Cloud Application Services – Technical Report

Deep Roychoudhury
Masters in Cloud Computing
National College of Ireland
Dublin, Ireland
x19109130@student.ncirl.ie

***Abstract*—This document is a technical report of the Cloud Application Services project that has been developed. The application that has been developed is a sports application using various public cloud platforms based upon the needs. The project has been made using two applications. One is a Web-service application and the other is a client application to access webservices from various sources. The report gives the technicalities used in the project based upon the background review and also provides the implementation details of the project.**

Keywords—AWS, Azure, Heroku, Object Storage and infrastructure in cloud, Java with Tomcat, Ruby on Rails, Azure Service Bus

I. INTRODUCTION (*HEADING 1*)

The business requirement are ever changing time to time. With the business requirements changing frequently, solving the business problems at one place becomes difficult. Hence, there is a need to split the tasks into several pieces such as, data can be kept in one place whereas the processing of data can be done on the other place. This means that applications should be created at different platforms, operating systems, different communication protocols and different development languages. But, this creates a question on how these different platforms interact with each other. The simple answer to this question becomes Web Services. A web service has the ability to interact with other applications which are written in different environments and networks. The web services provides standardized web protocols such as HTTP and HTTPS to communicate for data through Extensible Markup Language(XML) or Javascript Object Notation(JSON). There are various types of web services used such as XML-RPC(Remote Procedure Call), UDDI(Universal Description, Discovery and Integration), SOAP and REST. The most common web services that are used widely are SOAP(Simple Object Access Protocol) and REST(Representational State Transfer). REST web service makes use of HTTP methods such as GET,POST,PUT or DELETE for CRUD-oriented services. RESTful web service is well known for its lightweight, human readable, is stateless and easier to build properties. On the other hand, SOAP uses structured data in the form of XML and uses SMTP(Simple Mail Transfer Protocol) or HTTP for transmission. SOAP uses WSDL(Web services Description Languages) to interact with other applications. SOAP has three major characteristics which are extensibility, neutrality and independence. SOAP web service is well known for its easy consumable ability and security. Some of the factors which forces developers to lean towards RESTful web services rather than using SOAP web services are the fact that SOAP web services are difficult to setup and coding a SOAP is harder. There are two types of authentication such as basic http authentication and OAuth authentication. The basic authentication is usually carried out using username and password which are encoded in Base64 format. The OAuth authentication takes place using the token system from client to server and server to client. The OAuth authentication checks for the authenticity of the user trying to access the service.

In the context of development and from the developers point of view PaaS(Platform as a Service) is always the best option to deploy the applications. Deploying an application to IaaS(Infrastructure as a Service) requires a lot of resources especially DevOps resource. There are various public platforms provided by various cloud providers such as Amazon, Microsoft and SalesForce. Amazon provides EBS(Elastic Beanstalk) and AWS Lambda(Serverless) platforms as PaaS. Microsoft provides Azure App Service and Salesforce provides Heroku as PaaS. The deployment of application to the respective PaaS provider depends upon the organizational requirements. In order to deploy application from various platforms the cloud providers provide sdk services to communicate with the cloud services. Comparing AWS EBS and Lambda, Lambda provides a serverless architecture which basically means that the infrastructure and platform is handled by the AWS itself. The developers just have to upload the jar or a zip file to the S3 bucket and use it to create a Lambda function. The Lambda function then, can be configured for various requests such as GET,POST and PUT. Apart from that AWS RDS a relational database by AWS, provides many platforms to create database such as MySQL, PostgreSQL, Oracle SQL, etc. Apart from database and platform cloud providers also provide the queue services to manage the data sent and received by consumers and providers. Examples for the messaging services are AWS SQS service and Azure Service Bus. There are various patterns which are followed for messaging. But the most popular one is the FIFO(First In First Out) pattern.

The sports project developed has been made up of two applications. One is a client application which is the sportsclientapp and the other is the services application which is SportsREST. The client side application has been made using Ruby on Rails and the srevices application has been made using Java Maven project using Jersey. The services application developed is a Restful webservice application. The client application utilizes the webservice as well as other webservices to provide details.

The section 2 gives the background research for the project. Section 3 is about the software architecture used to develop the sports application. The fourth section talks about the implementation details of the applications developed as well as the findings during the design and development of the applications. Finally, Testing and Conclusion section gives the overview of the project developed.

II. BACKGROUND RESEARCH

A. Object Storage Infrastructure in Cloud

The number of available cloud providers are ever increasing and the demand for processing and storage power are also

increasing. The cloud object storage is a format for storing unstructured data in the cloud.

The object based storage service offered by Amazon is the Amazon S3. The AWS S3 has been built to store the data and retrieve it from anyplace in the world. It is based upon simple web services interfaces and hence also uses web services for its integration. S3 comprises of buckets, storage objects and the S3 REST API to implement with any code. The S3 objects can be hosted in any region to minimize the latency as well as for cost optimization. A bucket created in AWS S3 has unique name for each bucket. The storage capacity of the AWS S3 is five Terrabytes(5TB). There are various languages which are supported by S3 such as Ruby,.NET, Java, C# etc. S3 provides 4 storage classes:

- STANDARD
- STANDARD_IA
- GLACIER
- RRS

Similar to the Amazon S3 Microsoft provides Azure Blob Storage. It is a scalable object based storage solution. The azure storage account is divided into containers which are know as blobs. Blobs are further classified into 3 types block, append and page. Binary files or the text files are stored in blocks. Each block has 100MB and can contain upto 50000 blocks. Logging and Input/Output operations can be done on append blob and contains 4MB space for each append blob. For read and write store page blobs can be used. Page blob contains 512-byte storage. Data saved in Azure can be retrieved from cloud and locally. The blobs or containers created are private objects which can be modified alter and are the valid DNS names. Methods such as download, upload delete etc. are available to the developers. The Azure Blob storage has much simple methods to access data.

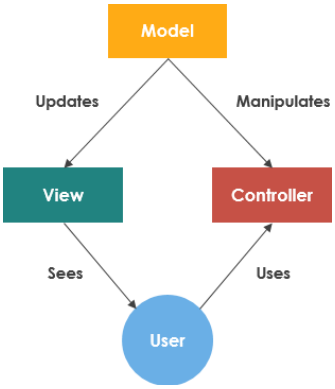
Google offers similar solutions as that of AWS and Azure. The data are stored in projects and buckets stores the objects. Only one operation is performed with a bucket from a project every 2 seconds. Objects are created using different programming languages. The cloud storage offered by Google are of two types, json and xml interface.

Object storage is elastic, flexible and has the ability of unlimited data growth. The main advantage of cloud object storage is that it is highly distributed which makes it reliable in the case of disaster or hardware failure. Also, object storage are much more cheaper than the traditional storage.

B. *Messaging Infrastructure in Cloud*

The applications are splitted into smaller parts which are independent of each other. This helps in easy development, deployment and easy maintenance of the application. To resolve the problem coordination and communication of these smaller parts message queues are used. The message queues improve the performance, reliability and scalability of the application. Message queues provides asynchronous communication. In short the users can send the requests in queues without halting for a response from the server. The queues are fault tolerant, for example, if one queue fails the other takes over the task to complete and hence, there is 100% availability. There are various features of message queue such as pull or push in which pull means continuous querying and push means a message is available, schedule or delay, atleast once delivery, exactly once delivery, FIFO, etc. Amazon uses AWS SQS (Simple Queue Service) and Azure makes use of Azure Service Bus and Azure Storage Queues.

C. *Model-View-Controller*



The Model View Controller design pattern has been used for the development of sports services application. The user interacts using the view and sends the request to controller which then manipulates and sends the data from model back to user view.

D. *Adapter Design Pattern*

The web services uses adapter design pattern to interact with each other. In the adapter design pattern the services becomes the adapter and the applications just hit the adapter to get the details from other sources. Two different application with different platforms can easily interact using the adapter pattern.

III. SOLUTION ARCHITECTURE

The sports application project has been built making use of AWS, Azure and Heroku as the cloud provider.

A. *The Cloud Deployment Model*

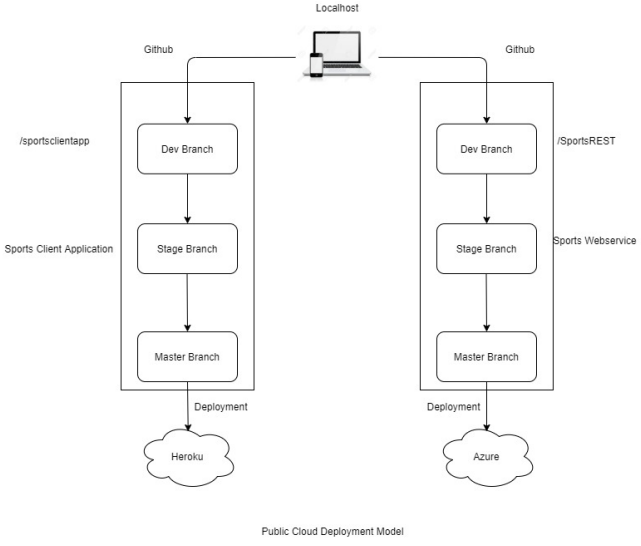


Figure 1

The architecture of the code deployment model that has been implemented is shown in the Figure 1

The steps that are involved in the architecture are :

1) Developing both client application(Ruby on Rails) and services application(Java) locally.

2) Storing the codes in Github with 3 branches(dev,stage and master).

3) Deploying the local code to dev branch. Testing the code with blackbox testing.

4) When the test in dev passes the code is deployed to stage branch.

5) Testing in stage branch takes place. If the code fails an hotfix is will be done onto the stage branch.

6) If the test passes then, the code is ready to be pushed to master.

7) The code will be deployed to master and will be tested again.

8) If the test is correct, the code will be deployed to respective cloud provider for hosting.

In the sports application, once the client application has the correct version it will be deployed to the Heroku platform. In the case of SportsREST the application will be deployed to Azure Web App.

B. The AWS RDS Connection

In the Figure 2, the code functionality is being displayed.

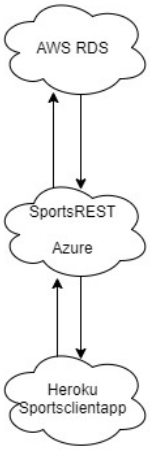


Figure 2

In this figure the client application deployed in Heroku sends a request to the services application(SportsREST). The services application sends the request to AWS RDS Postgresql and retrieves the data. The result is sent back to services and then to client application.

C. The Messaging Service

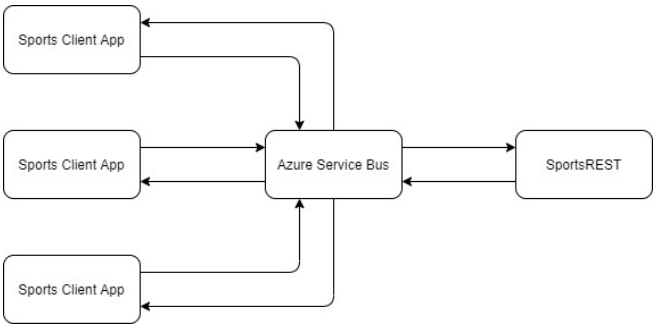


Figure 3

Figure 3 shows the messaging service connection with the client application and services application. The queue messaging service will be implemented to provide FIFO(First In First Out) functionality to the project. When there are multiple request to the services application the queue in the Azure Service Bus will apply the FIFO system to the requests and will send the request to the services application. The data retrieved will be sent back to the client with the same pattern.

IV. IMPLEMENTATION

The implementation of the sports client application and the sports services are done using two different cloud providers. The sports client has been deployed to the Heroku platform whereas the services application has been deployed to the Azure Web App Service.

Services Application Composition – Java 8, Tomcat 9.0, Jersey, Maven, Azure Web App

Client Application Composition – Ruby on Rails, Heroku

Figure 4 shows the actual implementation and the interaction both the applications make.

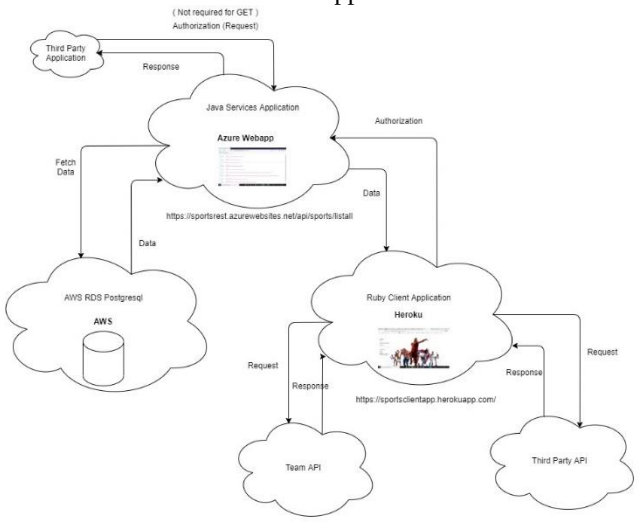


Figure 4

The sports client has been made using Ruby on Rails. The sports client uses the below web services and fetches the details :

1) https://sportsrest.azurewebsites.net/api/sports/listall

This web service is the webservice of SportsREST application. This webservice displays different kinds of sports played across different countries. This is a GET request and is publicly available to everyone.

2) https://sportsrest.azurewebsites.net/api/sports/{id}

This GET web service is used to fetch the details of a specific country with its id and is only accessible to admin. This service will need basic authentication for accessing the details.

3) https://sportsrest.azurewebsites.net/api/sports/{country}

Similar to {id} this is a GET service to access the details using the country name but is only accessible to the admin using basic authentication.

4) https://sportsrest.azurewebsites.net/api/sports/add

This service is a POST request of SportsREST service application which only accessible to the admin. This service is used to add new country and sports played on that country. Basic authentication has been used.

5) https://hotelapi20200806072002.azurewebsites.net/invoice/getitems?invoiceNumber=11&customerName=shraddha&productName=P1&productName=P2&productPrice=2&productPrice=32&totalAmt=43&balanceAmt=5&transactionType=cash&siteName=dsf

This is a team GET service which is used to generate invoice of the items bought in the sports client application.

6) https://www.thesportsdb.com/api/v1/json/1/all_sports.php

This is a GET third party service which is used to fetch details of various sports played around the world. This can be viewed when an user clicks the sports dictionary tab.

The sports client app makes use of facebook omni-auth gem which is used for authorizing the users through social media login. The omniauth uses OAuth to check the authenticity of the device used. It is a token based approach.

The Services application on the other hand uses Basic authentication for checking if the user is authorized to access the service. Below code shows the implementation of POST request and the authentication mechanism used to determine if the user is eligible.

```
@POST
        @Path("/add")
        @Consumes("text/plain,text/html,application/octet-stream")
        @Produces("application/json")
        public Response addData(InputStream inputdata, @HeaderParam("authorization") String authString) {
                if(!isUserAuthenticated(authString))
        {
                return Response.status(401).entity("Invalid Credentials").build();
        }
                StringBuilder builder = new StringBuilder();
                SportsObject sportsobject = new SportsObject();
                try {
                        BufferedReader in = new BufferedReader(new InputStreamReader(inputdata));
                        String line = null;

                while((line=in.readLine())!=null) {

                builder.append(line);
                        }
                }catch(Exception e) {
                        System.out.println("Error");
                }
                System.out.println("Country :"+builder.toString());
                JsonObject jsonobj = new JsonParser().parse(builder.toString()).getAsJsonObject();
                String country = jsonobj.get("country").getAsString();
                String sportsplayed = jsonobj.get("sportsplayed").getAsString();
                System.out.println("Country : "+country+" and sports played are : "+sportsplayed);
                sportsobject.setCountry(country);
                sportsobject.setSportsplayed(sportsplayed);
                PostgresqlJdbcConnection mydata = new PostgresqlJdbcConnection();
                mydata.addData(country,sportsplayed);
                return Response.status(201).entity(inputdata).build();
        }
```

```
private boolean isUserAuthenticated(String authString){
```

```
        String decodedAuth = "";
        // Header is in the format
"Basic 5tyc0uiDat4"
        // We need to extract data
before decoding it back to
original string
        String[] authParts =
authString.split("\\s+");
        String authInfo =
authParts[1];
        // Decode the data back to
original string
        byte[] bytes = null;

        bytes =
Base64.getDecoder().decode(authI
nfo);

        decodedAuth = new
String(bytes);

System.out.println(decodedAuth);
        String[]
decodedAuthsplit=decodedAuth.spl
it(":");
        String username =
decodedAuthsplit[0];
        String password =
decodedAuthsplit[1];

if(username.equals("admin")&&pas
sword.equals("secret")) {
        return true;
        }
        return false;
}
```

The Services application makes connection to the AWS RDS Postgresql to retrieve the data. In order to allow RDS to make connection to the other application IAM Role(postgresports) has been set with policy of AmazonRDSFullAccess and AmazonVPCFullAccess. The inbound rules of RDS VPC security group has been edited for application access from azure.

Findings during the design and development of the application : -

1) Ruby on Rails is much easier when compared to Java. Although both follow object-oriented programming principles.

2) Ruby on Rails provide fewer lines of code and is easy to debug and fix the bugs when error occurs. This results in speedy development.

On the other hand, Java code needs to compiled and interpreted which may also result into failure.

3) Java code execution is much faster than that of Ruby. On the other hand, Ruby offers much flexibility than Java.

4) Deployment of Java in Azure Web App requires less configuration and takes less time when compared to deployment on Amazon AWS Elastic Beanstalk(EBS).

5) Deployment of Java project in EBS sometimes results in failure due to version mismatch in AWS and actual Maven project. For Example, deploying a Java Restful webservice with Tomcat 9.0 and java 8 into AWS EBS resulted in an error due to version mismatch of Dynamic Web Module. The Dynamic Web Module of the Project in development was 4.0, whereas that available in AWS EBS was 3.0. In order to change the version to a lower version, org.eclipse.wst.common.project.facet.core.xml file in .settings folder needed to be updated. While in the case of Azure Web App, the updated version of Dynamic Web Module was present.

6) Deployment in AWS Elastic Beanstalk also fails sometimes due to misconfiguration in VPC and the security group.

7) Deployment using AWS CLI using students account may result into failure due to the updation of Access Key Id and Access Key Secret on time to time basis. Also, there are less privileges for deployment using AWS CLI such as region creation during the environment setup.

8) Deploying the Java Webservice using Tomcat is easy in AWS Lambda with comparison to deployment in AWS Elastic Beanstalk. There are much less configuration and the code can be either deployed using the AWS Toolkit for Eclipse or through AWS CLI. After the deployment, a minimal configuration is required in the Lambda function to set the different kinds of request such as GET,POST and PUT. After this configuration, an additional configuration of VPC is neeeded and an IAM Role needs to be created for letting the AWS Function to access the data from RDS Database.

9) Deploying the application in azure Web App was done easily using the Azure Toolkit in eclipse with authorization. To connect to AWS Relational Database Service, an IAM Role was created for the access. And hence, the deployment was much more easier than the rest.

10) Deploying a Ruby Application also needed a more of configuration in Azure and AWS both and Heroku turns out to be a faster deployment platform for Ruby on Rails Application. The application is easily deployed using the heroku cli with only three to four commands.

11) Development of a Client application turns out to be faster when compared to deploying the client application in Java. Since, Ruby uses embedded ruby for its view, it is much more secure when compared to javascript, jsf and html programming. Also, Ruby has many inbuilt libraries to help the developer to code efficiently in very less time.

12) During the development of webservice in order to utilize the SSL for Java Webservice, Tomcat had to be configured. As a result, keystore id had to be generated and

configured on the Tomcat web server. In case of Ruby on Rails, there was only one line which was added to the development.rb file in config/environments folder.

V. Functional & Non-Functional Requirements

A. *Functional Requirements* –

- Whenever the admin enters an incorrect data the admin must be notified.
- The normal users should be able to buy from the sports kit and will get an invoice for the same.
- Only the available products must be displayed in sportskit.
- The admin only must have the authority to update the kits and the sports.
- The admin only should have the authority to make a POST request to the Web-service. All other users should be notified as unauthorized access.
- The application should provide basic knowledge about sports.

B. *Non-Functional Requirements*–

- The application should be scalable and elastic.
- The application should provide a secure connection to data.
- Users should get an easy navigation website to search for the sports kits they are looking for.
- The application should be cost effective with minimum Total Cost of Ownership.
- The application should be portable.
- Should follow SSL protocols.

VI. TESTING

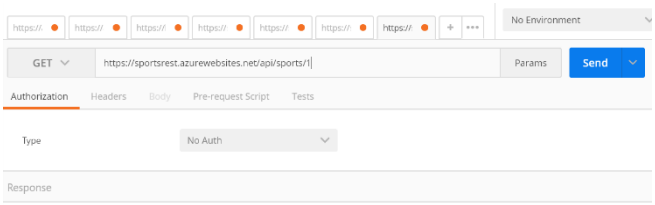
The basic authentication technique has been done using the postman application to test the services application.

A. *Checking for Unauthorized access*

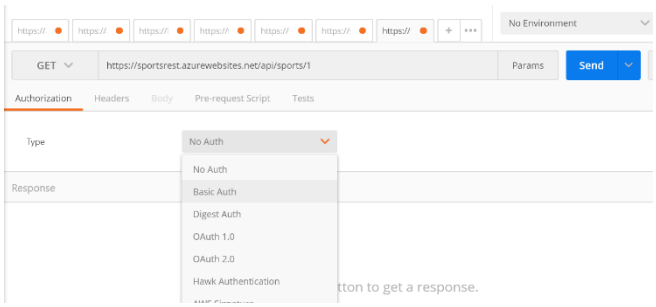
Steps involved in testing the unauthorized access for the service.

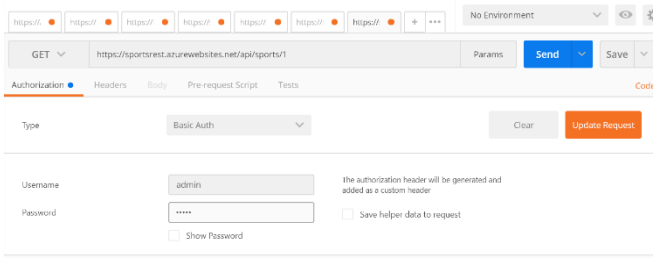
GET
https://sportsrest.azurewebsites.net/api/sports/country/Ireland

1) Enter the GET request and the url in postman.

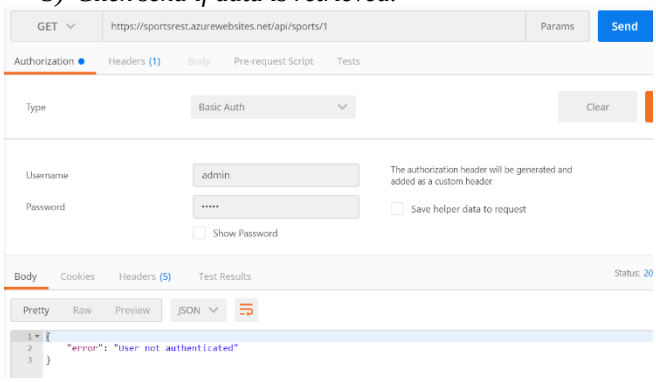


2) Enter any random credentials

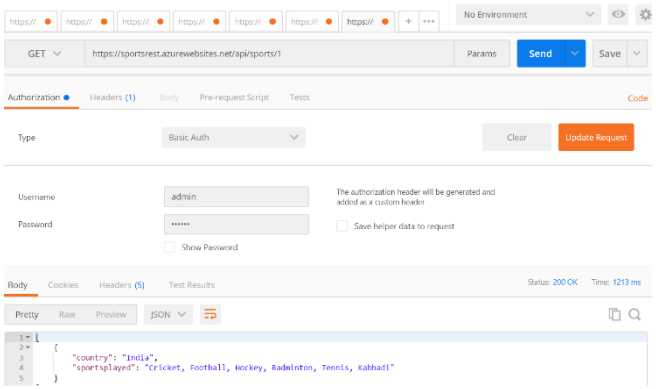




3) Click send if data is retrieved.



Hence, we get an unauthorized access. Whenever an admin logs in through his credential, he will get success as shown below.



The same principle applies for POST Request.

For the client application black box testing has been performed to check if normal user is able to update the data in RDS Postgresql.

VII. Conclusion

The sports service application shows how the client application interacts with the services application. By using a message queue system with azure service bus, the

application implements the Messaging Oriented Middleware(MOM) integration pattern. The basic authentication provides only admin to access the services application. The OAuth authentication is provided by omniauth where the user is checked for authenticity by token based system.

VIII. Github and Deployed App Links

A. **Github Link for SportsREST Webservice** - https://github.com/DeepRoychoudhury/SportsREST
B. **Deployed Application url** - https://sportsrest.azurewebsites.net/api/sports + (/listall (GET), /add (POST), /{id} (GET), /country/{country} (GET))
C. **Github Link for sportsclientapp Application** - https://github.com/DeepRoychoudhury/sportsclientapp
D. **Deployed Application url** - https://sportsclientapp.herokuapp.com/

References

[1] https://devcenter.heroku.com/articles/getting-started-with-ruby

[2] https://azuredevopslabs.com/labs/vstsextend/tomcat/

[3] https://docs.aws.amazon.com/lambda/latest/dg/services-rds-tutorial.html

[4]https://medium.com/@chinnatiptaemkaeo/integrate-omniauth-facebook-to-rails-5-1389d760d92a

[5] https://www.javatpoint.com

[6] https://moodle.ncirl.ie/course/view.php?id=2375

[7]https://www.java2novice.com/restful-web-services/http-basic-authentication/

[8]https://medium.com/paris-rb/deploying-your-rails-postgresql-app-on-microsoft-azure-180f8a9fab47

[9] https://medium.com/@jameshamann/deploying-rails-5-app-using-elastic-beanstalk-and-postgresql-8ca19bc7648a

[10] https://dzone.com/articles/aws-lambda-with-mysql-rds-and-api-gateway