Distributed Ski-Resorts Systems using Spring, MongoDB and AWS

Abishek Arumugam Thiruselvi - 40218896
Department of Electrical and Computer Engineering
Concordia University
Montreal, Canada
abishekarumugam@icloud.com

***Abstract*— To develop a distributed system that enables the digitization of ski resorts. The application uses an RFID ticket reader that collects the data of skiers, resorts and other events and updates them into the MongoDB. The data is then interpreted and analyzed.**

Keywords—AWS, Spring Framework, JUnit, Multi-threading, Elastic Beanstalk, CodePipeline, CodeBuild, SwaggerHub and Postman.

I. INTRODUCTION

A Client/Server application is developed using the spring framework. The application collects and gives data on skier, resort, season, and time.

The application is deployed in AWS Elastic Beanstalk through CI/CD.

Fig. 1. 1. Architecture Design

II. APPLICATION STRUCTURE

1. *pom.xml :*

A Project Object Model is an XML file containing the configuration of the application and its versions.

2. *buildspec.yml :*

The buildspec file contains the instruction to carry out CI/CD.

3. *Source: src/main/java*

   a) *Package: cu.ski.controller*

   This package has the file "SkiController.java". It is the REST Controller, where endpoints and their operations are defined.

   b) *Package: cu.ski.model*

   This package has the classes of skier, resort, and latency records.

   c) *Package: cu. assignment.utils*

   This package has the class ServerlessSpringApplication. It is the start of the spring boot application. Here we have configured a thread pool of 4.

   d) *Package: cu. assignment.repository*

   This package has the interface SkiRepository. It extends the MongoRepository class and has the dependency springboot-data-monogdb.

   e) *Package: cu. assignment.service*

   This package has the class SkiService and PerformanceAnalysis. It implements the interface.

4. *Source: src/test/java*

   1. *Package: cu. ski.utils*

   This package has the JUnit test case RestfulWebServicesApplicationTests. It has test cases to test the client functionality by post concurrently. Which will be explained in the upcoming section.

5. *Source: src/main/resources*

The source folder contains the application properties file. The file contains the details about MongoDB credentials, actuator metrics and OpenAPI documentation.

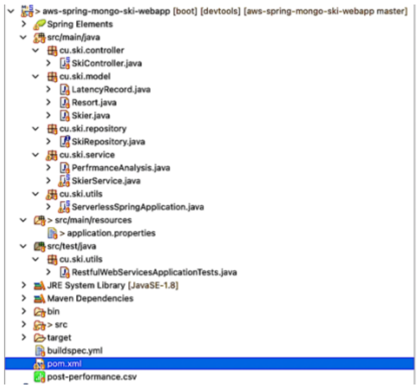

Fig. 2. 1. Project Structure

III. REST CONTROLLER

A Controller handles 4XX and 5XX cases.

1. *GET: /api/ski/resorts*

Lists all the resorts in the database.

2. *POST: /api/ski/create*

Create a skier in the database.

3. *GET: /api/ski/resorts/{resortID}/seasons/{seasonID}/day/{dayID}/skiers*

   Get the number of unique skiers at the resort/season/day.

4. *GET: /api/ski/resorts/{resortID}/seasons*

   Get a list of seasons for the specified resort.

5. *POST: /api/ski/resorts/{resortID}/seasons*

   Add a new season for a resort.

6. *POST: /api/ski/skiers/{resortID}/seasons/{seasonID}/days/{dayID}/skiers/{skierID}*

   Create a new lift ride for the skier. Uses ConcurrentLinkedQueue for generated in a single dedicated thread and made available to the threads that make API calls.

7. *GET: /api/ski/skiers/{resortID}/seasons/{seasonID}/days/{dayID}/skiers/{skierID}*

   Get ski day vertical for a skier.

8. *GET: /api/ski/skiers/{skierID}/vertical*

   Get the total vertical for the skier for specified seasons at the specified resort.

9. *GET: /api/ski/statistics*

   Getting the API performance statistics uses sprig-boot-actuator dependency to get the metrics.

10. *OpenAPI on SwaggerHub*

    - OpenAPI test works only for cloud deployment. Does not work for localhost.
    - Also, the postman was used to test the API functionality.

IV. CLIENT

The client is multi-threaded, which can configure lift rides, create seasons, and get performance analysis. The client uses TestRestTemplate for achieving HTTP Get and POST.

1. *Data Generation:*

The client sends 10K POST requests with the below configuration.

1) skierID - between 1 and 100000
2) resortID - between 1 and 10
3) liftID - between 1 and 40
4) seasonID - 2022
5) dayID - 1
6) time - between 1 and 360

2. *Multithreaded Client:*

To POST 10K data, we use ExecutorService which runs the tasks asynchronously. At the start 32 thread pools were created and each thread sends 1K POST request and shuts down on completion.

A lift ride event is generated by a single thread and is made available for others. This is achieved by ConcurrentLinkedQueue which polls the event and ensures thread safety. The basic HTTP validation is handled in the application.

3. *Client JUnit Testing:*

After the completion of the spring-boot test. The client's characteristics are as follows.

1) All the requests are asserted success.
2) Zero failure request.
3) The total time taken for both the client process and individual posts to complete is calculated. The individual data is stored as a CSV file.
4) From the CSV file, we were able to calculate the min, max, median, and 99 percentiles.

4. *Handling Errors:*

The requests will retry about 5 times with an interval of 500 milliseconds. The basic HTTP validation is done.

V. PROFILING PERFORMANCE

The CSV generated by running the JUnit test is used for calculating the performance of the client.

The location of the class performing the analysis is "src/main/java/cu/ski/service/PerformaneAnalysis."

1. *Mean:*

$$mean = (endtime - starttime)\ /32000$$

2. *Median:*

$$index = \ sort(32000)/2$$

$$median = \ get(index)$$

3. *Throughput:*

$$throughput = \text{total request/wall time}$$

4. *99 Percentile:*

$$index = \ 0.99 * \ 32000$$

$$p99 = get(index)$$

VI. CLOUD DEPLOYMENT

1. *Elastic Beanstalk:*

Create an application with the below configuration.

a) Choose a platform as java.
b) Version 3.4.4, Corretto 17 – Linux 2.

2. *Github:*

Maintain the github repository and generate git tokens if necessary.

3. *CodePipeline:*

Create a buildspec file which has the commands for pre-build and post-build. Webhook the github repository to the code pipeline. Configure the deploy stage as Elastic Beanstalk and click save. The CI/CD starts automatically.

4. *Route 53:*

Used for switching the domain from "skiresort-env.eba-dx7ddhji.us-east-1.elasticbeanstalk.com" to "concordia.abishekarumugam.com".

VII. Instructions to Run the Application

Steps to install, run and test the application locally.

STEP 1. Prerequisite: Git, Java17 or greater, maven 3.9 or higher.

STEP 2. git clone https://github.com/abishekat/aws-spring-mongo-ski-webapp

STEP 3. cd aws-spring-mongo-ski-webapp

STEP 4. mvn spring-boot: run

STEP 5. Use Postman or OpenAPI (yaml: src/resources) to test the application.

STEP 6. While running part 1 and part 2 clients.

a. Delete “post-performance.csv” every time before testing.

b. Uncomment “@test” for the client that needed to be tested.

VIII. Results

Requirements satisfied are given below,

- Server-side APIs that are defined by Swaggerhub are implemented.
- Basic HTTP validation.
- The client is built to perform 32*1000 posts.
- Lift ride events must be generated in a single dedicated thread and be made available to the threads that make API calls.
- Handling errors with 5 times retry before making it a failed request.
- Successful requests are 32000 and unsuccessful requests are 0.
- Profile performance of mean, median, wall time, throughput and p99 is logged in the console.
- Throughput within 5% of Client Part 1

  Throughput = 1600/83 = 19.27.
- Actual Throughput Vs. Little’s Law predictions.

  Expected Throughput: $L=\lambda*W$; no of items = **32000**; average arrival rate = **0.016**; Expected Throughput = 62.5

  Actual Throughput = 65.8.
- When analysing the throughput of clients, those with multithreading exhibit superior results in comparison to those without.
- If the images are in less resolution, please check out the readme file, thanks.

Performance Analysis Output = {

```
"p99": 1055,
"min": 0,
"median": 146,
"max": 1300,
"mean": 483.733875,
"throughput": 65.84362139917695
}
```

Fig. 8.1. Performance Analysis w/o multithread

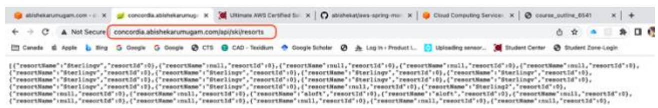

Fig. 8.2. Ski resorts && Domain Configured

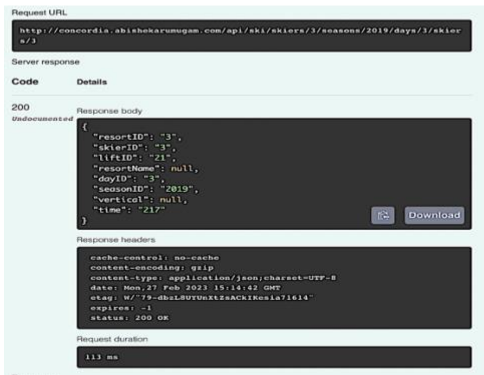

Fig. 8.2. Create a ski activity.

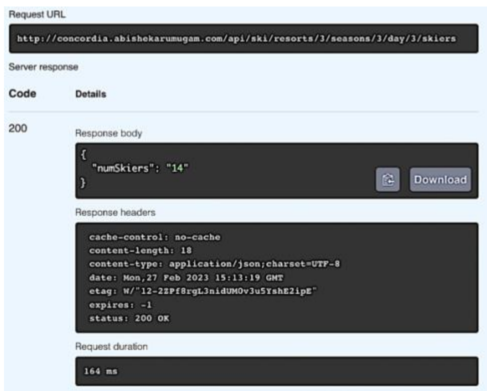

Fig. 8.3. Get: the number of unique skiers at the resort/season/day

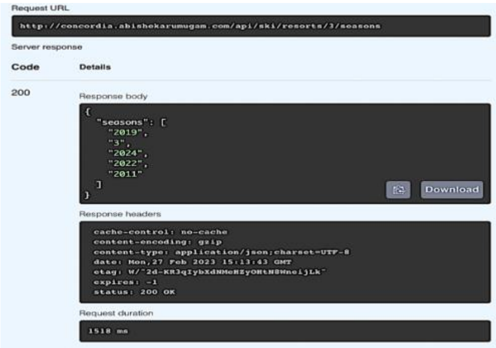

Fig. 8.4. Get: a list of seasons for the specified resort.

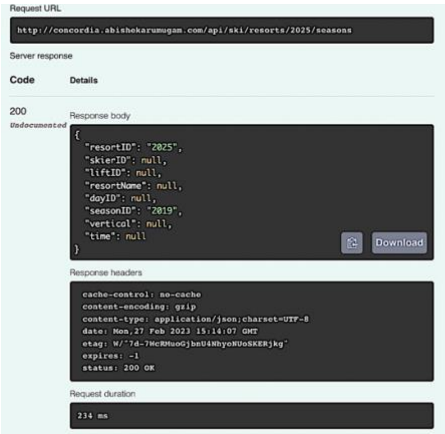

Fig. 8.5. Add a new season for a resort.

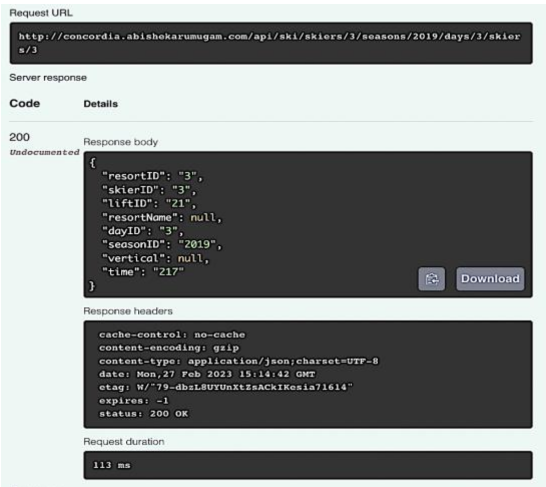

Fig. 8.6. Create a new lift ride for the skier.

Fig. 8.7. Get: ski day vertical for a skier.

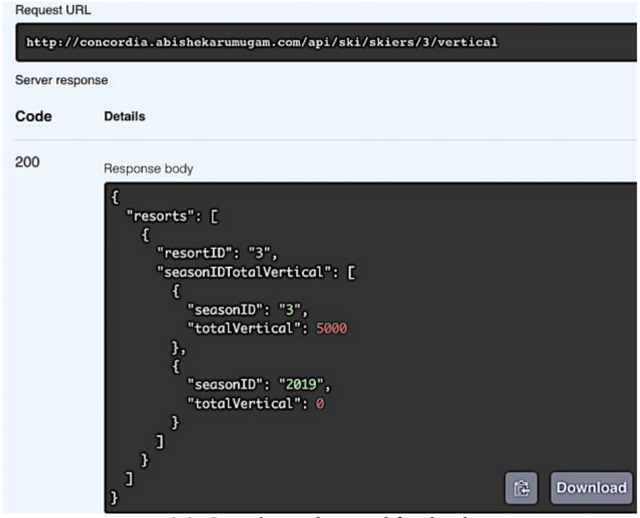

Fig. 8.8. Get: The total vertical for the skier

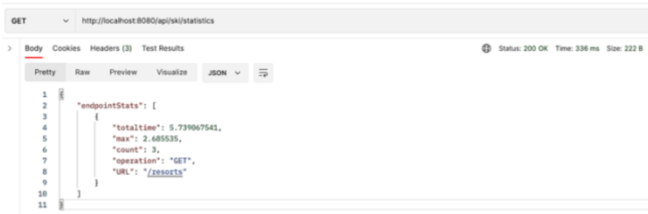

Fig. 8.9. Get: Ski statistics

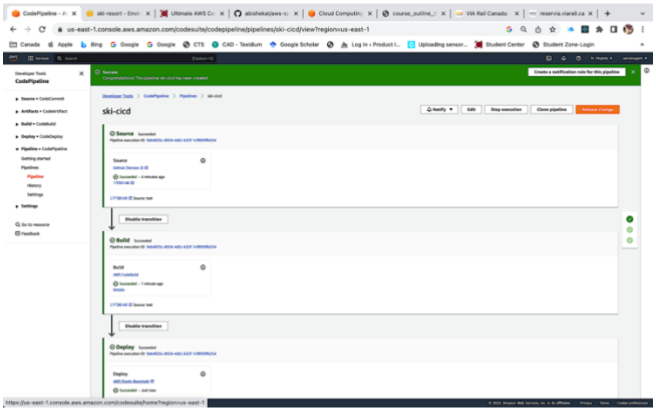

Fig. 8.10. CI-CD Pipeline

REFERENCES

[1] https://moodle.concordia.ca/moodle/mod/lti/view.php?id=3433629

[2] https://medium.com/@contactkumaramit9139/spring-boot-integration-with-mongodb-c24e48f12ba7

[3] https://github.com/youyinnn/distributed_system_jetty_helloworld/blob/main/Oracle%20Cloud%20VM%20Setup.md