DALHOUSIE UNIVERSITY

Faculty of Computer Science

CSCI 5409-Cloud Computing

Final Design Report:

Explore Canada

A tourism application that helps you connect with Canada

March 29, 2020

Submitted By:

Group 13

Akshay Singh	B00814753
Bhagyashree Pandit	B00827861
Keerthi Gowda	B00837999
Yash Shah	B00841980

Table of Contents

List of Figures

Project Requirement:

The project focuses on building a native mobile application and website for tourism Canada. The application helps the user to search places like National parks, major beaches and important cities. The user can book a bus ticket for the desired location by creating an account for them.

The functional requirement of the application are as follows:

1) Mobile application should interact with the server for each query submission from the user.
2) All modules must be loosely coupled in the project and must be tested before integrating and deployed.
3) All the transmitted between the server and client must be secured and stored in the database.
4) User authentication, computing, and all the functionalities must be performed at the server end.

Technical requirement of the application are as follows:

1) AWS Elastic Bean Stalk to employs Auto Scaling and Elastic Load Balancing to scale and balance workloads.
2) AWS EC2 which provides the virtual server in the cloud.
3) AWS Relational Database-RDS
4) Android Studio- To build an android application for the project
5) IDE such as Eclipse/ IntelliJ Idea
6) AWS S3 Storage for storing the images for the application
7) Spring boot Framework for building APIs

1 Introduction

In this project, a web application and mobile application called Explore Canada, supported by cloud computing, is designed and implemented to guide tourists to major places in Canada. It also provides an online travel services that includes ticket booking for the tourist destinations. Canada preserves its stunning mountains, rivers, forests, and beautiful cities which makes Canada on top of the list of tourist destinations. In this project, the application uses an API end points where the user can make a request from the web browser or from the mobile application and gets a response from the server. All the computation, load balancing, storing the data and image, authentication, encryption and decryption, user validation are performed at the server side of the application.

The project was divided into different phases, Firstly, the creation of API, secondly building front end for the web application as well as for the mobile application. Next, connecting the application to cloud services. Finally, the deployment of application on to docker running on AWS Elastic Beanstalk. The development of application also gave an insight of using Git for distributed version control system. The web application was built using Spring Boot framework, which was tested by writing a sample unit test cases in Junit. The project work reinforces skills that are relevant to both individual and group work such as planning and managing time, breaking complex tasks into parts and steps, stronger communication skills, refine understanding through discussion and explanation and delegating roles and responsibility.

1.1 Feasibility

During the initial design of the project, the Explore Canada application was a three-tier architecture which consisted of presentation layer, domain layer, and data layer. The REST endpoints map to the presentation layer, the booking domain model maps to the domain layer, and the database maps to the data source layer.

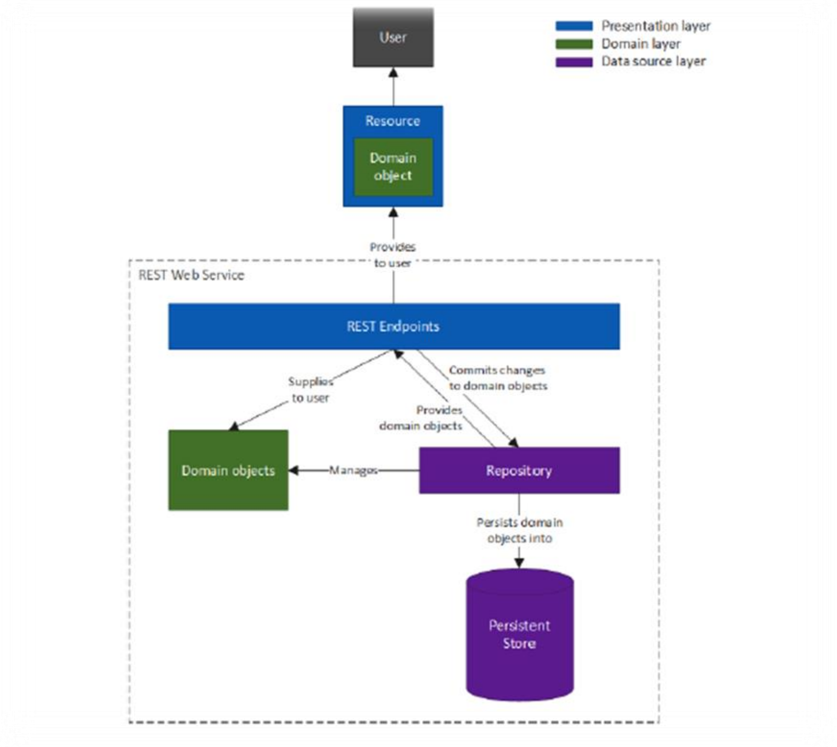


Figure 1: Three Tier Architecture

The project had an architecture to follow but the programming language, Database, framework, and the cloud services to develop the application was not defined when the feasibility report was created. The programming language and the framework to develop the API end point for the application was undecided. There were many database systems which AWS provides, but there was an uncertainty whether to use a document DB or MySQL Database as they had their own advantages. The knowledge about the services provided by AWS was limited to the team members, which took time to figure out the services required to build an application.

1.2 Final Design

Building a microservice-based online travel reservation system obviously entails developing separate microservices for the catering individual functionalities within the system. In this context, explore Canada microservices can be broadly divide into five microservices namely:

- **User Registration service:** This service provides create, read, update, and delete (CRUD) operations on user entity.
- **Search service:** This service provides criteria-based search functionality to look for places and tourism locations.
- **User Login/Authentication service:** This service is responsible for authenticating and authorizing users registered in the system.
- **Booking service:**
- This service is responsible for checking availability of tickets and booking tickets asper the customer requirements.
- **Payment service:** Once the booking is finalized payment service acts as middle layer between the customer's bank and merchant payment gateway.
- **Notification service:** This service is built on top of Amazon SES service; this service is used for sending system generated notifications to the users and administrators of the system.

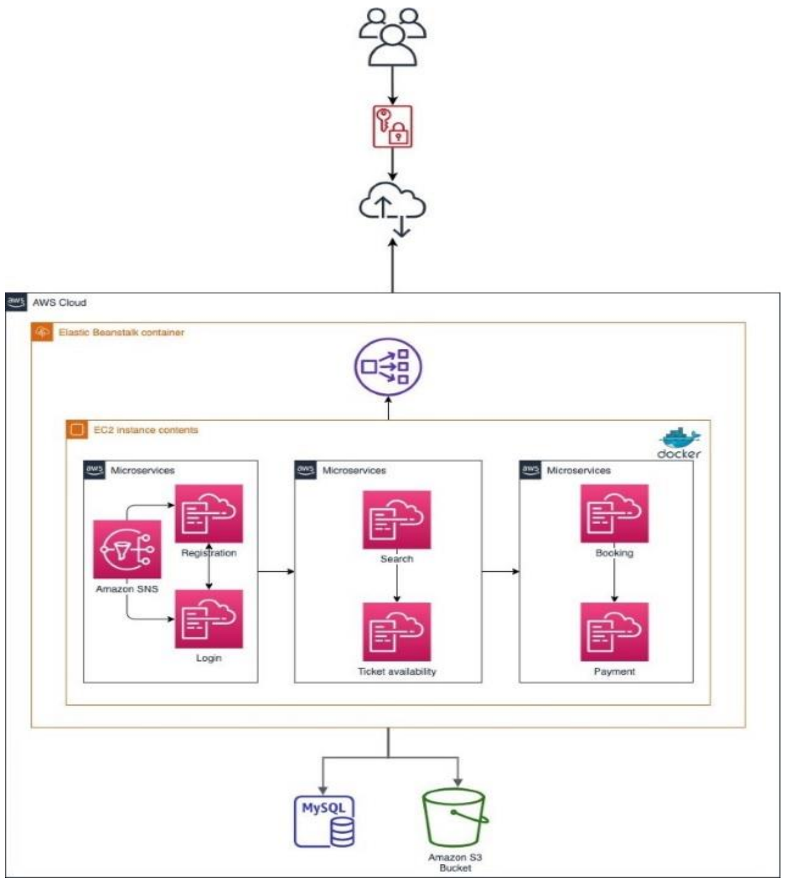


Figure 2: Final Design

The above flowchart shows how each microservice works independently and also communicates with each other. This is the most significant advantage of using microservices—every microservice can be developed, enhanced, and maintained separately, without affecting the others. These services can each have their own layered architecture and database; plus, there are no restrictions in terms of using different technologies/languages to develop the respective services.

1. **User Registration microservice**:

The User Registration microservice will be exposed to the external world using REST endpoints for consumption. The Registration microservice has the following endpoints.

Endpoint	POST /api/register	
Parameters		
Name	Description	
None		
Request		
Property	Type	Description
UserInfo	UserInfo	UserInfo object or Json in the respective format for registration
Response		
Property	Type	Description
UserInfo	UserInfo object	UserInfo object registered in the system with a specific email-Id

2. **Login Microservice:**
   GET /login?email=emailvalue&password=passwordencryptedvalue
3. **Search Microservice:**
   GET /search?searchInfoid=locationIdvalue&locationname=locNameval&locationtype=loctypeval
4. **Payment Microservice**:
   POST /payment parameter = PayamentInfo object
5. **Notification Microservice**:
   POST /notification?email=emailval&message=messageval
6. **Ticket Availability Microservice**:
   GET /ticket/availability?locationId=locidval&startdate=val&enddate=val
7. **Booking Microservice**:
   POST /booking parameter = booking objec

2. Implementation (4 pages with some screenshots)

2.1 Sign in and Register

Sign in, Register and Forgot Password are part of the same microservice. For registration, users are required to provide their first name, last name, email address, and preferred password. This service is combined with two-factor authentication which is discussed below. The details provided by the users are validated, and post that users are registered successfully using the two-factor authentication service. When existing users wish to access their account, they are asked to sign in using the credentials they had provided during registration. The credentials provided by them are authenticated using the data stored in the database.

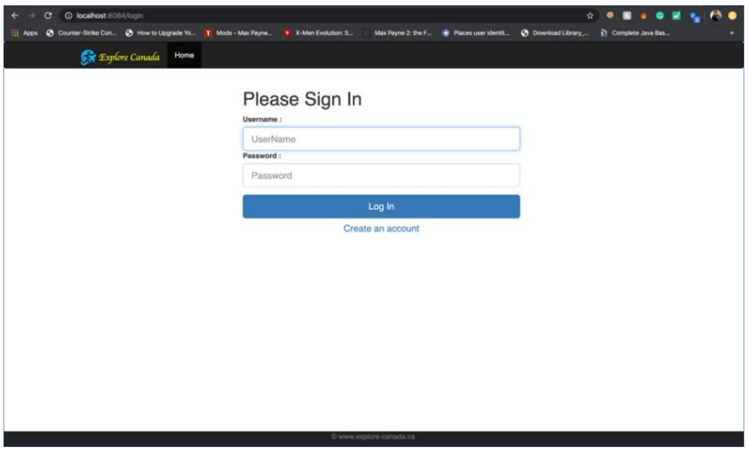


Figure 3: Sign In page

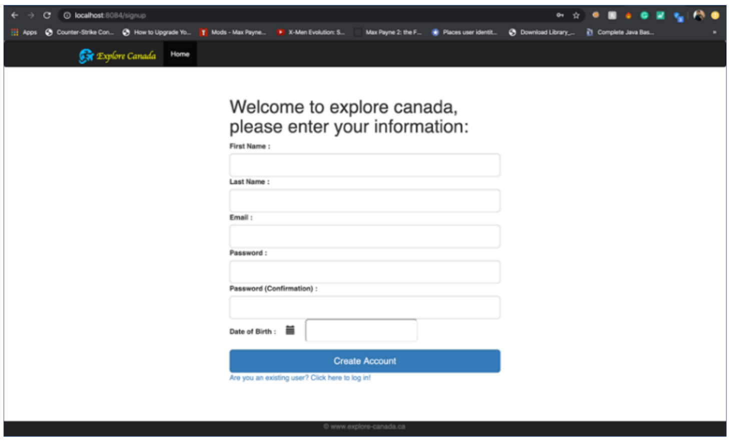


Figure 4:User Registration Page

2.2 Search

One of the requirements for the project is to have search functionality, where a user can search for the tourist places in Canada. Currently, the user can search for major cities, beaches, park and hill station. When the use logs in to the application, the list of places with name, description about the place, image of the place and a booking option to book the ticket for the destination place. All the data are stored in the AWS Relational Database. When the user searches for the places, the API fetches the result for the user and displays it on the screen.

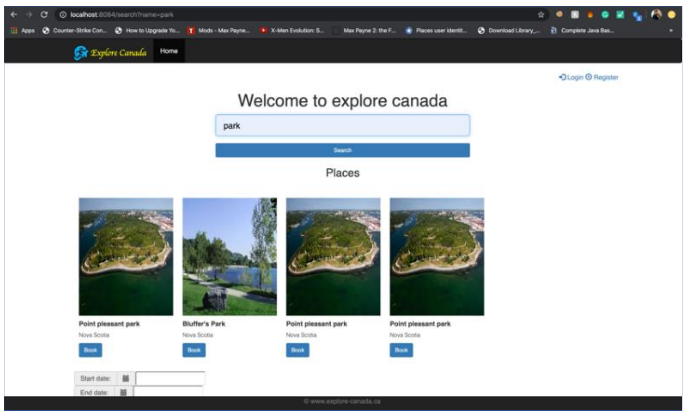


Figure 5: Search

The Search bar in the application lets the user to filter the places based on their interest. The user now able to see the result for the keyword searches by entering cities, parks, hills, and beaches.

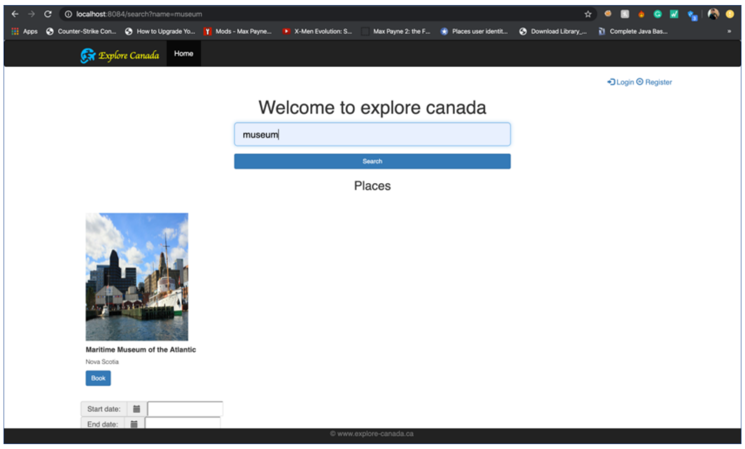


Figure 6: Display for Museum

2.3 Payment

It is a separate module for the project, yet part of the same microservice as ticket booking. Payment Controller has only one method to carry out the payment for the user. This method is called internally and returns true or false to the Ticket Booking module. The payment page on the frontend takes the user card details from the user. These card details are checked with the database table 'card_details'. Generally, this part done by the third-party authentication services. For this project, we have created one table with the dummy data stored in the table, as specified in the requirements. This table is not connected with any other tables in the database, as it is not part of our system. The payment method first checks the expiry month and year of the card, if the card is expired a relevant message is passed to the user. After that, all the card details are checked against the stored database. If all the value matches, the payment method continues. On the other hand, if the details do not match, invalid card message is sent to the user without proceeding. After the successful validation of details, the total amount is checked with the available balance. If the balance is greater than the amount, payment is successful. The total amount is then deducted from the balance column in the card_details table. The method returns true to the ticket booking method if all the tasks are successful.

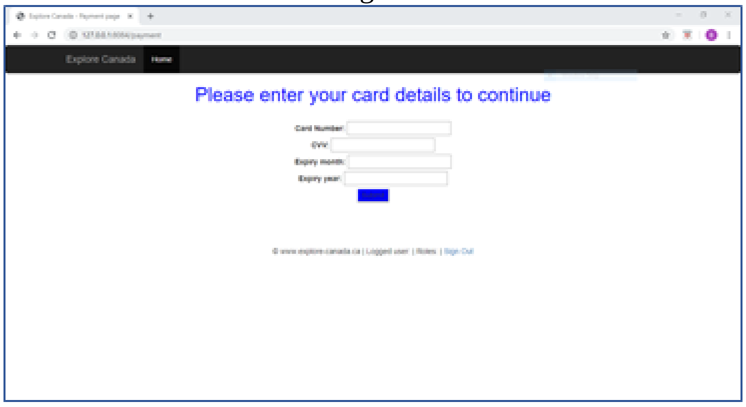
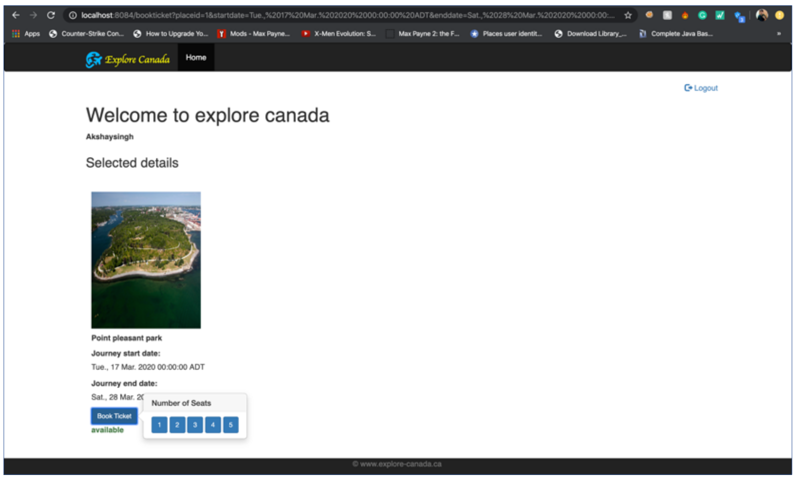

Explore Canada - Payment page
127.0.0.1:8084/payment
Explore Canada
Home
Please enter your card details to continue
Card Number
CVV
Expiry month
Expiry year
© www.explore-canada.ca | Logged user | Roles | Sign Out

Figure 7:Payment Page

2.4 Ticket Booking

Ticket booking and Payment module are part of the same microservice as they both are coupled together. The Model View Controller (MVC) architecture of the microservice permits the low coupling between the modules. The 'TicketInfo' and 'DBConnection' class acts as the model. TicketInfo stores the information and provides getters and setters methods about the ticket.
The DBConnection class creates the connection with the database and passes the connection and

statement objects when called by other classes. ‘TicketController’ and ‘PaymentController’ classes acts as the controllers. TicketController has the methods for finding bus, booking a ticket and displaying the ticket.
User has the facility to book a ticket for a bus ride from a specific source to destination. Before booking a ticket, the system checks if the user is already signed in or not. If the user is not logged in, the control is passed over to the Login microservice. After the successful login, the user will be able to see the available buses running between two spots. Then the user can select the desired bus and select the number of seats as well. The bus information is stored in the database in ‘bus_details’ table, where bus_id is the primary key. The bus will be identified across the system with this bus_id. The required information is fetched from the database using the source and the destination entered by the user. Different rate is provided for the adults and children. Depending on the number of seats booked, a total billing is generated, and the control is handed over to the payment module internally. If the payment is successful, it passes the successful message to the ticket booking module.
User will then be able to view the ticket on the next screen. There is also an option provided to download the ticket. Once the payment is successful, at the backend side, the total number of seats booked will be deducted from the bus_details table. In addition, a new transaction with the bus_id, payment amount and user information will be entered in the transaction table for the company’s record as well as to fetch it again in the future.


localhost:8084/bookticket?placeid=1&startdate=Tue.,%2017%20Mar.%202020%2000:00:00%20ADT&enddate=Sat.,%2028%20Mar.%202020%2000:00:...
Explore Canada
Home
Logout
Book Ticket
Number of Seats
1
2
3
4
5
© www.explore-canada.ca
Welcome to explore canada

Akshaysingh

Selected details

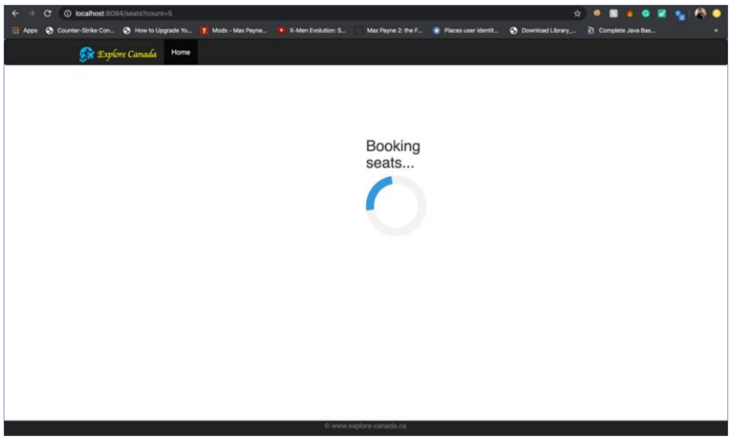
Point pleasant park

Journey start date:
Tue., 17 Mar. 2020 00:00:00 ADT

Journey end date:
Sat., 28 Mar. 20

available

Figure 8: Ticket Booking



Booking seats...

Figure 9: Booking status

2.5 2-Factor Authentication

Two-factor authentication is a separate microservice which works closely with the Sign in and Register service. During registration, after successful validation of users' first name, last name, email address, and preferred password, a verification link is sent to the users on their email address. Only on clicking that link, the user will be allowed to log in and provided with a session token. To log in, after successful authentication of the users' email address and password, the users are sent a six-digit verification code that they must enter on the website to get a session token and successfully log into the system. In the scenario were users forget their password, using two-factor authentication, they are sent a verification link on their email address, and on clicking this link the users will be redirected to a webpage where they will have to enter a new password and log in again using the new password.

2.6 Encryption and Decryption

Securing the data over the transmission channel is one of the major goals when developing the application. In this project, the data has been encrypted and decrypted using AES algorithm. AES algorithm is a symmetric key encryption where the same secret key is used to encrypt and decrypt the messages. In this algorithm, the plain text will be separated into blocks size of 128 bits. Then derive the set of round keys from the cipher key. Third step is to add the initial round key to the starting state array and then perform nine rounds of state manipulation. Finally, copy the final state array out as the encrypted data (ciphertext). The decryption follows the same process but in a reverse order.

2.7 Services

Simple Storage Service (S3): In S3, the images required for the application are stored in the S3 buckets. S3 bucket is similar to file folders, store objects, which consist of data and its descriptive metadata. This service provides the application to store images on the remote server and easy to manage images that need to be displayed to the user.

AWS SNS: It is a notification service provided and catered by Amazon AWS, we have utilized this service for sending notification for two-factor authentication, for sending forgot password generation link, failure messages to administrators, etc.

AWS Elastic Beanstalk: It is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS. The code can simply be uploaded to Elastic Beanstalk and it automatically handles everything starting from deployment, capacity provisioning, load balancing, auto-scaling to application health monitoring.

Amazon Elastic Compute Cloud (Amazon EC2): EC2 is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Application deployed by Elastic Beanstalk is ultimately deployed and hosted on EC2 instance.

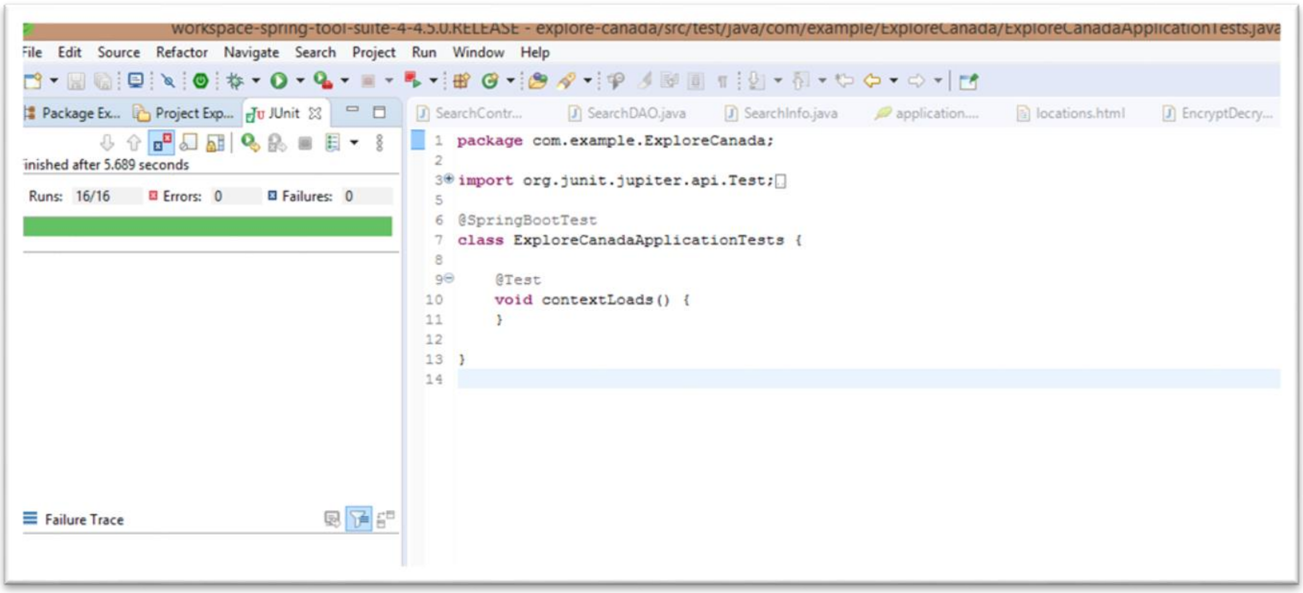
Amazon Relational Database Service (Amazon RDS): RDS makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups. We have used MySQL RD for hosting the database for our app Explore Canada.

3. Milestones

Sr. No.	Milestone	Status
1	Understanding the project requirements and constraints thoroughly, followed by careful study of the system components and the technologies that are to be used	Completed
2	Designing the final architecture of the system and determining its feasibility	Completed
3	Developing all the micro-services, namely sign in and registration, search, two-factor authentication, payment, ticket-booking, and encryption & decryption	Completed
4	Building the website and mobile application which includes all the required functionalities in addition to designing the database on AWS	Completed
5	Containerizing all the services using Docker, deploying them on AWS, and connecting the website and mobile application to the them	Completed
6	Thorough testing and structuring the code, and making the entire system presentable	Not Completed

4. Testing

Unit testing is essential because it is one of the earliest testing efforts performed on the code and the earlier defects are detected and the defects can be fixed at the earlier stage. In this project, Junit is used for unit testing of the application.
 JUnit includes an annotation named @Test, which informs JUnit that the public void form under which it is used is a test case. A small test cases were written to test the User registration and login functionality. The test cases check the email validation and user validation for the application. The below screenshot shows the test result for the user validation.


workspace-spring-tool-suite-4-4.5.0.RELEASE - explore-canada/src/test/java/com/example/ExploreCanada/ExploreCanadaApplicationTests.java
File Edit Source Refactor Navigate Search Project Run Window Help
Package Ex... Project Exp... JUnit
SearchContr... SearchDAO.java SearchInfo.java application... locations.html EncryptDecry...
Failure Trace

```
inished after 5.689 seconds
Runs: 16/16    Errors: 0    Failures: 0
```

```
package com.example.ExploreCanada;

import org.junit.jupiter.api.Test;

@SpringBootTest
class ExploreCanadaApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

Figure 10: Test Result

5. Knowledge Sharing

Akshay Singh	The idea behind adopting spring-boot was to apply the knowledge and best practices, Akshay learned during Advanced SDC class. After working on the project, he is now more confident with spring-boot, microservices, cloud infrastructure, containerization and much more. This project also gave Akshay an opportunity to revise the concepts of mobile computing and android development covered during the previous coursework.
Bhagyashree Pandit	This project was the first exposure for Bhagyashree to the Spring boot framework. She first learned about the technology and then started the implementation. Now, she has a good knowledge about this technology and cloud infrastructure. She was fairly new to the API calling as well in the beginning. She has learned other skills as well from this project, such as working with the team members remotely over the internet and managing multiple tasks together.
Keerthi Gowda	This project helped Keerthi in learning and configuring the different cloud services to the application. With this project, he has also gained knowledge of spring boot framework for calling an API for the application. This project also gave the knowledge on Android studio and the development of Mobile application. The project also helped him in improving non –technical skills such as presentation, team collaboration, report writing.
Yash Shah	This project allowed Yash to explore the domain of cloud computing and learn more about AWS and its services. Coming from hardly knowing anything about cloud, he learned about microservices and their implementation; learned about containerizing using Docker; learned about developing, integrationg, and deploying an application like this on cloud. Apart from this, he improved his presentation, team collaborating, and report-writing skills.

6. Job Role

Akshay Singh	Akshay was assigned the task to implement two microservices for performing user authentication and user registration. He is responsible for developing the respective microservices, login and registration front end database tables and stored procedures and also integration of the front-end application with the backed end microservices API. Akshay has also contributed in designing the back-end API and documentation of all the reports.
Bhagyashree Pandit	Bhagyashree was assigned the task to implement a microservice for the ticket booking and payment. She has developed the backend, frontend and the database for the given microservices. The connection between the frontend and backend i.e. the API call is also configured by her. Apart from the implementation, she has helped in the design phase of the project. Bhagyashree has contributed equally in the documentation part for all the reports.
Keerthi Gowda	Keerthi was assigned the task to implement two microservices for Searching the Places and Encryption and Decryption for the application. He has developed the front end, backend, and the database for the given microservices. He has also worked on configuring the S3 storage for the application. Keerthi has also equally contributed in the designing the architecture, and documentation part for all the reports.
Yash Shah	Yash was assigned the task of implementing a single microservice of two-factor authentication. He is responsible for developing the respective microservice and integrating it with the login and registration service. He worked on the database design of the entire application and contributed in researching about the technologies and how certain microservices can be implemented. Lastly, he equally contributed in all the documentation work, like every other member.

7. Future Work

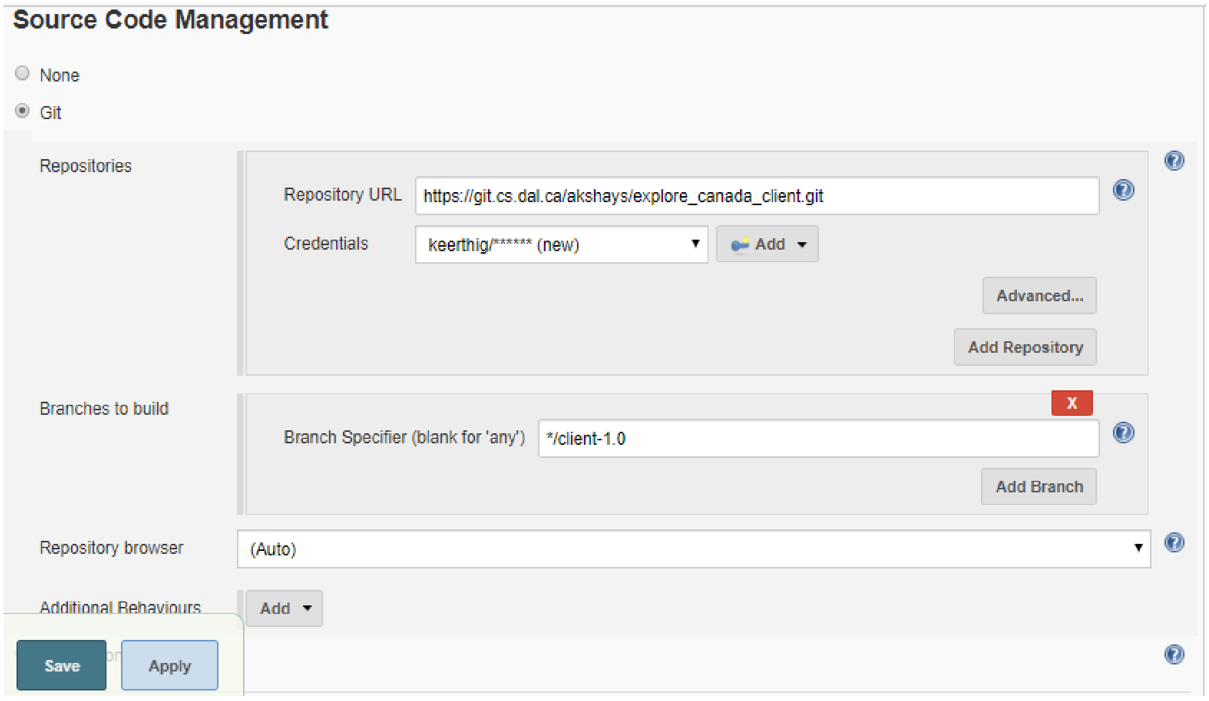
The current model undoubtedly performs all the functionalities in an acceptable way but there is scope for improvement. The user interface (UI) could be more user-friendly and well organized, making the feel of the web and mobile application more pleasing. The concept of the project can be quite beneficial to the users and hence, if a greater number of places were displayed on the application then it would be better. User payment could be made more convenient by using different modes of payment. The cloud services could be used in a better way by proper organization of the microservices; which would include better structuring of the code, minimal amount of calls in the code, and an optimal number of API calls between services to make the entire system efficient. Ticket generation and validation can be implemented in a smoother and hassle-free way.

8. Additional Work

8.1 CI/CD Development

Continuous Integration (CI) refers to the development process where the developers frequently commit their code to a common repository and integrate it with the existing code in that repository. The project CI/ CD is setup using Jenkins. Whenever the code is pushed to the repository, Jenkins checks the build and test for the branch that is committed and then deploy the code to the Elastic Bean stalk.

The development of the project is carried out in Spring Boot Maven project. The Jenkins configuration checks the Maven build and the source code management is pointed to the Git Repository being used. The below screenshot shows the branch and the git repository used.


Source Code Management

None

[✓ Git]

Repositories

Repository URL: https://git.cs.dal.ca/akshays/explore_canada_client.git

Credentials: keerthig/****** (new)

Branches to build

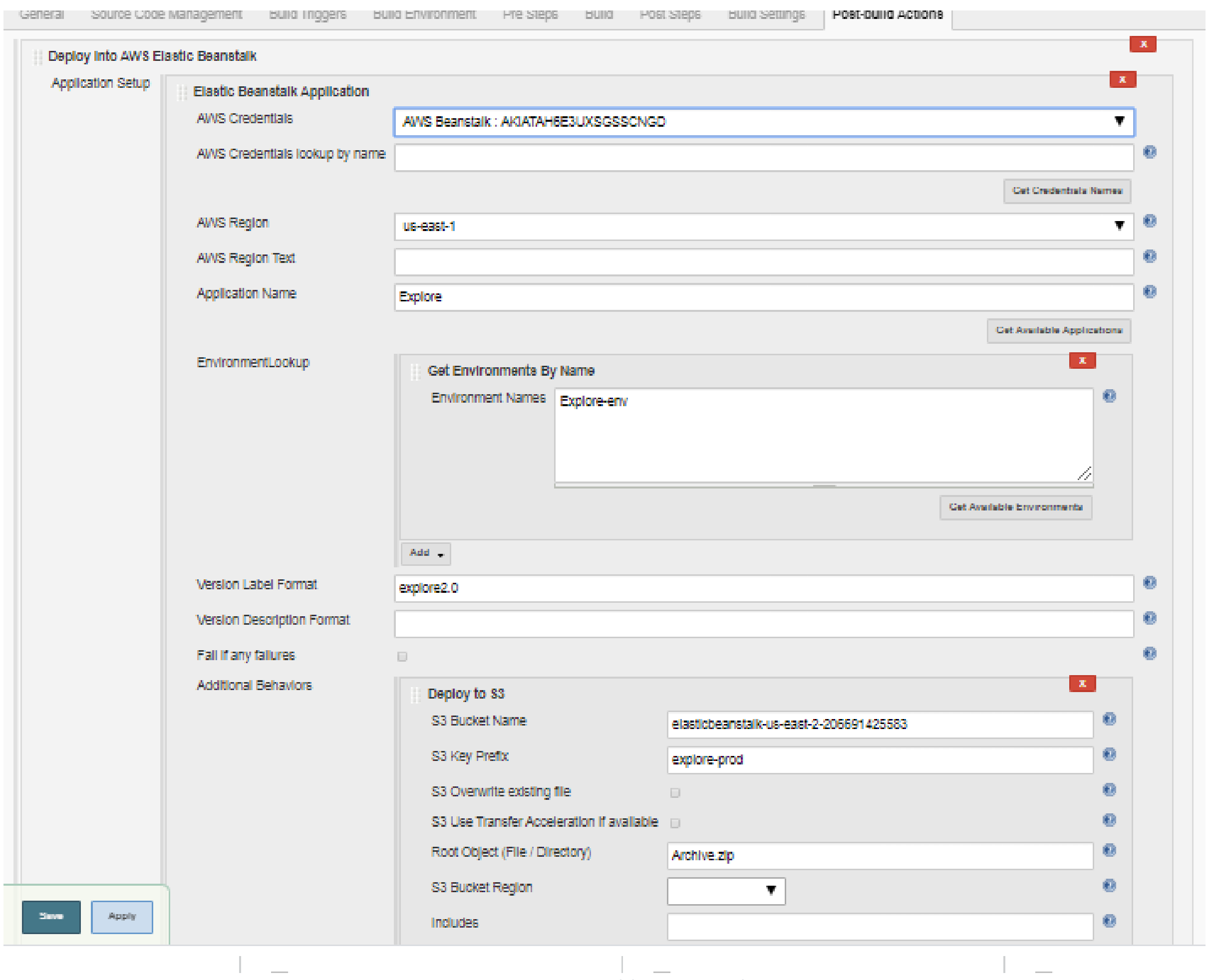
Branch Specifier (blank for 'any'): */client-1.0

Repository browser: (Auto)

Additional Behaviours

Figure 11: Source Code Management

Once the build and test are executed successfully, Jenkins checks for the post-build action specified. Here, the post-build action provided is to deploy the application to elastic bean stalk. The Jenkins pipeline was downloaded, and credentials were provided to connect Jenkins with AWS elastic Bean stalk. The below screenshot shows the configuration for the Jenkins used in the project to achieve continuous deployment to elastic beanstalk.



Deploy into AWS Elastic Beanstalk

Application Setup

Elastic Beanstalk Application

AWS Credentials: AWS Beanstalk : AKIATAH6E3UXSGSSCNGD

AWS Credentials lookup by name:

AWS Region: us-east-1

AWS Region Text:

Application Name: Explore

EnvironmentLookup

Get Environments By Name

Environment Names: Explore-env

Version Label Format: explore2.0

Version Description Format:

Fail if any failures

Additional Behaviors

Deploy to S3

S3 Bucket Name: elasticbeanstalk-us-east-2-206691425583

S3 Key Prefix: explore-prod

S3 Overwrite existing file

S3 Use Transfer Acceleration if available

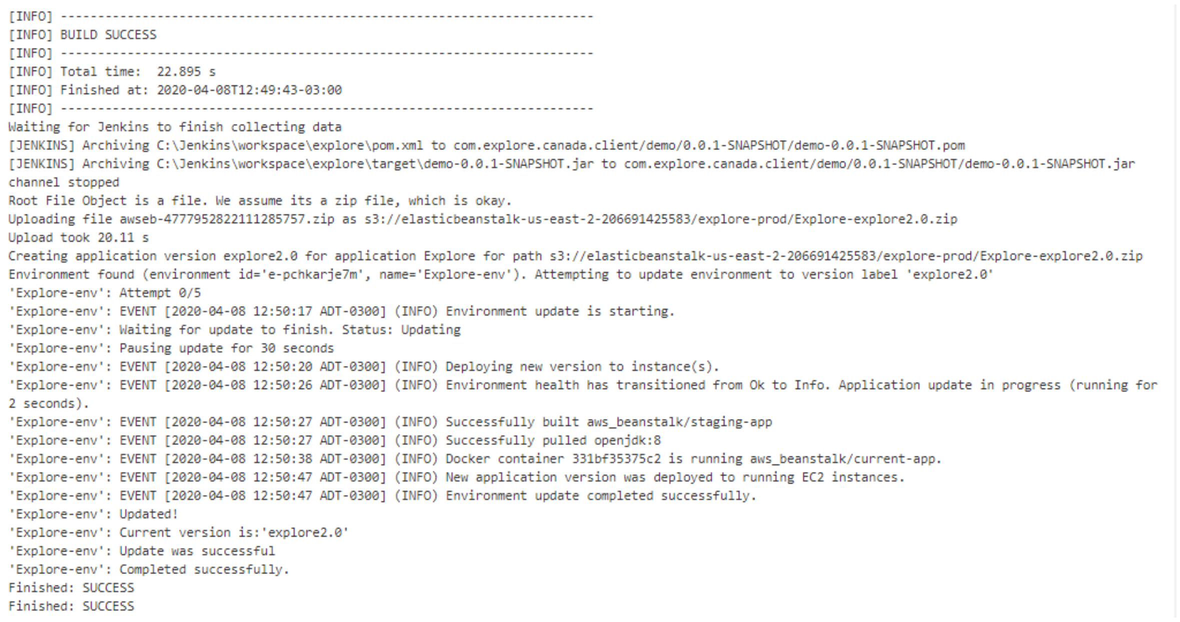
Root Object (File / Directory): Archive.zip

S3 Bucket Region:

Includes:

Figure 12: Post-Build Action Jenkins

The Application was successfully deployed to AWS Elastic Beanstalk and below is the sample of the log from the Jenkins output console.

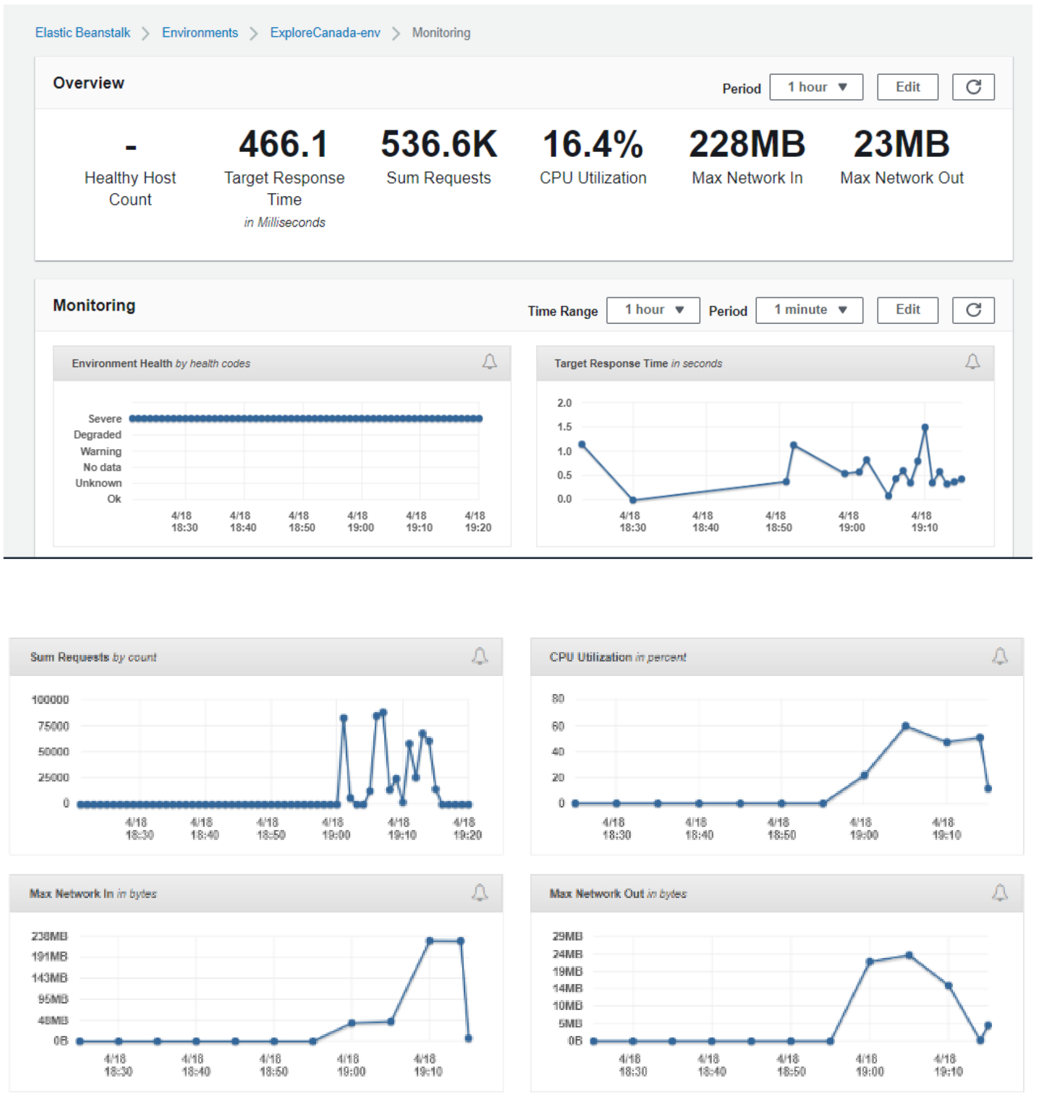


```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  22.895 s
[INFO] Finished at: 2020-04-08T12:49:43-03:00
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving C:\Jenkins\workspace\explore\pom.xml to com.explore.canada.client/demo/0.0.1-SNAPSHOT/demo-0.0.1-SNAPSHOT.pom
[JENKINS] Archiving C:\Jenkins\workspace\explore\target\demo-0.0.1-SNAPSHOT.jar to com.explore.canada.client/demo/0.0.1-SNAPSHOT/demo-0.0.1-SNAPSHOT.jar
channel stopped
Root File Object is a file. We assume its a zip file, which is okay.
Uploading file awseb-4777952822111285757.zip as s3://elasticbeanstalk-us-east-2-206691425583/explore-prod/Explore-explore2.0.zip
Upload took 20.11 s
Creating application version explore2.0 for application Explore for path s3://elasticbeanstalk-us-east-2-206691425583/explore-prod/Explore-explore2.0.zip
Environment found (environment id='e-pchkarje7m', name='Explore-env'). Attempting to update environment to version label 'explore2.0'
'Explore-env': Attempt 0/5
'Explore-env': EVENT [2020-04-08 12:50:17 ADT-0300] (INFO) Environment update is starting.
'Explore-env': Waiting for update to finish. Status: Updating
'Explore-env': Pausing update for 30 seconds
'Explore-env': EVENT [2020-04-08 12:50:20 ADT-0300] (INFO) Deploying new version to instance(s).
'Explore-env': EVENT [2020-04-08 12:50:26 ADT-0300] (INFO) Environment health has transitioned from Ok to Info. Application update in progress (running for
2 seconds).
'Explore-env': EVENT [2020-04-08 12:50:27 ADT-0300] (INFO) Successfully built aws_beanstalk/staging-app
'Explore-env': EVENT [2020-04-08 12:50:27 ADT-0300] (INFO) Successfully pulled openjdk:8
'Explore-env': EVENT [2020-04-08 12:50:38 ADT-0300] (INFO) Docker container 331bf35375c2 is running aws_beanstalk/current-app.
'Explore-env': EVENT [2020-04-08 12:50:47 ADT-0300] (INFO) New application version was deployed to running EC2 instances.
'Explore-env': EVENT [2020-04-08 12:50:47 ADT-0300] (INFO) Environment update completed successfully.
'Explore-env': Updated!
'Explore-env': Current version is:'explore2.0'
'Explore-env': Update was successful
'Explore-env': Completed successfully.
Finished: SUCCESS
Finished: SUCCESS
```

Figure 13: Jenkins Build Status

8.2 Auto Scaling and Load Balancing:

To check Auto Scaling and Load Balancing for the application, JMeter is used to send the http request for the application. The below screenshot shows the CPU utilization, requests for the application. The elastic Beanstalk utilizes two Amazon CloudWatch alarms to activate scaling operations. The default triggers scale when the average outbound network traffic from every instance is greater than 6 MB or lesser than 2 MB around an interval of five minutes. Once the threshold is reached, it creates a new EC2 instance which will balance the http request by distributing it to multiple EC2 environment.

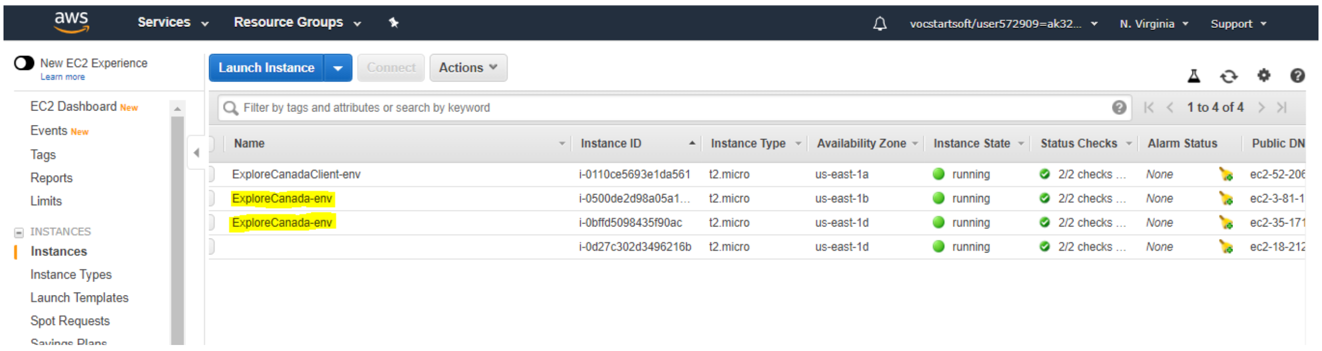

Elastic Beanstalk > Environments > ExploreCanada-env > Monitoring

Overview — Period: 1 hour

- Healthy Host Count: -
- Target Response Time (in Milliseconds): 466.1
- Sum Requests: 536.6K
- CPU Utilization: 16.4%
- Max Network In: 228MB
- Max Network Out: 23MB

Monitoring — Time Range: 1 hour; Period: 1 minute

Figure 14: Auto Scaling



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DN
ExploreCanadaClient-env	i-0110ce5693e1da561	t2.micro	us-east-1a	running	2/2 checks ...	None	ec2-52-206
ExploreCanada-env	i-0500de2d98a05a1...	t2.micro	us-east-1b	running	2/2 checks ...	None	ec2-3-81-1
ExploreCanada-env	i-0bffd5098435f90ac	t2.micro	us-east-1d	running	2/2 checks ...	None	ec2-35-171
	i-0d27c302d3496216b	t2.micro	us-east-1d	running	2/2 checks ...	None	ec2-18-212

Figure 15: Load Balancer

9. References

[1] T. Lelek, “Leveraging Spring Boot Integration Toolkit,” REST Applications with Spring, March 2019.
[2] “Spring Boot CRUD Operations - javatpoint,” www.javatpoint.com. [Online]. Available: https://www.javatpoint.com/spring-boot-crud-operations. [Accessed: 30-Mar-2020].
[3] “Building REST services with Spring,” Spring. [Online]. Available: https://spring.io/guides/tutorials/rest/. [Accessed: 04-Mar-2020].
[4] “JUnit Annotations Tutorial with Example,” Guru99. [Online]. Available: https://www.guru99.com/junit-annotations-api.html#6. [Accessed: 10-Mar-2020].
[5] F. North, “Getting started,” Amazon, 1998. [Online]. Available: https://aws.amazon.com/getting-started/tutorials/deploy-docker-containers/. [Accessed: 14-Mar-2020].
[6] “S3,” *Amazon*, 2002. [Online]. Available: https://aws.amazon.com/s3/getting-started/?nc=sn&loc=5. [Accessed: 16-Mar-2020].
[7] R. Treichler and C. Hardmeier, “Schlagwortnormdatei Schweiz für allgemeine öffentliche Bibliotheken: SNS,” *Amazon*, 2005. [Online]. Available: https://aws.amazon.com/sns/getting-started/. [Accessed: 17-Mar-2020].
[8] F. North, “Getting started,” *Amazon*, 1998. [Online]. Available: https://aws.amazon.com/getting-started/tutorials/create-mysql-db/. [Accessed: 18-Mar-2020].