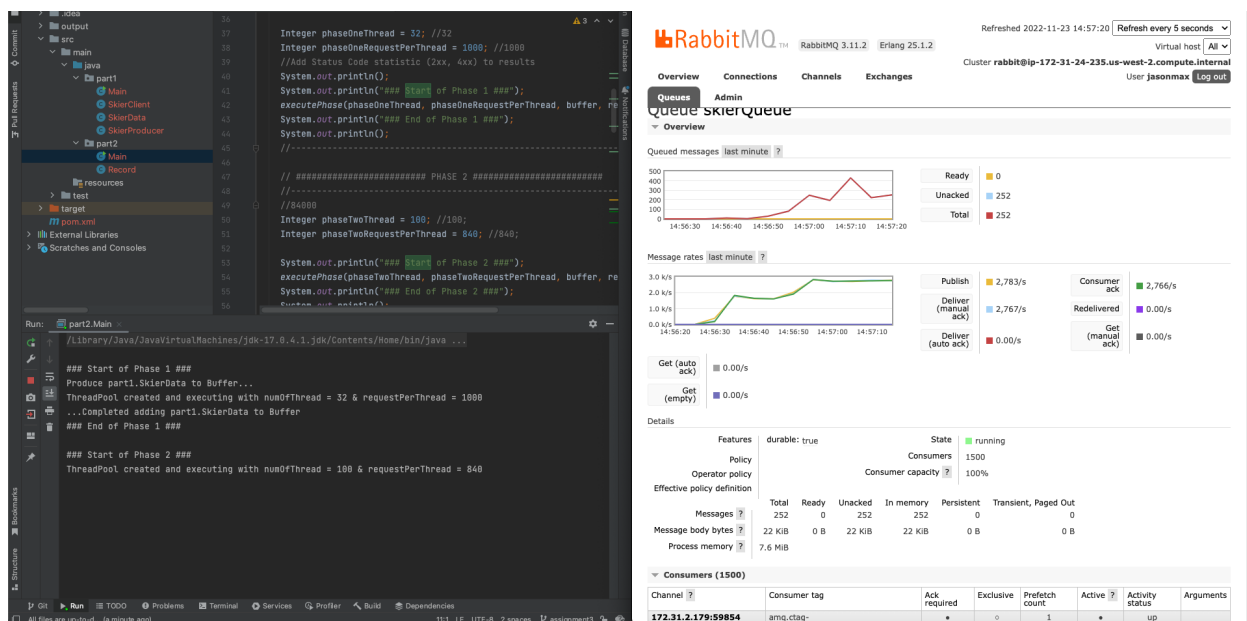
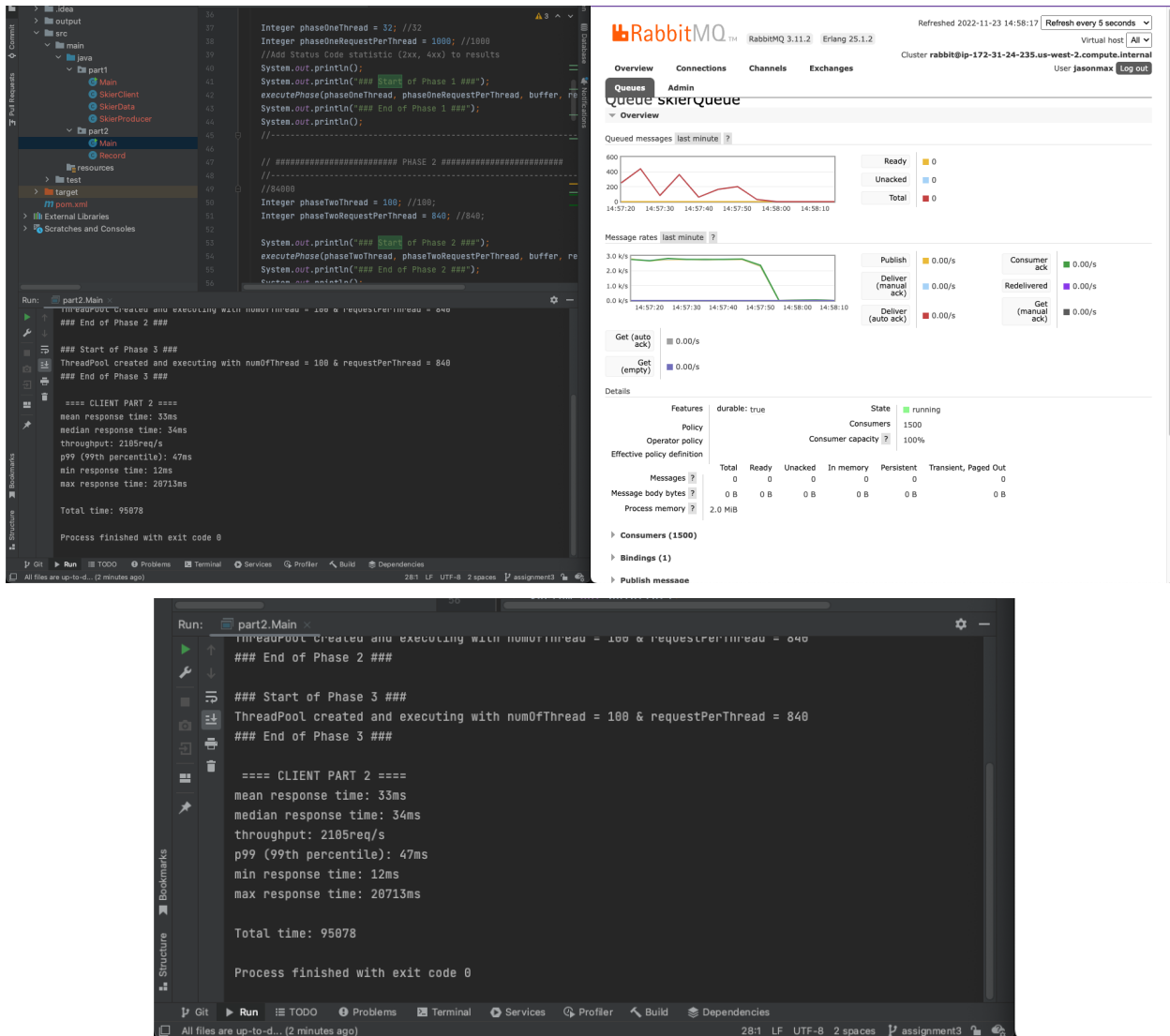


Image showing client is in phase 2 with queue under 500 messages and constant 3,000 message/s rates



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