CS6650 Assignment3

# 1. Github repo url:

https://github.com/IreneZ723/CS6650Assignment3

# 2. Server Design:

## 2.1 LikeServlet:

This servlet handles HTTP POST requests related to album likes or dislikes. It is mapped to the "/review/\*" URL pattern. The servlet uses RabbitMQ to send messages indicating album likes or dislikes. Messages are sent to the "review" queue.

### Methods:

* init(): Initializes the servlet by creating a RabbitMQ connection factory.
* doPost(HttpServletRequest request, HttpServletResponse response): Handles HTTP POST requests, extracting like/dislike information from the request URL, updating the likes/dislikes, and sending a message to the RabbitMQ queue.
* performReview(String likeOrNot, String albumID): Implements the logic to update likes or dislikes for a given albumID.
* sendToQueue(String msg): Sends a message to the RabbitMQ queue.

## 2.2Consumer:

This class is responsible for consuming messages from the RabbitMQ queue and updating the database accordingly.

### Methods:

* main(String[] argv): Initializes the necessary components, creates a pool of threads to consume messages, and shuts down the thread pool when done.
* consumeMessages(): Consumes messages from the RabbitMQ queue and updates the database.

### Dependencies:

LikeDao: Handles database operations related to album likes and dislikes.

DBCPDataSource: Provides a DataSource for obtaining database connections.

ExecutorService: Manages a pool of threads for message consumption.

### RabbitMQ Integration:

The consumer connects to the "review" queue in RabbitMQ.

Multiple threads are used to concurrently consume messages from the queue.

Each thread processes the received message by updating the database using LikeDao.

## 2.3 Relationships:

The LikeServlet and Consumer share a common RabbitMQ queue named "review".

When a user likes or dislikes an album through the servlet, a message is sent to the RabbitMQ queue.

The consumer, running in multiple threads, listens to the RabbitMQ queue, consumes messages, and updates the database accordingly using LikeDao.

The database connection is managed by DBCPDataSource.

Both applications use the RabbitMQ Java client library for interacting with RabbitMQ.

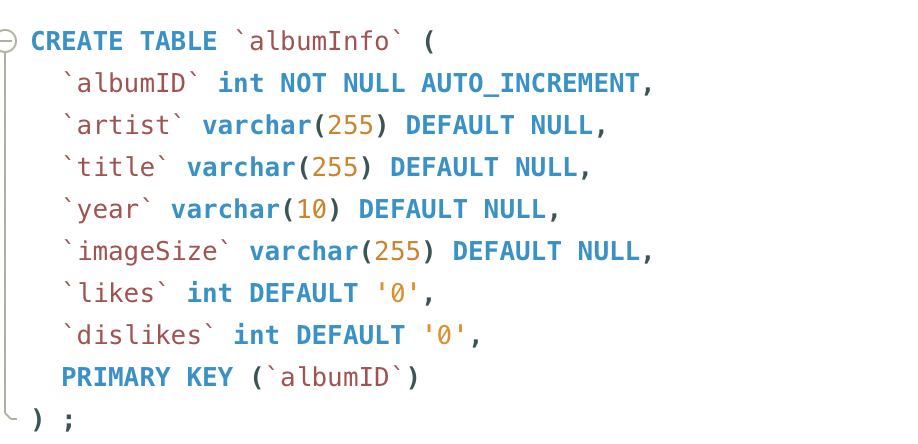
Exception handling is implemented for potential errors during initialization, message processing, and database operations.

Threading is employed in the consumer to achieve concurrent message processing.

## 2.4 Data Model Design

Add Two columns like and dislike to store the number of (dis)like received for particular albumID.

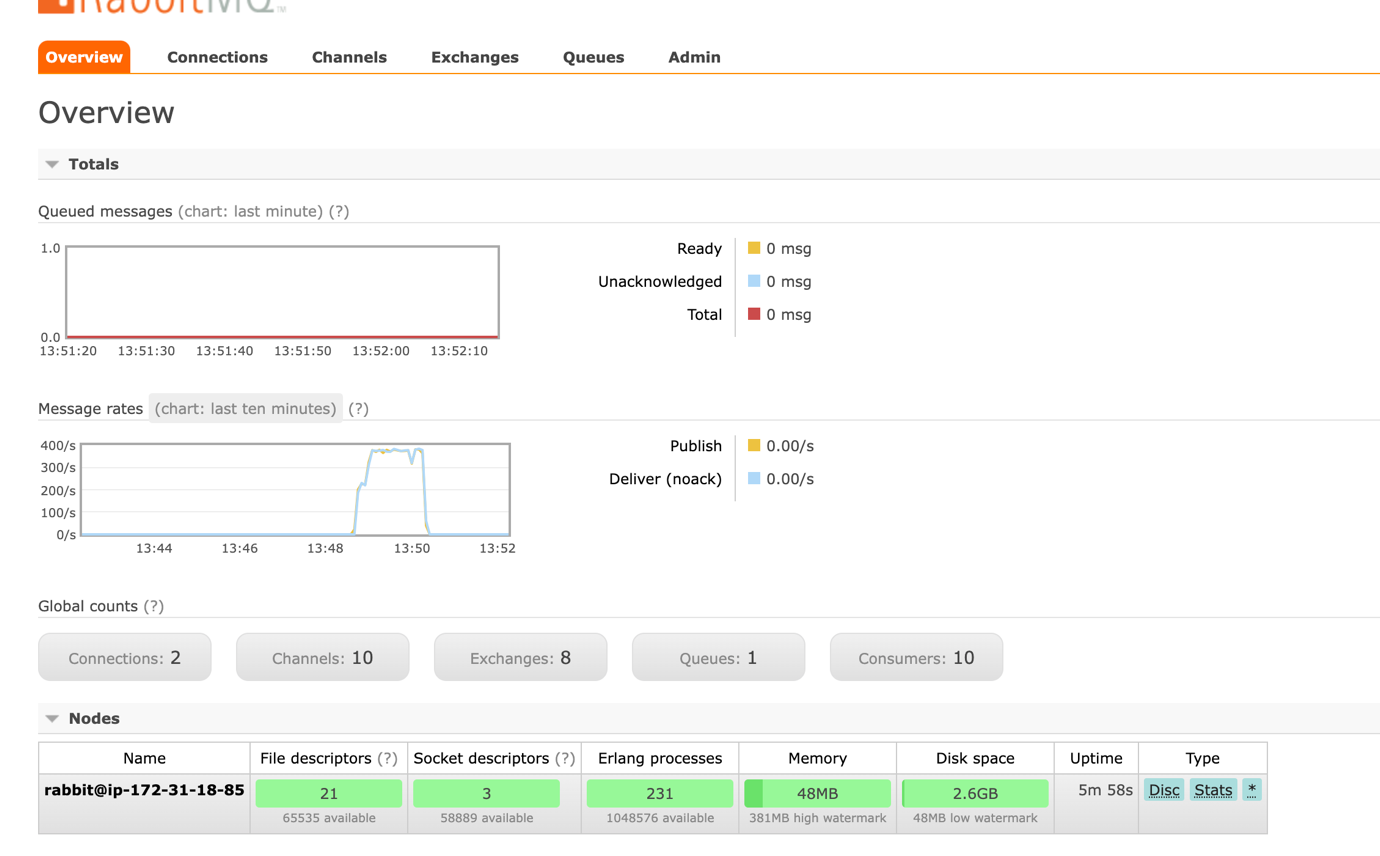
This create statement as follows:



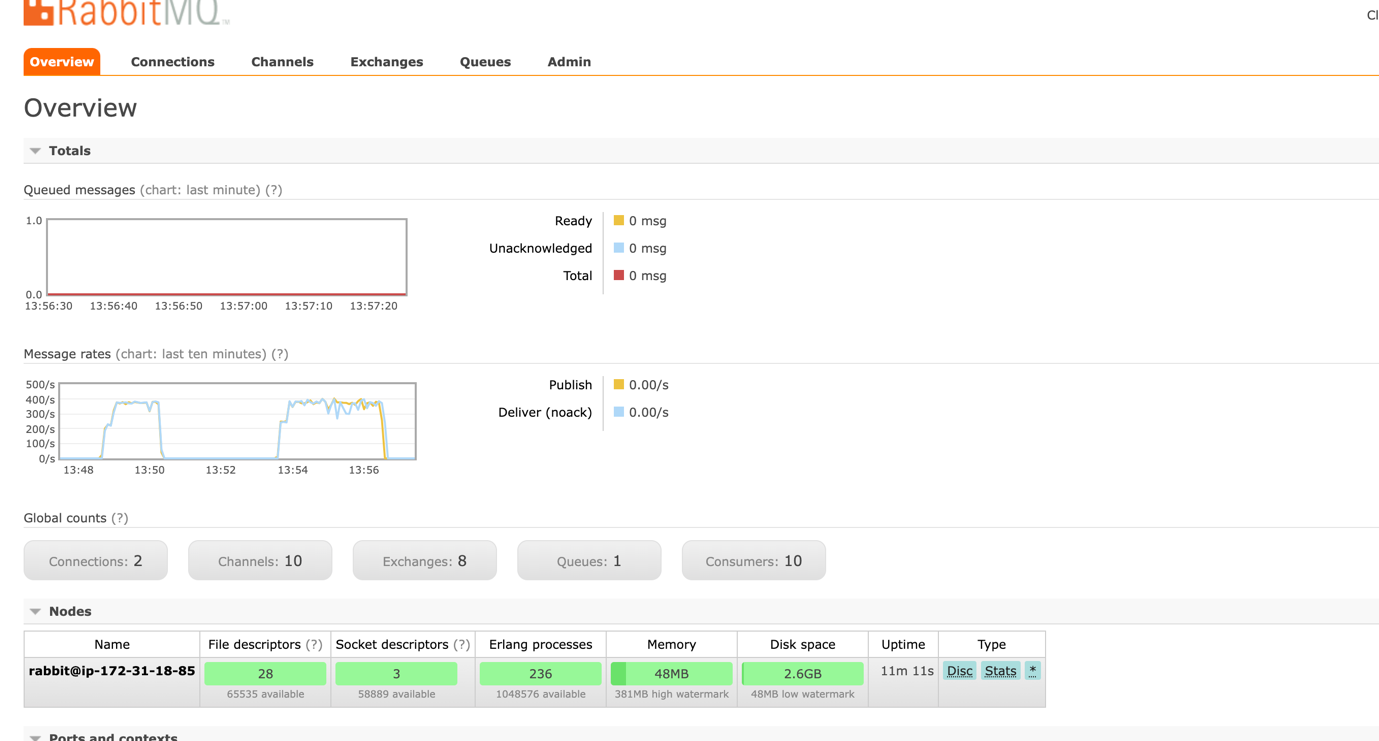
# 3. Test result

The throughputs are ‘roughly’ the same as assignment 2. In all tests, production\_rate ≈ consumption\_rate thus the queue length remains small. Also all message rate charts show an increase to a plateau like /¯¯¯¯\ and the plateau is less than a 1000.

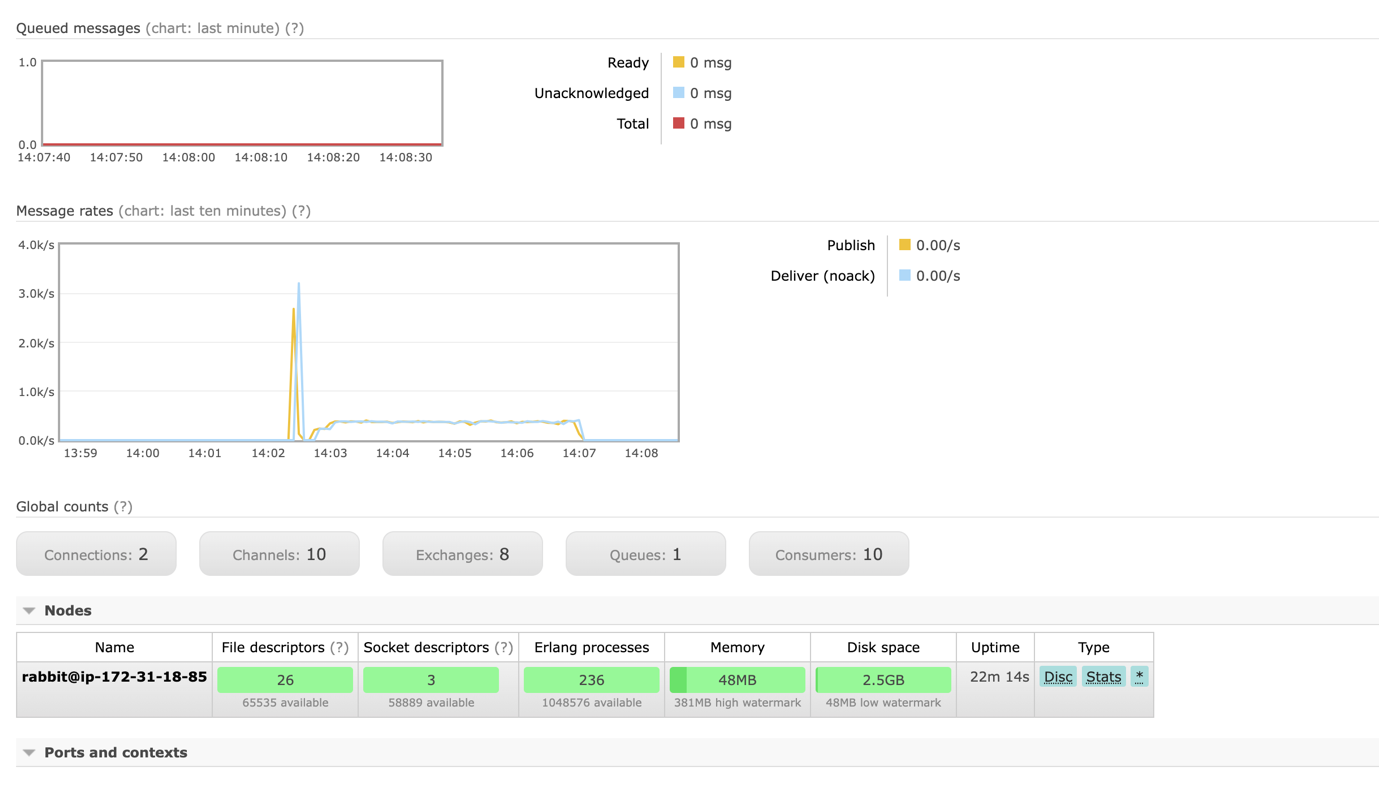
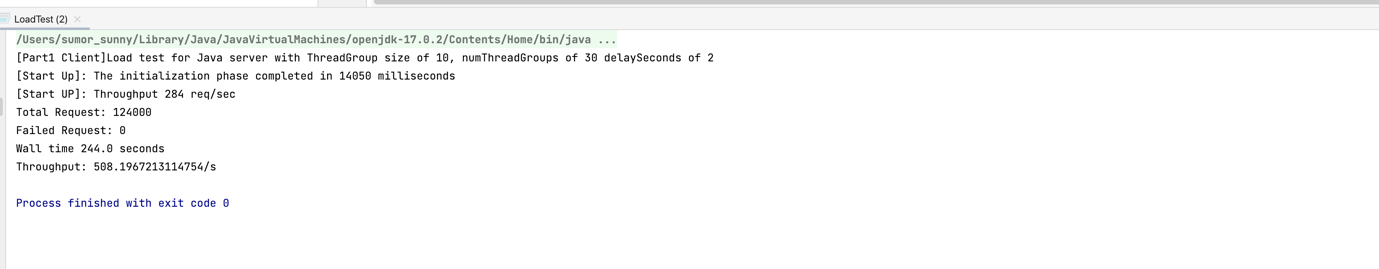
## 3.1 threadGroupSize = 10, numThreadGroups = 10, delay = 2 consumer = 10



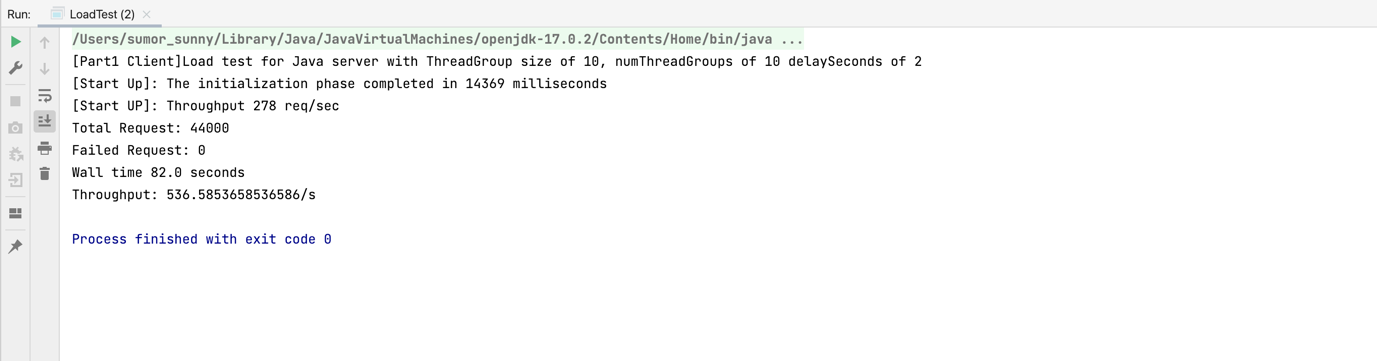
## 3.2 threadGroupSize = 10, numThreadGroups = 20, delay = 2 consumer = 10



## 3.3 threadGroupSize = 10, numThreadGroups = 30, delay = 2 consumer = 10

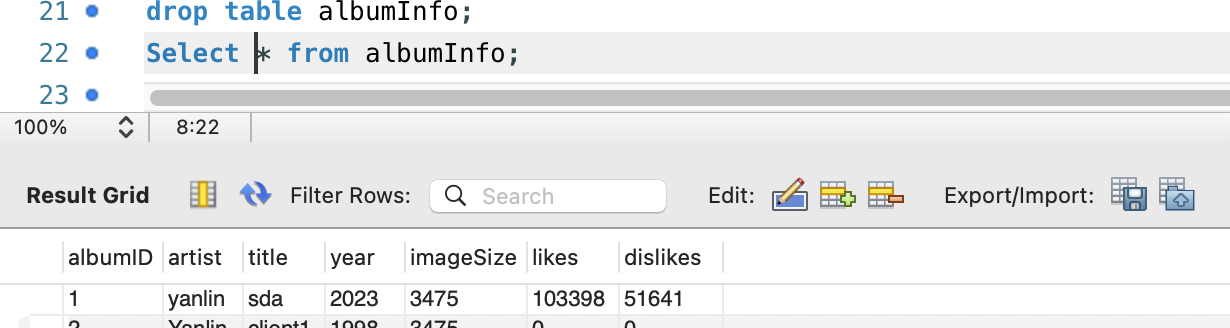


## 3.4 threadGroupSize = 10, numThreadGroups = 10, delay = 2 consumer = 30

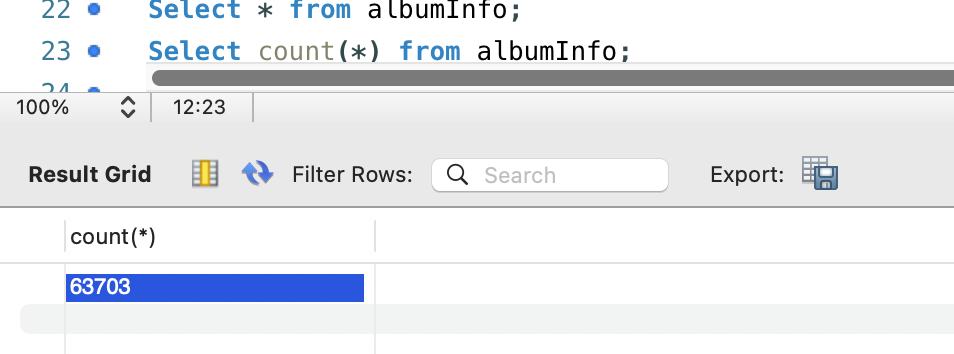


## 3.5 Database updated

### 3.5.1 The number of dislike and like



### 3.5.2 the number of albums



## 3.6 Monitoring EC2 instance

