Delivery of assignment3 - 6650 by QingChen

URL

- **Consumer**: `https://github.khoury.northeastern.edu/chenqing/consumer-hw2`
- **Server**: `https://github.khoury.northeastern.edu/chenqing/cs6650_assignment1_server`
- **Client**: `https://github.khoury.northeastern.edu/chenqing/cs6650_assignment1_client`

Database Design

1. DynamoDB Table Design

This project involves two DynamoDB tables to store skier and resort data.
The tables are structured to support queries for ski rides and resort information based on skier and resort IDs.

SkierDB Table

- **Partition Key**: `skierID` (Number)
  This is the unique identifier for each skier.
- **Sort Key**: `seasonID-resortID-dayID-liftID-time` (String)
  A composite sort key, which uniquely identifies each lift ride during a specific day of the season at a resort.
- **Other Attributes**:
  - `seasonID` (String)
  - `resortID` (Number)
  - `dayID` (String)
  - `liftID` (Number)
  - `time` (Number)

The `SkierDB` table stores information about each skier's lift rides during the ski season, including which resort, day, and lift were involved, as well as the time spent on the lift.

ResortDB Table

- **Partition Key**: `resortID` (Number)
  This is the unique identifier for each resort.
- **Sort Key**: `dayID-seasonID-liftID-time-skierID` (String)
  A composite sort key that combines the day, season, lift, time, and skier ID, which helps uniquely identify each lift ride per day at the resort.
- **Other Attributes**:
  - `seasonID` (String)
  - `dayID` (String)
  - `liftID` (Number)
  - `time` (Number)
  - `skierID` (Number)

The `ResortDB` table stores the lift ride details for each resort, including the season, day, and specific lift ride information for each skier.

AWS Deployment Topology

1. Instance Types

For the EC2 instance hosting the consumer service, we recommend using the following instance type, depending on the scale of your system:

- **Instance Type**: fee-tier t2.micro
- **Number of Instances**: load balancer using 2-4 instances for auto-scaling

2. DynamoDB Configuration

- **Default setting**: For this project, **On-demand** capacity mode is ideal as it automatically adjusts the throughput capacity based on the workload. This is beneficial for unpredictable workloads and allows automatic scaling.

Data Flow

1. Message Consumption from RabbitMQ

The `Consumer` service consumes messages from RabbitMQ. Each message contains data related to a skier's lift ride at a resort. The messages are processed using a multi-threaded approach with a thread pool, allowing multiple consumers to process messages in parallel.

2. Data Insertion into DynamoDB

For each message received from RabbitMQ:

- **Skier Data**: The message is parsed, and the skier's lift ride information is saved into the `SkierDB` DynamoDB table.
- **Resort Data**: The message is also parsed, and resort-specific data is stored into the `ResortDB` DynamoDB table.

This ensures that each skier's lift ride data is recorded and indexed both by their skier ID (for personal history) and by the resort ID (for resort-specific insights).

How to increase capacity

- **batchWrite**: I tried batch write to handle multi requests but see no difference
- **Elastic Load Balancer (ELB)**: deployed an ELB
- **DynamoDB provisioned + auto scaling**

throughput screenshot

client2 - 168threads

```
Number of successful requests sent: 200000
Number of unsuccessful requests sent: 0

--- Test 168 Threads for remaining events ----
Posts were sent! Time taken: 82s
Total throughput is: 2439 requests per second

Process finished with exit code 0
```

client2 - 210threads

```
--- Test 210 Threads for remaining events ----
Posts were sent! Time taken: 130005ms
Total throughput is: 1538 requests per second

--------Calculation Part-----------
Mean response time: 79.38185809809448 ms
Median response time: 58.0 ms
99th percentile response time: 325.0 ms
Min response time: 15 ms
Max response time: 703 ms
Throughput: 1538 requests per second
```

client1 - 168threads

```
Number of successful requests sent: 200000
Number of unsuccessful requests sent: 0

--- Test 168 Threads for remaining events ----
Posts were sent! Time taken: 82s
Total throughput is: 2439 requests per second

Process finished with exit code 0
```

client1 - 100threads

```
--- Test 100 Threads for remaining events ----
Posts were sent! Time taken: 118s
Total throughput is: 1694 requests per second

Process finished with exit code 0
```

RMQ