



Dublin Business School

excellence through learning

Networks and Systems Administration Assignment

Building & Deploying secure (SSL) Portfolio website on cloud platforms using containerization methods and establishing communication between two servers.

Module Title: Networks and Systems Administration (B9IS118)

Module Leader: Mr. Obinna Izima

Student Name: Saurabh Devade (10531140)

TABLE OF Contents

TABLE OF Contents	1
TABLE OF FIGURES.....	2
LIST OF TABLES.....	2
1. INTRODUCTION	3
2. BACKGROUND	4
2.1. WHY WEBSITE FOR PORTFOLIO?.....	4
Individuals who can consider to have a portfolio website are: (Ragaine).....	4
2.2. VIRTUALIZATION AND DOCKER	4
2.3. HOW DOCKER WORKS?	5
2.4. DOMAIN NAME SYSTEM (DNS)	6
2.5. HTTP, HTTPS, SSL	6
2.6. How SSL Works?	7
2.7. SMTP – SIMPLE MAIL TRANSFER PROTOCOL.....	7
2.9. APACHE AND TOMCAT SERVER	7
2.10. MySQL	8
2.11. CLOUD PLATFORMS	8
3. TECHNICAL DESCRIPTION OF THE PROOF OF CONCEPT	9
3.1. Website’s HTML, CSS & JavaScript.....	9
3.2. DEPLOYING WEB APPLICATION USING DOCKER	9
3.3. MAPPING OF DOMAIN NAME TO IP ADDRESS.....	17
3.4. COMMUNICATION BETWEEN TWO SERVERS AND SENDING EMAILS USING AWS’S SES AND SMTP PROTOCOL	18
4. TESTING, EVALUATION & DEMONSTRATION OF PROOF OF CONCEPT 21	
4.1. Testing & Evaluation whether HTML is working correctly	21
4.2. Testing Docker images	21
4.3. Testing Docker Container.....	21
4.4. Testing whether our container is running properly	22
4.5. Testing image is pushed on docker hub	22
4.6. Testing docker hub image on production server	23
4.7. Testing if sites work properly on production server.....	24
4.8. Testing of domain Mapping	24

4.9. Testing whether API is successfully and data is stored.....	25
4.10. Testing whether API is successfully called on other servers and email is sent successfully using SMTP.	25
5. Conclusion	26
6. REFERENCES & BIBLIOGRAPHY	27
7. APPENDIX.....	29

TABLE OF FIGURES

Figure 1 - Docker Vs Virtual Machine (Preeth <i>et al.</i> , 2016).....	6
Figure 2 – Dockerfile.....	10
Figure 3 - SSL Letsencrypt Certbot Command (Mahdi Mashrur Matin).....	10
Figure 4 - Virtual Host Configuration File.....	11
Figure 5 - Directory	11
Figure 6 - Docker Build Command	11
Figure 7 - Docker Images Command	12
Figure 8 - Docker Run Command	12
Figure 9 - Docker Ps Command	12
Figure 10 - Localhost Website	13
Figure 11 - SSL Certificate Details	14
Figure 12 - Docker Tag and Docker Push Command	14
Figure 13 - Docker Hub Repository	15
Figure 14 - Docker Pull Command	16
Figure 15 - Live Site.....	17
Figure 16 - Big Rock DNS Management	17
Figure 17 - Java Controller Code	18
Figure 18 - Inspect Element Snippet	19
Figure 19 - Database Snippet	19
Figure 20 - AWS SES Service.....	19
Figure 21 - application.properties file	20
Figure 22 - Java Mail Service Code	20
Figure 23 - Ping Command	24
Figure 24 - Received Mail Snippet.....	25

LIST OF TABLES

Table 1 - Before Virtualization VS After Virtualization.....	4
Table 2 - Container-Based VM VS Docker-based VM	5

1. INTRODUCTION

This project shows how to build a responsive portfolio website and host it on the server/cloud using containerization and virtualization techniques using Docker so it can be accessed by anyone who has an active internet connection. It also includes how to access a website over domain name so it reduces human efforts to remember IP addresses of each website they are supposed to visit. Also, to secure the data transmission from client to server and vice versa, SSL (Secure Socket Layer) certificate is used so the website can only be accessed over the SSL layer (i.e. <https://isaurabh.me>). To register the visitor's requirement, a separate database is developed and maintained on the other server. To demonstrate the communication between two servers with different IP addresses this project makes use of a JAVA-based API / Webservice (Application Programming Interface). Further, to acknowledge the visitor that his requirement is reached to the owner successfully, the mail is being sent to the visitor using an SMTP (Simple Mail Transfer Protocol) Protocol using Amazon's SES (AWS) (Simple Email Services). This project also demonstrates how to transfer files from a local machine to the server over a Secure File transfer protocol and also how to access a server from an SSH using various client-side Applications.

2. BACKGROUND

2.1. WHY WEBSITE FOR PORTFOLIO?

In this era of the Internet and various technologies, one must have an online presence. In older days people used to write their portfolio on papers and carry it with them all the time but, as the technologies are growing rapidly people are choosing online platforms for their portfolio. It is easy, handy and accessible by everyone. You can show all of your past working experiences, your past projects, your future goals, etc on your website. By having one such a website, clients will be able to see your work and more clients will be attracted to you. (Pibernik, Kanižaj and Frljak, 2012)

Individuals who can consider to have a portfolio website are: (Ragaine)

- Graphic Designers
- Website Designers
- Website Developers
- Application Developers
- Interior Designers
- Photographers

2.2. VIRTUALIZATION AND DOCKER

This project makes use of virtualization technology using Docker

From the name Virtualization, it is very clear that it has something to do with virtuality. In the context of this project, virtualization is basically deploying/creating a number of instances on a single operating system.

It utilizes the host systems hardware in the most efficient way by creating multiple virtual instances on a single machine. (Jain and Choudhary, 2016)

BEFORE VIRTUALIZATION	AFTER VIRTUALIZATION
1. Only one operating system was able to run on the entire hardware	1. Multiple instances of an Operating system can run on a single hardware
2. Wastage of resources	2. Minimal wastage of resources
3. Very difficult and time-consuming to transfer the same configuration on a newer machine	3. Very easy and less time consuming to transfer it to another machine

Table 1 - Before Virtualization VS After Virtualization (Jain and Choudhary, 2016)

Docker is an open-source platform and was introduced in 2013 to solve the developer's problem i.e one application would run perfectly on one environment and it would fail in some other environment because applications require a lot of dependencies and libraries and a lot of configurations. While developing an application developer will

configure his environment for that particular application but if he wants to make his application live, before Docker he had to configure the Production environment also. Otherwise, the application would fail to run and to do this it would take a lot of time and resources and he has to follow all the steps again. By using Docker there is no need for this extra work to configure the system again.

While Developing, the Developer will put all the Dependencies and libraries in one Image and simply run that image in any environment he wants and it would run as it should be without any extra efforts (Klinbua and Vatanawood, 2018).

2.3. HOW DOCKER WORKS?

Docker doesn't create a new OS on the host operating system like Virtual Machine (Oracle) does. Instead, it creates a package called a container that includes all the external dependencies. And later it can be transferred to any other environment to replicate that application. (Preeth *et al.*, 2016)

CONTAINER-BASED VM - DOCKER	HYPERVISOR-BASED VM
Performance of the container-based virtualization is enhanced because it shares the kernel between multiple instances	Performance is not very optimum because it uses the same hardware as the host hardware uses
Easy to distribute or transfer to another machine	Inconvenient and needs lots of resources for transformation
Provides isolation between the containers as a security feature	Hard to obtain isolation
It is more resource efficient	It is less resource efficient

Table 2 - Container-Based VM VS Docker-based VM (Preeth *et al.*, 2016)

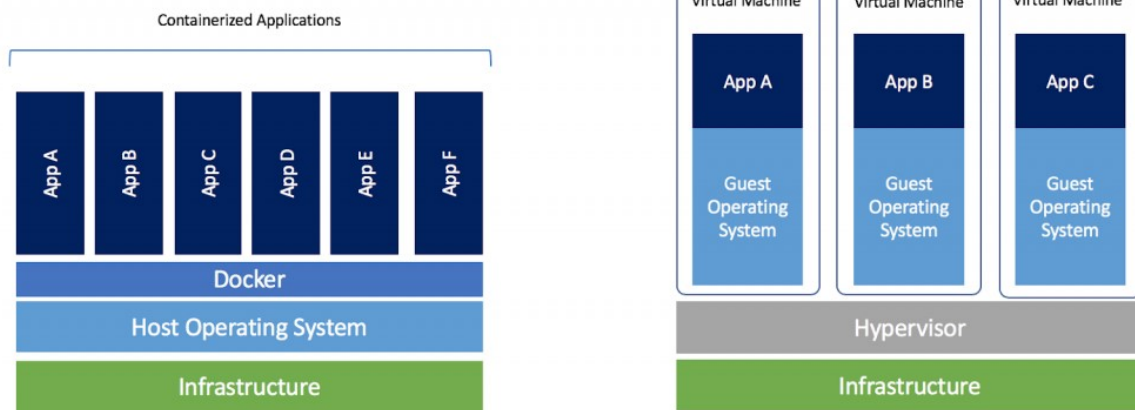


Figure 1 - Docker Vs Virtual Machine (Preeth *et al.*, 2016)

2.4. DOMAIN NAME SYSTEM (DNS)

In the network, every machine has its unique IP address(139.59.59.220) and if we want to access that machine we have to access it by its IP address. same goes for the websites also. But for humans, it is highly impossible to remember the IP addresses for so many websites. In addition, the computer doesn't identify systems like humans do. So to fill this void concept of DNS was introduced. SO what DNS does is it resolves the human-readable domain names like isaurabh.me to computer understandable IP addresses. This is how when we type any URL in the browser it converts that URL to IP address and fetches data from that server with that IP address. (“Cloudflare”; P. Mockapetris)

The working of DNS take place in number of steps It Mainly involves four different servers to resolve a domain name

- **DNS Recursive Resolver**
- **DNS Root Nameserver**
- **DNS TLD Nameserver**
- **Authoritative Nameserver**

2.5. HTTP, HTTPS, SSL

HTTP is HyperText Transfer Protocol and it is one of the most widely used protocols in the world. When we want to visit any website it works on the HTTP protocol e.g <http://isaurabh.me>. All the data transmission takes place under this protocol is in plain text format and is highly insecure. Anyone who is aware of data transmission on certain sites can get access to the data (Hacking) and can misuse it knows as Man in Middle attack. To avoid this HTTP is usually used in secure mode i.e **HTTPS**. HTTPS is nothing but HTTP with security mode **SSL**. For accessing the site over HTTPS we have to install a certificate for that domain and after that, all the data transmission takes place

over the secure network i.e all the data transmitted is encrypted. Therefore even if the hacker gets the data he cannot read it because it is encrypted that way, security is maintained. HTTPS is highly recommended for e-commerce or banking domain websites. (Kvachko et al.; P. Mockapetris)

2.6. How SSL Works?

SSL or secure sockets layer is a protocol that is used to ensure security on the internet. It uses public-key encryption to secure data. When a computer connects to a website that uses SSL the computer's web browser will ask the website to identify itself, then the webserver will send the computer a copy of its SSL certificate. SSL certificate is a small digital certificate that is used to authenticate the identity of a website. Basically it's used to let your computer know that the website you're visiting is trustworthy so then the computer's browser will check to make sure that it trusts the certificate and if it does, it will send message to the webserver, then after the webserver will respond back with an acknowledgment so SSL session can proceed. Then after all these steps are complete, encrypted data can now be exchanged between your computer and the webserver. (Dierks)

We are using a free SSL certificate provided by Letsencrypt.

2.7. SMTP – SIMPLE MAIL TRANSFER PROTOCOL

No one is new to the mails in this era of the internet. Everyone has used mails at some point in their life. The **SMTP** protocol is built over **TCP/IP** protocol which is used by most applications for sending emails across the internet. It is mainly used with a few other protocols like POP3 or IMAP because it lacks queue management. The reason for using it with other protocols is because a user can save messages in the mailbox and can download them according to his need or when he logs in. (J. Klensin)

2.8. How SMTP works? (J. Klensin)

Working with SMTP takes in three steps. It works in client-server architecture.

1. First, it uses clients like Gmail or outlook to send messages from client to email server
2. Then email server uses SMTP for sending mail to the receiving server
3. Then after receiving mail at the receiving server, it uses SMTP with IMAP or POP to download the mails and store it to the user's inbox.

2.9. APACHE AND TOMCAT SERVER

Apache and Tomcat are open source HTTP web server (Apache Tomcat Project; Group). They enable us to deploy our web application into them. And using different ports we can access our websites from that server. Apache is mainly used for PHP based applications and Tomcat is used for Java-Based Applications. Tomcat9 is used in this project.

2.10. MySQL

For storing visitors data in our database we are using **MySQL DBMS 5.7**. MySQL is built on top of **SQL**. (MySQL Documentation)

2.11. CLOUD PLATFORMS

For this project, we are using two cloud platforms. One is **Google Cloud Platform (GCP)** and the other is **DigitalOcean**. Our frontend is on GCP and the backend is on DigitalOcean. These platforms allow us to buy storage on the cloud, create virtual machines and it provides a lot of other services as well. (Google Cloud,)

3. TECHNICAL DESCRIPTION OF THE PROOF OF CONCEPT

This section discusses how all the technologies mentioned above are combined together to build a better application that is easy to Build, Deploy and Maintain.

3.1. Website's HTML, CSS & JavaScript

The aim of this project is not about discussing how to write HTML and CSS for a better responsive website instead it focuses more on how to deploy it on network reliably.

Therefore, this project has used a free HTML Template which is taken from online source BOOTSRAPMADE (BOOTSTRAPMADE). Few changes are done according to the needs. This template uses **HTML5**, **CSS3**, and **JavaScript** for its better performance. HTML5 is the latest HTML version released on **28, October 2014** and it is supported by almost all the browsers.

Now that we are ready with a markup of our site we are ready to deploy it on our server.

3.2. DEPLOYING WEB APPLICATION USING DOCKER

For making a website online, which will be accessible for anyone and from everywhere we have to deploy it on the server. A server is nothing but space we own on a cloud, it is also called as **Virtual Machines**.

It is just a computer located somewhere in the world having memory, space, Operating System, etc like our local machine. There are many companies that provide cloud services like **Google Cloud Platform (GCP)**, **Microsoft Azure**, **Amazon Web Services (AWS)**, **DigitalOcean**, etc. While buying a server we can choose our Operating System Like **Ubuntu**, **Windows**, etc. For this project, we have two virtual machines i.e two instances on GCP and DigitalOcean. The frontend of this project is deployed on GCP and the backend is on DigitalOcean.

This project is using the **Apache 2** as a webserver for keeping all the web-related files on a server. As discussed earlier we are using docker for developing our application so let's see step by step guide to install and run the application

1. First, docker has to be installed on your local machine. Depending on the OS you are using you can download the .exe file from the official site of docker or can install it from the command line. At the time of this project, we are using dockers latest version which is **19.03.2** ("Install Docker Desktop on Windows")
2. Once the Docker is installed successfully, we will start building our Dockerfile (Butler). Dockerfile is nothing but a file with a name **Dockerfile** which contains a set of instructions that we want to execute one by one. It may contain installing various software, downloading dependencies and libraries, giving permissions, etc.



```

1 FROM php:7.1-apache
2 RUN apt-get update && \
3     apt-get install -y \
4         zlib1g-dev
5
6 COPY portfolio /var/www/html/portfolio
7 COPY isaurabh /etc/apache2/ssl/isaurabh
8 COPY dev.conf /etc/apache2/sites-enabled/dev.conf
9 RUN docker-php-ext-install mysqli pdo pdo_mysql zip mbstring
10 RUN a2enmod rewrite
11 RUN a2enmod ssl
12 RUN service apache2 restart

```

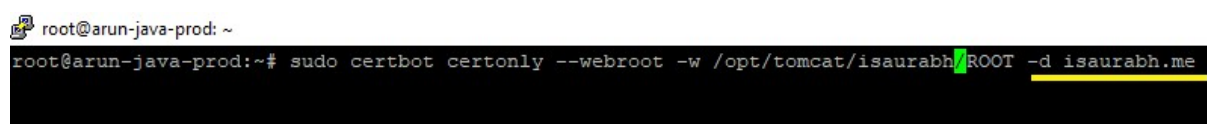
Figure 2 – Dockerfile

This is docker file for our project

First, it downloads the apache server from the docker hub and Runs the command which is written in front of the **RUN** keyword.

After downloading the Apache server, we need to move our website's files to the /var/www/html/portfolio directory which is done by the **COPY** keyword. Because all the files which we see on the browser are shown from html directory of our apache server.

Then we also want to move our SSL files i.e Certificates to our ubuntu server which is inside a directory named isaurabh.



```

root@arun-java-prod: ~
root@arun-java-prod:~# sudo certbot certonly --webroot -w /opt/tomcat/isaurabh/ROOT -d isaurabh.me

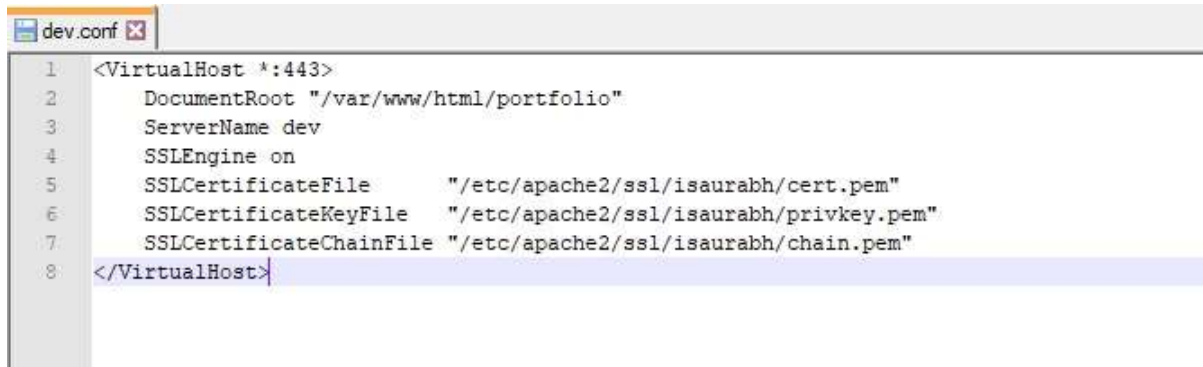
```

Figure 3 - SSL Letsencrypt Certbot Command (Mahdi Mashrur Matin)

We have Obtained SSL certificates i.e cert.pem, chain.pem, fullchain.pem, privkey.pem files using letencrypt's certbot command. (Mahdi Mashrur Matin)

Isaurabh.me is the domain we want to obtain the certificate for and we have to specify the path in which domain resolves to.

If we don't configure ports of our server, the default port is **80** and it will run on http i.e without security. But when we want it to run on https we need to make some modifications to apache's con file. Because https runs on port **443**. Those modifications are written on dev.conf file and we **COPY** that dev.conf file to etc/apache2/sites-enabled/dev.conf directory.



```

1 <VirtualHost *:443>
2     DocumentRoot "/var/www/html/portfolio"
3     ServerName dev
4     SSLEngine on
5     SSLCertificateFile      "/etc/apache2/ssl/isaurabh/cert.pem"
6     SSLCertificateKeyFile   "/etc/apache2/ssl/isaurabh/privkey.pem"
7     SSLCertificateChainFile "/etc/apache2/ssl/isaurabh/chain.pem"
8 </VirtualHost>

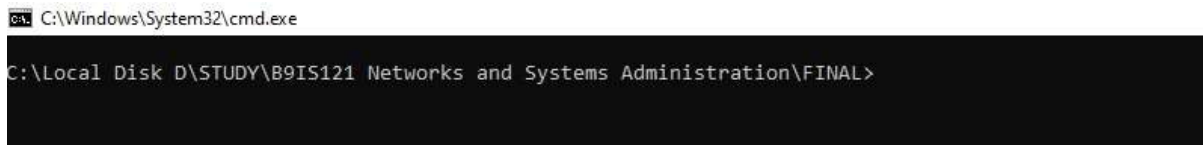
```

Figure 4 - Virtual Host Configuration File

In this conf file, we give the path of our cert.pem, privkey.pem, chain.pem files which are required for https.

3. Once the docker file is ready with all the commands and instructions we will now create a **docker image** (“**Docker Image**”) from the above docker file, and then this docker image will be ready to move on any other server where Docker is installed.

To create image first navigate to the directory where Docker file is placed



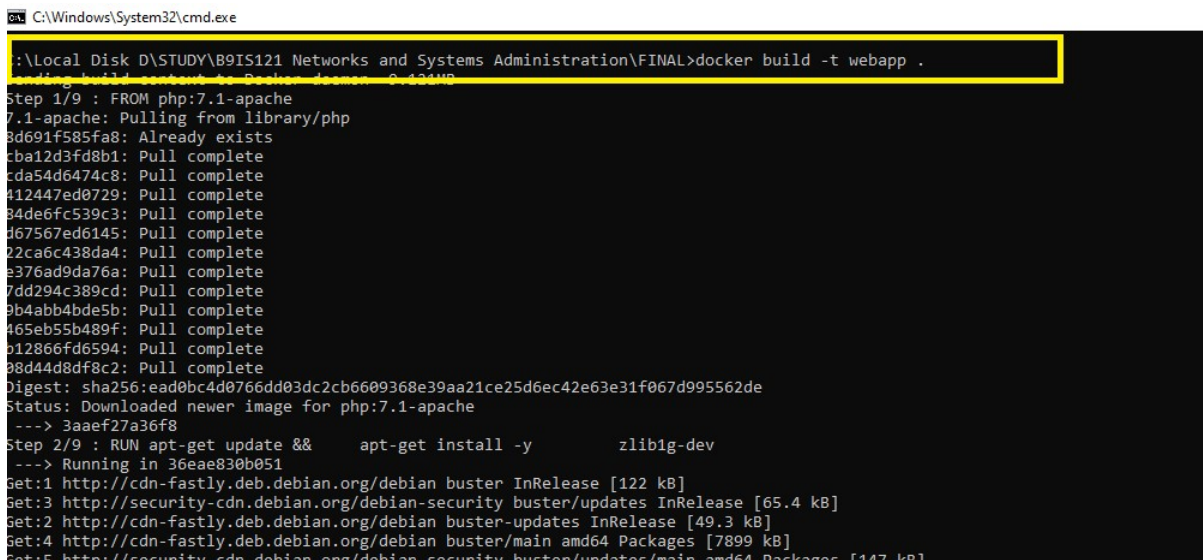
```

C:\Windows\System32\cmd.exe
C:\Local Disk D:\STUDY\B9IS121 Networks and Systems Administration\FINAL>

```

Figure 5 - Directory

After that, we will run a **docker build command** (“**Docker Build**”) to create a docker image



```

C:\Windows\System32\cmd.exe
C:\Local Disk D:\STUDY\B9IS121 Networks and Systems Administration\FINAL>docker build -t webapp .
Sending build context to Docker daemon 0.438MB
Step 1/9 : FROM php:7.1-apache
7.1-apache: Pulling from library/php
8d691f585fa8: Already exists
c8a12d3fd8b1: Pull complete
cda54d6474c8: Pull complete
412447ed0729: Pull complete
34de6fc539c3: Pull complete
d67567ed6145: Pull complete
22ca6c438da4: Pull complete
e376ad9da76a: Pull complete
7dd294c389cd: Pull complete
9b4abb4bde5b: Pull complete
465eb55b489f: Pull complete
b12866fd6594: Pull complete
88d44d8df8c2: Pull complete
Digest: sha256:ead0bc4d0766dd03dc2cb6609368e39aa21ce25d6ec42e63e31f067d995562de
Status: Downloaded newer image for php:7.1-apache
--> 3aaf27a36f8
Step 2/9 : RUN apt-get update && apt-get install -y zlib1g-dev
--> Running in 36eae830b051
Get:1 http://cdn-fastly.deb.debian.org/debian buster InRelease [122 kB]
Get:3 http://security-cdn.debian.org/debian-security buster-updates InRelease [65.4 kB]
Get:2 http://cdn-fastly.deb.debian.org/debian buster-updates InRelease [49.3 kB]
Get:4 http://cdn-fastly.deb.debian.org/debian buster/main amd64 Packages [7899 kB]
Get:5 http://security-cdn.debian.org/debian-security buster-updates/main amd64 Packages [147 kB]

```

Figure 6 - Docker Build Command

As you can see as soon as we run the docker build command all the dependencies get downloaded and one image gets created with some random unique id. We can pass the parameter for mentioning the name of our image.

```
C:\Windows\System32\cmd.exe
C:\Local Disk D:\STUDY\B9IS121 Networks and Systems Administration\FINAL>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
webapp               latest             12d7abb1da7f       10 minutes ago     432MB
php                  7.1-apache         3aaef27a36f8       2 weeks ago        402MB
```

Figure 7 - Docker Images Command

Now that we have generated an image from our docker file we will run that image and create a container using the **docker run** (“Docker Image Push”) command.

```
C:\Windows\System32\cmd.exe
C:\Local Disk D:\STUDY\B9IS121 Networks and Systems Administration\FINAL>docker run -d -p 80:80 -p 443:443 12d7abb1da7f
5707211b5bba9c3c61abcc072d2669b8e76f8b07712423a161713ebfb6d22547
C:\Local Disk D:\STUDY\B9IS121 Networks and Systems Administration\FINAL>
```

Figure 8 - Docker Run Command

What docker run command does is it creates a container out of the image which we are specifying. It also takes parameters -p for running that container on specified port Here we want to run it on 80 as well as 443 port for SSL so we are specifying two ports.

```
C:\Windows\System32\cmd.exe
C:\Local Disk D:\STUDY\B9IS121 Networks and Systems Administration\FINAL>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
16041a2af538       12d7abb1da7f       "docker-php-entryp..." 6 seconds ago       Up 4 seconds        0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp  wizardly_pare
```

Figure 9 - Docker Ps Command

Now, our container is working and to test this we go to the browser and see if we can see our website form 80 and 443 port

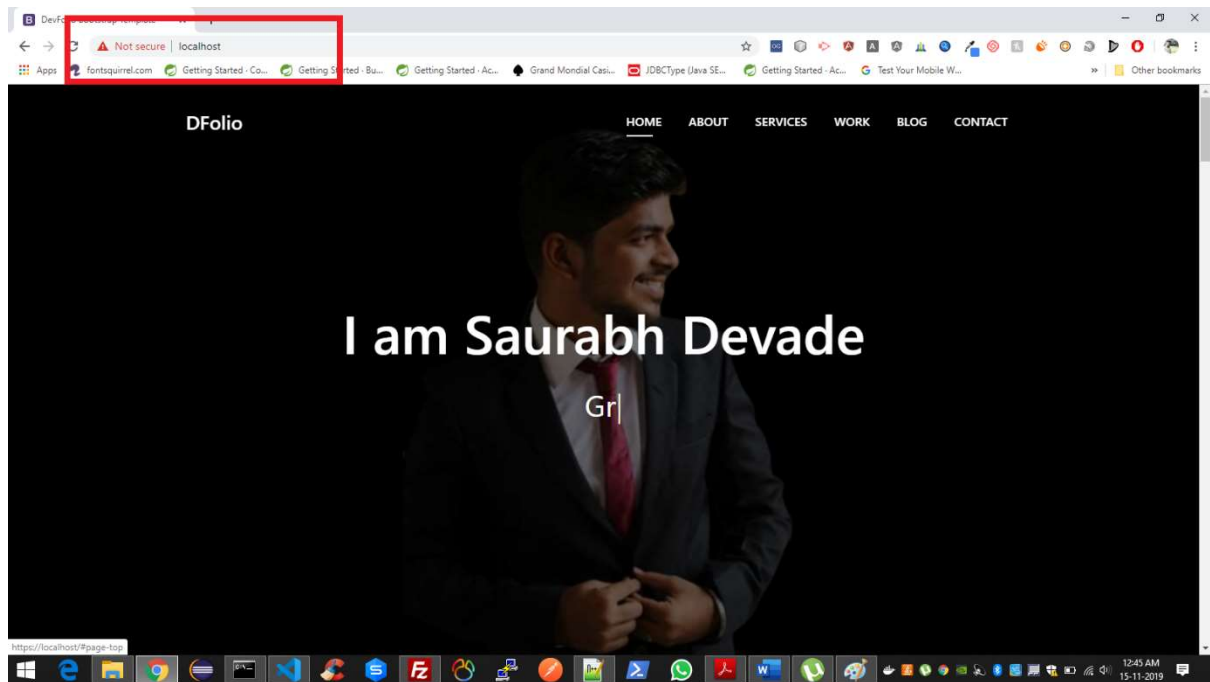


Figure 10 - Localhost Website

If the container is running properly our site should be visible on **https://localhost** and **http://localhost**. Port 80 and 443 are not visible because those are by default hidden by the browser. The browser showing that the site is not secured because we have obtained the certificate for the **isaurabh.me** domain and not for the localhost domain.

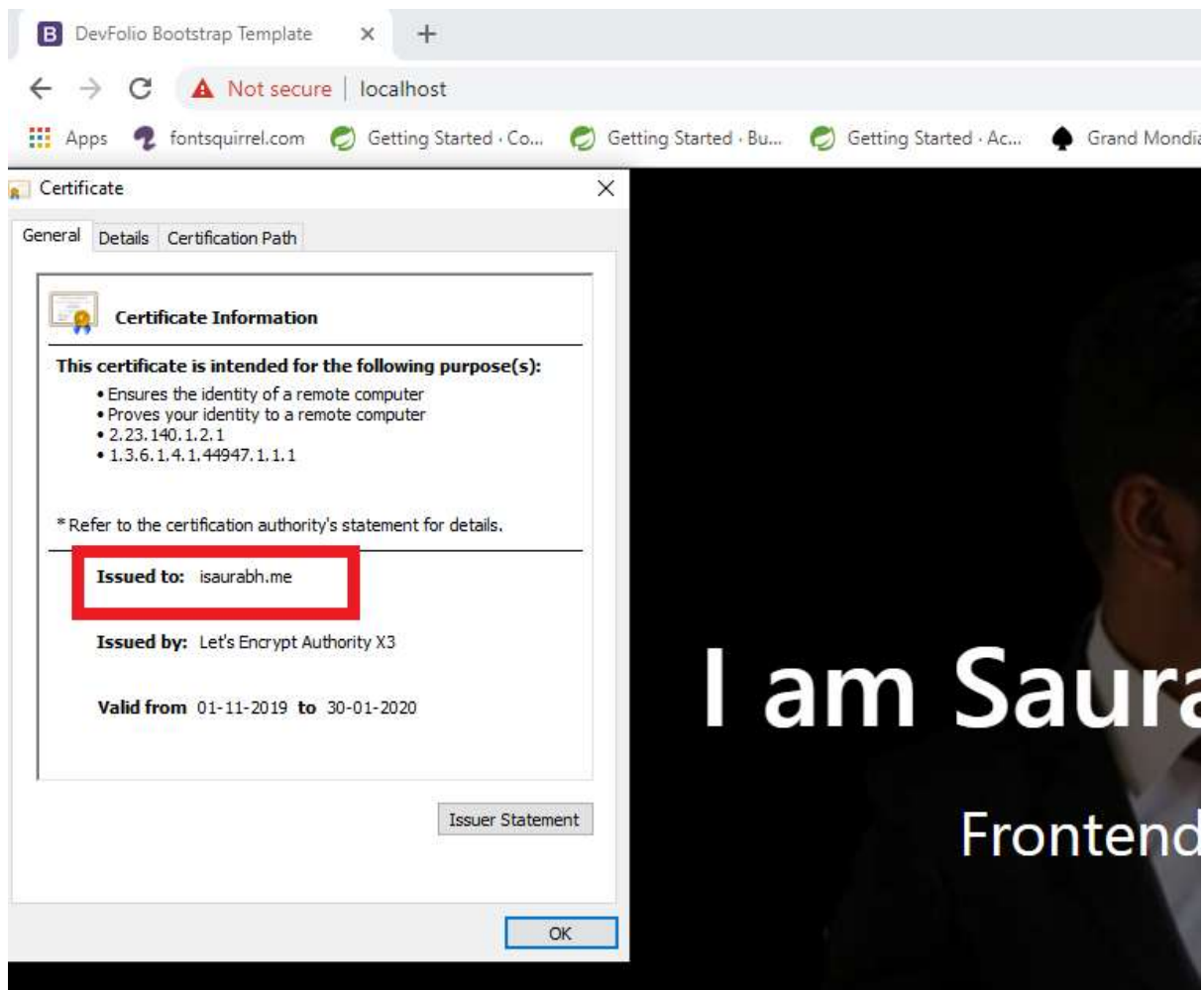


Figure 11 - SSL Certificate Details

The next step is to push the docker image to the server. So that you or anyone else can just pull your image and application will be up and ready. In this project, we are pushing images so we can replicate the development environment to the production environment as it without any extra efforts for the development environment.

Let's see how to push the image to the docker hub.

For pushing an image to the docker hub you should have an account on docker hub and using that account you must be logged in on the system where you want to pull and run that images. First, you need to add tag to images using **docker tag** ("Docker Tag") command and the **docker push** ("Docker Image Push") command is used for pushing the existing image to the Docker hub,

```
C:\Local Disk D\STUDY\B9IS121 Networks and Systems Administration\FINAL>docker tag webapp saurabh1995/webapp
C:\Local Disk D\STUDY\B9IS121 Networks and Systems Administration\FINAL>docker push saurabh1995/webapp
```

Figure 12 - Docker Tag and Docker Push Command

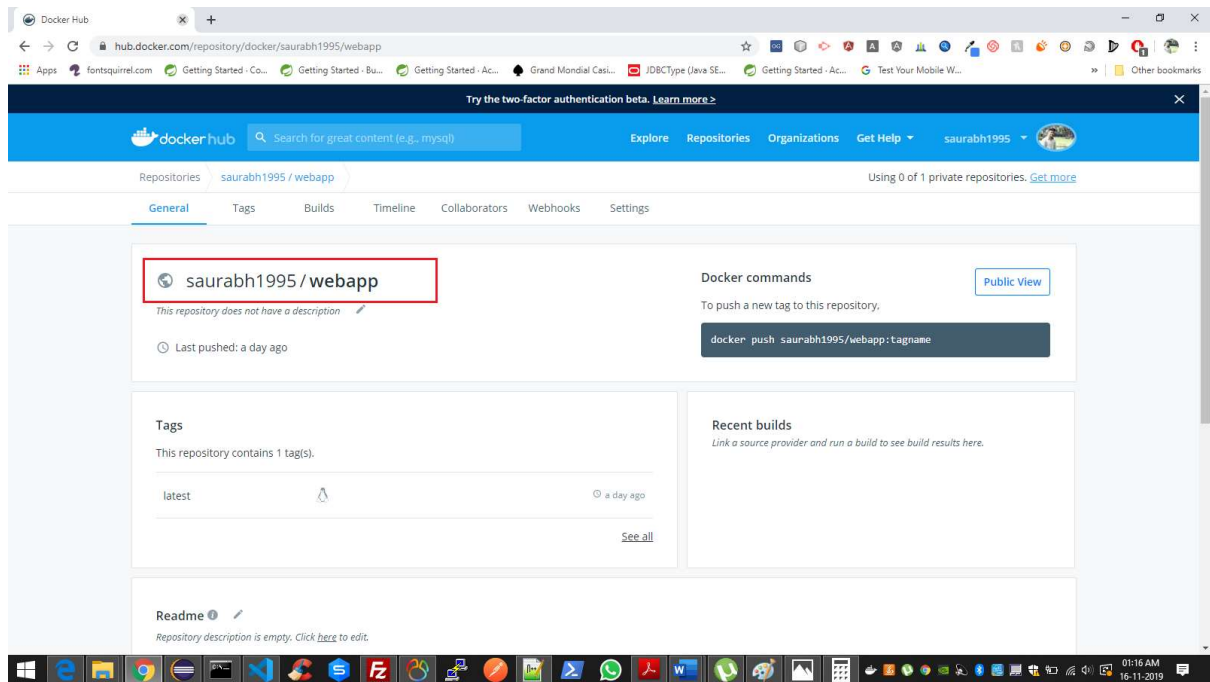


Figure 13 - Docker Hub Repository

The process for creating a docker image for our application is done now. We will now run in on a production server and check if our website opens with the IP of that server. For that, we have to log in to our virtual machine which is on GCP through ssh. Then run command **docker pull saurabh1995/webapp** (“Docker Pull”) for downloading images from the docker hub.

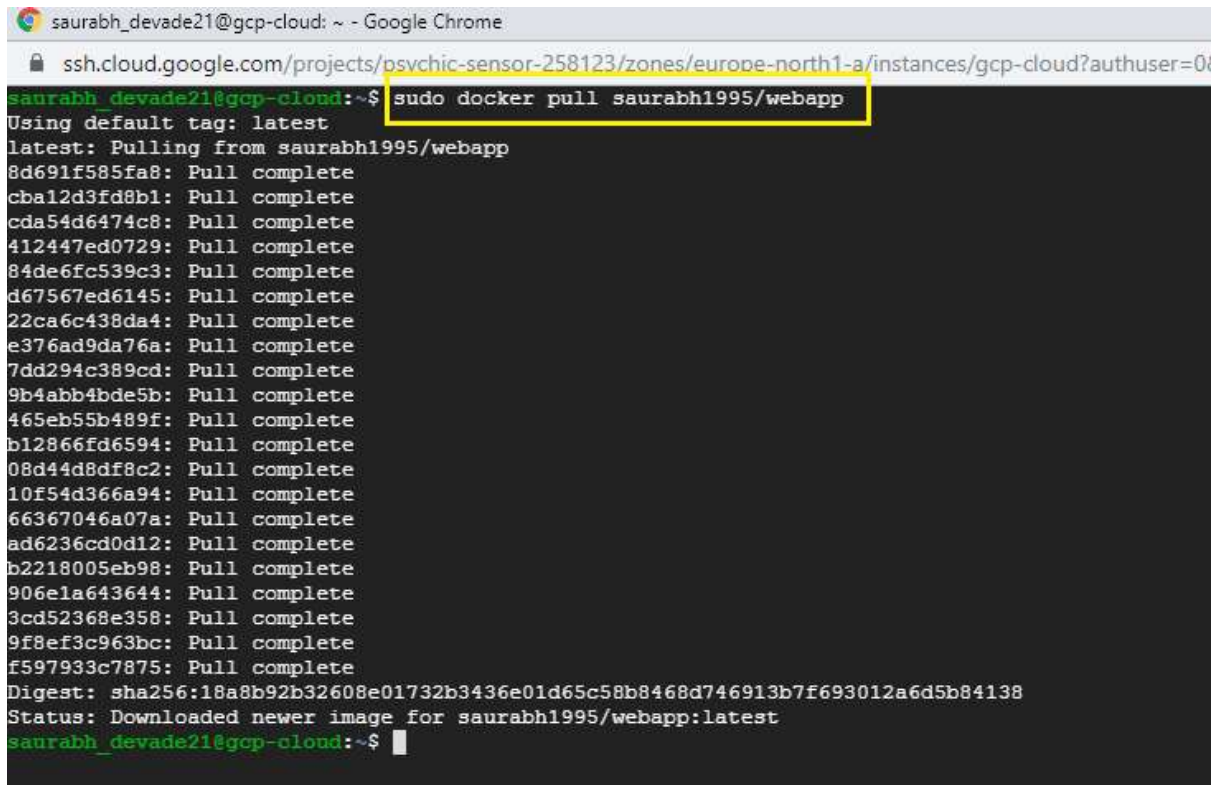
A screenshot of a terminal window titled 'saurabh_devade21@gcp-cloud: ~ - Google Chrome'. The address bar shows 'ssh.cloud.google.com/projects/psvchic-sensor-258123/zones/europe-north1-a/instances/gcp-cloud?authuser=0'. The terminal shows the command 'sudo docker pull saurabh1995/webapp' being executed. The output indicates that the latest tag is being pulled from the repository. A list of 20 layers is shown, each with a digest and the status 'Pull complete'. The final digest is 'sha256:18a8b92b32608e01732b3436e01d65c58b8468d746913b7f693012a6d5b84138'. The status is 'Downloaded newer image for saurabh1995/webapp:latest'. The prompt returns to 'saurabh_devade21@gcp-cloud:~\$'.

Figure 14 - Docker Pull Command

From logs in the screenshot, we can see that the image is downloaded.

Now for running the image, we will execute the same command which we did in localhost earlier **docker run -d -p 80:8080 -p 443:443 webapp** (“Docker Run”).

Once this command runs successfully our website should be working on **https://isaurabh.me**. i.e it works on 443 port using SSL certificate also, it doesn't show certificate invalid error the way it used to show on our local machine. It shows the padlock icon indicating the site is secure.

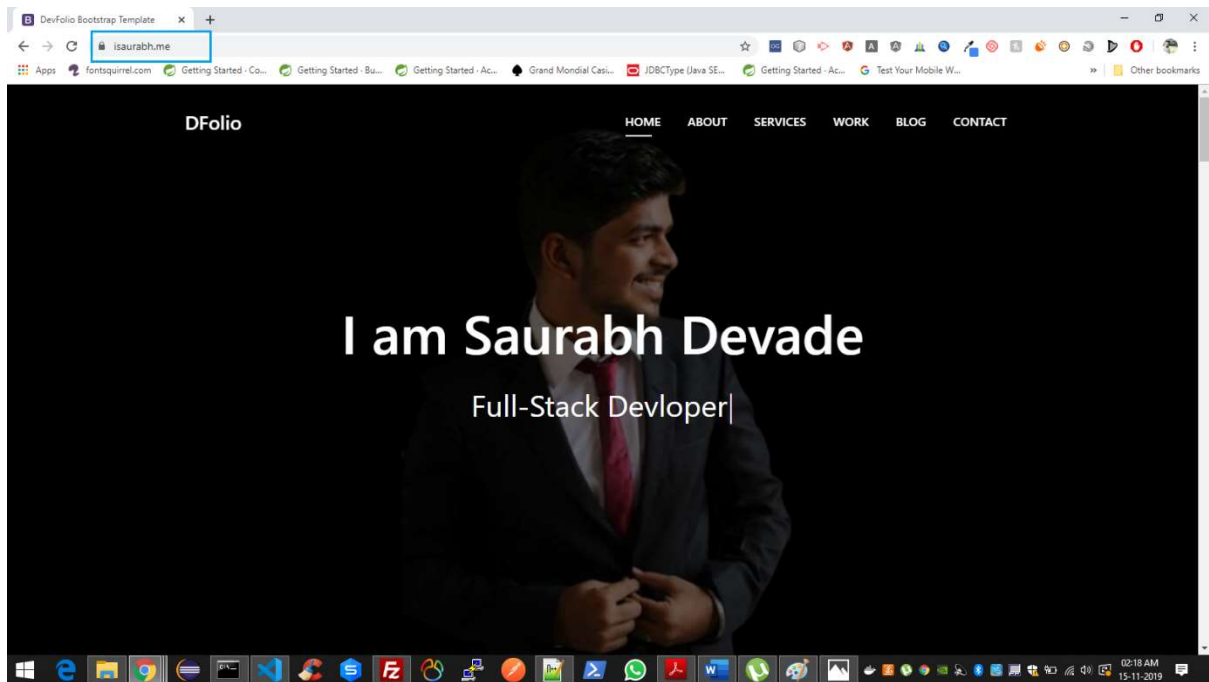


Figure 15 - Live Site

3.3. MAPPING OF DOMAIN NAME TO IP ADDRESS

To map the domain name, In this case, **isaurabh.me** to IP address **35.228.90.221** we need to add the entry of our IP address in the **A RECORDS** of DNS management section of the domain name which is provided by the domain name provider in this case **BIGROCK** (“Managing DNS Resource Records | KnowledgeBase”).

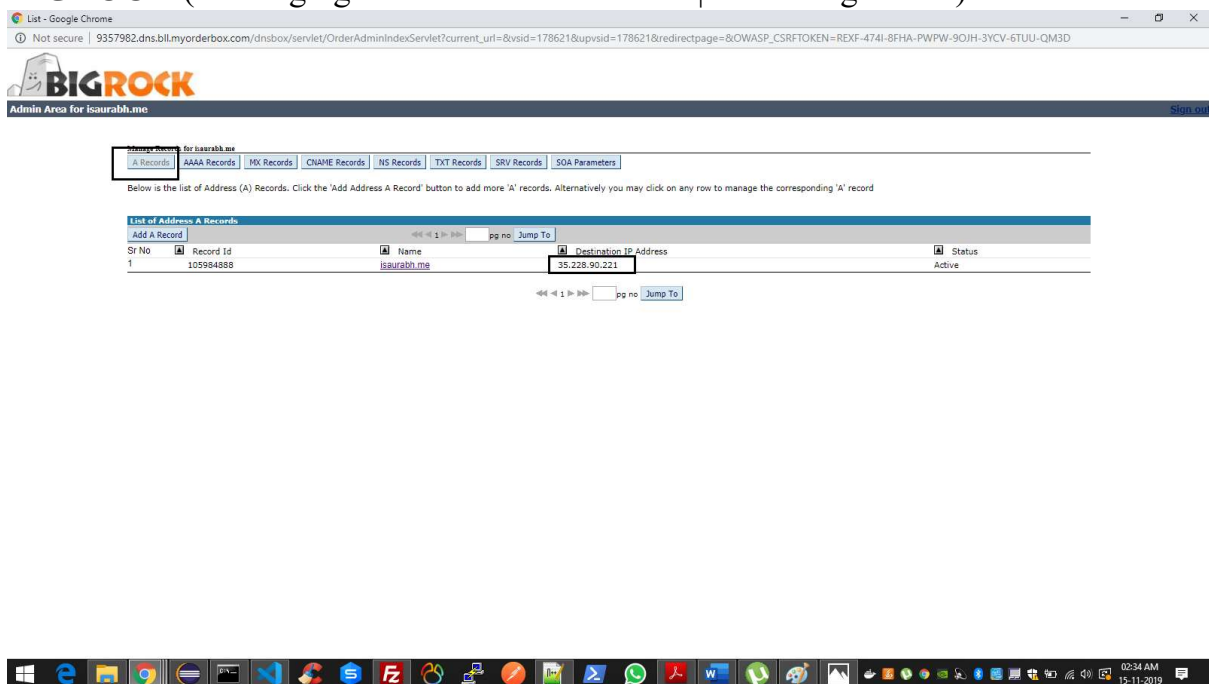
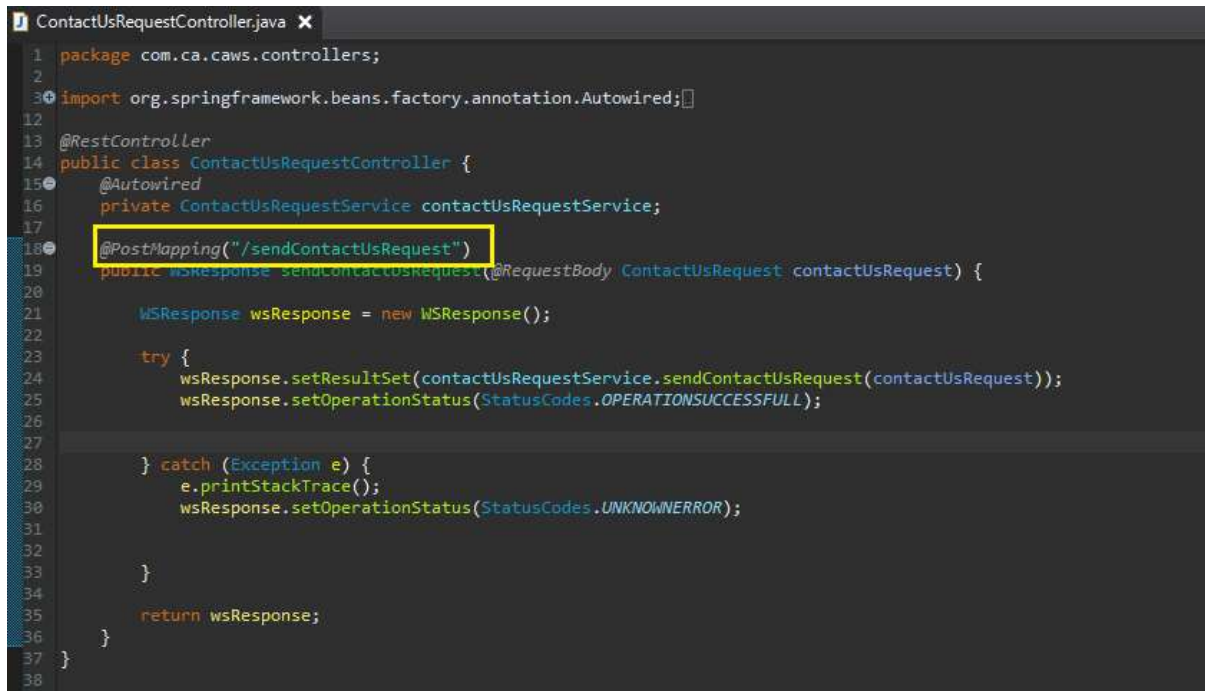


Figure 16 - Big Rock DNS Management

Now we will proceed to see how we will send emails using SMTP, who visit our site for business and store their data in our database for further communication. We will also discuss how we can establish a connection between the two servers.

3.4. COMMUNICATION BETWEEN TWO SERVERS AND SENDING EMAILS USING AWS'S SES AND SMTP PROTOCOL

Our one server is on **GCP** and the second server is on **DigitalOcean**. So, to establish communication between two servers we are using **JAVA** based **API** (Java API).

A screenshot of a code editor showing the implementation of a Java REST controller. The file is named 'ContactUsRequestController.java'. The code includes package declarations, imports for Spring's @Autowired and @RestController, and a class definition. Inside the class, there is an @Autowired field for 'ContactUsRequestService' and a @PostMapping method for '/sendContactUsRequest'. The method takes a 'ContactUsRequest' object as a request body and returns a 'WSResponse'. The method body creates a new 'WSResponse' object, calls 'sendContactUsRequest' on the service, sets the result set and operation status to 'OPERATIONSUCCESSFULL', and returns the response. A catch block handles exceptions by printing the stack trace and setting the operation status to 'UNKNOWNERROR'.

```
1 package com.ca.caws.controllers;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @RestController
6 public class ContactUsRequestController {
7     @Autowired
8     private ContactUsRequestService contactUsRequestService;
9
10    @PostMapping("/sendContactUsRequest")
11    public WSResponse sendContactUsRequest(@RequestBody ContactUsRequest contactUsRequest) {
12
13        WSResponse wsResponse = new WSResponse();
14
15        try {
16            wsResponse.setResultSet(contactUsRequestService.sendContactUsRequest(contactUsRequest));
17            wsResponse.setOperationStatus(StatusCodes.OPERATIONSUCCESSFULL);
18
19        } catch (Exception e) {
20            e.printStackTrace();
21            wsResponse.setOperationStatus(StatusCodes.UNKNOWNERROR);
22
23        }
24
25        return wsResponse;
26    }
27 }
```

Figure 17 - Java Controller Code

The IP of this server is **139.59.59.220**. If we want to communicate with this server we have to call API deployed on this server. For example, we want to call the **sendContactUsRequest** API which will store the visitor's details in our database and send the email to the visitor as well as the owner of the site saying that new inquiry is registered.

We have to call the API by using the IP of that server followed by the API URL for eg. **http://139.59.59.220/sendContactUsRequest** and we have to pass the request body which contains visitor name, email, etc. Once this server receives the request it first stores the data in the database named **portfoliodb** which we have created. This project uses **MYSQL DBMS** for storing data.

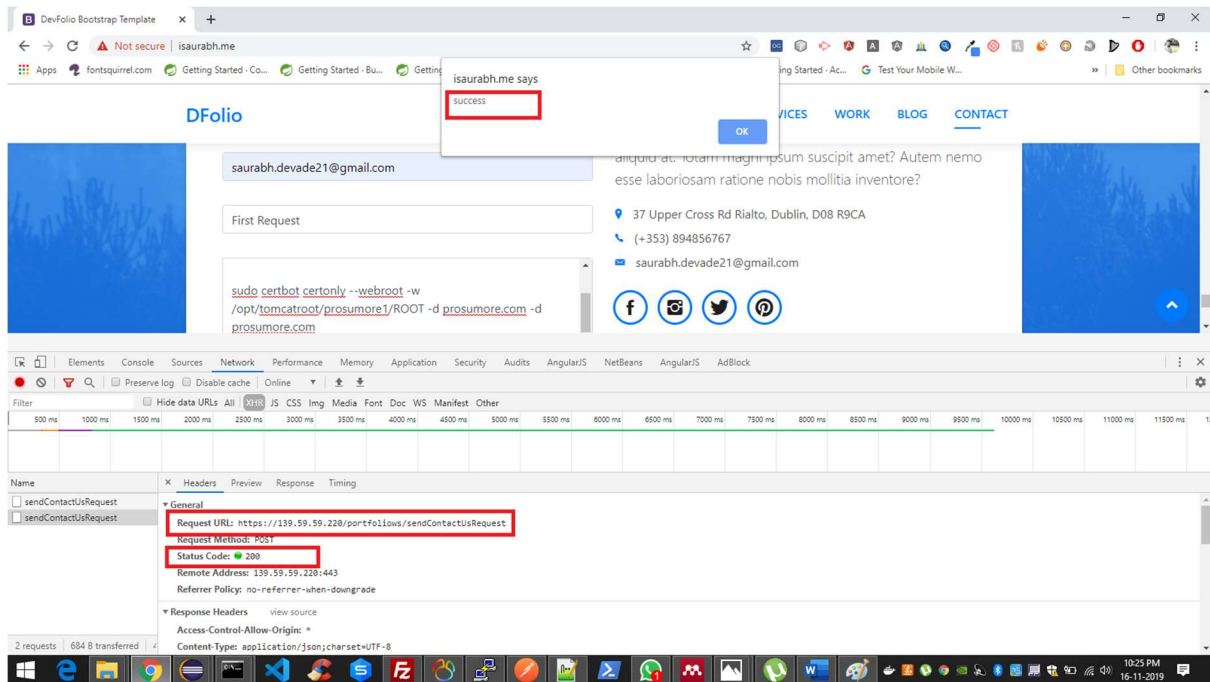


Figure 18 - Inspect Element Snippet

IF we go to the XHR section in the inspect element in the browser, we can see the one request is called to the other server using the IP of that server which is nothing but our API and status code is **200** i.e success code.

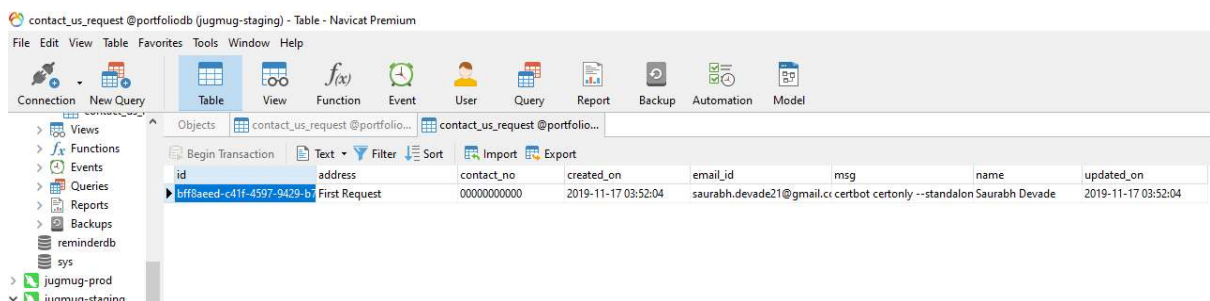


Figure 19 - Database Snippet

Once the data is stored we want to send emails to the visitors as well as the owner of the site. For that, we are using **AWS Simple Email Service**. For using SES we have to whitelist our email address from AWS.

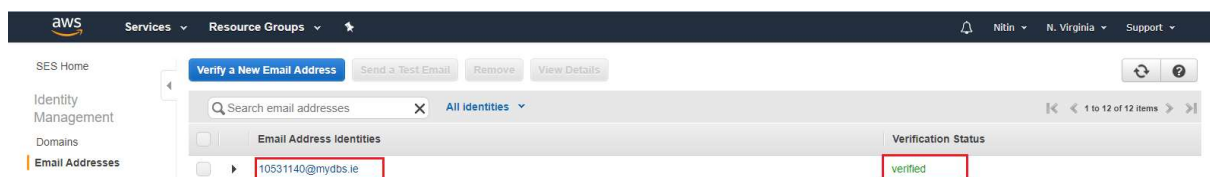
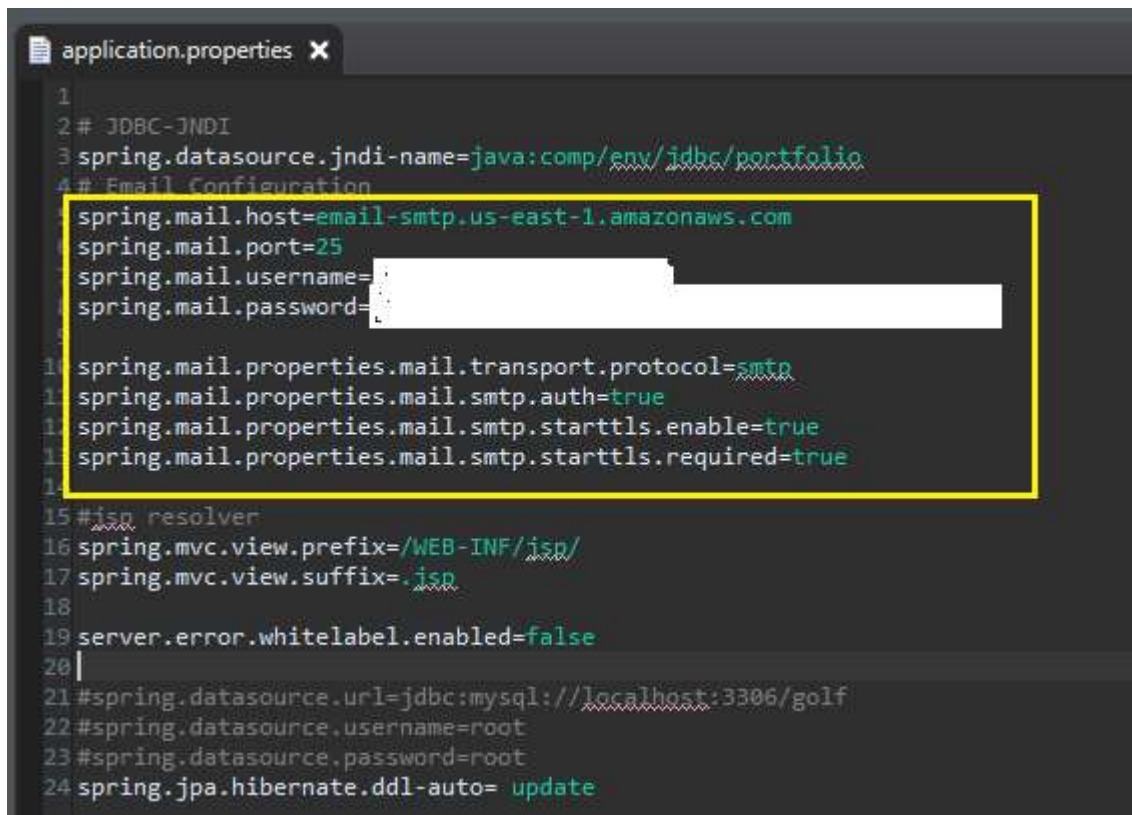


Figure 20 - AWS SES Service

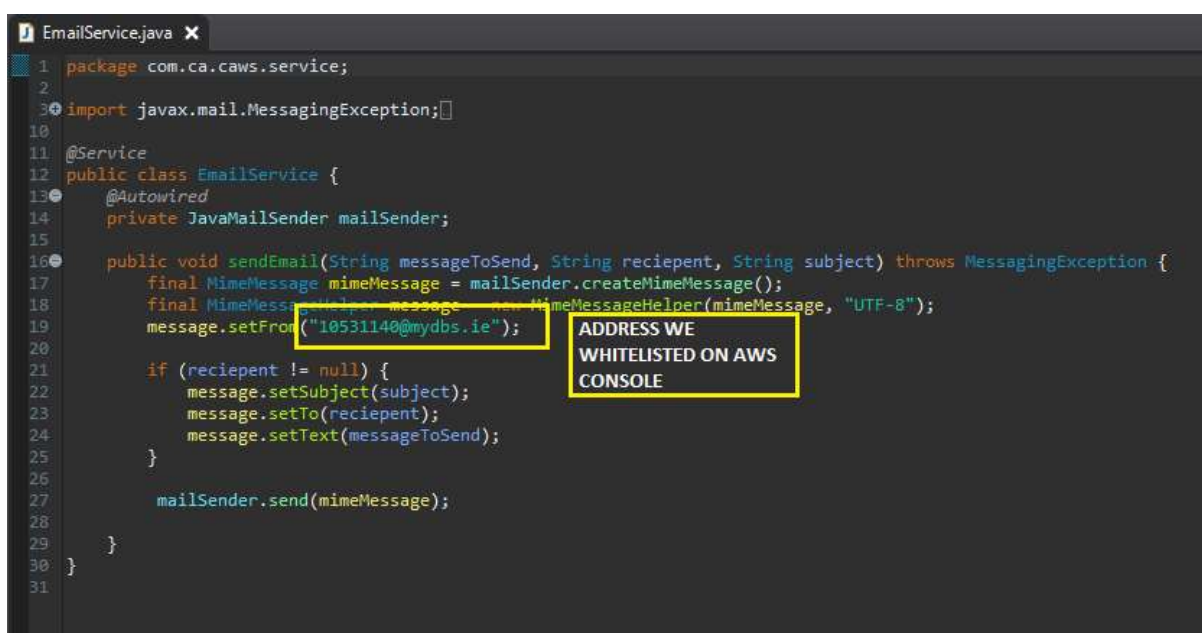
Once the email is verified we get the SMTP port, host, other settings, and credentials which we put in our code's **.properties** file

A screenshot of a code editor showing the 'application.properties' file. The file contains various Spring configuration properties. A yellow rectangular box highlights the email configuration section, which includes the SMTP host, port, username, and password. The password field is obscured by a white redaction box. Other visible properties include JDBC settings, MVC view settings, and server error handling.

```
1
2 # JDBC-JNDI
3 spring.datasource.jndi-name=java:comp/env/jdbc/portfolio
4 # Email Configuration
5 spring.mail.host=email-smtp.us-east-1.amazonaws.com
6 spring.mail.port=25
7 spring.mail.username=[REDACTED]
8 spring.mail.password=[REDACTED]
9
10 spring.mail.properties.mail.transport.protocol=smtp
11 spring.mail.properties.mail.smtp.auth=true
12 spring.mail.properties.mail.smtp.starttls.enable=true
13 spring.mail.properties.mail.smtp.starttls.required=true
14
15 #jsp resolver
16 spring.mvc.view.prefix=/WEB-INF/jsp/
17 spring.mvc.view.suffix=.jsp
18
19 server.error.whitelabel.enabled=false
20
21 #spring.datasource.url=jdbc:mysql://localhost:3306/golf
22 #spring.datasource.username=root
23 #spring.datasource.password=root
24 spring.jpa.hibernate.ddl-auto= update
```

Figure 21 - application.properties file

Using these settings and JavaMailSender we send emails to the visitor.

A screenshot of a code editor showing the 'EmailService.java' file. The code defines a service class for sending emails. A yellow rectangular box highlights the 'setFrom' method call in the 'sendEmail' function, which uses a specific email address. A callout box points to this address, stating it is whitelisted on the AWS console. The code also shows the creation of a MimeMessage and the use of a JavaMailSender interface.

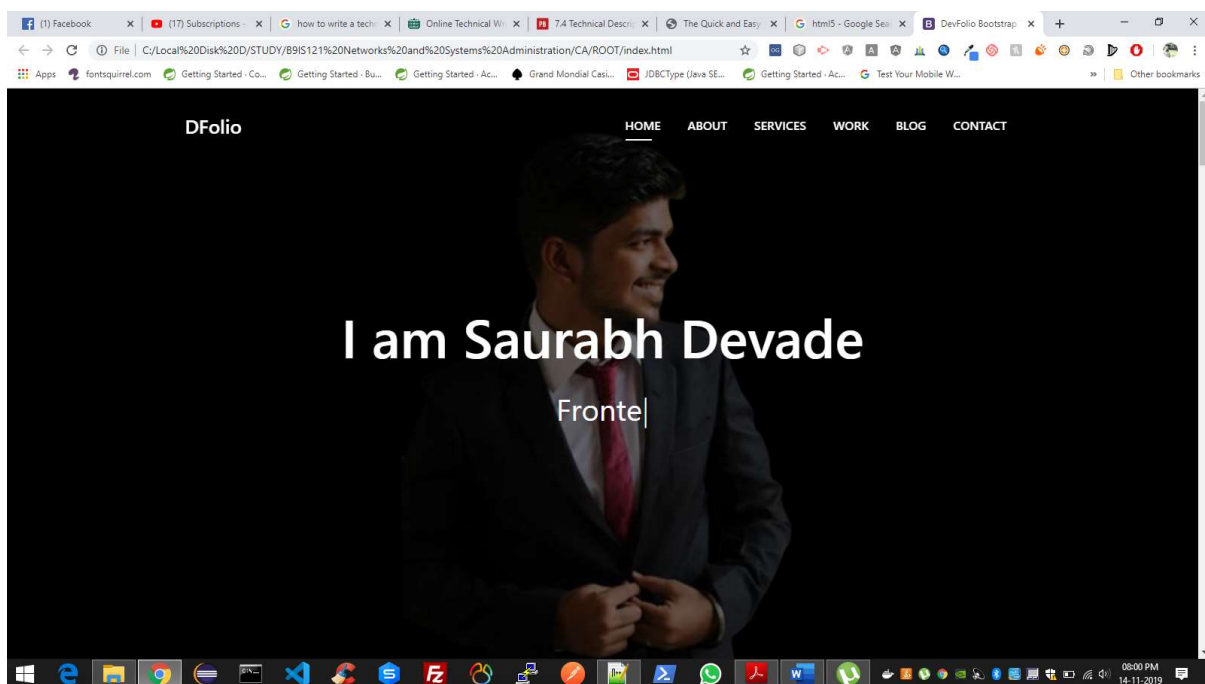
```
1 package com.ca.caws.service;
2
3 import javax.mail.MessagingException;
4
5
6 @Service
7 public class EmailService {
8     @Autowired
9     private JavaMailSender mailSender;
10
11     public void sendEmail(String messageToSend, String recipient, String subject) throws MessagingException {
12         final MimeMessage mimeMessage = mailSender.createMimeMessage();
13         final MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, "UTF-8");
14         message.setFrom("10531140@mydbs.ie");
15
16         if (recipient != null) {
17             message.setSubject(subject);
18             message.setTo(recipient);
19             message.setText(messageToSend);
20         }
21
22         mailSender.send(mimeMessage);
23     }
24 }
25
26
27
28
29
30
31
```

Figure 22 - Java Mail Service Code

Now that we have built and deployed our network, let's test it whether it's working properly

4. TESTING, EVALUATION & DEMONSTRATION OF PROOF OF CONCEPT

4.1. Testing & Evaluation whether HTML is working correctly - As shown in the below Screenshot if we open index.html file in the browser it shows proper site that means markup is working correctly.



4.2. Testing Docker images - Now you can see the newly created image with the name webapp by typing **docker images** command. It indicates that the docker image is created successfully. Shown below.

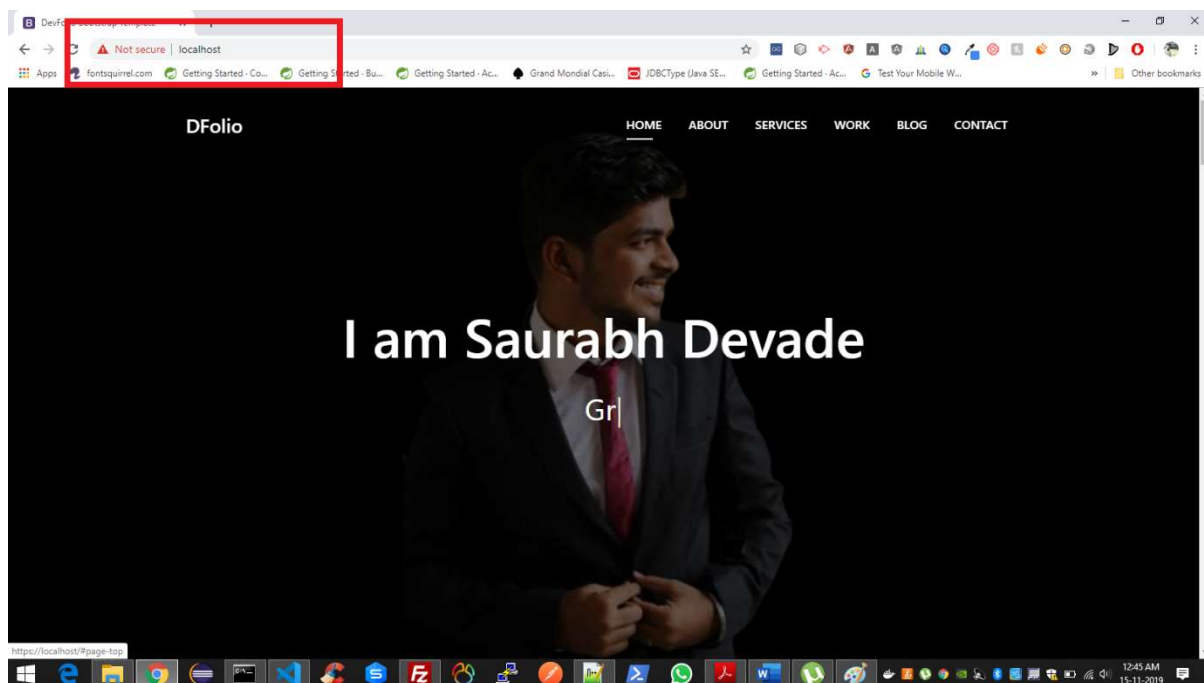
```
C:\Windows\System32\cmd.exe

C:\Local Disk D\STUDY\B9IS121 Networks and Systems Administration\FINAL>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
webapp               latest              12d7abb1da7f       10 minutes ago     432MB
php                  7.1-apache         3aaet27a36f8       2 weeks ago        402MB
```

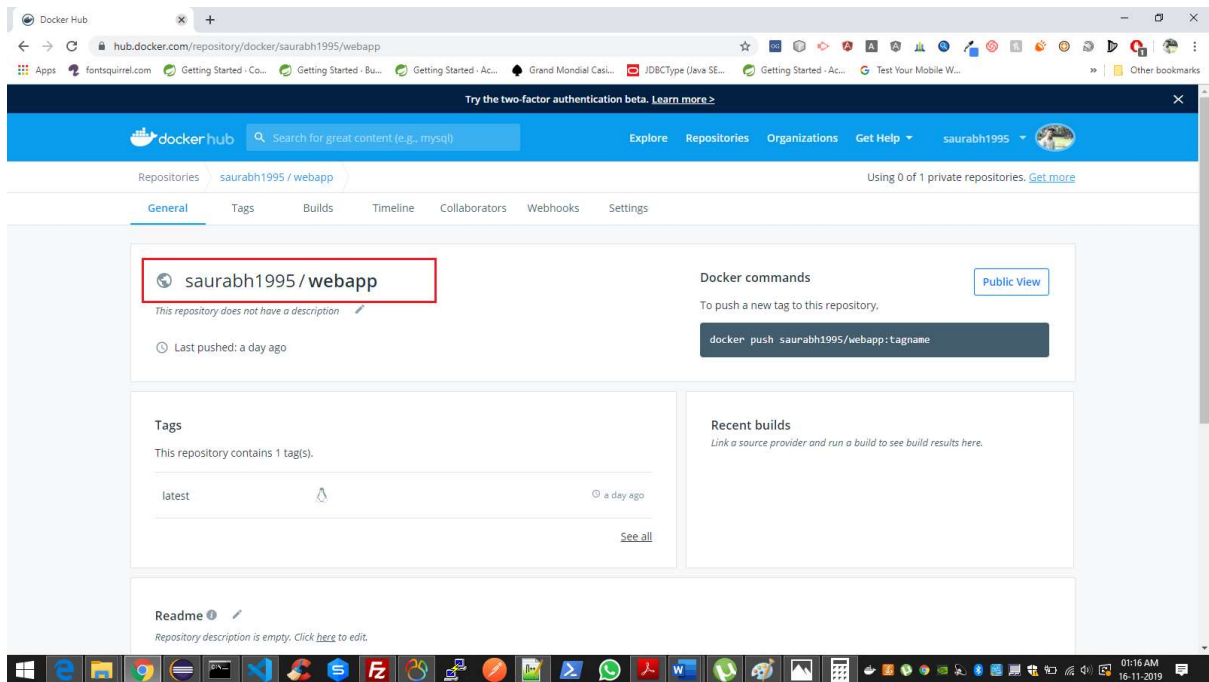
4.3. Testing Docker Container – If we run command **docker ps**, it will show all working containers. As we can see our container is working this test is successful as well.

```
C:\Windows\System32\cmd.exe
C:\Local Disk D:\STUDY\B9IS121 Networks and Systems Administration\FINAL>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
6041a2af538        12d7abb1da7f       "docker-php-entryp..." 6 seconds ago       Up 4 seconds       0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp  wizardly_pare
```

4.4. Testing whether our container is running properly – If the container is running properly our site should be visible on **https://localhost** and **http://localhost**. Port 80 and 443 are not visible because those are by default hidden by the browser. The browser showing that the site is not secured because we have obtained the certificate for our isaurabh.me domain and not for the localhost domain. This test is also executed successfully.



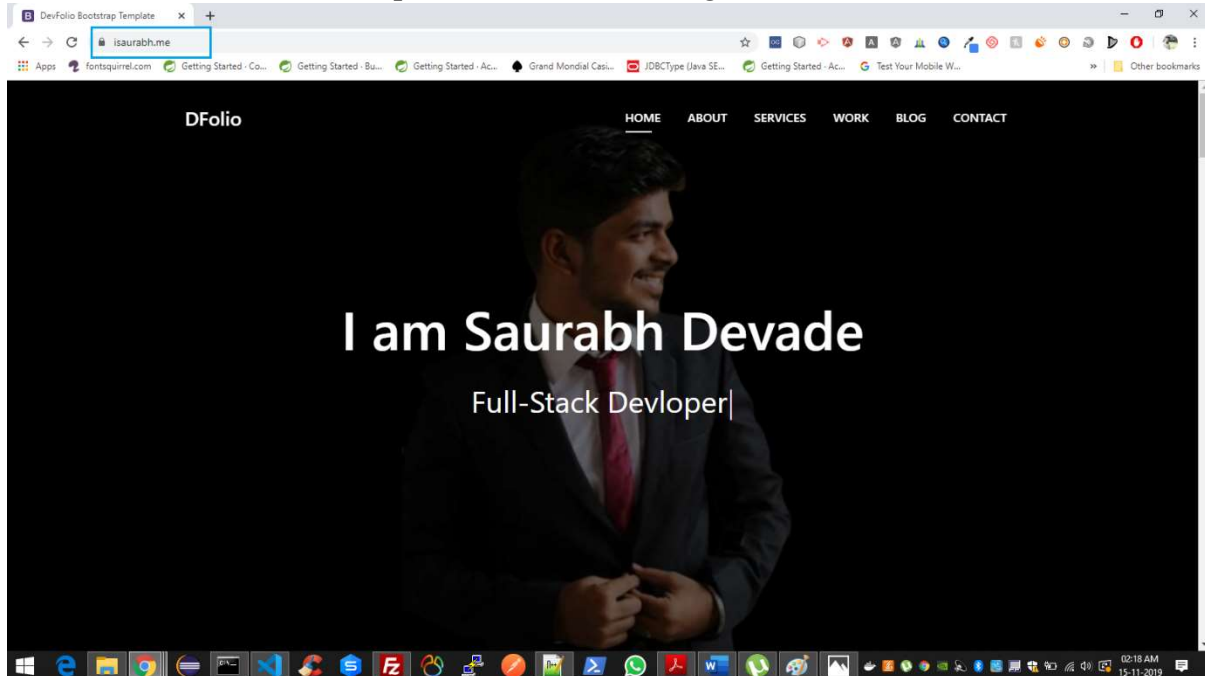
4.5. Testing image is pushed on docker hub – If we can see the image on our docker hub account then we can assume that this test runs successfully.



4.6. Testing docker hub image on production server - From logs in the screenshot, we can see that image is downloaded

```
saurabh_devade21@gcp-cloud: ~ - Google Chrome
ssh.cloud.google.com/projects/psvchic-sensor-258123/zones/europe-north1-a/instances/gcp-cloud?authuser=0
saurabh_devade21@gcp-cloud:~$ sudo docker pull saurabh1995/webapp
Using default tag: latest
latest: Pulling from saurabh1995/webapp
8d691f585fa8: Pull complete
cba12d3fd8b1: Pull complete
cda54d6474c8: Pull complete
412447ed0729: Pull complete
84de6fc539c3: Pull complete
d67567ed6145: Pull complete
22ca6c438da4: Pull complete
e376ad9da76a: Pull complete
7dd294c389cd: Pull complete
9b4abb4bde5b: Pull complete
465eb55b489f: Pull complete
b12866fd6594: Pull complete
08d44d8df8c2: Pull complete
10f54d366a94: Pull complete
66367046a07a: Pull complete
ad6236cd0d12: Pull complete
b2218005eb98: Pull complete
906e1a643644: Pull complete
3cd52368e358: Pull complete
9f8ef3c963bc: Pull complete
f597933c7875: Pull complete
Digest: sha256:18a8b92b32608e01732b3436e01d65c58b8468d746913b7f693012a6d5b84138
Status: Downloaded newer image for saurabh1995/webapp:latest
saurabh_devade21@gcp-cloud:~$
```


4.7. Testing if sites work properly on production server - Once **docker run -d -p 80:8080 -p 443:443 webapp** command runs successfully our website should be working on **https://isaurabh.me**. i.e it works on 443 port using SSL certificate also, it doesn't show certificate invalid error the way it used to show on our local machine. It shows the padlock icon indicating the site is secure.



4.8. Testing of domain Mapping – If we type ping command for our domain and if it shows IP of our server then it is properly mapped to our server. From the screenshot below we can say our this test is working

```
C:\Windows\system32\cmd.exe

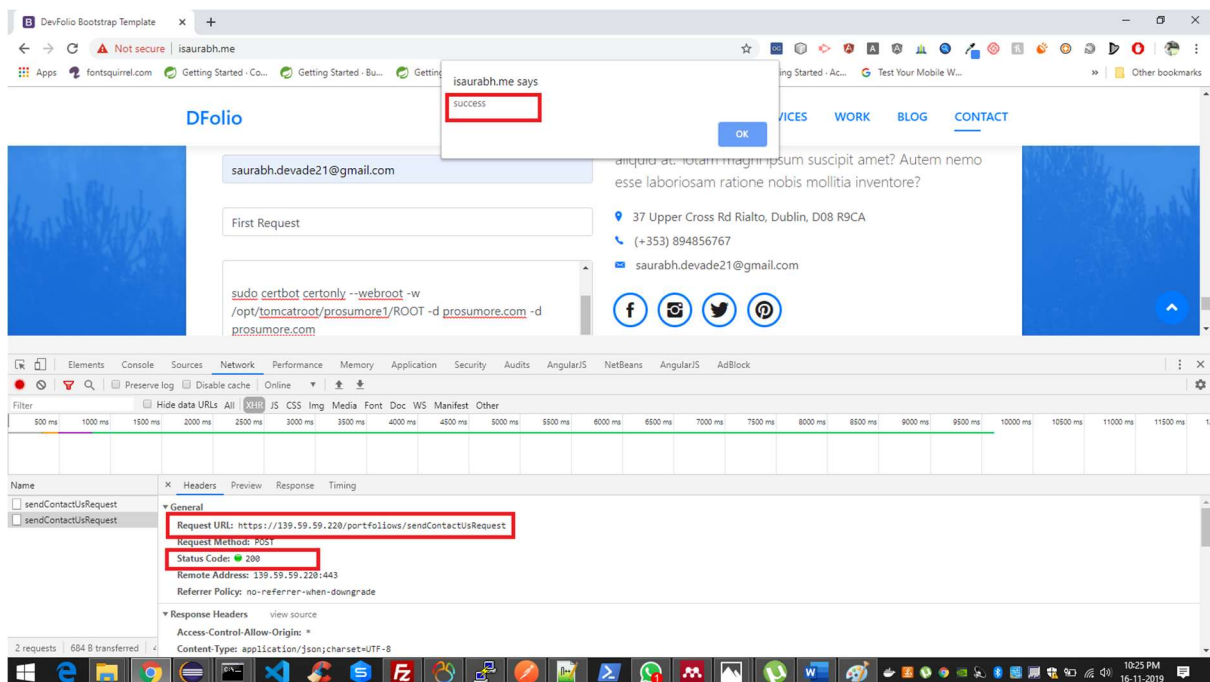
C:\Users\Saurabh>ping isaurabh.me

Pinging isaurabh.me [35.228.90.221] with 32 bytes of data:
Reply from 35.228.90.221: bytes=32 time=58ms TTL=55
Reply from 35.228.90.221: bytes=32 time=58ms TTL=55
Reply from 35.228.90.221: bytes=32 time=61ms TTL=55
Reply from 35.228.90.221: bytes=32 time=58ms TTL=55

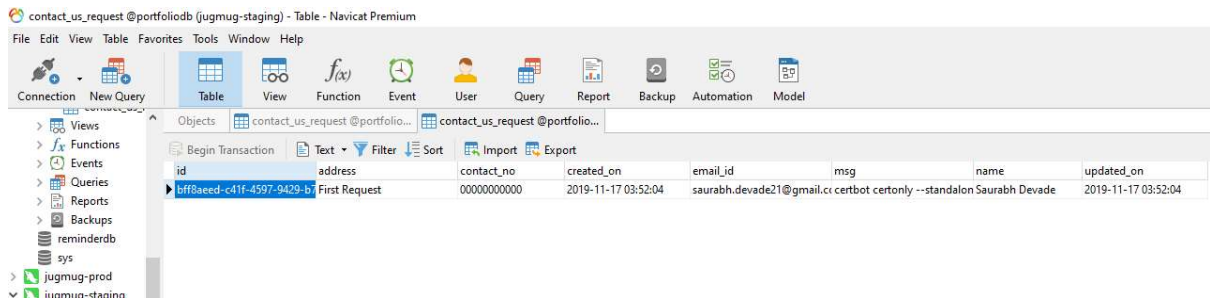
Ping statistics for 35.228.90.221:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 58ms, Maximum = 61ms, Average = 58ms
```

Figure 23 - Ping Command

4.9. Testing whether API is successfully and data is stored



From the screenshot below, we can see that the data which we entered into form is stored successfully.



4.10. Testing whether API is successfully called on other servers and email is sent successfully using SMTP.

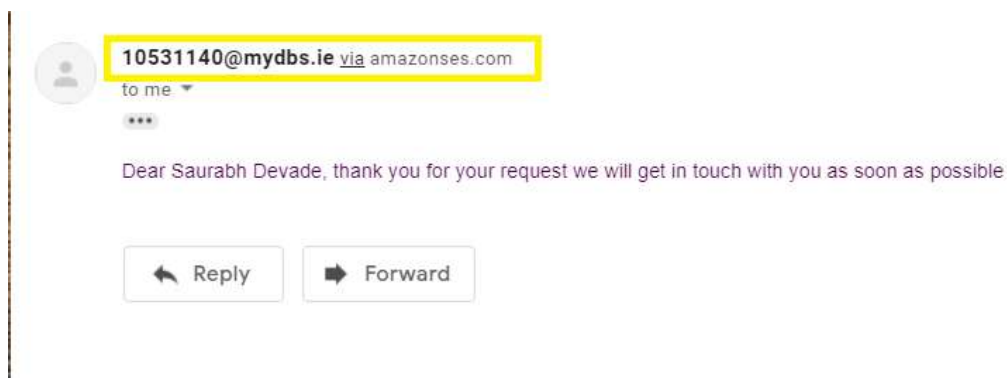


Figure 24 - Received Mail Snippet

From the screenshot, we can see that the visitor has received the success message from the email address we registered on AWS. So this test also runs successfully.

5. Conclusion

From the status of all the test cases above, finally, we can conclude that our Artifact is working as we expected.

6. REFERENCES & BIBLIOGRAPHY

Apache Tomcat Project. “Apache Tomcat® - Welcome!” *Apache.Org*, 2019, tomcat.apache.org/. Accessed 5 July 2019.

BOOTSTRAPMADE. “DevFolio – Bootstrap Portfolio HTML Template | BootstrapMade.” *BootstrapMade*, 5 Dec. 2018, <https://bootstrapmade.com/devfolio-bootstrap-portfolio-html-template/>. Accessed 17 Nov. 2019.

Butler, Tim. “What Is a Dockerfile?” *Via @codeship*, 28 Sept. 2015, blog.codeship.com/what-is-a-dockerfile/. Accessed 17 Nov. 2019.

Cloud, Google. “What Is Google Cloud Platform (GCP)? - Definition from WhatIs.Com.” *SearchCloudComputing*, 2019, searchcloudcomputing.techtarget.com/definition/Google-Cloud-Platform.

“Cloudflare.” *Cloudflare*, Cloudflare, 2019, www.cloudflare.com/learning/dns/what-is-dns/.

Dierks, T. “The Transport Layer Security (TLS) Protocol Version 1.2.” *Ietf.Org*, 2019, tools.ietf.org/html/rfc5246.

“Docker Build.” *Docker Documentation*, 15 Nov. 2019, docs.docker.com/engine/reference/commandline/build/.

“Docker Image.” *Docker Documentation*, 15 Nov. 2019, docs.docker.com/engine/reference/commandline/image/. Accessed 17 Nov. 2019.

“Docker Image Push.” *Docker Documentation*, 19 Aug. 2019, docs.docker.com/v17.12/engine/reference/commandline/image_push/.

“Docker Pull.” *Docker Documentation*, 15 Nov. 2019, docs.docker.com/engine/reference/commandline/pull/. Accessed 17 Nov. 2019.

“Docker Run.” *Docker Documentation*, 15 Nov. 2019, docs.docker.com/engine/reference/commandline/run/.

“Docker Tag.” *Docker Documentation*, 15 Nov. 2019, docs.docker.com/engine/reference/commandline/tag/.

Group, Documentation. “Welcome! - The Apache HTTP Server Project.” *Apache.Org*, 2015, httpd.apache.org/.

“Install Docker Desktop on Windows.” *Docker Documentation*, 15 Nov. 2019, docs.docker.com/docker-for-windows/install/. Accessed 17 Nov. 2019.

Jain, N. and Choudhary, S. (2016) ‘Overview of virtualization in cloud computing’, *2016 Symposium on Colossal Data Analysis and Networking, CDAN 2016*, pp. 79–82. doi: 10.1109/CDAN.2016.7570950.

J. Klensin. “Simple Mail Transfer Protocol.” *Ietf.Org*, 2019, tools.ietf.org/html/rfc2821. Accessed 17 Nov. 2019.

Java API. “What Is Java and Why Do I Need It?” *Java.Com*, 30 July 2014, java.com/en/download/faq/whatis_java.xml.

Klinbua, K. and Vatanawood, W. (2018) ‘Translating TOSCA into docker-compose YAML file using ANTLR’, *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, 2017-Novem, pp. 145–148. doi: 10.1109/ICSESS.2017.8342884.

Kvochko, Elena, et al. “Embracing HTTPS.” *Open Blog*, 10 Jan. 2017, open.blogs.nytimes.com/2014/11/13/embracing-https/.

Mahdi Mashrur Matin. “A Step-By-Step Guide to Securing a Tomcat Server With LetsEncrypt SSL Certificate.” *Medium*, Medium, 24 Apr. 2019, medium.com/@mashrur123/a-step-by-step-guide-to-securing-a-tomcat-server-with-letsencrypt-ssl-certificate-65cd26290b70.

“Managing DNS Resource Records | KnowledgeBase.” *Bigrock.In*, 2019, manage.bigrock.in/kb/node/636. Accessed 17 Nov. 2019.

MySQL Documentation. “MySQL :: MySQL 8.0 Reference Manual :: 1.3.1 What Is MySQL?” *Mysql.Com*, 2019, dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html.

Pibernik, J., Kanižaj, B. and Frljak, L. (2012) ‘Conceptual models for better quality of portfolio web site experience’, *Proceedings Elmar - International Symposium Electronics in Marine*, (September), pp. 161–164.

Preeth, E. N. *et al.* (2016) ‘Evaluation of Docker containers based on hardware utilization’, *2015 International Conference on Control, Communication and Computing India, ICCCI 2015*. IEEE, (November), pp. 697–700. doi: 10.1109/ICCC.2015.7432984.

P. Mockapetris. “Domain Names - Concepts and Facilities.” *Ietf.Org*, 2019, tools.ietf.org/html/rfc1034.

R. Fielding. “Hypertext Transfer Protocol -- HTTP/1.1.” *Ietf.Org*, 2019, tools.ietf.org/html/rfc2616.

Ragaine, Linda. "The Anatomy of a Perfect Portfolio Website to Showcase Your Work." *Kinsta Managed WordPress Hosting*, 18 Jan. 2019, kinsta.com/blog/portfolio-website/.

7. APPENDIX

All the well-documented code files are maintained on GitHub repository link given below.

GITHUB REPO - <https://github.com/saurabh-github-1995/netwoking-ca>