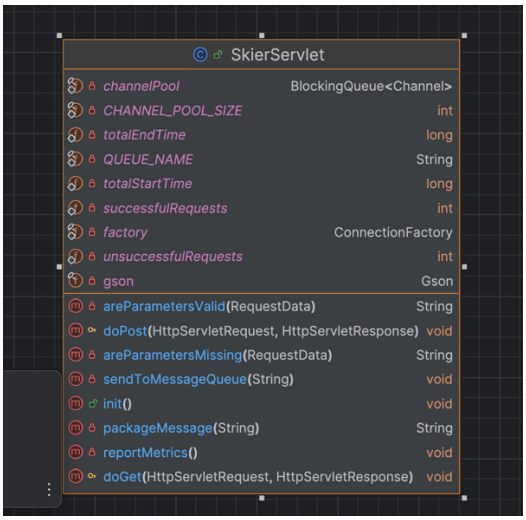
Assignment 2

GitHub Repo:
https://github.com/xq443/distributed-system/tree/main/Assignment/2

Server Design Overview:
The server is designed to handle incoming requests for ski lift rides using a RESTful approach through a servlet, while also leveraging RabbitMQ for message queuing. The design comprises two main components: the **SkierServlet** for handling HTTP requests and the **LiftRideConsumer** for processing messages from the RabbitMQ queue.

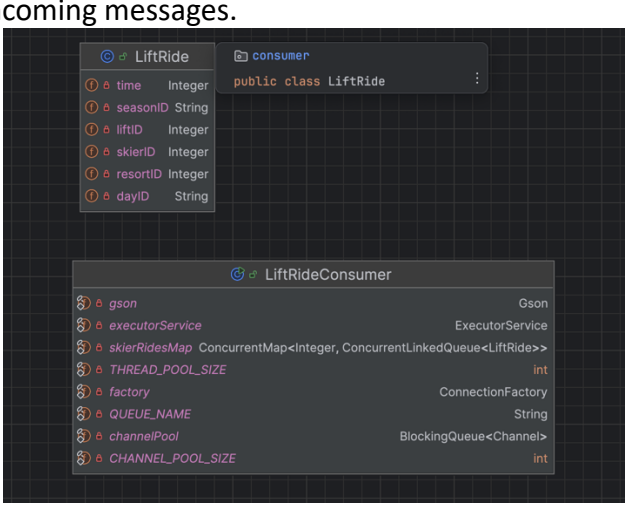
Major Classes and Packages

1. **SkierServlet: server**
    - **Responsibilities:**
        - Handle HTTP POST requests to register skier lift rides.
        - Validate incoming request data.
        - Serialize data into JSON format and send it to RabbitMQ.
        - Maintain metrics for successful and unsuccessful requests.
    - **Key Methods:**
        - doPost(HttpServletRequest request, HttpServletResponse response): Main method for processing POST requests.
        - sendToMessageQueue(String message): Sends the serialized message to the RabbitMQ queue.
        - reportMetrics(): Outputs metrics after processing a specified number of requests.



2. **LiftRideConsumer: consumer**
    - **Responsibilities:**
        - Consume messages from the RabbitMQ queue.

        - Deserialize JSON messages into LiftRide objects.
        - Store lift ride data in a thread-safe map for concurrent access.
    - **Key Methods:**
        - main(String[] args): Entry point to initialize the consumer.
        - initChannelPool(Connection connection): Initializes a pool of RabbitMQ channels.
        - getConsumer(Channel channel): Returns a consumer that processes incoming messages.



3. **RequestData and ResponseData (Package: com.cathy.bean)**
    - **RequestData**: Represents the incoming request data, containing fields like skierID, resortID, liftID, seasonID, dayID, and time.
    - **ResponseData**: Represents the outgoing response data, providing feedback for requests, including success and error messages.
4. **LiftRide**
    - Represents a lift ride with relevant attributes (e.g., skierID, resortID, seasonID..).

Relationships and Interactions

- **Servlet to Message Queue**:
    - The SkierServlet initializes a connection to RabbitMQ and sends serialized JSON messages containing lift ride data to the SkierQueue.
    - A pool of channels is maintained for efficient message sending, preventing the overhead of repeatedly opening and closing connections.
- **Message Consumer**:
    - The LiftRideConsumer establishes its own connection to RabbitMQ and consumes messages from the same SkierQueue.
    - Upon receiving a message, it deserializes the JSON content into a LiftRide object and stores the information in a concurrent data structure to allow for thread-safe access and modification.

Message Sending and Receiving

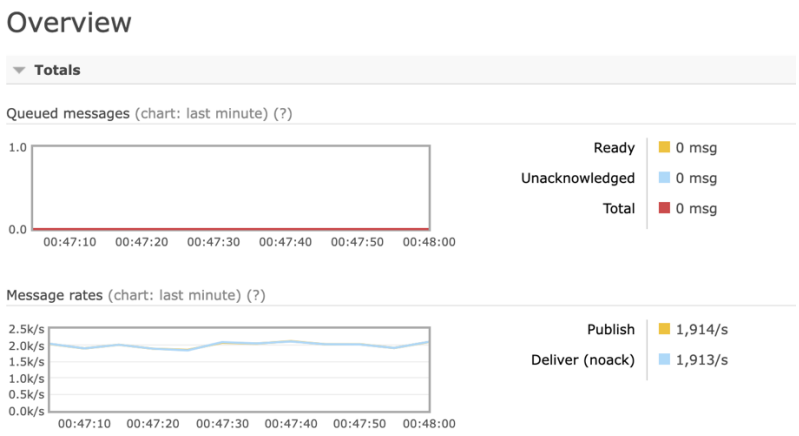
1. **Sending Messages**:

- Upon receiving a valid POST request, the servlet constructs a message string from the request body, then calls sendToMessageQueue(), which retrieves a channel from the pool and publishes the message to RabbitMQ.

2. **Receiving Messages**:
   - The consumer, upon initialization, creates a pool of channels and subscribes to the SkierQueue. Each message received is processed by the handleDelivery() method, which extracts the message content, deserializes it, and stores it in a thread-safe map.

Load Test Result for single EC2 instance:

Screenshot for an interval:

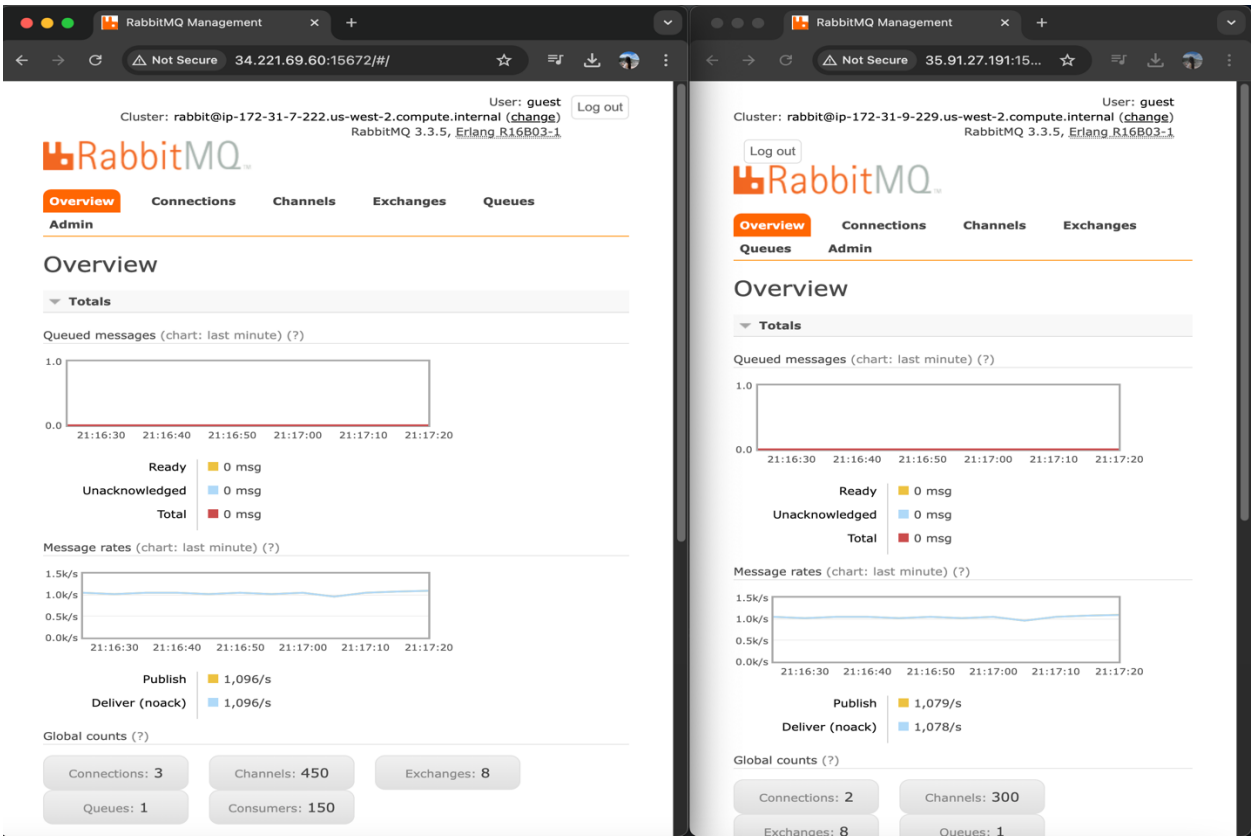



```
Thread counts: 280
Successful requests: 200000
Failed requests: 0
Total time: 103218 ms
Throughput: 1937.646534519173 requests/second
```

Queue length: 0
Best Throughput: 1937 requests/second
Number of consumer/producer channels: 150

Load Test Result for 2 Target Instances:

Screenshot for an interval:




```
Thread counts: 280
Successful requests: 200000
Failed requests: 0
Total time: 91601 ms
Throughput: 2183.3822774860537 requests/second
```

Queue length: 0
Best Throughput: 2183 requests/second
Number of consumer/producer channels: 150