Assignment 3 Report

Github Repo:

https://github.com/CloudyZ524/6650-Distributed-System/tree/main/Assignment/Assignment3

Database Design

Datebase: DynamoDB
Table Name: SkiResortData
Attributes:

- **SkierId** (Number): The ID of the skier, used as the hash key (partition key).
- **SeasonDayId** (String): A combined sort key made up of the season and day, used as the range key.
- **ResortId** (Number): The ID of the resort where the skiing activity took place.
- **LiftRide** (Map): The lift ride object generated by each request.
- **Time** (Number): The time associated with a particular lift ride.
- **LiftID** (Number): The ID of the lift used for the ride.
- **Billing Mode**: PAY_PER_REQUEST, which means i can bill for read and write throughput on demand.

Partition key: SkierId (Number)
Sort key: SeasonDayId (String)

This design leverages DynamoDB's ability to handle semi-structured data and its flexible schema to store varied data types. The primary key is a composite of SkierId (partition key) and SeasonDayId (sort key), allowing query efficiently for all the rides of a particular skier on a specific day in a season.

Deployment Topology on AWS:

The deployment is composed of an EC2 instance and a DynamoDB table within AWS. The EC2 instances run a consumer application that reads messages from a queue on RabbitMQ and writes to the DynamoDB table.

Consumer Design

1. SkiRecord Model:

This Java class models the ski resort data. It has properties to store skier, season, day, and resort IDs, along with a LiftRide object.

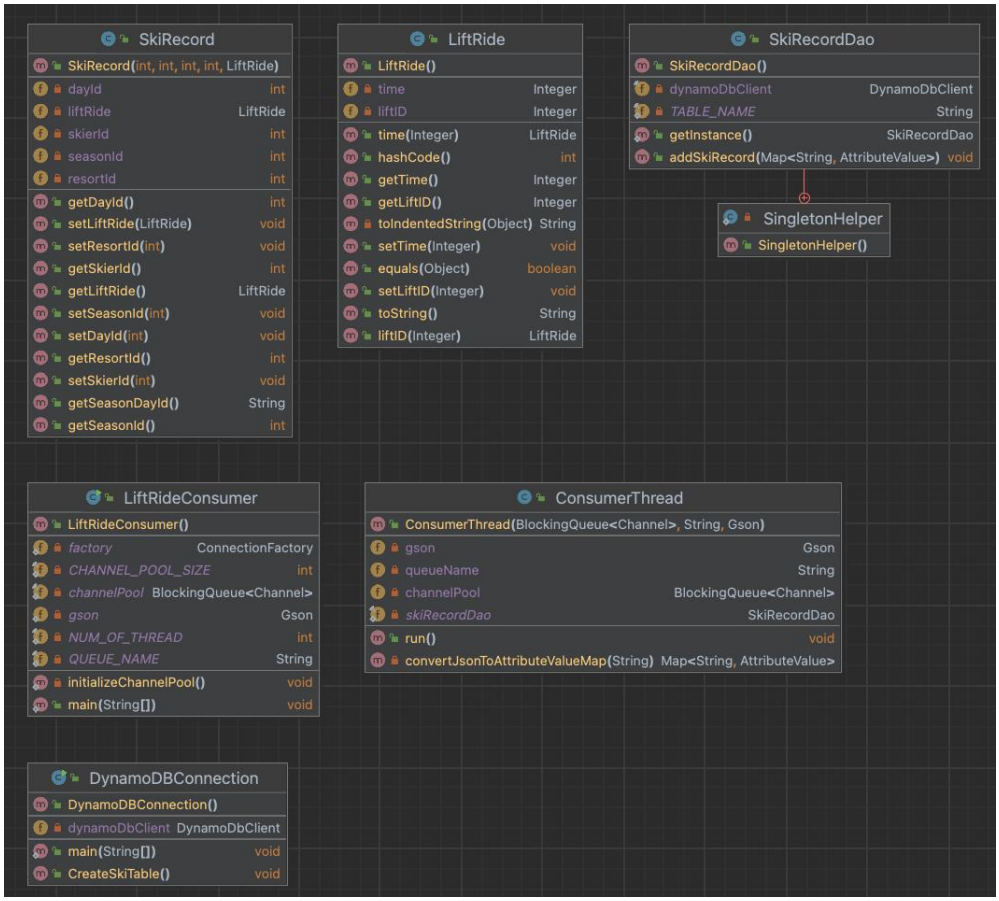
The class provides getters and setters for its properties and a method getSeasonDayId() to generate a compound key used as a sort key in DynamoDB.

2. SkiRecordDao:

This Data Access Object (DAO) class is responsible for interacting with DynamoDB. It abstracts the underlying database operations from the consumer logic.
It provides the method addSkiRecord(SkiRecord skiRecord) which takes a SkiRecord object, converts it to a DynamoDB item, and writes it to the database using the AWS SDK.

3. DynamoDBConnection

This class is useful for setting up the initial database schema required for storing ski resort data. It ensures that the necessary table exists before the consumer application starts processing messages and writing data to DynamoDB.



Consumer Workflow:

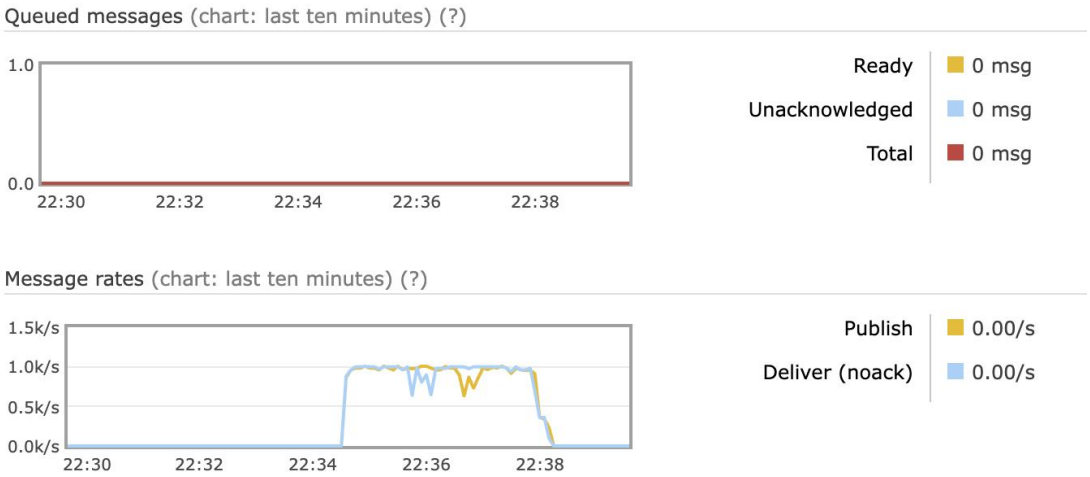
Message Consumption and Model Creation:

When a ConsumerThread receives a message from the queue, it first deserializes the message into a SkiRecord object using getSkierIdFromMessage() and getLiftRideFromMessage(), provided by LiftRideConsumer.

Database Interaction:

The ConsumerThread then calls the addSkiRecord(SkiRecord skiRecord) method of SkiRecordDao. This method is responsible for transforming the SkiRecord object into a format suitable for DynamoDB and executing the PutItem operation.

Result of client throughput and RMQ console:




```
Successful requests: 200000
Failed requests: 0
Total time: 217380 ms
Throughput: 920.0478424878094 requests/second
```

Length of Queue: 0
Throughput: 920 requests/second
Number of producer channels: 120
Number of consumer channels: 200
Number of consumer threads: 200