**Assignment 4**

Our project has two parts.

In the first part, we use AWS SQS to store the requests at server side and use lambda to read the messages out from SQS to process each request including write to/read from database. Since we SQS can retry unsuccessful requests and batch write into database, we expect lower latency and more reliable service by using lambda.

In part two, we use DynamoDB instead of MySQL to store skier lift data. With DynamoDB autoscaling enabled, we expect faster and more reliable database access and lower latency.

**Part 1**

**Result:**

Original server without SQS or lambda.

A screenshot of text

Description automatically generated

With SQS and lambda

A screenshot of text

Description automatically generated

The following metrics graphs show the resource usage when sending requests to the original server and server with SQS + lambda. 13:45-13:50 is roughly when we sent requests to the original server. 13:50-13:55 is roughly when we sent requests to server with SQS + lambda.

A screenshot of a social media post

Description automatically generated

We can see that EC2 server CPU utilization is reduced using SQS + lambda.

A screenshot of a cell phone

Description automatically generated

We can see the using SQS batch write to database, the database connection number of largely reduced.

A screenshot of a social media post

Description automatically generated

**Part 2**

In part 2, we use DynamoDB as our data storage. The partition key is concatenation of ResortId, SeasonId, DayId and SkierId. The primary sort key is time. We use the partition key and primary sort key to ensure each record is unique. We set the write provisioned capacity to 100 units since autoscaling takes time and we want the server to respond fast at the beginning. Also, the cost of RDS instance we are using is 0.068/h. This configuration has the similar cost which is 49.33/month. We can see with similar cost, DynamoDB has shorter wall time 164s compared to RDS 189s.

Configuration:

**Provisioned capacity:**

Read capacity units: Table 10

Write capacity units: Table 100

**Autoscaling:**

Read capacity:

Target utilization: 70%

Minimum provisioned capacity: 10 units

Maximum provisioned capacity: 40000 units

Write capacity:

Target utilization: 70%

Minimum provisioned capacity: 100 units

Maximum provisioned capacity: 40000 units

Result:

A close up of text on a black background

Description automatically generated

We also experimented with much higher provisioned read capacity and write capacity as follows. The cost of 2000 minimum provisioned write units and 100 provisioned read units is about 1000/month. It achieved similar performance as using SQS + lambda, but still cost less than the total cost of RDS+SQS+lambda.

**Provisioned capacity:**

Read capacity units: Table 100

Write capacity units: Table 2769

**Autoscaling:**

Read capacity:

Target utilization: 70%

Minimum provisioned capacity: 100 units

Maximum provisioned capacity: 40000 units

Write capacity:

Target utilization: 70%

Minimum provisioned capacity: 2000 units

Maximum provisioned capacity: 40000 units

**Result:**

**A screenshot of text

Description automatically generated**

**Addition Result:**

**DynamoDB (Provisioned and autoscaling) Result**

At first, we experimented with 4 different provisioned capacity configurations for DynamoDB and test with client (256 threads). We can see that higher minimum capacity gives shorter wall time. These four tests were conducted on the same day. Interestingly, we noticed that the latency and wall time can vary a lot when we test on different days. So, we tried to compare the tests on the same day to rule out the influence of other uncontrolled variables. Tests and results above are on the same day. The tests and results below are on another day.

**Config 1:**

**Provisioned capacity:**

Read capacity units: Table 100

Write capacity units: Table 100

**Autoscaling:**

Read capacity:

Target utilization: 70%

Minimum provisioned capacity: 100 units

Maximum provisioned capacity: 40000 units

Write capacity:

Target utilization: 70%

Minimum provisioned capacity: 100 units

Maximum provisioned capacity: 40000 units

A close up of text on a black background

Description automatically generated

**Config 2:**

**Provisioned capacity:**

Read capacity units: Table 100

Write capacity units: Table 500

**Autoscaling:**

Read capacity:

Target utilization: 70%

Minimum provisioned capacity: 100 units

Maximum provisioned capacity: 40000 units

Write capacity:

Target utilization: 70%

Minimum provisioned capacity: 500 units

Maximum provisioned capacity: 40000 units

**A close up of text on a black background

Description automatically generated**

**Config 3:**

**Provisioned capacity:**

Read capacity units: Table 100

Write capacity units: Table 1000

**Autoscaling:**

Read capacity:

Target utilization: 70%

Minimum provisioned capacity: 100 units

Maximum provisioned capacity: 40000 units

Write capacity:

Target utilization: 70%

Minimum provisioned capacity: 1000 units

Maximum provisioned capacity: 40000 units

**A close up of text on a black background

Description automatically generated**

**Config 4:**

**Provisioned capacity:**

Read capacity units: Table 100

Write capacity units: Table 2769

**Autoscaling:**

Read capacity:

Target utilization: 70%

Minimum provisioned capacity: 100 units

Maximum provisioned capacity: 40000 units

Write capacity:

Target utilization: 70%

Minimum provisioned capacity: 2000 units

Maximum provisioned capacity: 40000 units

A screenshot of text

Description automatically generated