Hotel Booking Service

Introduction

You're building an application that allows for booking hotel reservations.

The provided code includes functionality for booking reservations, canceling reservations, and modifying reservations.

The main entity in our service is:

- Reservation: Stores information about a hotel reservation, including cost and check-in date.
    - Partition key: reservationId

Our service will support these operations related to reservations:

- BookReservation - creates a new reservation
- CancelReservation - cancels an existing reservation with a full refund.
- ModifyReservation - modifies an existing reservation (changing the check-in date and/or number of nights) and modifies the total cost based on any change in the number of nights. We provide a partial refund if the number of nights of the stay is shortened and charge more if the stay is extended.

The code base follows the Activity-DAO-DynamoDBMapper pattern that we've come to know and love. The various Activity classes each implement one operation that our service supports. Each Activity depends on `ReservationDao` to accomplish their use cases.

Disclaimer: One difference that you'll notice here is that our Activity classes don't yet accept/return Response/Result objects. They're accepting/returning individual values. We'll do this until it's time to create our service infrastructure and create the necessary Coral models. That retrofit is beyond the scope of this activity, but will be fairly easily accomplished in the Activity classes themselves when the time comes.

Phase 0: Set up

1. Create the tables we'll be using for this activity by running these aws CLI commands:

    ```
    aws cloudformation create-stack --region us-west-2 --stack-name
    metrics-reservation-table --template-body
    file://cloudformation/reservations_table.yaml --capabilities
    CAPABILITY_IAM
    ```

2. Log into your Unit 6 AWS account and verify that the table exists and has sample data.
3. Review the different attributes and provided activities to make sure you understand what they represent.
4. Based on what you've learned about the service, think about what metrics you could log and add your ideas to the discussion board.
5. As a final verification, run `Phase0Test` and make sure it passes.

GOAL: Reservations table is created in your AWS Account, and the attributes make sense to you.

Phase 0 is complete when:

- You understand the `Reservation` data type and the provided activities.
- You've added some ideas for metrics to track in the discussion board.
- Reservation table exists with some sample data
- `Phase0Test` passes

Phase 1: But who's counting?

Our business team wants to track the number of bookings, cancellations, and modifications to reservations, so that they can see easily view booking trends over time.

Your task to is add metrics that log a metric every time we have a successfully completed reservation, cancellation, and modification. We can track this by updating each of the corresponding Activity classes with a count metric.

Create and log these 3 separate metrics:

1. BookedReservationCount: This is the number of booked reservations, based on the calls to BookReservationActivity. When implementing this metric, use the metric name `BookedReservationCount`.
2. CanceledReservationCount: This is the number of canceled reservations we get. When implementing this metric, use the metric name `CanceledReservationCount`.
3. ModifiedReservationCount: This is the number of times a reservation has been modified, based on the calls to ModifyReservationActivity. When implementing this metric, use the metric name `ModifiedReservationCount`.

For all three metrics, set the unit to be `StandardUnit.Count`.

We have provided a `MetricsPublisher` class that provides a helper method called `addMetric` that publishes metrics to CloudWatch. Take a look at the class and the method so that you understand how to use it for recording your metrics. You'll see that we already have two dimensions to your metric: Service, which maps to the name of our service name `ATAHotelReservationService`, and Marketplace, which currently only maps to `US` (we'll update this logic when we launch in other marketplaces!). You don't need to add any other dimensions for this activity. The `MetricsPublisher` has already been injected into the activity classes for you to use.

We have also provided a `MetricsConstants` class that contains some constant strings used for recording metrics. Be sure to add the names of the metrics you create.

Run `Phase1Test`. This makes some calls to your service. If things are working, then you should see your metrics in CloudWatch!

To check your CloudWatch metrics:

1. Login to your AWS Account
2. Make sure you're in the US West (Oregon) region
3. Go to Services->CloudWatch
4. Go to Metrics

5. Under Custom Namespace, click Unit4MetricsClassroomActivity
6. Click Marketplace, Service
7. You should see your new metrics listed! Click the checkbox on the left to graph them above.
8. In the graph metrics tab, make sure to update the statistic to sum to get the proper amounts graphed.
9. Also update the graph aggregation to a period of 1 minute or less, so that you can see the new datapoints update more often. Otherwise, you'll need to wait longer to see new metrics!
10. Run the tests a second time and check CloudWatch again for the updates! Feel free to run the tests multiple times to generate more metrics.

Phase 1 is complete when:

- You've added metrics that publish the `BookedReservationCount`, `CanceledReservationCount`, and `ModifiedReservationCount` to CloudWatch.
- You've viewed your metrics in CloudWatch.
- `Phase1Test` passes

Phase 2: Show me the money

Your hotel booking service seems to be doing pretty well, and your business team wants more insight into how well it's doing. They want to know how much income they're generating from bookings, so that they can easily check if the revenue they're generating is trending upward over time.

Create and log this metric:

Revenue: This is the amount on money made from bookings, based on the `totalCost` of a Reservation. We'll want to log metric information each time we book a reservation and generate revenue, and also when we modify or cancel a reservation, which also can generate or cause a loss of revenue. When implementing this metric, use the metric name `ReservationRevenue` and the unit `StandardUnit.None`. Remember to add the `ReservationRevenue` metric name string to the `MetricsConstants` class -- this will allow you to easily reference the metric name in different activities.

For this metric, make sure to consider each of these scenarios:

1. When a reservation is booked, the totalCost of the reservation is the amount of revenue generated.
2. When a reservation is canceled, the totalCost gets updated to a negative number, indicating that a refund was issued (so this revenue is lost).
3. When a reservation is modified, revenue may be generated or lost depending on how the reservation is modified. For example, if the reservation was initially for 3 nights, but is modified to be 5 nights, then we gained revenue from the extra 2 nights added. However, if that same reservation was instead modified to 2 nights, then we lost some revenue because we'd have to give a refund for 1 night. Remember that when the reservation was first booked, we logged the original revenue that was generated.

Note that the `modifyReservation` method in the `ReservationDao` returns an object called `UpdatedReservation`. This object contains two `Reservation` objects: one is the original `Reservation` object, and the other is the modified `Reservation` object. When calculating the revenue metric to log for `ModifyReservationActivity`, use the `totalCost` values in the original and modified reservations to calculate the revenue metric.

Run `Phase2Test` to test your changes and to generate some traffic that should log some metrics.

Check your CloudWatch metrics again using the same steps as Phase 1. You should now see a new `Revenue` metric!

Hints:

- I'm not sure how to log the metric for cancelations.
- I'm not sure how to log the metric for modified reservations.

Phase 2 is complete when:

- You've added metrics that publish the `ReservationRevenue` metric to CloudWatch.
- You've viewed your new metric in CloudWatch.
- `Phase2Test` passes

Extensions

Extension 1

Implement any of the other metrics that were brought up in the brainstorming! Some examples:

- Latency of the APIs
- Count of modifications that shorten the reservation vs those that lengthen the reservation.

Extension 2

Save your metrics to a CloudWatch dashboard to easily view all your metrics at once!

To create a new dashboard in CloudWatch, follow these steps:

1. Open CloudWatch in your AWS console.
2. Click Dashboards in the menu on the left.
3. Click Create dashboard.
4. Give a name to your dashboard and press submit.
5. In the "Add to this dashboard" window, select "Line: Compare metrics over time" and click Configure.
6. Search for your metric that counts the number of queries for your cache and add it to the graph (by clicking the checkbox on the left).
7. Click Create widget. You should now see your metric on your new dashboard! You can add more metrics to your dashboard by clicking "Add widget" at the top of your dashboard and following steps 5-7 above.