Dublin Business School

excellence through learning

Networks and Systems Administration Assignment

Web application in Cloud

Module Title: Networks and Systems Administration (B9IS121)

Module Leader: Mr. Obinna Izima

Student Name: Manik Mahashabde (10518579)

Table of Contents

List of Diagrams

List of Figures

1: Introduction:

This is an artifact which combines web application development with Amazon Web Service cloud terminologies along with the usage of containerization using **Docker** and Simple Mail Transfer Protocol(**SMTP**). The project is about a Java web application deployed in Amazon Web Service cloud. An Ubuntu container of Docker acts as a client and gives **RestAPI** webservice call to the end point in the application server which is hosted in Linux Server of **AWS cloud** and gets the response from it. User can also hit endpoint and receive a mail from Java Spring boot in his Gmail. Let us understand the project in more details in Technical section:

1.1 Aim of the Project

The aim of the project is combining different terminologies such as web development, cloud, containers, Secure Shell, mail configuration and creating a network artifact.

1.2 Scope and approach of the project

The scope of the project is using advance technologies and expanding a simple web application towards multitude of possibilities using cloud, docker, STMP, SSH etc. Once a web application with SMTP configuration was developed locally and tested. Cloud platform was introduced for its deployment for easy storage and scalability. After successfully getting the response from the application in cloud through local client **PostMan**, Docker was introduced into the project. First a container of ubuntu image was created and CURL(client URL) was installed which acted as a client. In terms of cost involvement, a free tier account of Amazon Web Service was created which gives 750 hours of monthly use of micro instance. Private key provided by Amazon was used to create .ppk extension putty key using PuttyGen. Docker Toolbox was installed and images were pulled from DockerHub then containers were made of the images. More about the approach is covered in technical description section.

1.3 Assumptions

Assumption for my artifact are that a person should have knowledge about web application development using Java and its frameworks such as spring boot. Then a person should also have knowledge about MySQL database , Hibernate JPA and RestAPI webservices . Also, about HTTP methods such as GET and POST. Working with Apache Tomcat server. How spring boot projects are created using Maven. Also, about configuring SMTP in spring boot.

The knowledge about AWS platform and services such as Elastic Beanstalk, Elastic Compute Cloud, Relational Database Service. He/she should also be able to configure AWS CLI in windows and setting up environment variables. After this knowledge about deployment of web application on cloud. Performing SSH connection from putty to server hosted in cloud. Some basic UNIX commands knowledge is also preferable. Knowledge about handling cloud instances in AWS console and checking logs.

Last but not the least knowledge about Docker and containers. How to install and configure Docker Toolbox in Windows 8.1. Knowledge about docker images, containers, pulling and pushing images from and to Docker Hub (image repository of Docker). Creating containers from images, creating your own image.

2: Background:

2.1 Background of Java web application development:

It is covered in detail in the appendix section **A.1**

2.2 Background of AWS and Linux:

In earlier times big monolithic applications were deployed in production environments hosted in data centers which resulted in having a greater number of resources for company. They were required to maintain servers and other hardware. Hence a need of cloud computing was felt.

Cloud Computing: Cloud computing is the on-demand delivery of computational power, database, storage, applications, and other IT resources via the internet with pay as per usage pricing. Whether you are using it to run applications that share photos to millions of mobile users or to support business critical operations, a cloud services platform provides rapid access to flexible and low-cost IT resources*(Amazon Web Services, Inc., n.d.)*.

Amazon Web Service: AWS provides cloud computing platform which provides various services such as computing, deployment of web application, database management, analytics etc.

Following are the AWS services used in the project:

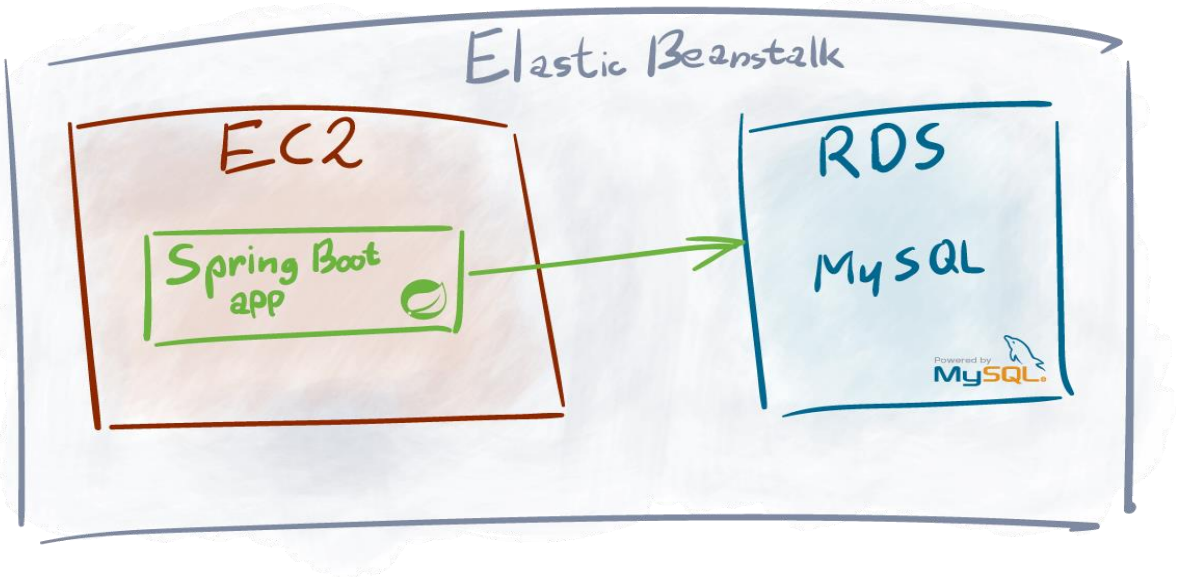

Diagram 2.1 AWS Elastic Beanstalk Components

AWS Elastic Beanstalk(EB) : AWS Elastic Beanstalk 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. It gives us an environment with EC2 instances and optional database (*Docs.aws.amazon.com, n.d.*).

AWS Elastic Compute Cloud(EC2) : Amazon Elastic Compute Cloud (Amazon 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. In this project I have deployed my application in Linux server(*Amazon Web Services, Inc., n.d.).*

AWS Relational Database Service(RDS) : Amazon RDS provides cost-efficient and scalable relational database capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups*(Us-west-2.console.aws.amazon.com, n.d.).*

SSH(Secure Shell) : SSH is a software package that enables secure system administration and file transfers over insecure networks. The SSH protocol uses encryption to secure the connection between a client and a server. All user authentication, commands, output, and file transfers are encrypted to protect against attacks in the network*(Ssh.com, n.d.)*.

Putty : PuTTY is a free implementation of SSH and Telnet for Windows and Unix platforms through which we can establish SSH connection.

PuttyGen : PuTTYgen is a key generator tool for creating pairs of public and private SSH keys. It is used to create keys so that secure SSH connection can be established.

Linux CentOS : The CentOS is a free software used as an operating system for Linux based machine having package manager as yum

Unix Cent OS commands : Linux command for Cent OS are required for performing useful operations such as finding about active processes running in the server, logs path of the application and other uses.

2.3 Background of Docker and Containers

In earlier times if you are using windows and wanted to work on different OS let's say Linux. You would have to install it in your system in different partition of hard disk. It resulted in occupying more space and processing power of the system. In another case if you are using windows 10 and wanted to test an application which runs only on Windows 7 it used to create compatibility issue. Hence a need of using virtual machine was felt in which we could install Linux or any other application in VM itself and use it on top of our host OS.

Virtual machines runs on a hypervisor (responsible for running virtual machines) and include its own guest operating system. This increased the size of the virtual machines significantly, makes setting up virtual machines more complex and requires more resources to run each virtual machine. Thereafter arrived a need to create a lightweight solution where Docker came to rescue.

Docker Hub : Docker Hub is a service provided by Docker for finding and sharing container images with your team*(Docker Documentation, n.d.).*

Docker Installation in Win 8.1: As Docker for Windows is for Windows 10 hence I installed docker toolbox because I use Windows 8.1*(Docker Documentation, n.d.).*

Why Docker : Docker provides containers which are light weight, independent applications which runs on top of Host Operating System with the minimum configuration files required to run properly. Unlike Virtual machine which requires complete independent Operating system environment for the application. Hence VM takes more space for operations. Containers are containing the binaries, libraries, and the application itself. Containers do not contain a guest operating system which ensures that containers are lightweight*(Medium, n.d.)*.

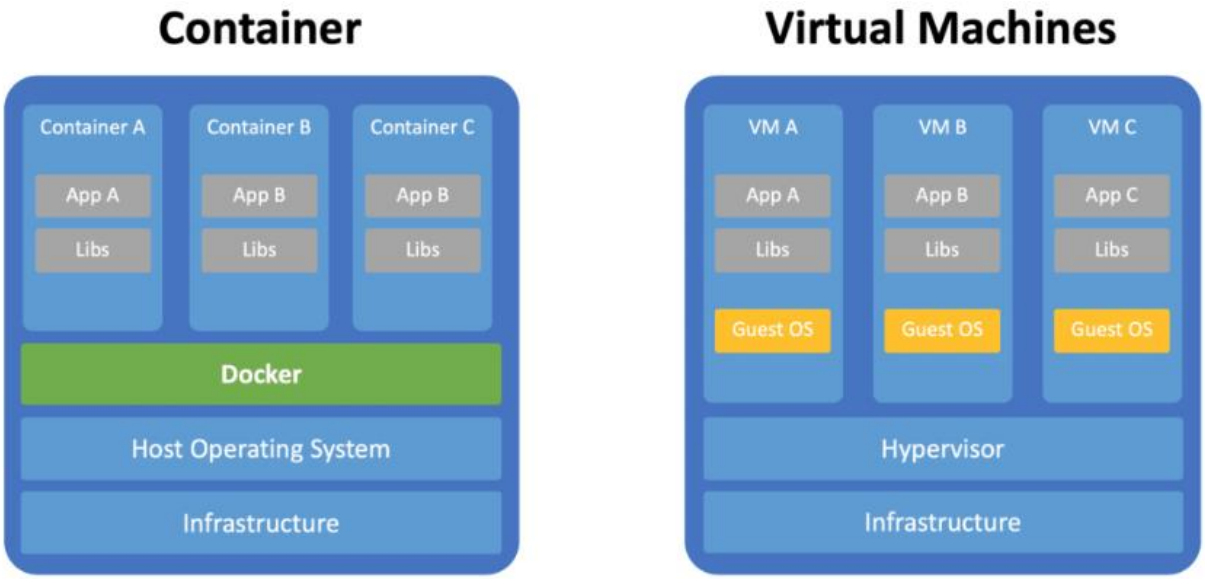

Diagram 2.2 Docker vs Virtual Machine *(Medium, n.d.)*

Docker Image and Container: A Docker image is containing everything needed to run an application as a container. A Docker container is a runtime instance of an image. From one image you can create multiple containers (all running the sample application) on multiple Docker platform *(Medium, n.d.)*.

Docker commands : It is required to pull an image from docker hub, creating new container of an image, starting an existing container, executing a command in running container, listing s container, making new image from a container and many more*(Medium, n.d.)*.

2.4 Benefits of the technologies used in the project

It is covered in detail in the appendix section **A.2**

3: Technical description and Demonstration:

Following are the details of software, technology and tools used along with version.

Software/Technology/Tool	Version
Spring tool Suite Compiler	3.9.9
JRE Library	JavaSE-1.8
Spring Boot	2.2
PostMan Client	5.5.4
MySQL Workbench	6.3
Putty	0.73
Elastic Beanstalk CLI	3.15
Amazon CentOS Linux OS	2018.03
Docker toolbox	19.03.1

The first step for starting the project was developing a java web application. It is covered in detail in the appendix section A.2

Once the Java application development was completed I configured **SMTP** (Simple Mail Transfer Protocol) in spring boot by adding spring-boot-starter-mail library in my **pom.xml** which imports the Java Mails library. I did it by creating an endpoint in spring boot application as below

http://localhost:5000/api/C1/Cars/email

Along with this I also gave my Gmail id the permission to allow access from less secure app and creating a password for spring boot which I configured in application.properties file as shown in appendix. Once everything is done I got the below mail to my Gmail as soon as I hit the mail API

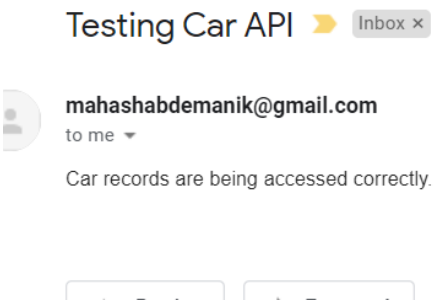

Fig 3.1: Mail from Spring Boot

Now after all these steps my application is successfully developed and it is running up and fine in local machine. The next step was to bundle up the application and create a JAR file which was deployed in Amazon Web Service cloud.

For creating a JAR file, I used Windows command prompt by first going to the directory where my spring boot project is present in the system. Then I used below command

$ mvn package.

The maven bundles up all the files and dependencies into a JAR file as shown below.

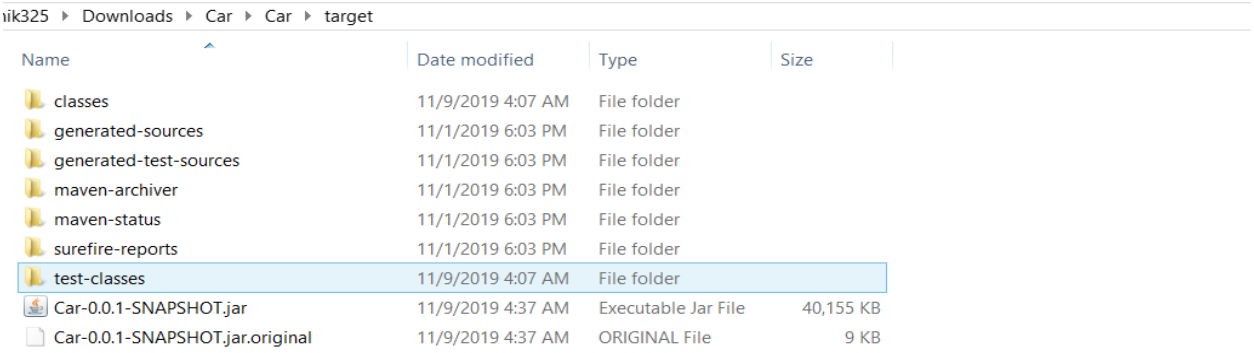

Fig 3.2: JAR file created by maven

First step for deploying my spring boot application was installing important plugins and configuring them in windows environment variables. Below plugins are important for the project.

```
C:\Users\Manik325>aws --version
aws-cli/1.16.270 Python/3.6.0 Windows/8 botocore/1.13.6

C:\Users\Manik325>eb --version
EB CLI 3.15.3 (Python 3.7.3)

C:\Users\Manik325>python --version
Python 3.7.3

C:\Users\Manik325>pip --version
pip 19.3.1 from c:\program files\python37\lib\site-packages\pip (python 3.7)
```

Fig 3.3: AWS and python plugins

Then I gave below command to instantiate Elastic Beanstalk(EB) environment which internally makes Elastic Computer Cloud(EC2) server or instance(termed by AWS) in which my java application was deployed.

$ eb init

Then select the region for my application I selected default region which is us-west-2.

After this I entered my AWS_SECRET_ID and AWS_SECRET_KEY provided by amazon while signing up.

Then I mentioned AWS config file path which is shown is next page. After this I selected the option to create new environment. My environment name is **Manik-AWS-SpringBoot-dev** and my application name is **Manik-AWS-SpringBoot.** The platform I selected was Java 8 and I allowed the SSH connection in options and created the new key-pair for my SSH(Secure Shell). I used **putty** for my SSH connection. Next option I did was selected database instance of **RDS** to store my Car records. Following command was used for this.

$ eb create –single –database

Note:- --Single tell EB to create only one database instance.

After this I selected a username and password of my RDS DB which is same as I selected in application.properties file in figure 2.

Once all the things are selected I gave next in command prompt.

My configuration file can be seen in figure 9 on next page.

```
branch-defaults:
  default:
    environment: Manik-AWS-SpringBoot-dev
deploy:
  artifact: C:\Users\Manik325\Downloads\Car\Car\target/Car-0.0.1-SNAPSHOT.jar
global:
  application_name: Manik-AWS-SpringBoot
  branch: null
  default_ec2_keyname: Manik-AWS-SpringBoot-Key
  default_platform: Java 8
  default_region: us-west-2
  include_git_submodules: true
  instance_profile: null
  platform_name: null
  platform_version: null
  profile: eb-cli
  repository: null
  sc: null
  workspace_type: Application
```

Fig 3.4: AWS Config.yml file

As it can be seen I gave my JAR file path in the config file. After this I gave the below command

$ eb setenv SPRING_DATASOURCE_URL=jdbc:mysql://aa1xpxyf4a8ds90.czpmd6pmuapj.us-west-2.rds.amazonaws.com:3306/ebdb SPRING_DATASOURCE_USERNAME=root SPRING_DATASOURCE_PASSWORD=****

I am setting the database environment of RDS to my EB environment which I can see by giving the command eb console to open the website. Finally, I used the command

$ eb setenv SERVER_PORT=5000

This is the default port where EB applications runs. Please note in figure 2 in appendix I used server.port=5000 to tell spring boot to use same port as by Elastic beanstalk.

Now after doing all the configurations. I gave the command **$eb deploy** to deploy the application in EB. After 5-10 minutes my application was deployed and I used eb status command to check my status

```
C:\Users\Manik325>eb status
Environment details for: Manik-AWS-SpringBoot-dev
  Application name: Manik-AWS-SpringBoot
  Region: us-west-2
  Deployed Version: app-191109_044136
  Environment ID: e-mmexfs6ynz
  Platform: arn:aws:elasticbeanstalk:us-west-2::platform/Java 8 running on 64bit
 Amazon Linux/2.10.0
  Tier: WebServer-Standard-1.0
  CNAME: Manik-AWS-SpringBoot-dev.us-west-2.elasticbeanstalk.com
  Updated: 2019-11-10 16:14:51.913000+00:00
  Status: Ready
  Health: Green
```

Fig 3.5: Checking application status through console

After giving eb console command I could see below status*(Amazon Web Services, n.d.)*

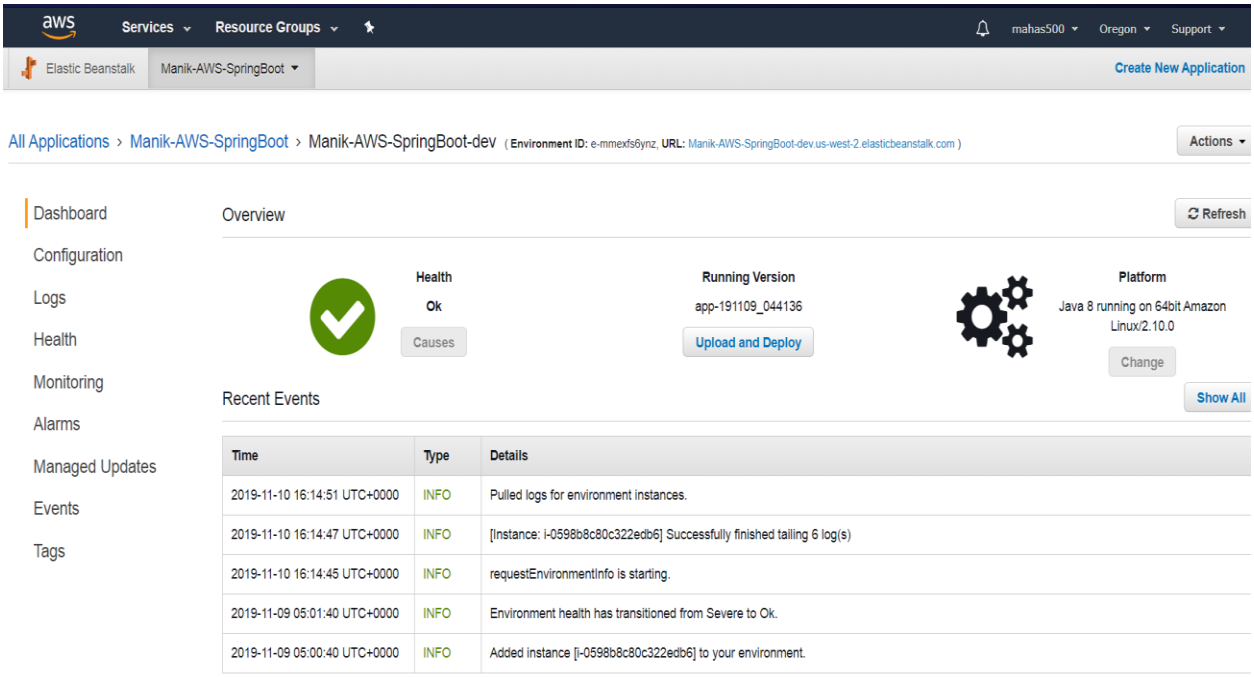

Fig 3.6: Console of the running application

Configuration which I gave in my config.yml file is given below:

Environment properties

The following properties are passed in the application as environment properties. Learn more

Name	Value
GRADLE_HOME	/usr/local/gradle
JAVA_HOME	/usr/lib/jvm/java
M2	/usr/local/apache-maven/bin
M2_HOME	/usr/local/apache-maven
SERVER_PORT	5000
SPRING_DATASOURCE_PAS	[redacted]
SPRING_DATASOURCE_URL	jdbc:mysql://aa1xpxyf4a8ds90.
SPRING_DATASOURCE_USE	root

Fig 3.7: Configuration of EC2 environment

The status of EC2 and RDS instance is also running fine as shown below

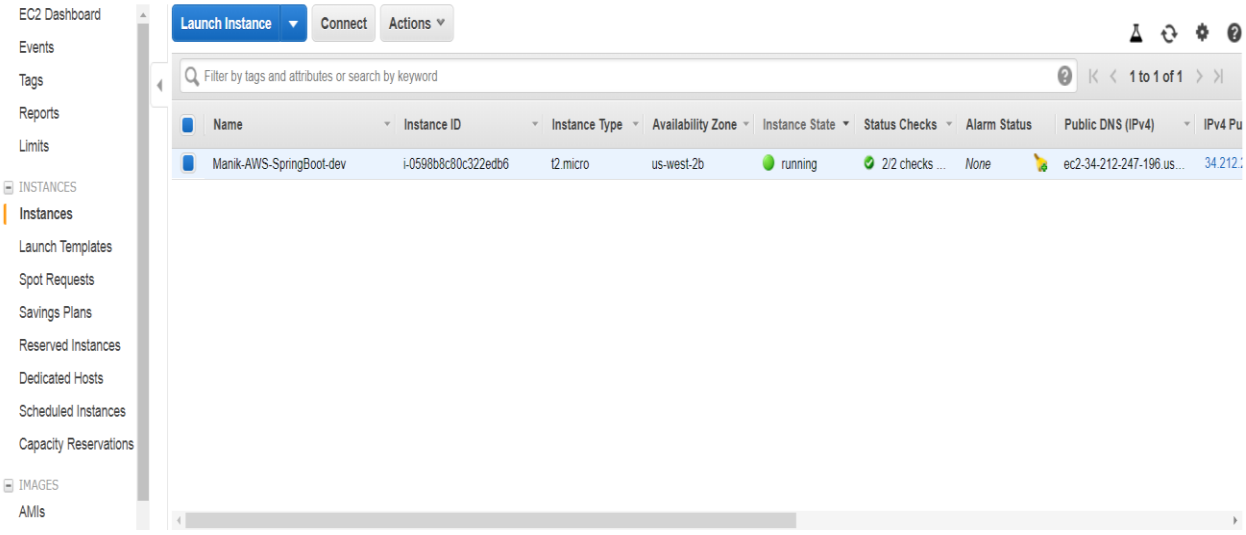

Fig 3.8: EC2 status

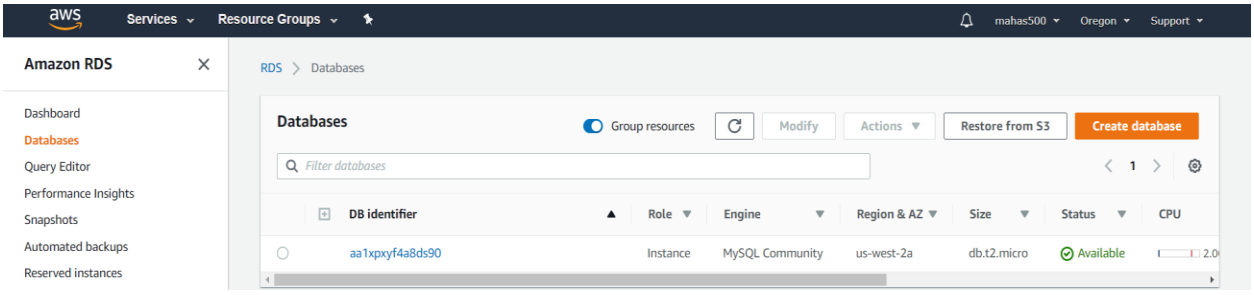

Fig 3.9: RDS status

RestAPI calls from postman client to my application hosted in Linux server in cloud are:

(i) To GET the data and POST the data URL is below

http://manik-aws-springboot-dev.us-west-2.elasticbeanstalk.com/api/C1/Cars

(ii) To send an email URL is below

http://manik-aws-springboot-dev.us-west-2.elasticbeanstalk.com/api/C1/Cars/email

(iii) To get a specific record URL is below

http://manik-aws-springboot-dev.us-west-2.elasticbeanstalk.com/api/C1/Cars/1

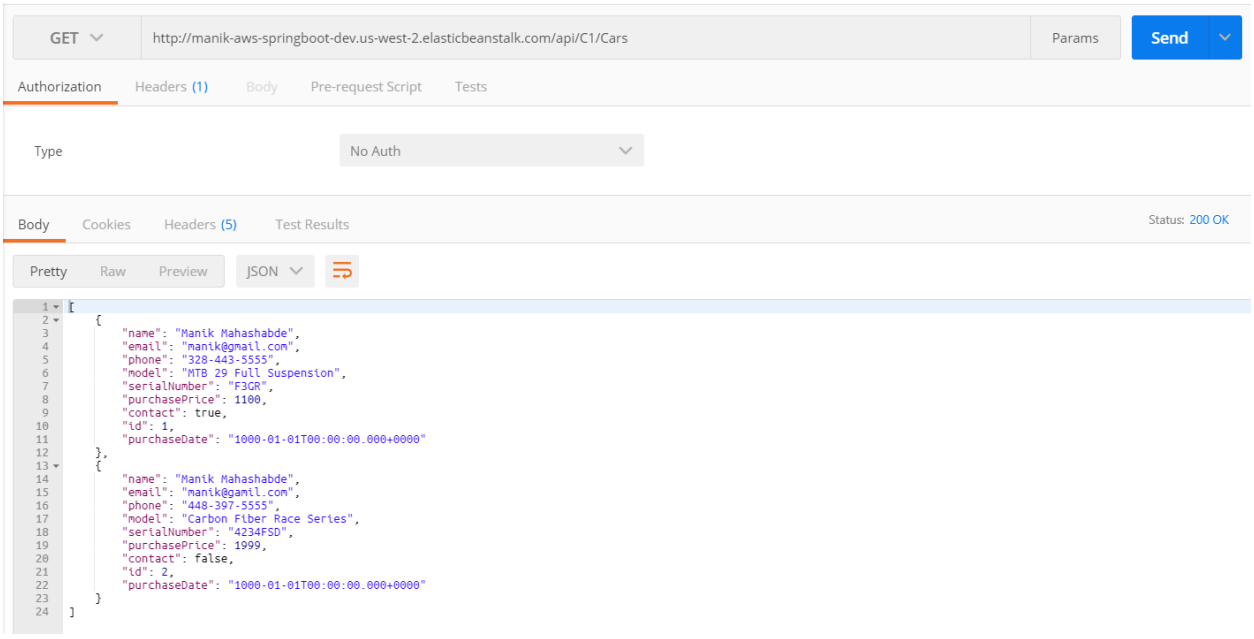

Fig 3.10: RestAPI call to AWS endpoint

After this I concluded that my application is deployed successfully in cloud. Now I needed to have an SSH connection to my application using putty. My first step was to create a .ppk file using puttyGen then later on importing it while connecting to my AWS CentOS Linux server. Below is the hostname for connection

Manik-AWS-SpringBoot-dev.us-west-2.elasticbeanstalk.com

Then I imported the .ppk private into putty and connected as **ec2-user.** Then after logging I could see my spring boot application logs as shown below. Logs are present in the location **/var/log/web-1.log**

cat /var/log/web-1.log (Note: cat command displays the content of our file)

Following is the IP address of AWS CentOS Linux Server

```
[ec2-user@ip-172-31-21-126 ~]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 02:A5:01:51:E8:A2
          inet addr:172.31.21.126  Bcast:172.31.31.255  Mask:255.255.240.0
          inet6 addr: fe80::a5:1ff:fe51:e8a2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
          RX packets:1066763 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1079548 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:354496696 (338.0 MiB)  TX bytes:355569001 (339.0 MiB)
```

Fig 3.11: IP address of AWS Linux Server

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::  (v2.2.0.BUILD-SNAPSHOT)

2019-11-09 05:00:12.654  INFO 3185 --- [           main] com.FordDemo.Car.CarApplication          : Starting CarApplication v0.0.1-SNAPSHOT on ip-172-31-21-126 with PID 3185 (/var/app/curre
nt/application.jar started by webapp in /var/app/current)
2019-11-09 05:00:12.661  INFO 3185 --- [           main] com.FordDemo.Car.CarApplication          : No active profile set, falling back to default profiles: default
2019-11-09 05:00:15.895  INFO 3185 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data repositories in DEFAULT mode.
2019-11-09 05:00:16.238  INFO 3185 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 287ms. Found 1 repository interfaces.
2019-11-09 05:00:17.840  INFO 3185 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration'
 of type [org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for
auto-proxying)
2019-11-09 05:00:18.837  INFO 3185 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 5000 (http)
2019-11-09 05:00:18.870  INFO 3185 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2019-11-09 05:00:18.870  INFO 3185 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.27]
2019-11-09 05:00:19.052  INFO 3185 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2019-11-09 05:00:19.053  INFO 3185 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 6174 ms
2019-11-09 05:00:20.245  INFO 3185 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2019-11-09 05:00:20.352  INFO 3185 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate Core {5.4.6.Final}
2019-11-09 05:00:20.687  INFO 3185 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
2019-11-09 05:00:20.829  INFO 3185 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2019-11-09 05:00:21.851  INFO 3185 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2019-11-09 05:00:21.894  INFO 3185 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
2019-11-09 05:00:24.093  INFO 3185 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platfo
rm.internal.NoJtaPlatform]
2019-11-09 05:00:24.112  INFO 3185 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2019-11-09 05:00:25.999  WARN 3185 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be perform
ed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2019-11-09 05:00:26.429  INFO 3185 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2019-11-09 05:00:27.631  INFO 3185 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 5000 (http) with context path ''
2019-11-09 05:00:27.639  INFO 3185 --- [           main] com.FordDemo.Car.CarApplication          : Started CarApplication in 16.917 seconds (JVM running for 20.092)
2019-11-09 05:49:37.955  INFO 3185 --- [nio-5000-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2019-11-09 05:49:37.957  INFO 3185 --- [nio-5000-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2019-11-09 05:49:38.001  INFO 3185 --- [nio-5000-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 44 ms
Hibernate: select car0_.id as id1_0_, car0_.purchase_date as purchase2_0_, car0_.contact as contact3_0_, car0_.email as email4_0_, car0_.model as model5_0_, car0_.name as name6_0_, car0_.ph
one as phone7_0_, car0_.purchase_price as purchase8_0_, car0_.serial_number as serial_n9_0_ from car car0_
Hibernate: select car0_.id as id1_0_, car0_.purchase_date as purchase2_0_, car0_.contact as contact3_0_, car0_.email as email4_0_, car0_.model as model5_0_, car0_.name as name6_0_, car0_.ph
one as phone7_0_, car0_.purchase_price as purchase8_0_, car0_.serial_number as serial_n9_0_ from car car0_
[root@ip-172-31-21-126 log]#
```

Fig 3.12: Spring boot logs in Linux server

After using the command which shows active process running

ps -ef

```
root      2610     1  0 Nov09 ?        00:00:01 sendmail: accepting connections
smmsp     2619     1  0 Nov09 ?        00:00:00 sendmail: Queue runner@01:00:00 for /var/spool/clientmqueue
root      2631     1  0 Nov09 ?        00:00:00 crond
root      2645     1  0 Nov09 ?        00:00:00 /usr/sbin/atd
ntp       2979     1  0 Nov09 ?        00:00:04 ntpd -u ntp:ntp -p /var/run/ntpd.pid -g
root      3117     1  0 Nov09 ?        00:00:20 /usr/bin/python /usr/local/bin/supervisord -c /etc/supervisor/supervisord.conf --nodaemon
root      3159     1  0 Nov09 ?        00:00:00 nginx: master process /usr/sbin/nginx -c /etc/nginx/nginx.conf
nginx     3162  3159  0 Nov09 ?        00:00:00 nginx: worker process
root      3184  3117  0 Nov09 ?        00:00:00 python /opt/elasticbeanstalk/private/lib/supervisor_listener.py
webapp    3185  3117  0 Nov09 ?        00:01:48 java -jar application.jar
root      3195     1  0 Nov09 ?        00:00:00 su -s /bin/bash -c healthd healthd
healthd   3197  3195  0 Nov09 ?        00:00:55 puma 2.11.1 (tcp://127.0.0.1:22221) [healthd]
root      3227     1  0 Nov09 ?        00:04:33 /usr/bin/python2.7 /opt/aws/bin/cfn-hup
root      3299     1  0 Nov09 ttyS0    00:00:00 /sbin/agetty ttyS0 9600 vt100-nav
root      3302     1  0 Nov09 tty1     00:00:00 /sbin/mingetty /dev/tty1
root      3307     1  0 Nov09 tty2     00:00:00 /sbin/mingetty /dev/tty2
root      3311     1  0 Nov09 tty3     00:00:00 /sbin/mingetty /dev/tty3
root      3314     1  0 Nov09 tty4     00:00:00 /sbin/mingetty /dev/tty4
root      3316     1  0 Nov09 tty5     00:00:00 /sbin/mingetty /dev/tty5
root      3318     1  0 Nov09 tty6     00:00:00 /sbin/mingetty /dev/tty6
root      3322  1571  0 Nov09 ?        00:00:00 /sbin/udevd -d
root      3323  1571  0 Nov09 ?        00:00:00 /sbin/udevd -d
root     26132  2577  0 16:37 ?        00:00:00 sshd: ec2-user [priv]
ec2-user 26135 26132  0 16:37 ?        00:00:00 sshd: ec2-user@pts/0
ec2-user 26136 26135  0 16:37 pts/0    00:00:00 -bash
root     26162 26136  0 16:37 pts/0    00:00:00 sudo su -
root     26163 26162  0 16:37 pts/0    00:00:00 su -
root     26164 26163  0 16:37 pts/0    00:00:00 -bash
root     26213 26164  0 16:38 pts/0    00:00:00 ps -ef
```

Fig 3.13: Running Process of Spring Boot Application

Now the next step was to introduce **containerization** using **docker** in the project which acted as a client for my application deployed in cloud. After signing up on docker website, I downloaded and installed Docker toolbox for my Windows 8.1

Next step was to pull ubuntu image from Docker Hub on command prompt*(Docker Documentation, n.d.)*

$ docker pull ubuntu

Next step was to create container from my Ubuntu image using below command

$ docker run -i -t ubuntu /bin/bash [-i for interaction, -t for terminal]

After creating Ubuntu container, I installed **CURL** using the below command

$ apt install curl [APT is Advanced Package Tool. Used for installing and handling packages in Linux]

Now my Ubuntu container had CURL installed. It stands for client URL through which I can hit the endpoint of my application.

I have already created an ubuntu Image container with ID **de98b092519c**

```
C:\Users\Manik325>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
de98b092519c        ubuntu              "/bin/bash"         7 days ago
Up 10 hours                             nice_johnson

C:\Users\Manik325>
```

Fig 3.14: Running Ubuntu Container

Next step was to create my own image. I created new image of the container using commit command:

$ docker commit de98b092519c manik-aws-image

Where **manik-aws-image** is the new image name with Image ID **e8903033dd32**. After this I pushed the image by first logging to my docker hub account in docker CLI. Command for this is

$ docker login

After this I gave my docker username and password. After logging in I gave the command

$ docker tag e8903033dd32 mahas500/manikrepo:manikSpringBootAWS

where **manikSpringBootAWS** is the tag name of my image on docker hub

Next step was to push image on DockerHub using command

$ docker push mahas500/manikrepo

Where mahas500 is my docker Hub username and manikrepo is my public repository name.

After 3-5 mins my image was pushed to DockerHub which can be seen on next page.

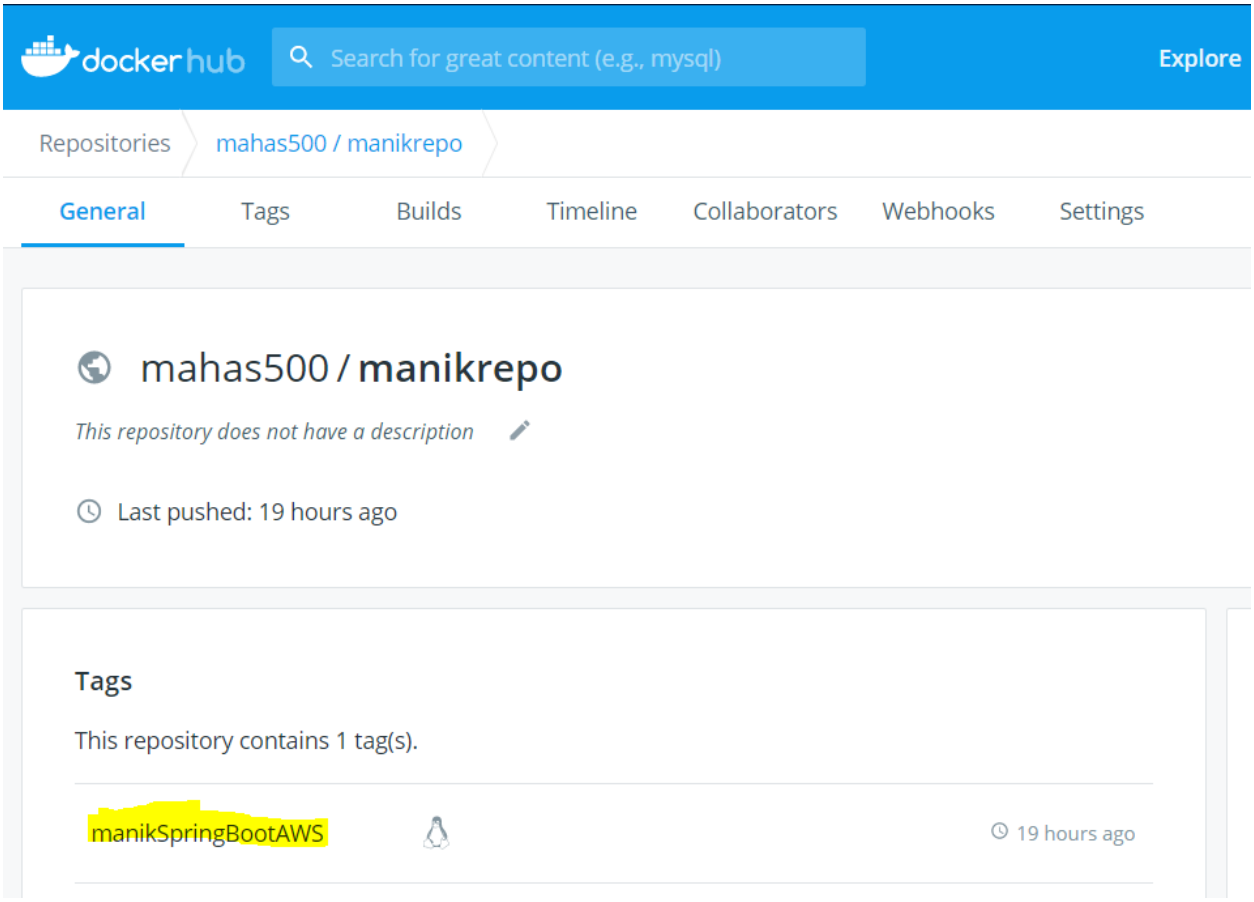

Fig 3.15: My docker image on DockerHub

Then I pulled the image to my local machine using below command.

$ docker pull mahas500/manikrepo:manikSpringBootAWS

The advantage of doing this step is that now I had new Ubuntu image with all the configuration which I did in my container with ID de98b092519c

I could see the image in my section using docker images

```
Manik325@Manik MINGW64 /c/Program Files/Docker Toolbox
$ docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED
          SIZE
manik-aws-image             latest              e8903033dd32        20 hours
ago         187MB
mahas500/manikrepo          manikSpringBootAWS  e8903033dd32        20 hours
ago         187MB
httpd                       latest              d3017f59d5e2        2 weeks a
go          165MB
ubuntu                      latest              cf0f3ca922e0        3 weeks a
```

Fig 3.16: Pulled image in docker

Then, I created container of my own image using

$ docker run -i -t e8903033dd32 /bin/bash

where e8903033dd32 is my Image ID. After this I gave exit. Now container d1938ef58e40 was created as shown below in highlighted text

```
Manik325@Manik MINGW64 /c/Program Files/Docker Toolbox
$ docker ps -a
CONTAINER ID        IMAGE                               COMMAND
  CREATED             STATUS                          PORTS                   NAMES
d1938ef58e40        e8903033dd32                        "/bin/bash"
  19 hours ago        Exited (255) 47 minutes ago                             agita
```

Fig 3.17: Newly created container from pulled Image

For accessing an existing container, I gave the command

$ docker start d1938ef58e40

Then to interact with running container I gave

$ docker exec -i -t d1938ef58e40 /bin/bash

Where -i is given to interact with the container

-t is given to open tty terminal

/bin/bash is the shell which I want to open in existing container

After logging to bash shell of my image I gave command to get RestAPI response from my application hosted in AWS cloud

```
Manik325@Manik MINGW64 /c/Program Files/Docker Toolbox
$ docker start d1938ef58e40
d1938ef58e40

Manik325@Manik MINGW64 /c/Program Files/Docker Toolbox
$ docker exec -i -t d1938ef58e40 /bin/bash
root@d1938ef58e40:/#
root@d1938ef58e40:/#
root@d1938ef58e40:/# curl http://manik-aws-springboot-dev.us-west-2.elasticbeans
talk.com/api/C1/Cars
[{"name":"Manik Mahashabde","email":"manik@gmail.com","phone":"328-443-5555","mo
del":"MTB 29 Full Suspension","serialNumber":"F3GR","purchasePrice":1100.00,"con
tact":true,"id":1,"purchaseDate":"1000-01-01T00:00:00.000+0000"},{"name":"Manik
Mahashabde","email":"manik@gamil.com","phone":"448-397-5555","model":"Carbon Fib
er Race Series","serialNumber":"4234FSD","purchasePrice":1999.00,"contact":false
,"id":2,"purchaseDate":"1000-01-01T00:00:00.000+0000"}]root@d1938ef58e40:/#
root@d1938ef58e40:/#
root@d1938ef58e40:/#
root@d1938ef58e40:/# curl http://manik-aws-springboot-dev.us-west-2.elasticbeans
talk.com/api/C1/Cars/email
Congratulations! Your mail has been sent to the user.root@d1938ef58e40:/#
root@d1938ef58e40:/#
```

Fig 3.18: RestAPI call to AWS cloud from container

As it can be seen above that I gave GET request and SMTP request to my application hosted in AWS cloud. I got the response from my application hosted in spring boot and a mail from spring boot in my Gmail.

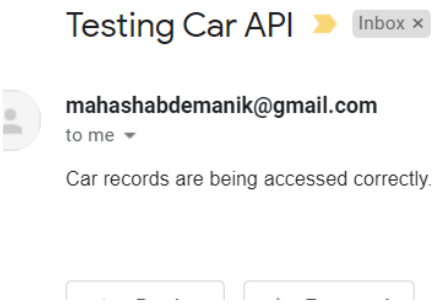

Fig 3.19: Mail from Docker

3.1 Testing of the application

Testing of the project is done by testing the endpoint URL's which are

http://manik-aws-springboot-dev.us-west-2.elasticbeanstalk.com/api/C1/Cars

http://manik-aws-springboot-dev.us-west-2.elasticbeanstalk.com/api/C1/Cars/email

http://manik-aws-springboot-dev.us-west-2.elasticbeanstalk.com/api/C1/Cars/1

I have already tested the passing test cases in the technical and demonstration section. Let us test some test cases which will fail.

Case 1: If the endpoint URL for GET and POST request is given wrong from the docker container client then we get a HTTP response as **404 error** as shown below

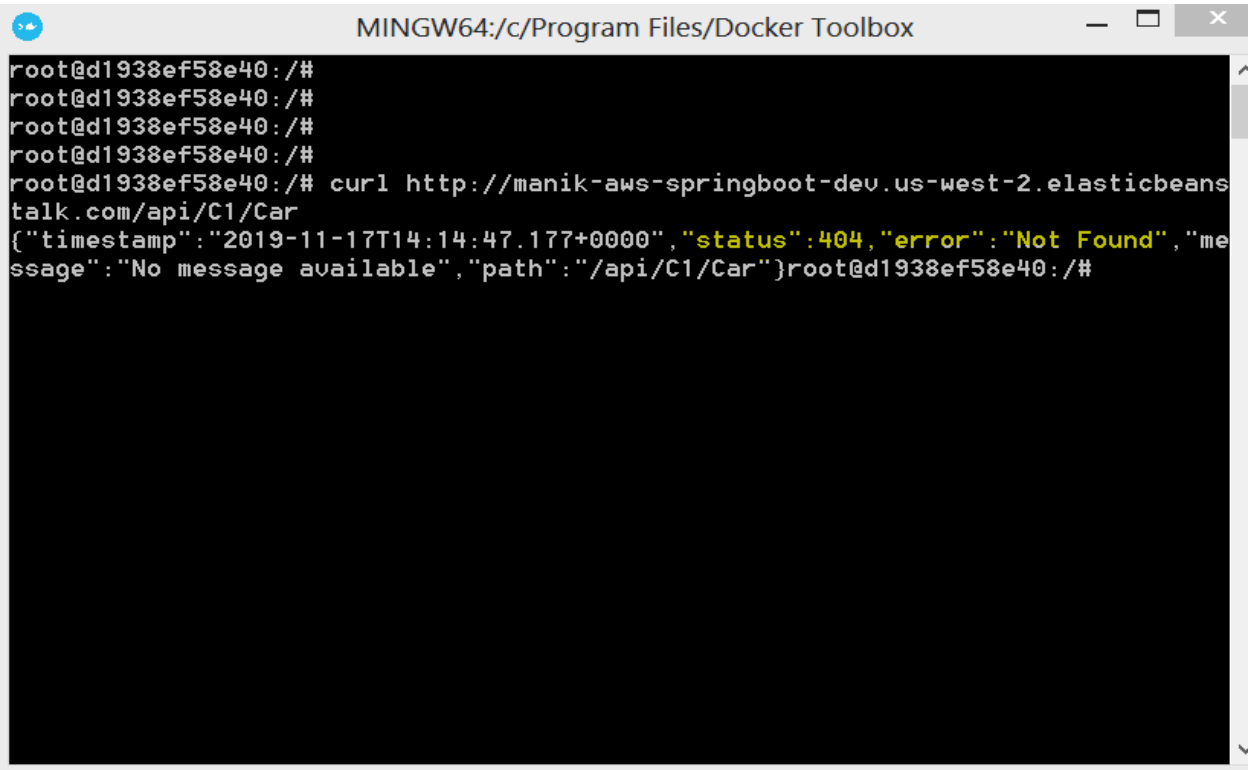

Fig 3.20: Test case of wrong endpoint

Case 2: If the endpoint URL in GET request is given of the ID which is not present in the database of the application we get a HTTP response as **500 error** which is a server side error as shown below

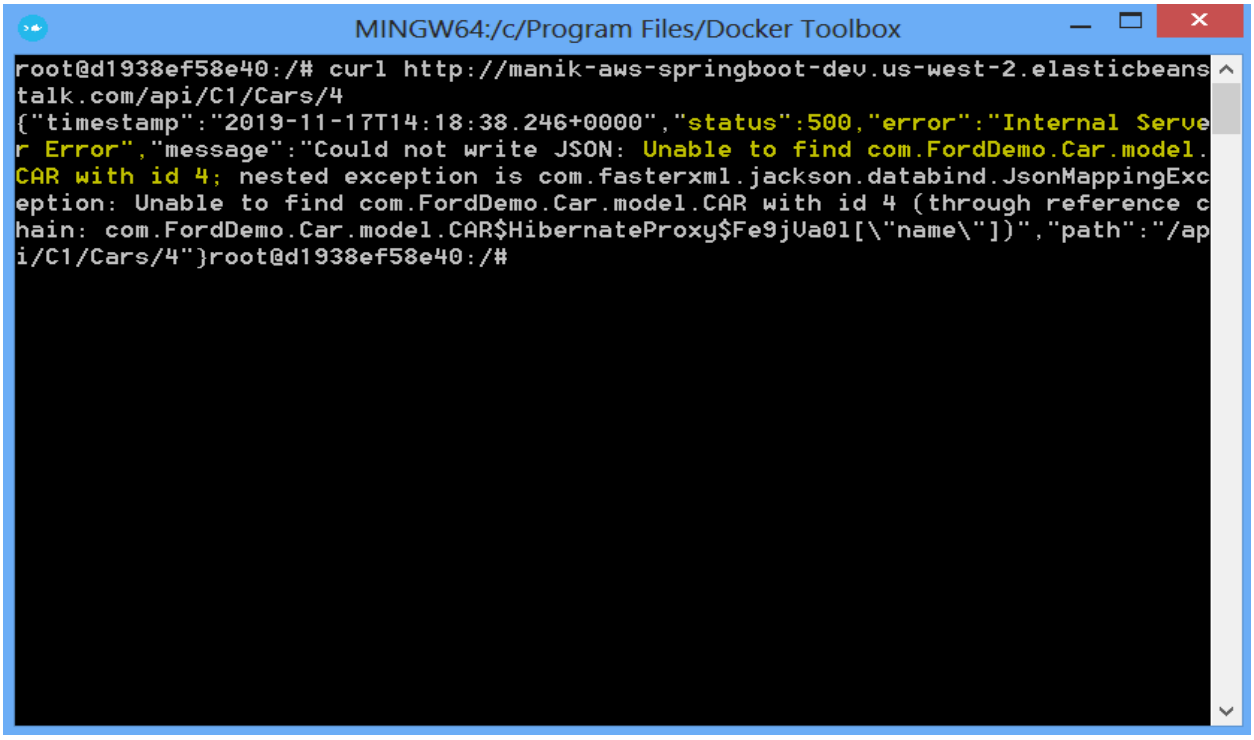

Fig 3.21: Test case of invalid ID in endpoint

Case 3: If the endpoint URL for SMTP is given wrong then again we get 404 error and mail won't be sent.

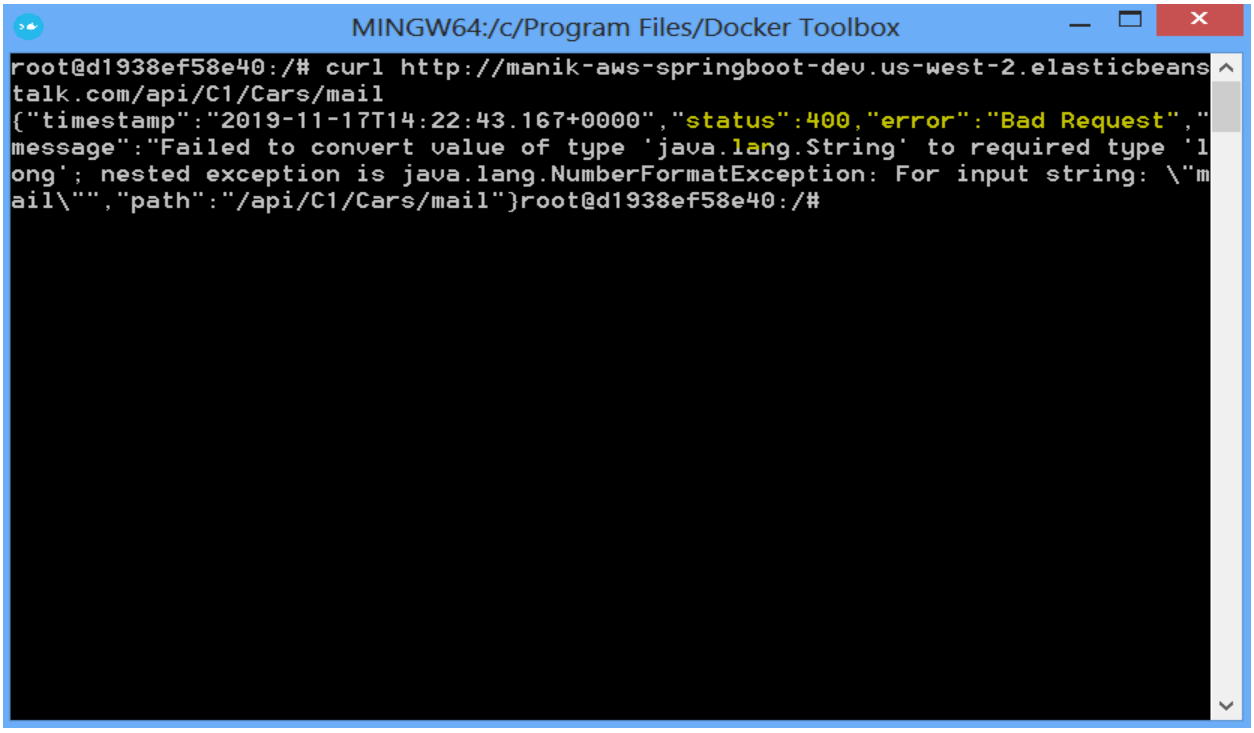

Fig 3.22: Test case of wrong email endpoint

3.2 Future Scope of the application

The future scope of the project is expanding the web application so that more instances(server) of EC2 can be created. Then a **load balancer** can also be set up which would have **reverse proxy** configuration. So that if there is more traffic in production environment then it can be redirected to any of the servers by load balancer.

Security can be added to the application. Spring security such as **OAuth2** can be added which would make the application more secure.

A frontend can be made of the application using technologies like **Angular**, React. Whole application can be deployed in a docker container and docker file can be used to build docker Image. Then the docker file can pushed on to AWS cloud using services like Elastic Container Service (**ECS**).

REFERENCES & BIBLIOGRAPHY

Amazon Web Services, Inc. (n.d.). *What is Cloud Computing*. [online] Available at: https://aws.amazon.com/what-is-cloud-computing/ [Accessed 17 Nov. 2019].

Docs.aws.amazon.com. (n.d.). *Manually Install the EB CLI - AWS Elastic Beanstalk*. [online] Available at: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-install-advanced.html [Accessed 17 Nov. 2019].

Amazon Web Services, Inc. (n.d.). *Amazon EC2*. [online] Available at: https://aws.amazon.com/ec2/?c=7&pt=1 [Accessed 17 Nov. 2019].

Us-west-2.console.aws.amazon.com. (n.d.). [online] Available at: *https://us-west-2.console.aws.amazon.com/rds/home?region=us-west-2#GettingStarted:* [Accessed 17 Nov. 2019].

Ssh.com. (n.d.). *SSH (Secure Shell) Home Page | SSH.COM*. [online] Available at: https://www.ssh.com/ssh/?utm_source=s&utm_medium=nav&utm_campaign=head [Accessed 17 Nov. 2019].

Docker Documentation. (n.d.). *Docker Hub Quickstart*. [online] Available at: *https://docs.docker.com/docker-hub/* [Accessed 17 Nov. 2019].

Docker Documentation. (n.d.). *Install Docker Toolbox on Windows*. [online] Available at: https://docs.docker.com/v17.12/toolbox/toolbox_install_windows/ [Accessed 17 Nov. 2019].

Medium. (n.d.). *Docker—Beginner's Guide—Part 1: Images & Containers*. [online] Available at: https://medium.com/codingthesmartway-com-blog/docker-beginners-guide-part-1-images-containers-6f3507fffc98 [Accessed 17 Nov. 2019].

Medium. (n.d.). *15 Docker Commands You Should Know*. [online] Available at: https://towardsdatascience.com/15-docker-commands-you-should-know-970ea5203421 [Accessed 17 Nov. 2019].

Amazon Web Services. (n.d.). *Deploying a Spring Boot Application on AWS Using AWS Elastic Beanstalk | Amazon Web Services*. [online] Available at: https://aws.amazon.com/blogs/devops/deploying-a-spring-boot-application-on-aws-using-aws-elastic-beanstalk/ [Accessed 18 Nov. 2019].

Docker Documentation. (n.d.). *docker exec*. [online] Available at: https://docs.docker.com/engine/reference/commandline/exec/ [Accessed 17 Nov. 2019].

Porter, B., Zyl, J. and Lamy, O. (n.d.). *Maven – Welcome to Apache Maven*. [online] Maven.apache.org. Available at: https://maven.apache.org/# [Accessed 17 Nov. 2019].

Spring.io. (n.d.). *Spring Projects*. [online] Available at: https://spring.io/projects/spring-boot [Accessed 17 Nov. 2019].

dzone.com. (n.d.). *Using Java @Annotations to Build Full Spring Boot REST API - DZone Java*. [online] Available at: https://dzone.com/articles/key-java-annotations-to-build-full-spring-boot-res [Accessed 17 Nov. 2019].

Baeldung. (n.d.). *Guide to Spring Email | Baeldung*. [online] Available at: https://www.baeldung.com/spring-email [Accessed 17 Nov. 2019].

Business.qld.gov.au. (n.d.). *Benefits of cloud computing | Business Queensland*. [online] Available at: https://www.business.qld.gov.au/running-business/it/cloud-computing/benefits [Accessed 17 Nov. 2019].

dzone.com. (n.d.). *Top 10 Benefits of Docker - DZone DevOps*. [online] Available at: https://dzone.com/articles/top-10-benefits-of-using-docker [Accessed 17 Nov. 2019].

APPENDIX

A.1 Background of Java web application development

Java: It is an Object-oriented programming language to develop web and enterprise application, servers, android applications etc.

Maven: Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information*(Porter, Zyl and Lamy, n.d.).*

Spring Boot: Spring Boot makes it easy to create stand-alone, production-grade Java Spring based Applications. It takes an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration*(Spring.io, n.d.).*

Java Persistence API(JPA): It create a table in database automatically and performs all the create, read, update, delete operations automatically.

RestAPI: It is used to create endpoints in controller class in Spring boot project. Client or frontend send an HTTP request to the endpoint located in server to fetch data or manipulate data depending upon the type of HTTP request.

Java Spring Boot Annotations @: Annotations are used by the developer to reduce the boilerplate code and improve the efficiency, readability of the program*(dzone.com, n.d.)*.

Spring Initializer: It is used to create spring project artifact through which we can select different dependencies and download the project. Then add the project workspace into our compiler.

Spring Mail: It is used to configure SMTP in our spring boot project for sending mails*(Baeldung, n.d.).*

Postman: It acts as client through which we can fetch RestAPI data, add the data by sending HTTP requests.

MySQL Workbench: MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more.

A.2 Benefits of the technologies used in the project

Benefits of using Cloud computing are:[

- **Reduced IT cost** : Moving to cloud computing may reduce the cost of managing and maintaining your IT systems. Rather than purchasing expensive systems and equipment for your business, you can reduce your costs by using the resources of your cloud computing service provider.
- **Scalability** : Your business can scale up or scale down your operation and storage needs quickly to suit your situation, allowing flexibility as your needs change.
- **Business Continuity** : Protecting your data and systems is an important part of business continuity planning. Whether you experience a natural disaster, power failure or other crisis, having your data stored in the cloud ensures it is backed up and protected in a secure and safe location*(Business.qld.gov.au, n.d.).*

Benefits of using Docker are:

- **Continuous Integration Efficiency** : Docker enables you to build a container image and use that same image across every step of the deployment process. A huge benefit of this is the ability to separate non-dependent steps and run them in parallel. The length of time it takes from build to production can be sped up notably.
- **Rapid Deployment** : Docker manages to reduce deployment to seconds. This is due to the fact that it creates a container for every process and does not boot an OS. Data can be created and destroyed without worry that the cost to bring it up again would be higher than what is affordable.
- **Isolation** : Docker ensures your applications and resources are isolated and segregated. Docker makes sure each container has its own resources that are isolated from other containers. You can have various containers for separate applications running completely different stacks*(dzone.com, n.d.).*

A.3 Technical details of the Java spring boot application

I thought of having a web application using the compiler Spring tool suite(version 3.9) which retrieves a car record from the MySQL database using a GET request. Adds a car record in database using a POST request. I implemented the application using spring Model-View-Controller framework by first downloading a **Spring Boot** (version 2.2.0) project targets from start.spring.io with Java version 8 and configuring them using a **Maven** project. Maven downloads all the dependencies automatically using a file **pom.xml**. Then a Model class is implementing JPA (Java Persistence Repository) which creates database instance automatically and stores them in database. JPA acts as an ORM on top of database which perform CURD(create, update, retrieve, delete) operations automatically. It has annotation such as **@Entity @Id** which are vary handy. After this MySQL database was created which was mapped to the application which stores the Car records created by Hibernate JPA. Then a controller is created in which annotations such as **@RequestMapping @RestController** were used.

```
1  package com.FordDemo.Car.Controller;
2
3  import java.util.List;
20
21 @RestController
22 @RequestMapping("api/C1/Cars")
23 public class CarController
24 {
25
26     @Autowired
27     private CarRepository carRepository;
28
29     @Autowired
30     private EMailService myEmailService;
31
32     @Autowired
33     private UserEmail userEmail;
34     //private CAR car;
35
36     @GetMapping
37     public List<CAR> getcars()
38     {
39         return carRepository.findAll();
40
41         }
42
43     @PostMapping
44     @ResponseStatus(HttpStatus.OK)
45     public void EnterCar(@RequestBody CAR car) {
46         carRepository.save(car);
47     }
48
49     @GetMapping("/email")
50     public String send() {
51
52         userEmail.setEmailAddress("mahashabdemanik@gmail.com");
```

Fig: A3.1 RestApi Controller

RestApi of GET and POST were created in this class. After this **application.properties** file was used to store important information as shown below.

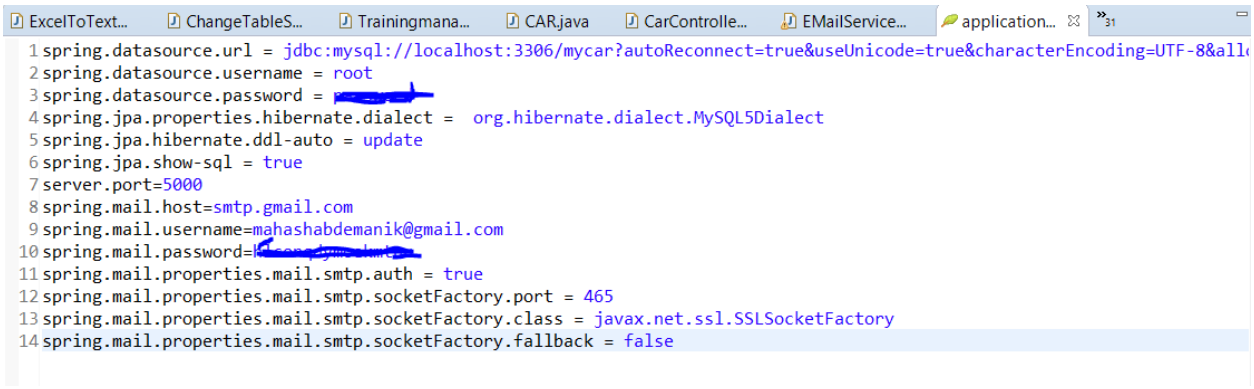

Fig: A3.2 application.properties configuration file

As seen above server port was defined, Simple mail configuration was done. Then application was started in Spring tool suite. Spring boot application usually comes with an internal Tomcat server in which application is hosted. Once application was up and running fine tomcat logs looked like below.

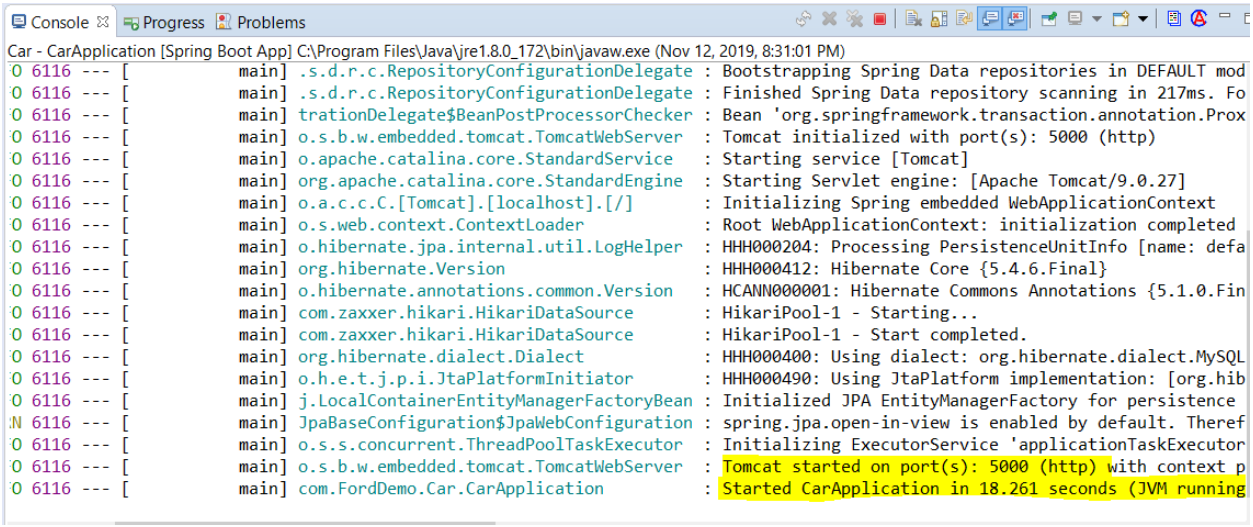

Fig: A3.3 Spring boot application successful running logs

The above highlighted text depicts that the application has been started successfully in local machine.

Next step is testing endpoints(a URL configured in the application) through Postman which acts as a client. As seen in Figure 3 application is running successfully on localhost with port number 5000. The endpoints can be seen in Figure 1. Below are the URL's

http://localhost:5000/api/C1/Cars :- To send a GET request to the server and getting data from the database. For adding a record into the database, the same URL is used but POST is selected instead of GET for adding records.

http://localhost:5000/api/C1/Cars/email:- this URL is used to send an email to the user telling that application is running fine. Below is postman request for one of the URL.

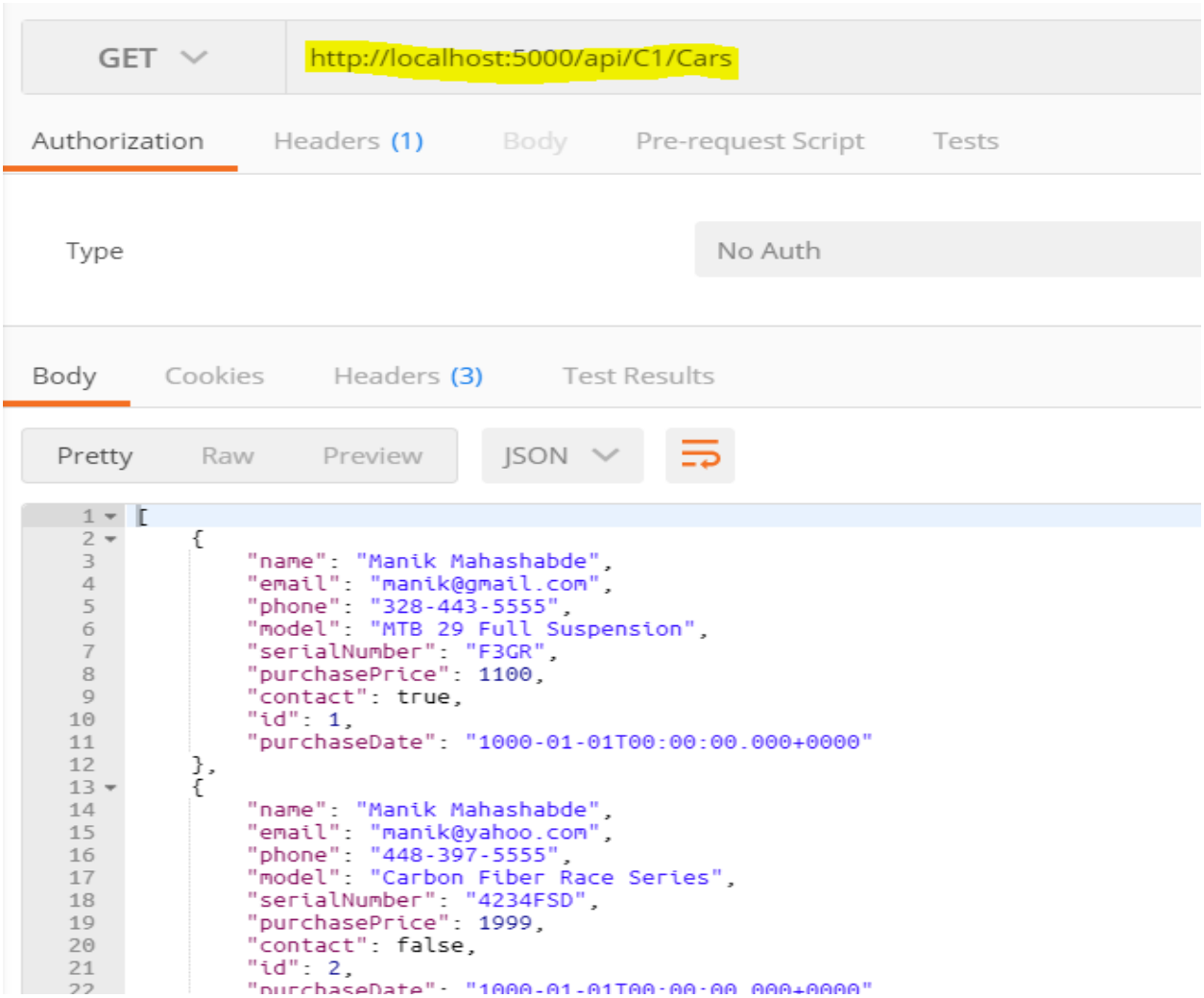

Fig A3.4: GET request to spring boot from PostMan

The MySQL database of the above application is shown below

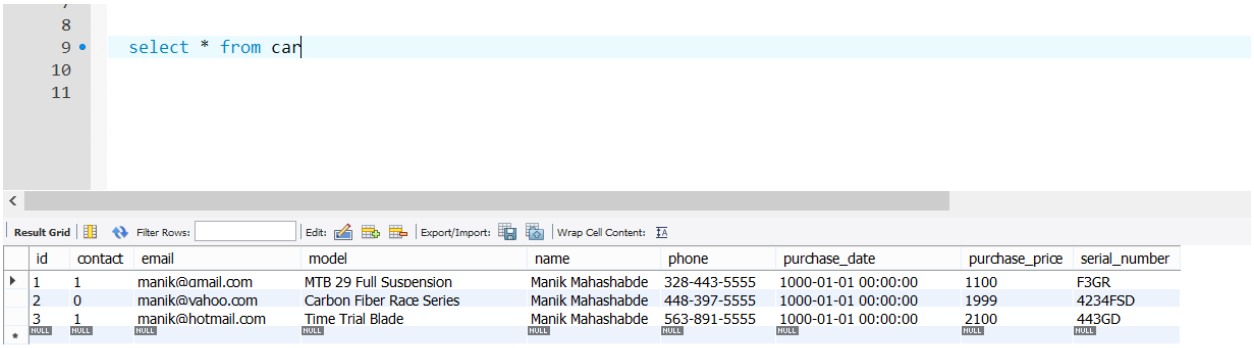

Fig A3.5: MySQL database of the application

A.4 Source code of Java Application

Source code of the application is mentioned in below GitHub URL

https://github.com/Manik2018/NetworkCA.git