Project Title

Patient Tracking System

Under the Guidance of

Prof. B. Thangaraju

Teaching Assistant

Harsh Tripathi

Submitted by

Harshabh Mahant - MT2019043
Manthan Fursule - MT2019061
Sameer Khurd - MT2019097
Vipin Rai P - MT2019134

Department of Computer Science & Engineering
International Institute of Information Technology
Bangalore
Bengaluru, Karnataka , India-560100

Contents

1 Abstract

Recently world is troubled with the spread of the pandemic COVID-19. Everyone is staying home to keep themselves safe and to co-operate with the authorities. But still there are some people which need to travel from one place to another for their livelihoods or some kinds of emergencies. If one of such person becomes infected with the virus then it becomes very difficult track all the people with whom that person came in contact with. In such critical cases this application can prove handy, which allows the authorities to track the travel histories of every person and visualize it easily with the help of google maps. Visualization of travel histories of people can easily determine hotspots where majority of people were present at the same time.

2 Introduction

2.1 Why DevOps?

Aim of developing this project under software production engineering subject is to demonstrate perks of using DevOps model instead of traditional ones in development. DevOps is an IT mindset that encourages communication, collaboration, integration and automation among software developers and IT operations in order to improve the speed and quality of delivering software.
Goals of DevOps :

1. Reduce time between deployments.
2. Cope up with user requirements with faster delivery.
3. Reduce the failure rate of application releases.
4. Fix the bugs in short span of time.
5. Improve recovery time.

2.2 About the application

Name of the application is "COVID-19 Patient Tracker". It is a web based application which allows the authorities to keep track of patient travel histories. This application is mainly developed for the easement of the government so that they can keep eye on every person travelling across the country. With the help of this application, we can add person's information into the centralized database, retrieve the data and visualize it on google maps and find hotspots where large amount people were present at certain time.

2.3 Technology stack for development

1. Front-end technologies - HTML, CSS, Bootstrap and Angular
2. Backend technologies - Node js
3. Database - MySQL

2.4 DevOps Tools Used

1. Source Code Management : Git
2. Container Platform : Docker
3. Testing tools : Jasmine and Karma
4. Continuous Integration : Jenkins
5. Continuous Deployment : Rundeck
6. Continuous Monitoring : ELK Stack

2.5 Services Used

1. Amazon Elastic Compute Cloud (Amazon EC2)
2. Microsoft Azure
3. Amazon Relational Database System (Amazon RDS)

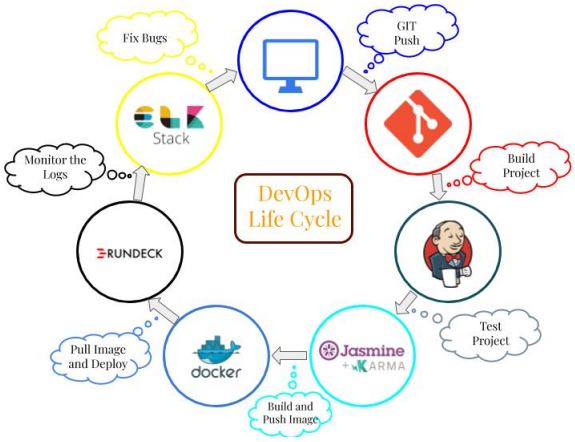


Figure 1: DevOps cycle of the project using mentioned tools

3 Configurations

3.1 Creating MySql DB in Amazon AWS

1. Signup and open the AWS console.
2. Search for RDS and open RDS.
3. Do the configuration for required DB as shown in ScreenShots below and click on Create database
4. You will see the below screen showing “creating”.
5. Once DB is created you will get endpoint URL, which can be used to access DB from anywhere (here Application).

Step 1 : Find the Relational Database System serivce in the AWS Management Console.

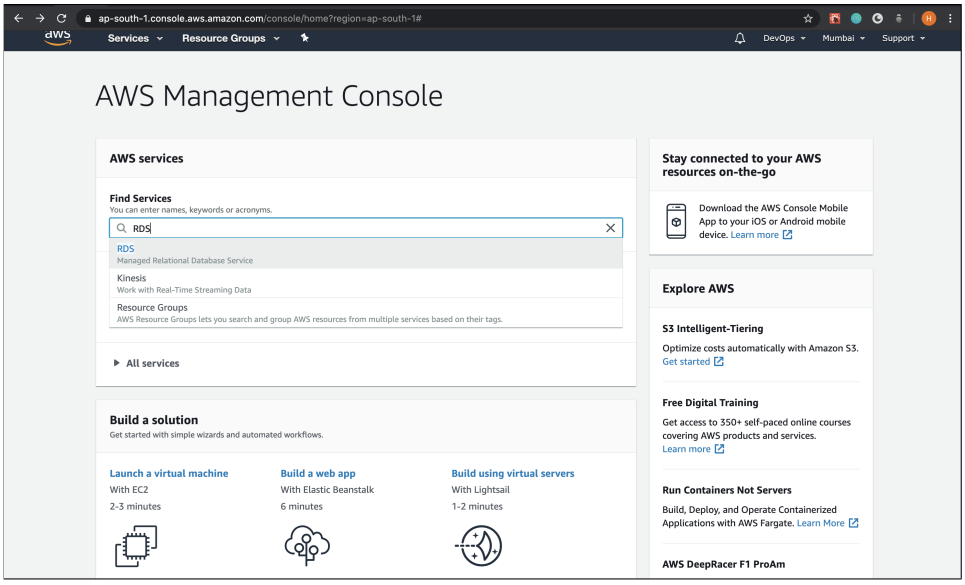


Figure 2: Finding RDS service on AWS Management Console

Step 2 : Click on "Create Database" option to start creating database.

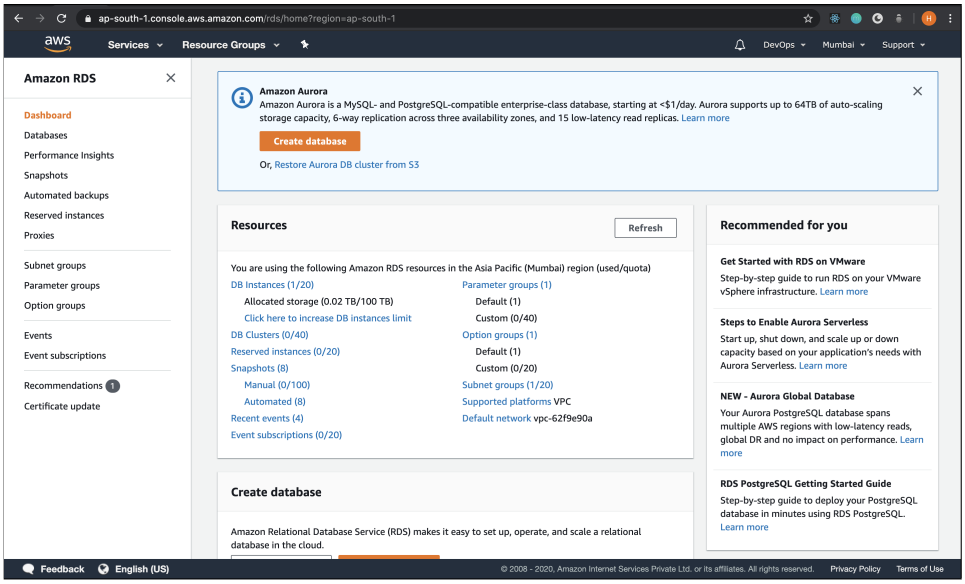


Figure 3: Start Creating Database

Step 3 : Choose the type of database engine to be created as MySQL.

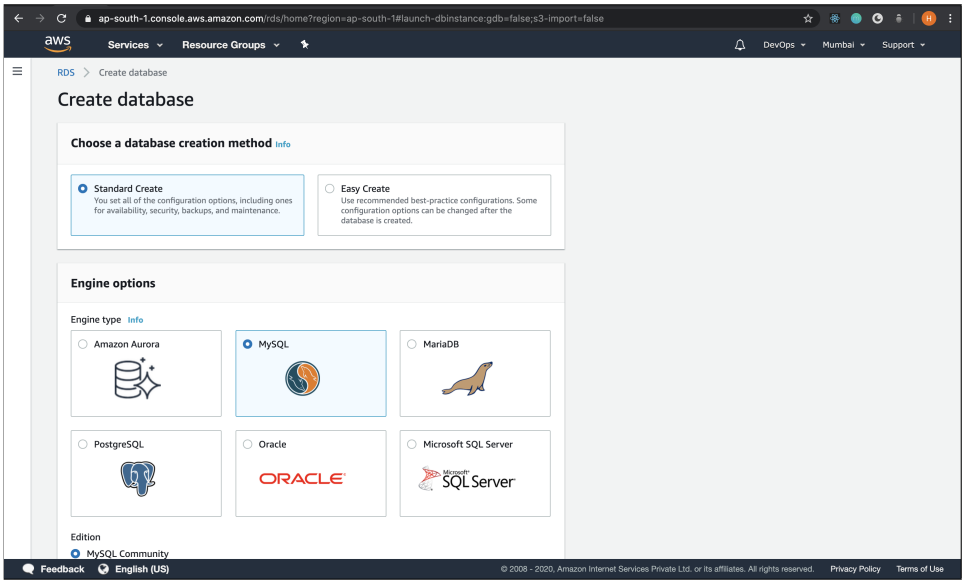


Figure 4: Select the Database Engine as MySQL

Step 4 : Choose the desired version of MySQL database.

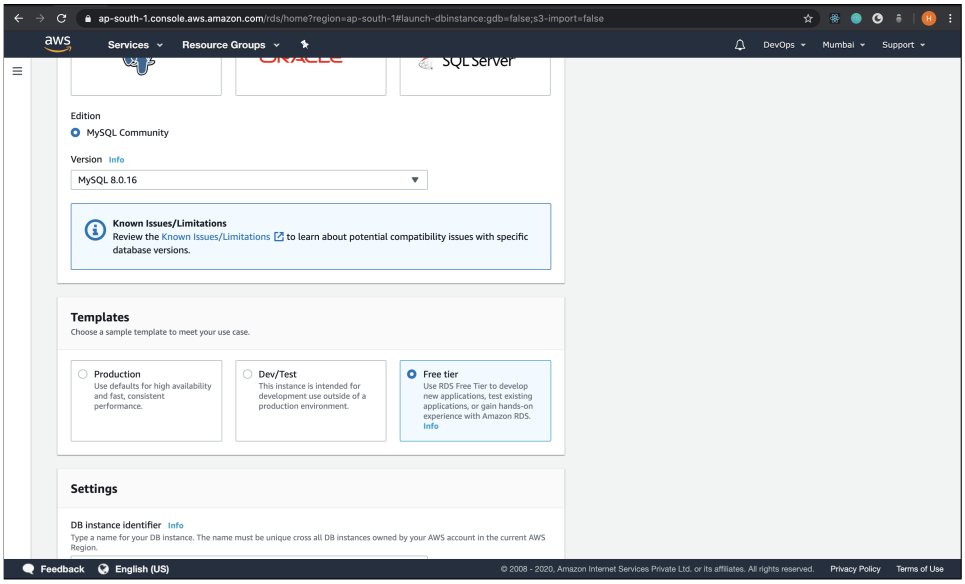


Figure 5: Select the MySQl Version

Step 5 : Set the username and password credentials which will be needed to access the database.

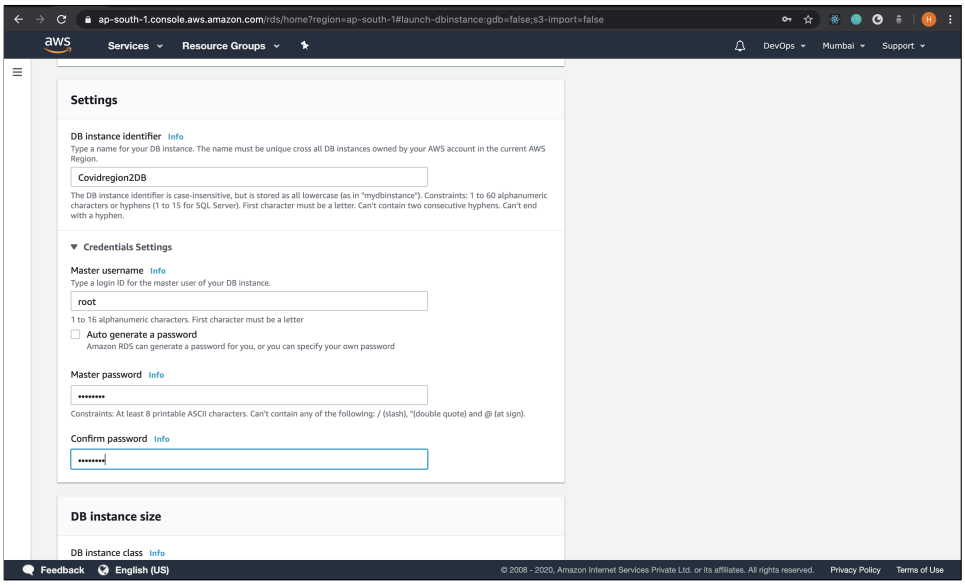


Figure 6: Set the credentials for Database Instance

Step 6 : Select the database instance size according to your development requirements.

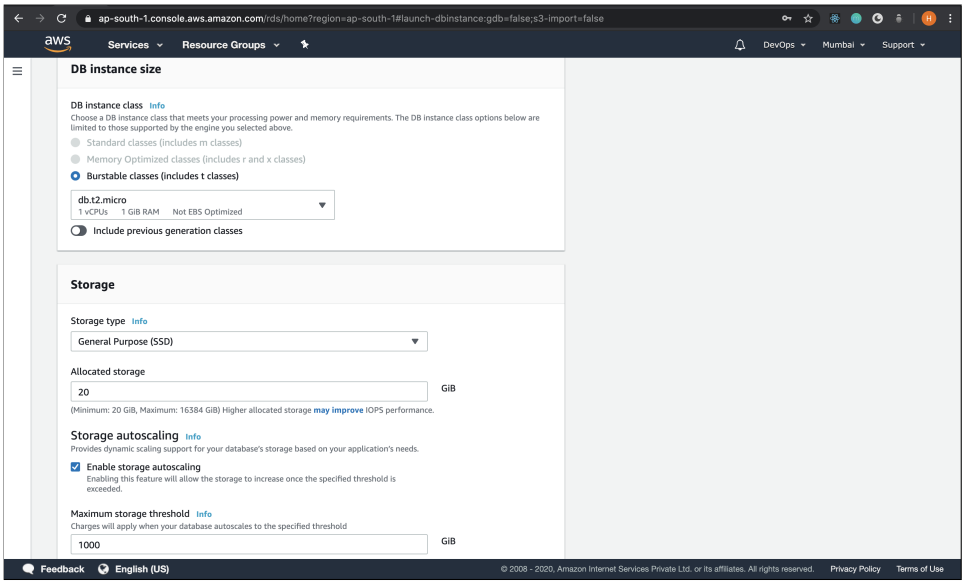


Figure 7: Select the database Instance Size and Storage

Step 7 : Select the accessibility type of the database : Public or Private.

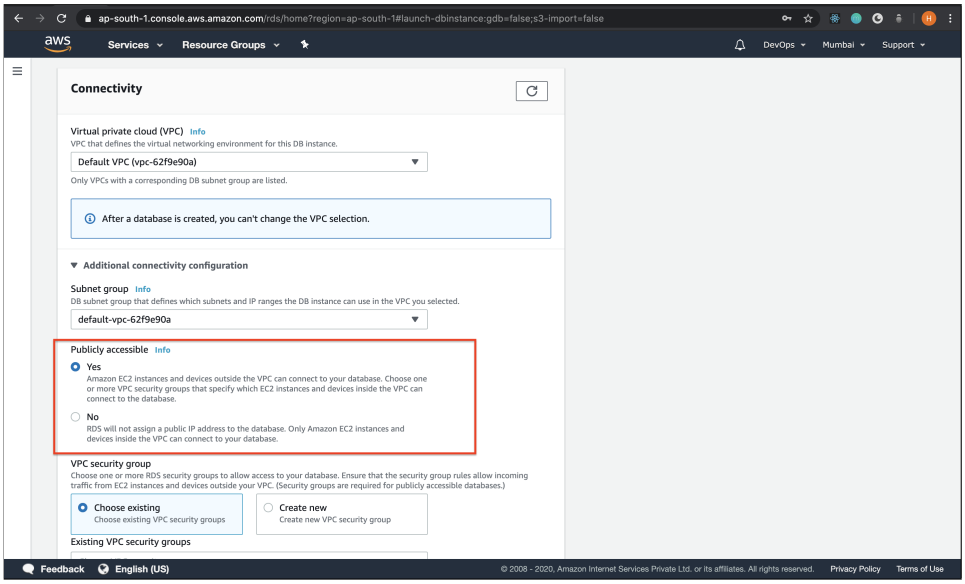


Figure 8: Set the Connectivity Settings

Step 8 : Select the VPC security group which acts as a virtual firewall for your instance to control inbound and outbound traffic.

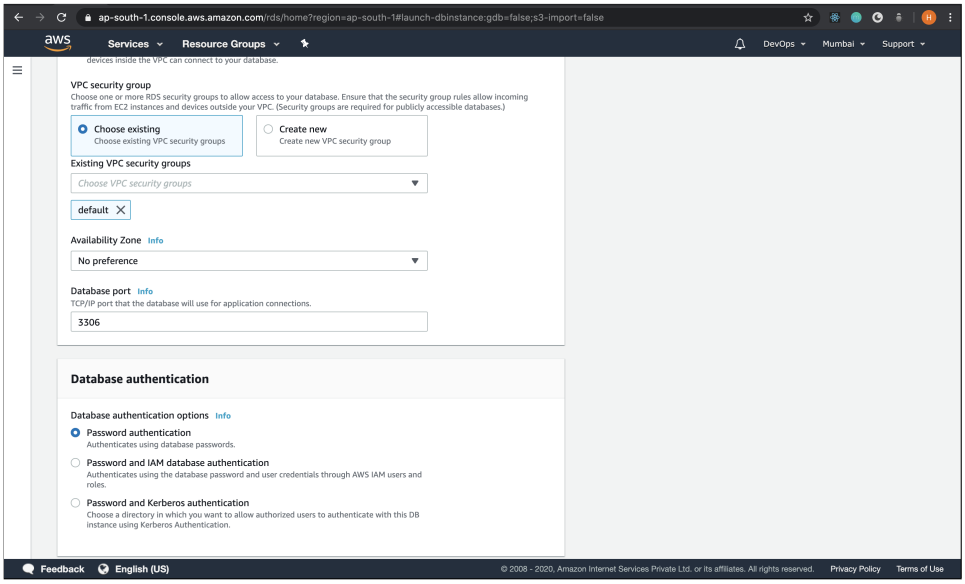


Figure 9: Set the Security Group and Authentication Options

Step 9 : Once all the specifications are chosen, click on "Create Database" to create the database.

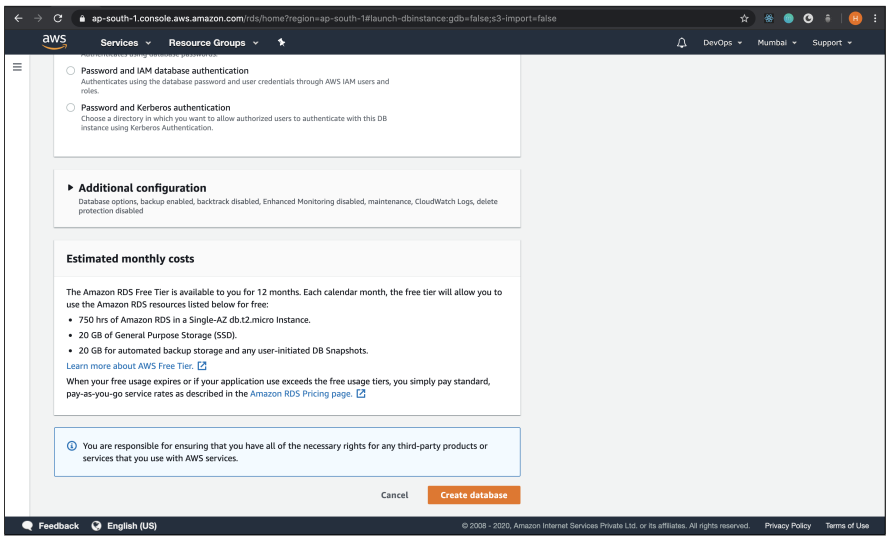


Figure 10: Create the Database

Step 10 : Once the database is created, you can view the database instance running on Amazon RDS dashboard.

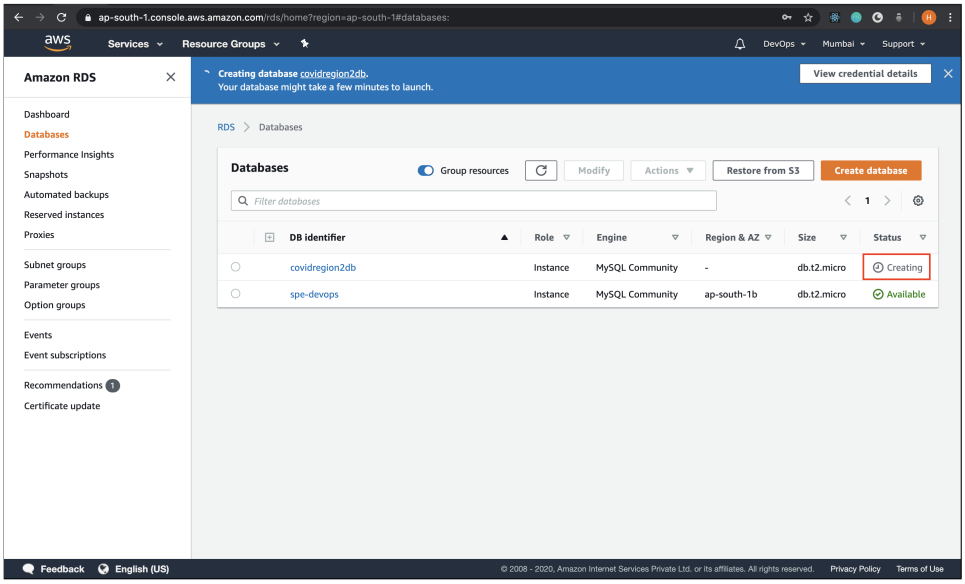


Figure 11: Dashboard View after creation of the Database

Step 11 : After database creation, you can see the status as "Successfully created database [database_name]"

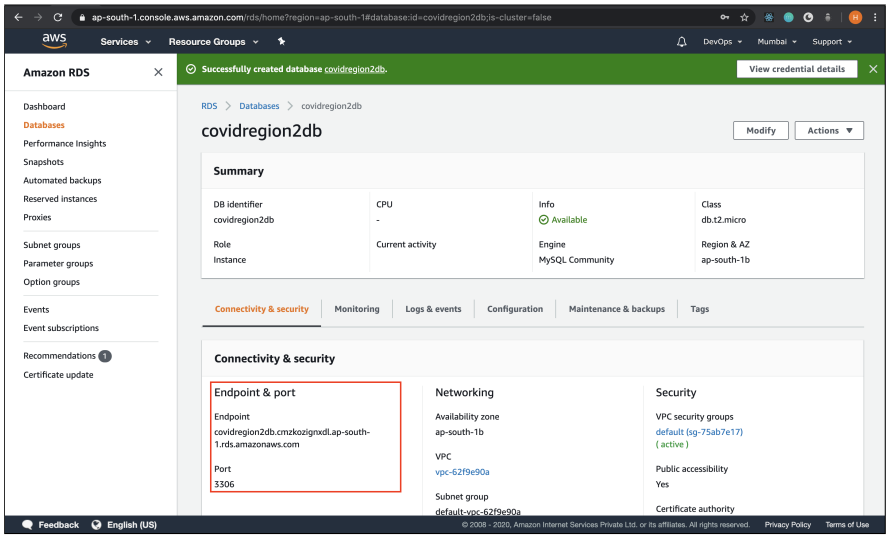


Figure 12: Status of Database Creation as Success

3.2 Creating AWS EC2 instance (Ubuntu 18.04):

1. Signup and open the AWS console.
2. Search for EC2 and open EC2.
3. Click on Launch Instance.
4. Select Ubuntu Server of required version and do the configuration as shown
5. Goto Configure security group and click on add Rule
6. Add ports which are required by the application and needs to be exposed to public.
7. Click on review and Launch
8. Once asked for key Pair, select “Create a new key pair”.
9. Give some name and click on download key Pair.
10. Store the downloaded file in safe place, it will be required to connect to the instance.
11. Once done click on Launch Instance then view instance.
12. It will take some time to create the instance.
13. Once instance is created You will get public DNS and IP using which you can connect to the instance.

Step 1 : Signup and open the AWS console. Search for EC2 and open EC2.

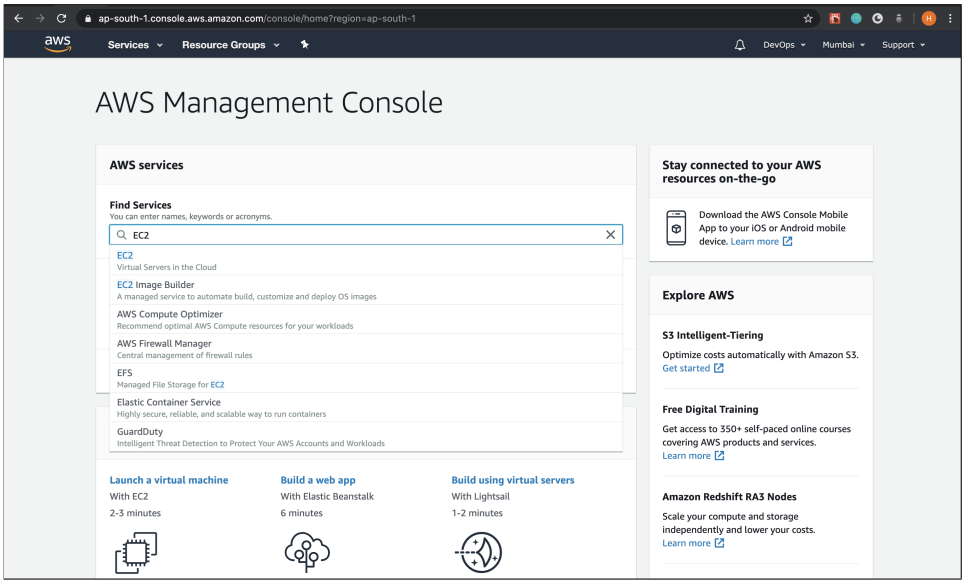


Figure 13: Finding EC2 service on AWS Management Console

Step 2 : Click on "Launch Instance" to initiate the process.

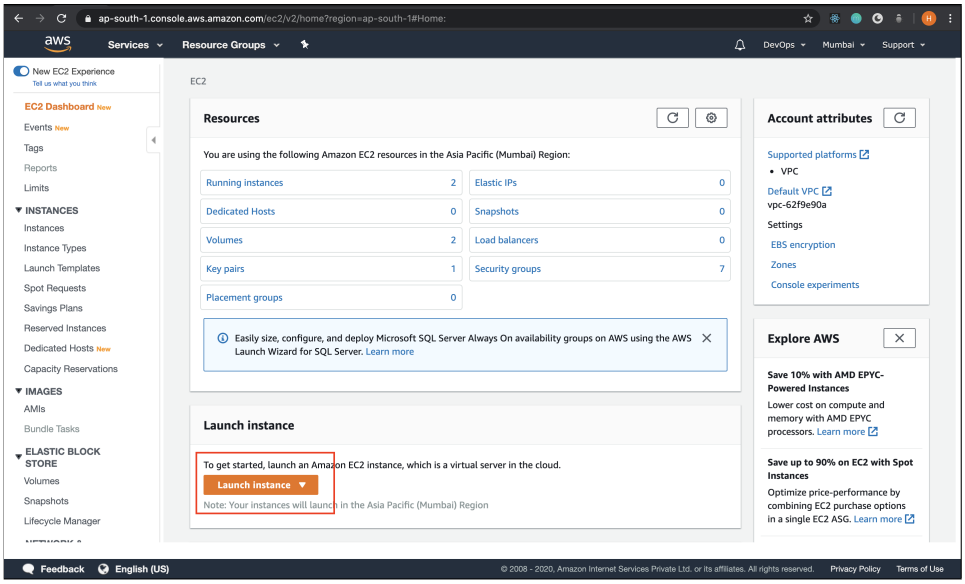


Figure 14: Select the Option to Launch Instance

Step 3 : Select Ubuntu Server of required version and do the configuration as shown. Here it is chosen as Ubuntu 18.04.

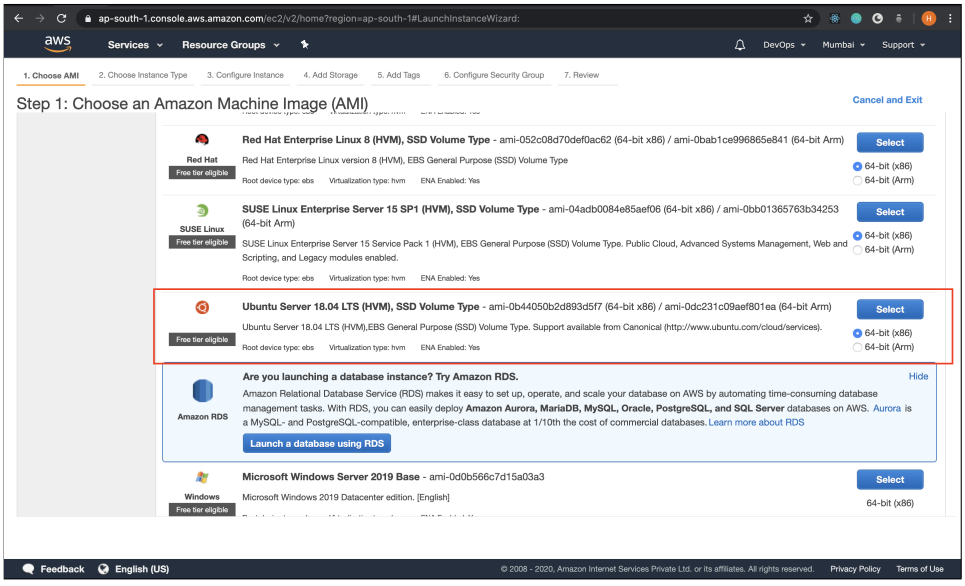


Figure 15: Select the Image as Ubuntu 18.04 LTS

Step 4 : Choose the instance type according to your development environment requirements (such as number of CPUs needed and memory requirement.)

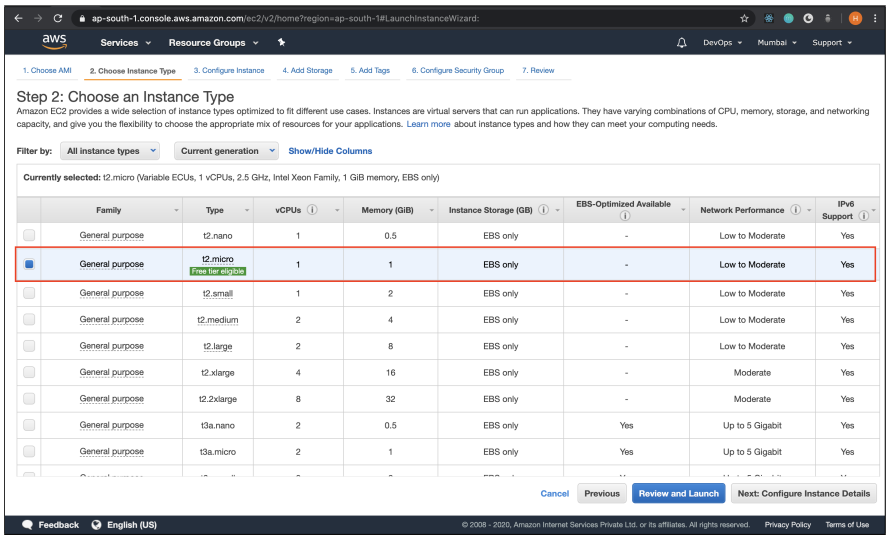


Figure 16: Choose the Instance Type

Step 5 : Configure the security group. To connect to your instance, your security group must have inbound rules that allow SSH access (for Linux instances).

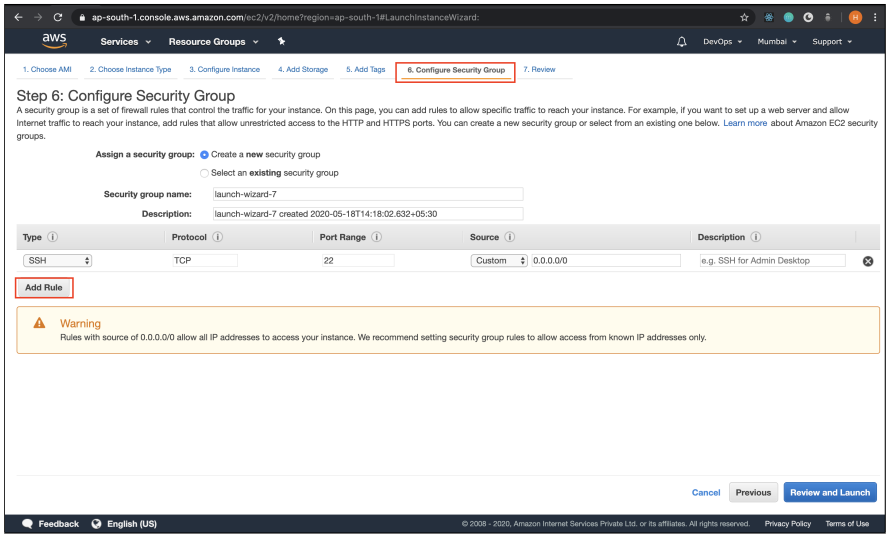


Figure 17: Configure Security Group - 1

Step 6 : In next step of configuring security group, choose the type as TCP and specify and port range and source.

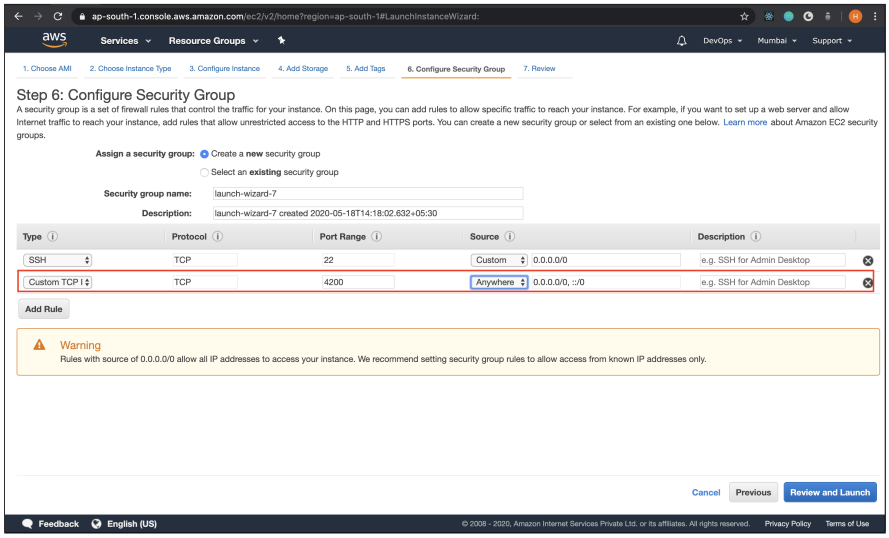


Figure 18: Configure Security Group - 2

Step 7: Select the existing key pair or create new one for your instance. This key pair provides added security to created instance access.

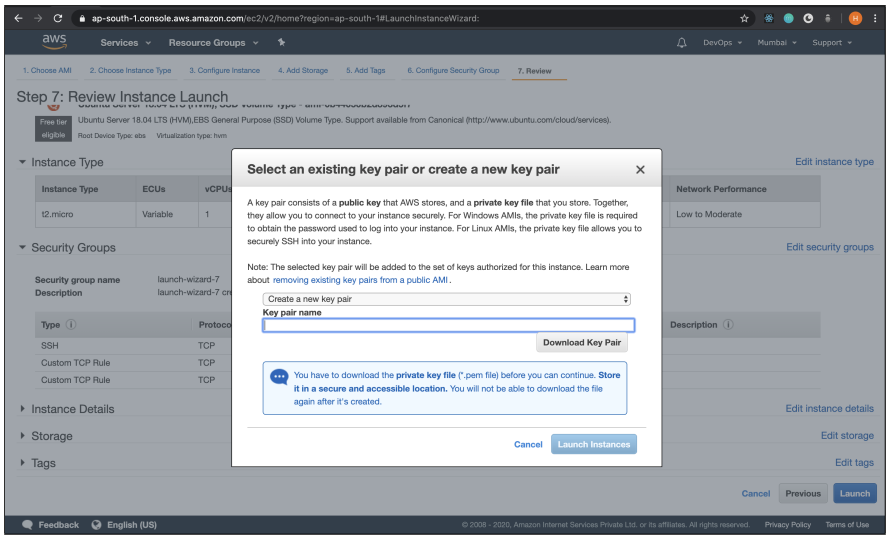


Figure 19: Select Key Pair

Step 8: Once you select key pair and click on "Launch Instance" you can view the status of instance creation process.

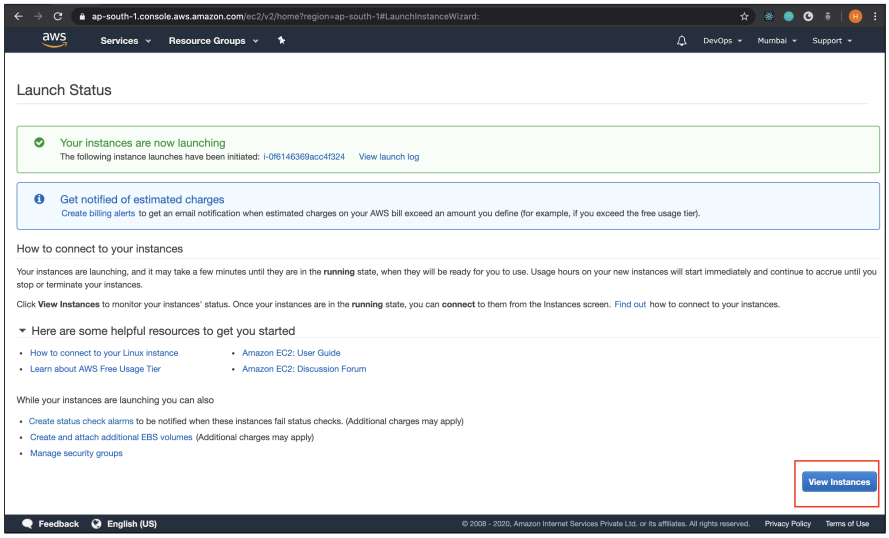


Figure 20: launch Status

Step 9. Once the instance is launched, you can view its status on AWS EC2 dashboard as running, stopped or terminated.

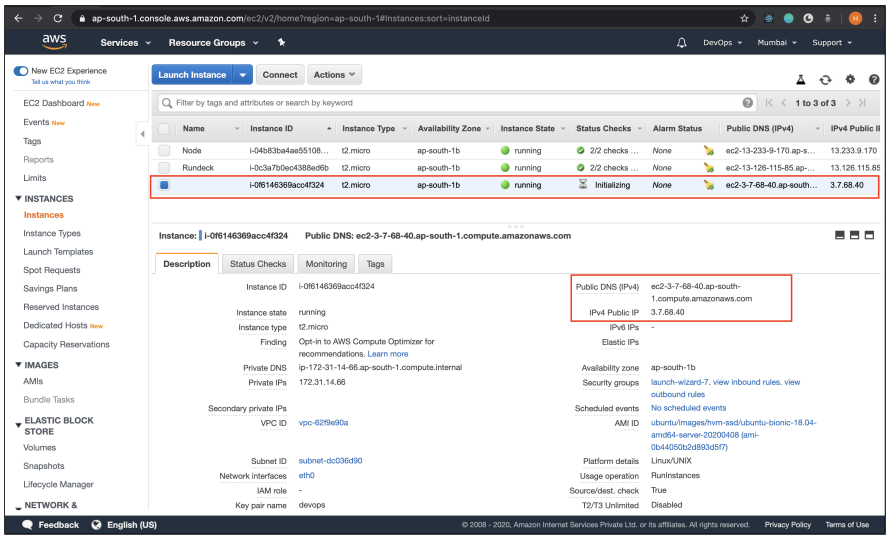


Figure 21: Dashboard view of Created Instance

3.3 Connecting to the Created Ubuntu Instance

1. Click on Connect as shown in below Screenshot
2. Copy the command to connect. “ssh -i ”devops.pem” ubuntu@ec2-3-7-68-40.ap-south-1.compute.amazonaws.com”
3. Run the above command in terminal and make sure the downloaded Key Pair file is in same location or provide the path to that file. (Here file name is devops.pem)

Step 1 : To connect to the created instance, first click in on the ”Connect” option.

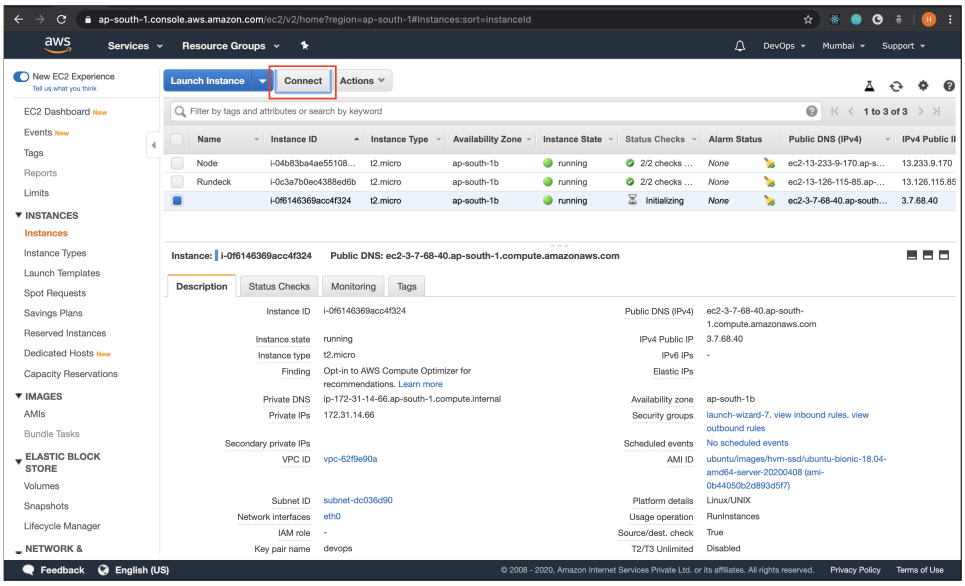


Figure 22: Connect to the Instance

Step 2 : Once you click on "Connect", a pop up window will appear which explains steps to connect to the instance. Follow the instructions to connect to the instance.

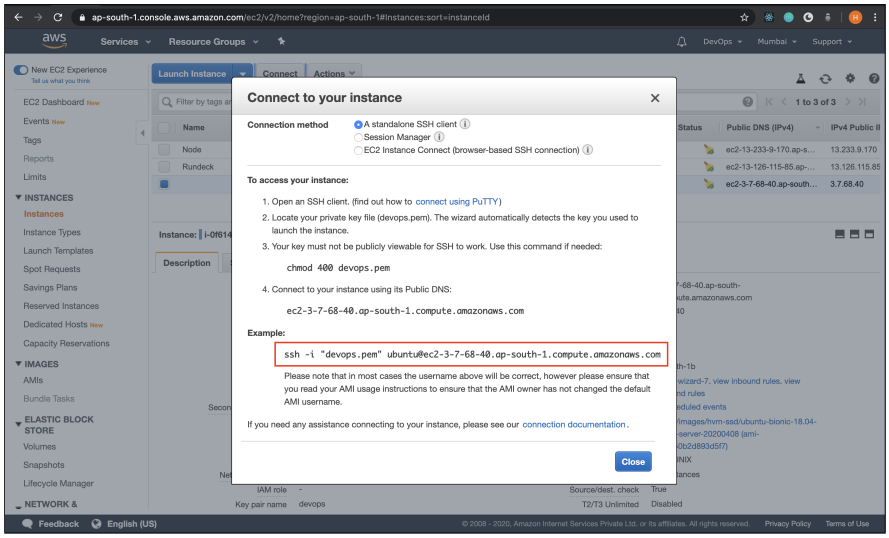


Figure 23: Command to Connect to the Instance

Step 3 : Run the ssh command displayed in the instructions from the terminal which will allow to you to access created instance from the cloud on your local system.

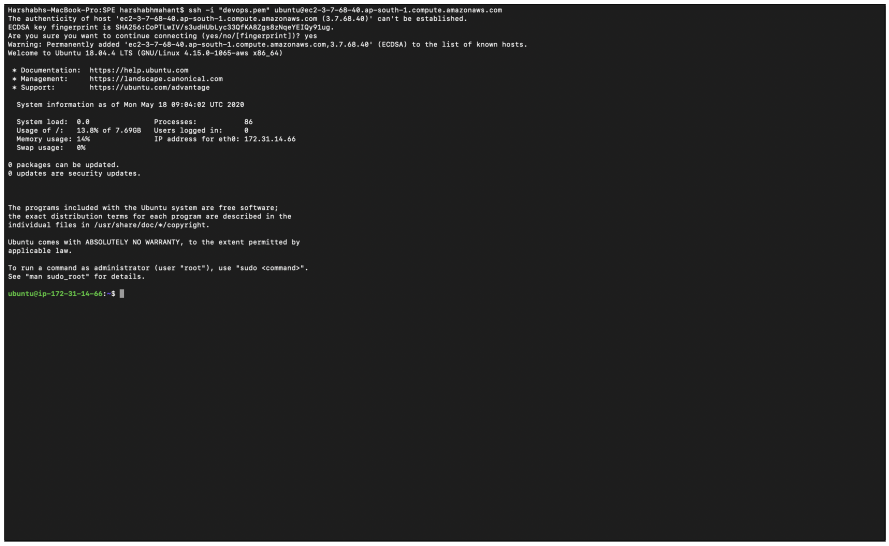


Figure 24: Connected to the instance

3.4 Jenkins and Dockerhub

To push created image to docker hub, it is necessary to add docker hub credentials to jenkins.

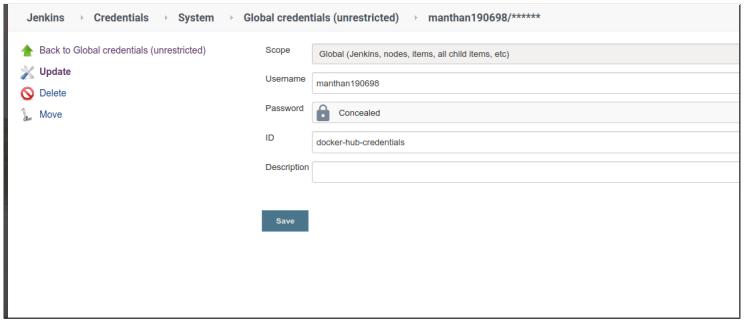


Figure 25: Add Docker Hub credentials to Jenkins

3.5 Jenkins and Rundeck

To trigger rundeck job through jenkins, rundeck credentials and URL must be added to the jenkins

Figure 26: Add Rundeck credentials to Jenkins

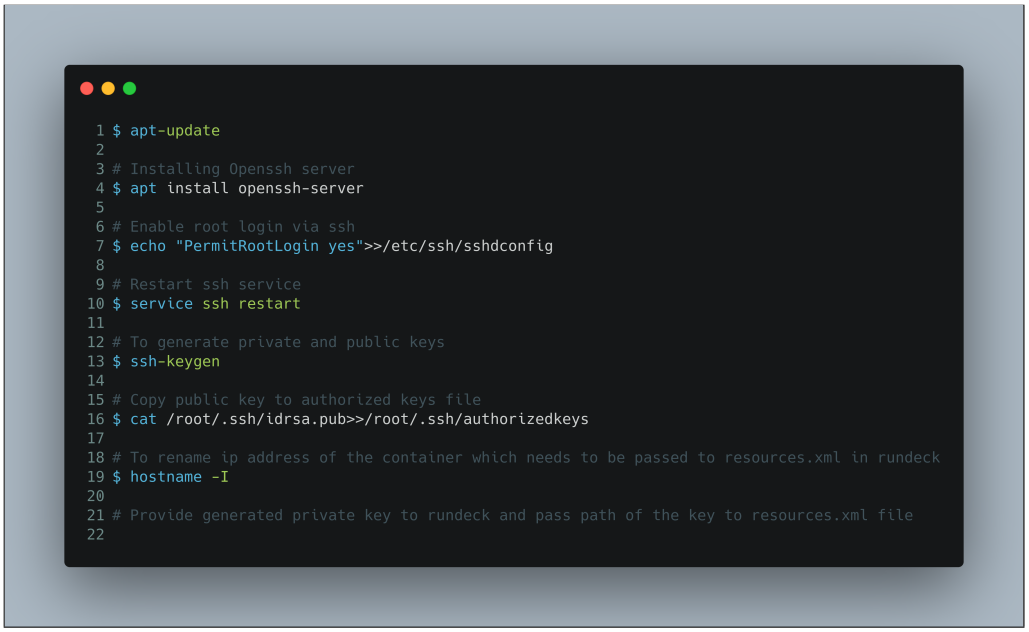
3.6 Jenkins and Docker

In order to build docker image via jenkins, jenkins need to be added to the docker user group. Following commands are used for this :

1. sudo usermod -a -G docker jenkins (Add jenkins to the docker user group)
2. systemctl restart jenkins (Restart jenkins service)

3.7 Node and Rundeck

In order to add a container as a node in rundeck, it is necessary to enable ssh service in container and provide generated private key to the rundeck.
Steps to enable ssh service and connect node to rundeck :



```
$ apt-update

# Installing Openssh server
$ apt install openssh-server

# Enable root login via ssh
$ echo "PermitRootLogin yes">>/etc/ssh/sshdconfig

# Restart ssh service
$ service ssh restart

# To generate private and public keys
$ ssh-keygen

# Copy public key to authorized keys file
$ cat /root/.ssh/idrsa.pub>>/root/.ssh/authorizedkeys

# To rename ip address of the container which needs to be passed to resources.xml in rundeck
$ hostname -I

# Provide generated private key to rundeck and pass path of the key to resources.xml file

```

Figure 27: Steps to connect node to rundeck

```
<?xml version="1.0" encoding="UTF-8"?>
   <project>
       <node name="DevOpsNode"
       description="Docker Container"
       hostname="15.206.185.233"
       osFamily="unix"
       osName="Ubuntu"
       username="root"
       ssh-key-storage-path="keys/devops/server.key"
       sudo-command-enabled="true"
       sudo-password-option="option.sudoPassword"/>
   </project>
```

Figure 28: resources.xml file of the project

Above is the resources.xml file used for this project.

1. Node can be created in rundeck with the help of this xml file.
2. "**node name**" attribute specify the name of node to be created on rundeck.
3. "**description**" attribute specifies the description of that node.
4. "**hostname**" attribute specify ip address of node on which application needs to deployed.
5. "**username**" attribute require username of the node which is generally root.
6. "**ssh-key-storage-path**" stores the path of private key which is generated on the node where application is deployed.

3.8 Node js and jenkins

As the the project uses node js as backend, while building pipeline it is necessary to add node js component to Jenkins.

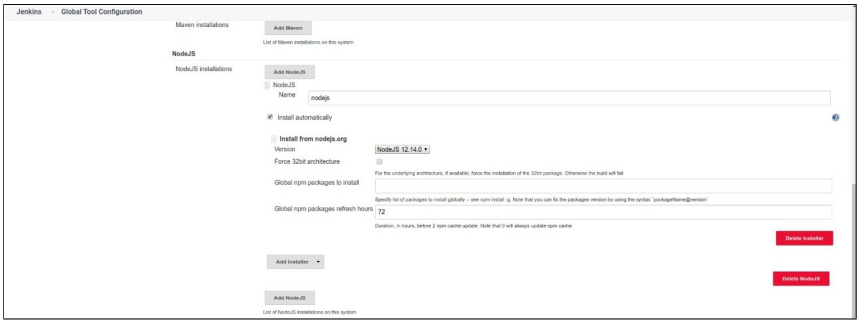


Figure 29: Add Node js to Jenkins

4 Software Development Life Cycle

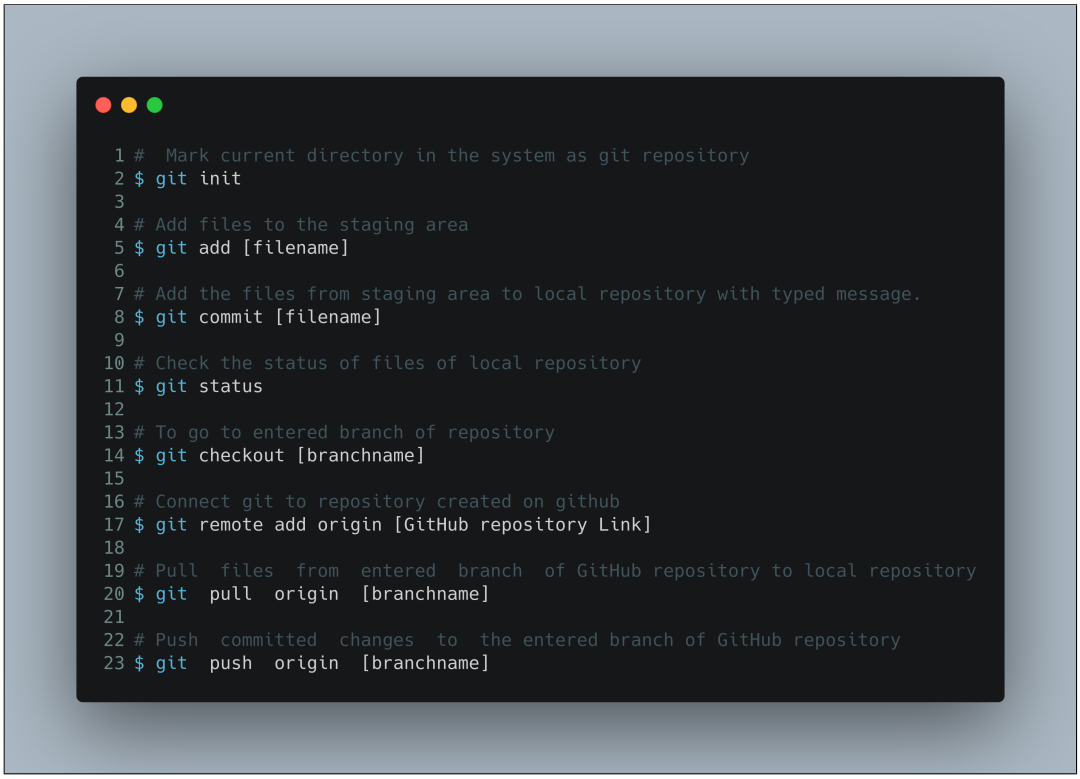
Automation of every step of the SDLC is the main motivation behind the use of DevOps model. This automation includes all the steps such as, code management, code building, code testing, configuration management, application deployment, application monitoring etc.

4.1 Source Code Management : GIT

Git is the source code management tool used in this project. Git is used to keep the track of newly created files and modification done to the existing files. Then git is connected to Github to push all the changes to central repository.
Creating Project Repository :

1. Login to github using credentials.
2. Clicked On "New" on left side for creating new repository
   (a) Repository name : **spe-covid19**
   (b) Check on Public
   (c) Create Repository

Git commands used :



```
#  Mark current directory in the system as git repository
$ git init

# Add files to the staging area
$ git add [filename]

# Add the files from staging area to local repository with typed message.
$ git commit [filename]

# Check the status of files of local repository
$ git status

# To go to entered branch of repository
$ git checkout [branchname]

# Connect git to repository created on github
$ git remote add origin [GitHub repository Link]

# Pull  files  from  entered  branch  of GitHub repository to local repository
$ git  pull  origin  [branchname]

# Push  committed  changes  to  the entered branch of GitHub repository
$ git  push  origin  [branchname]
```

Figure 30: Git commands

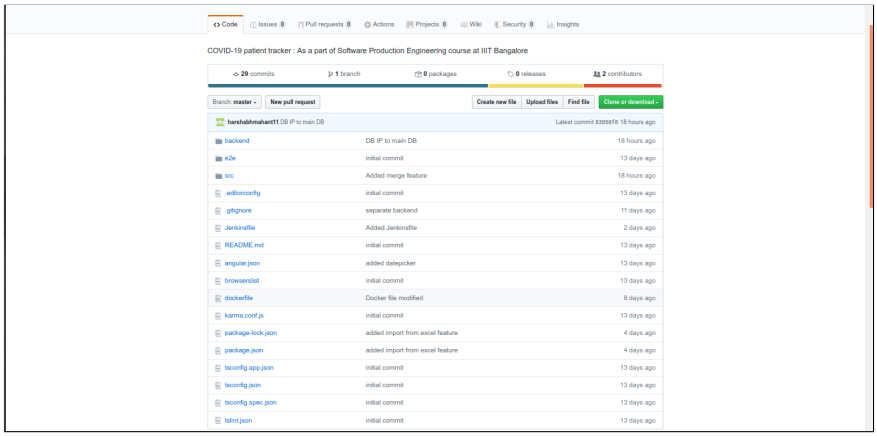


Figure 31: View of the Project GitHub Repository

4.2 Artifact : Docker

Docker is used to combine all the required steps to create an image which will deploy the application. With the help of dockerfile we can build the image of our application.[1]

```
FROM ubuntu:latest

RUN apt-get update
RUN apt install curl -y
RUN apt install nodejs -y
RUN apt install npm -y
RUN npm -g install forever deep-equal@1.1.1
RUN apt-get install -y apache2
RUN mkdir -p /var/lock/apache2
RUN mkdir -p /var/run/apache2
RUN mkdir /usr/backend
COPY backend /usr/backend

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_PID_FILE /var/run/apache2.pid
ENV APACHE_RUN_DIR /var/run/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_LOG_DIR /var/log/apache2
ENV LANG C

COPY dist/sp-covid19 /var/www/html

WORKDIR /usr/backend

RUN chmod 777 script.sh
CMD ./script.sh
EXPOSE 80
EXPOSE 3000
```

Figure 32: Dockerfile of the project

Steps performed in the dockerfile :

1. Set base image for image to be built as Ubuntu using FROM command.

2. Set all the necessary installations for the project deployment using RUN command.

3. Set the necessary environment variables using ENV command.

4. Copy project folder to destination folder and mark that folder as working directory.

5. Assign appropriate privileges to script which will launch the server.

6. Expose necessary ports for the deployment of the application.

Docker image can be built with the help of following command :
$ docker build . -t [**tag of docker image**].

4.3 Unit Testing : Jasmine test framework

- **Jasmine test framework :**

  Jasmine is a framework for testing. Test cases are define in the file [**componentname**]**.spec.ts** .Initially global function *describe* is called. It has two parameters, a string and a function. String is the title for the test suite and function is the block of code that implements the test suite. *Specs* which mean test specifications are defined inside *it* block. It takes two parameter string and function. String is the title of the test and the function is the test block. Function contains block of code and expections. If Expectations are. met then the test is passed.

```
it('should create', () => {
  expect(component).toBeTruthy();
});

it('should update submit variable', () => {
  component.onFormSubmit();
  expect(component.submitted).toEqual(true);
});

it('should invalidate form if fields left empty', () => {
  component.onFormSubmit();
  expect(component.registerForm.invalid).toEqual(true);
})
```

Figure 33: Unit tests defined inside *it* blocks

- **Karma :**

  It is a test runner which runs source code on a browser and will get feedback about the tests conducted. *PhantomJS* is a headless browser which is used to run tests.

```
// Karma configuration file, see link for more information
// https://karma-runner.github.io/1.0/config/configuration-file.html

module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular-devkit/build-angular/plugins/karma'),
      require('karma-phantomjs-launcher')
    ],
    client: {
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    coverageIstanbulReporter: {
      dir: require('path').join(__dirname, './coverage/sp-covid19'),
      reports: ['html', 'lcovonly', 'text-summary'],
      fixWebpackSourcePaths: true
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['PhantomJS'],
    singleRun: true,
    restartOnFileChange: true
  });
};
```

Figure 34: Karma Configuration File

```
it('should match userData', async(() => {
    //* arrange
    const userData = {
      PersonID: '1',
      Address: 'HSR Layout,Bengaluru',
      City: 'Bengaluru',
      State: 'Karnataka',
      Infected: true
    }
    let spy = spyOn(addPersonService,'addPersonDetails')
              .and.returnValue();

    fixture.detectChanges();
    component.testing = true;
    fixture.debugElement.query(By.css('#id')).nativeElement.value = userData.PersonID;
    fixture.debugElement.query(By.css('#address')).nativeElement.value = userData.Address;
    fixture.debugElement.query(By.css('#city')).nativeElement.value = userData.City;
    fixture.debugElement.query(By.css('#state')).nativeElement.value = userData.State;
    fixture.debugElement.query(By.css('#infected')).nativeElement.checked = userData.Infected;

    //let temp = fixture.debugElement.queryAll(By.css('.location'))[0].nativeElement.value;

    //* act
    fixture.whenStable().then(() => {
      component.onFormSubmit();
      // * assert
      expect(component.userData).toEqual(userData);
    })
  }));
```

Figure 35: Test to match user data entered

- **Running the tests**

  Tests can be started using the following command.

```
$ ng test
```

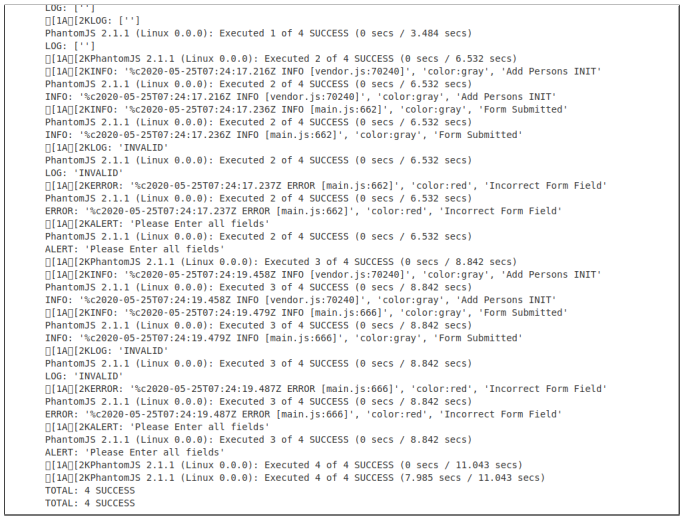


Figure 36: Jenkins console log

4.4 Monitoring : ELK Stack

In ELK, E is Elasticsearch, L is for Logstash and K is for Kibana.

1. Elasticsearch: the search and analytics engine
2. Logstash: the data collection, enrichment, and transportation pipeline
3. Kibana: the data visualisation platform

Logstash configuration file :

```
input {
    http {
    response_headers => {
            "Access-Control-Allow-Origin" => "*"
            "Access-Control-Allow-Headers" => "Content-Type, Access-Control-Allow-Headers,
Authorization, X-Requested-With"
            "Access-Control-Allow-Methods" => "*"
            "Access-Control-Allow-Credentials" => "*"
        }
        host => "0.0.0.0" # default: 0.0.0.0
    port => 3323        # default: 8080
  }
}
output {
  elasticsearch { hosts => ["http://52.170.156.240:9200"] }
  stdout { codec => rubydebug }
}
```

Figure 37: Logstash Configuration File

http input plugin is used to retrieve logging data from Angular and the elasticsearch output plugin to send it to an Elasticsearch index. According to the configuration file, Logstash will launch a HTTP server which will listen on 0.0.0.0, port 3323, and after retrieving the data, it will be sent to elesticsearch which is launched on 52.170.156.240:9200. And elesticsearch sends this data to kibana on which it can be visualized.[4][5][6][7]

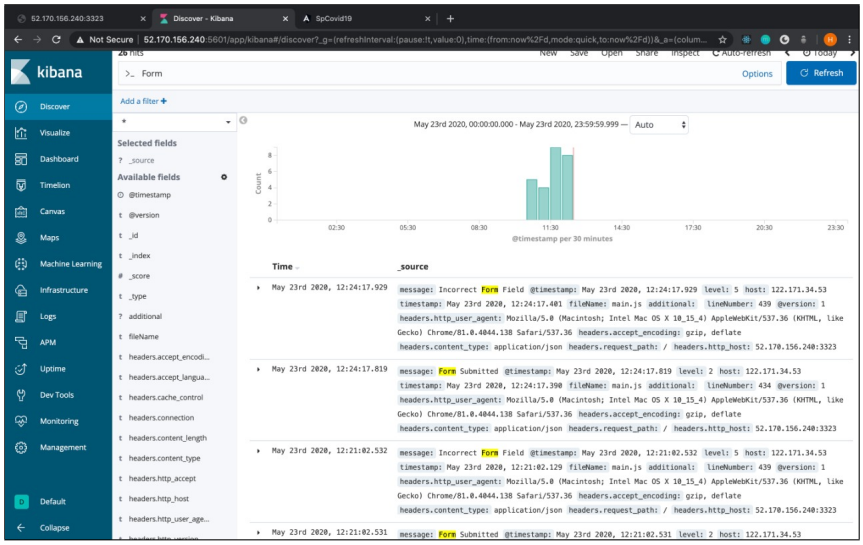
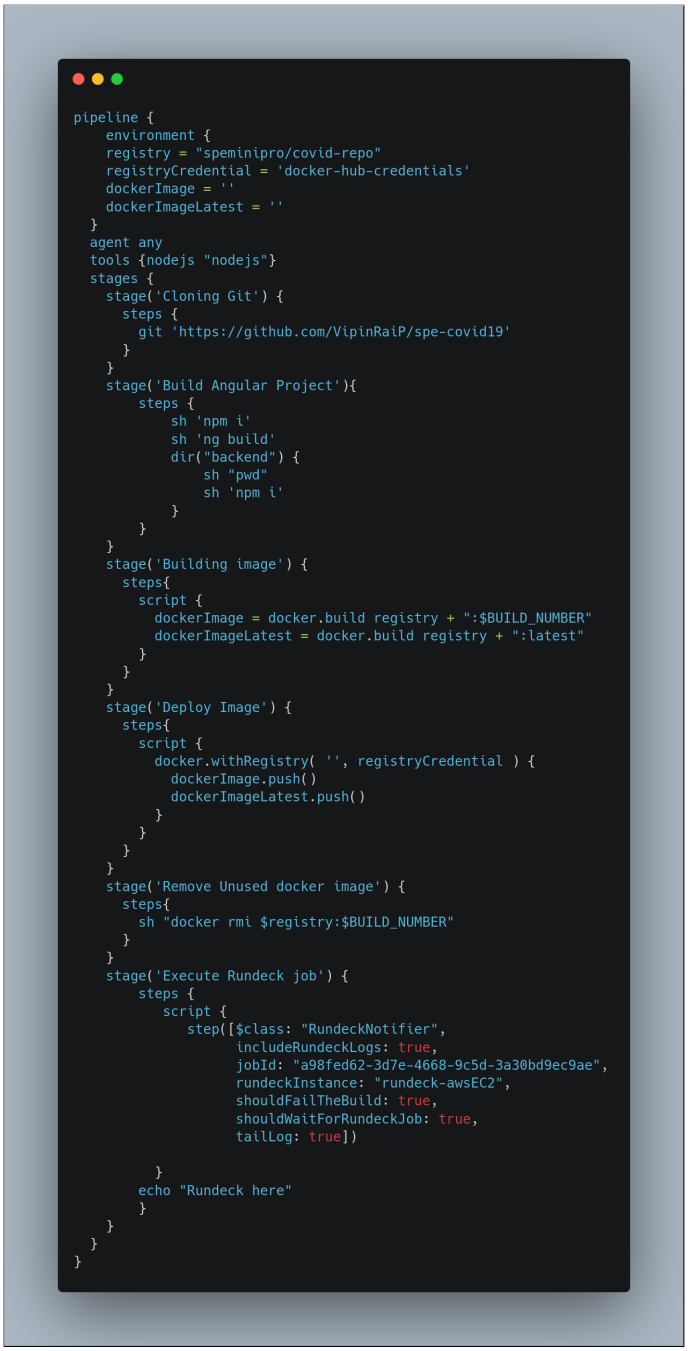


Figure 38: Kibana Dashboard

5 CI/CD Pipeline : Jenkins

With the help of Jenkins, CI/CD pipeline is executed in this project. Jenkins uses Jenkinsfile which is a groovy script, in which all the stages of pipeline are mentioned. To make the integration and deployment continuos, Poll SCM is added to this pipeline, which will allow pipeline to run automatically whenever there will be new features added to the application.[8][9]



```
pipeline {
    environment {
    registry = "speminipro/covid-repo"
    registryCredential = 'docker-hub-credentials'
    dockerImage = ''
    dockerImageLatest = ''
  }
  agent any
  tools {nodejs "nodejs"}
  stages {
    stage('Cloning Git') {
      steps {
        git 'https://github.com/VipinRaiP/spe-covid19'
      }
    }
    stage('Build Angular Project'){
        steps {
            sh 'npm i'
            sh 'ng build'
            dir("backend") {
                sh "pwd"
                sh 'npm i'
            }
        }
    }
    stage('Building image') {
      steps{
        script {
          dockerImage = docker.build registry + ":$BUILD_NUMBER"
          dockerImageLatest = docker.build registry + ":latest"
        }
      }
    }
    stage('Deploy Image') {
      steps{
        script {
          docker.withRegistry( '', registryCredential ) {
            dockerImage.push()
            dockerImageLatest.push()
          }
        }
      }
    }
    stage('Remove Unused docker image') {
      steps{
        sh "docker rmi $registry:$BUILD_NUMBER"
      }
    }
    stage('Execute Rundeck job') {
        steps {
           script {
              step([$class: "RundeckNotifier",
                    includeRundeckLogs: true,
                    jobId: "a98fed62-3d7e-4668-9c5d-3a30bd9ec9ae",
                    rundeckInstance: "rundeck-awsEC2",
                    shouldFailTheBuild: true,
                    shouldWaitForRundeckJob: true,
                    tailLog: true])

          }
        echo "Rundeck here"
        }
    }
  }
}
```

Figure 39: Jenkinsfile of the project

Stages of the Jenkinsfile are as follows:

1. Clone the git repository
2. Build the angular project
3. Build the image from the dockerfile
4. Push the created image to DockerHub
5. Pull the image on Rundeck node to launch the application

5.1 Screnshots of the execution

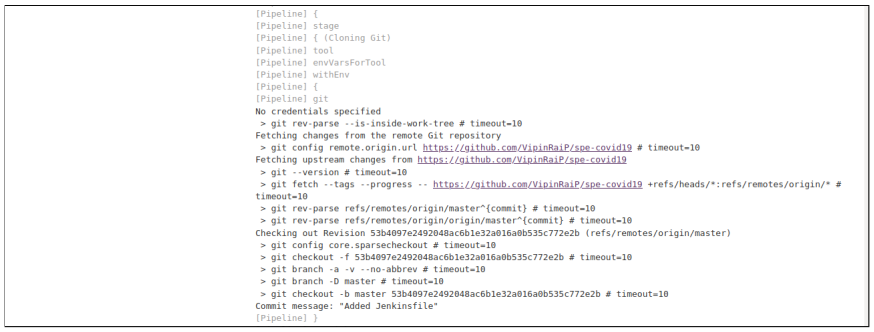


Figure 40: Stage 1 : Cloning the Git Repository

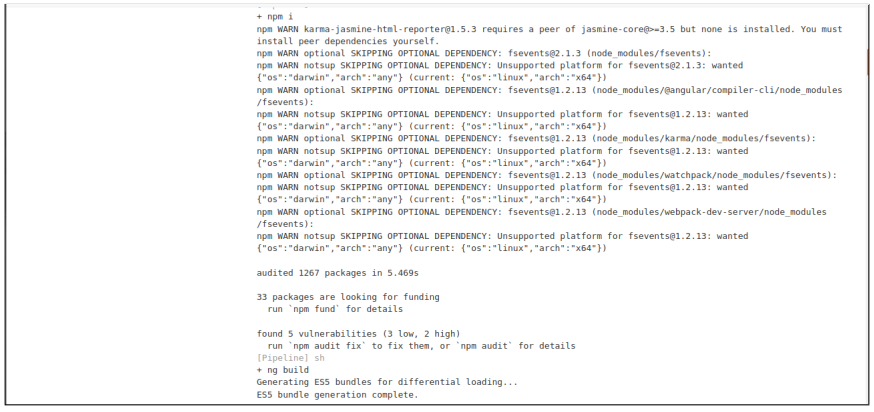


Figure 41: Stage 2 : Build the angular project - 1

Figure 42: Stage 2 : Build the angular project - 2

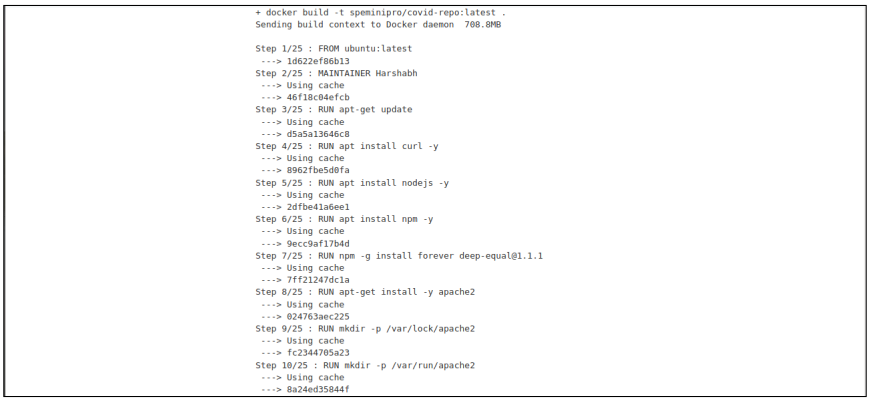


Figure 43: Stage 3 : Build the docker image - 1

Figure 44: Stage 3 : Build the docker image - 2

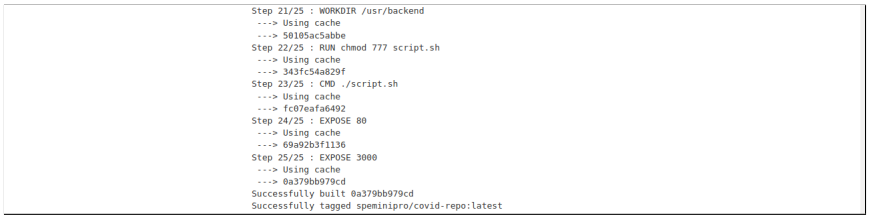


Figure 45: Stage 3 : Build the docker image - 3

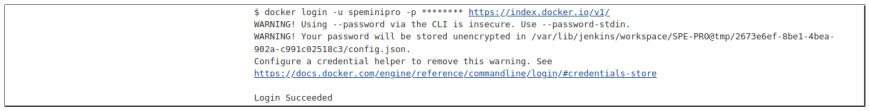


Figure 46: Stage 4 : Connect to the DockerHub

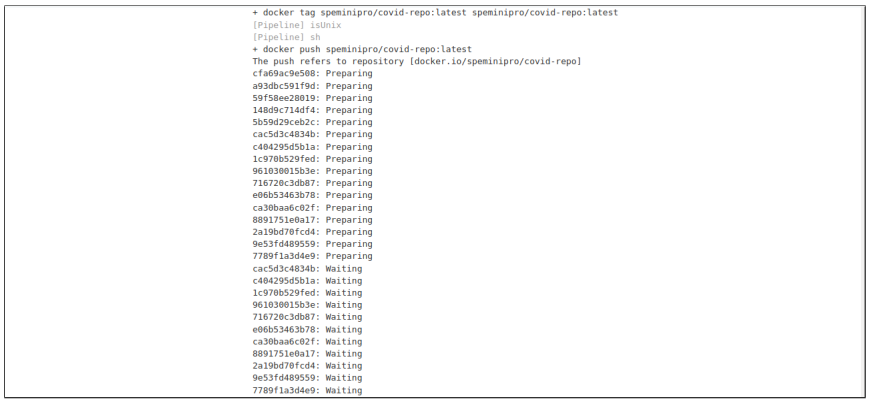


Figure 47: Stage 4 : Push the image to DockerHub - 1

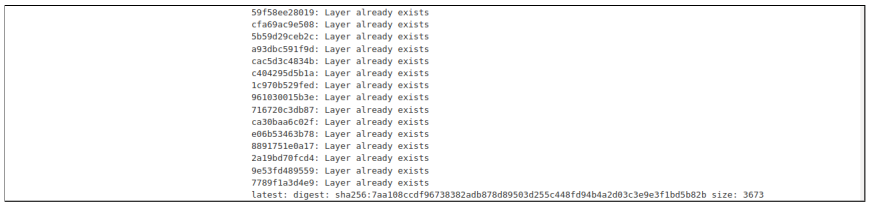


Figure 48: Stage 4 : Push the image to DockerHub - 2

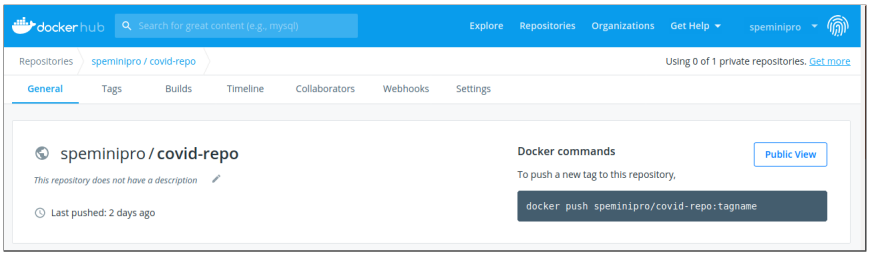


Figure 49: Stage 4 : Image pushed to DockerHub

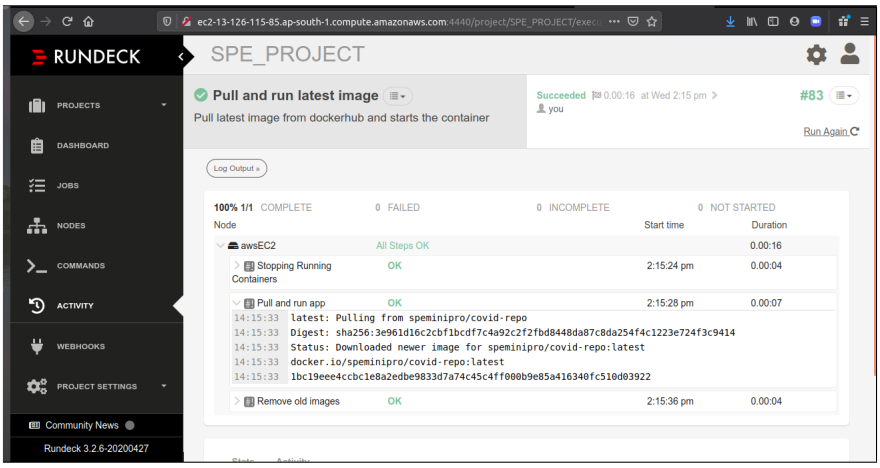


Figure 50: Stage 5 : Image pulled on Rundeck node

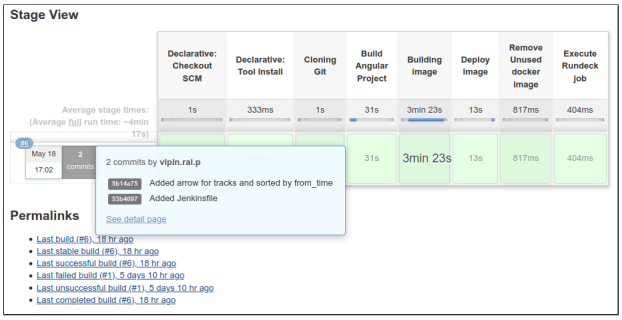


Figure 51: Successful execution of CI/CD pipeline

6 Project Architecture and workflow

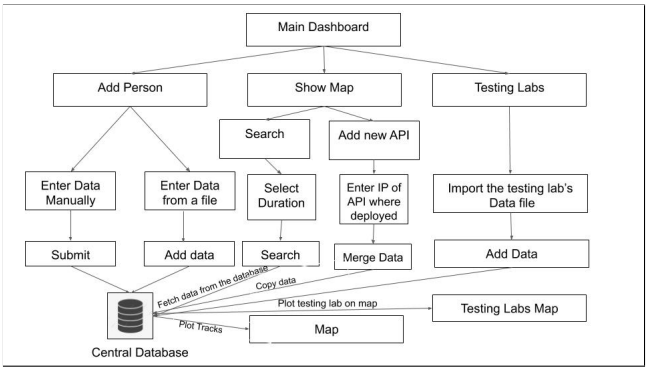


Figure 52: Execution workflow of each functionality of the application

1. Main Dashboard : Once the application is launched, main dashboard appears. This dashboard has three options : Add person, show Map, and Testing Labs.
   With the help of add person, entry of new person can be added to the database. With the help of show map, details present in the database can be viewed on the map. With the help of testing labs option, if new testing lab is created anywhere in the country then its data can be added to the database by importing its files.

2. Add Person : Once clicked on Add Person, person data can be added by two ways : Import through file and add manually
   If data is present in file, then simply by importing that file, entire data can be added to the database.
   Else data can be added manually where the information needed to be added include, ID, Address, City, State, whether person is infected or not and travel history which contain - location, mode of transport and date-time of journey.

3. Show Map : Once clicked on Show Map, date-time span needs to inspected, needs to be entered in respective fields. Once the date-time is entered, by clicking plot tracks, travel history of every patient can be viewed on google maps. Another option is to merge data from two different regions. If application is deployed in two different regions and we need to merge these two region's data then by adding the ip address of other region's application, we can merge its database with the central database.[10][11][12]

4. Testing Labs : If any new testing lab opens in the country, then with the help of this option, its data can be added to the central database. File which contains the testing data needs to be imported and with the help

of add data option, the data can be added to the central database of the application.[13]

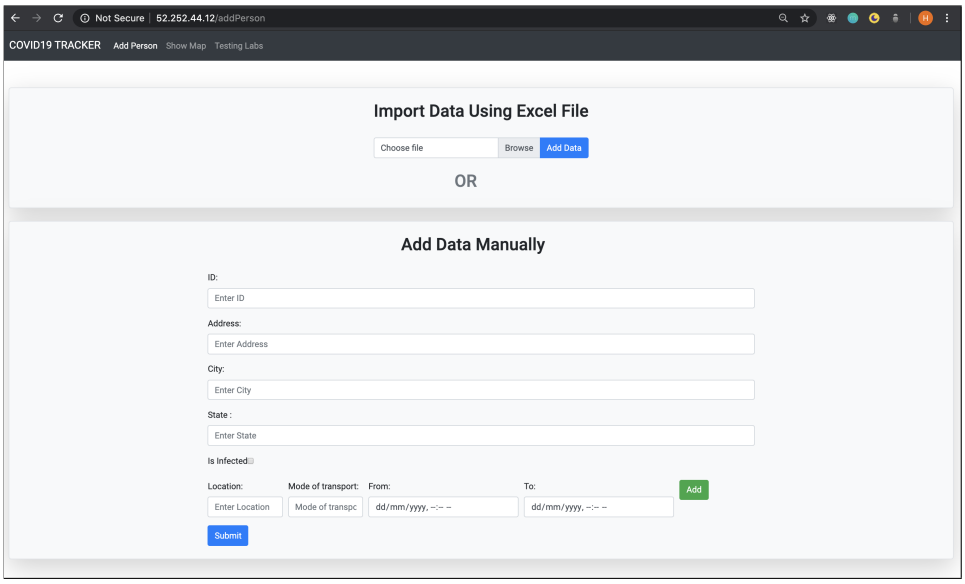


Figure 53: Add Person functionality

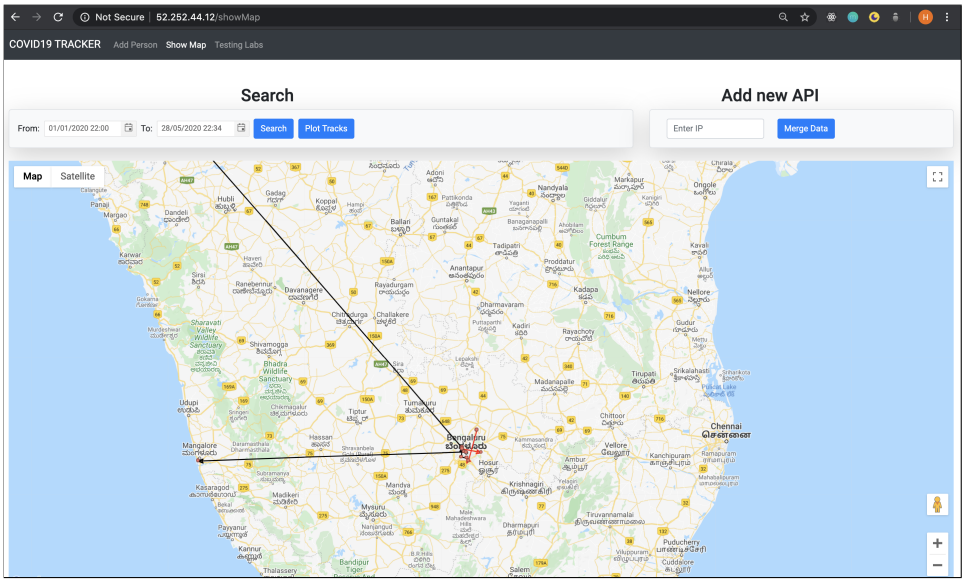


Figure 54: Show Map functionality

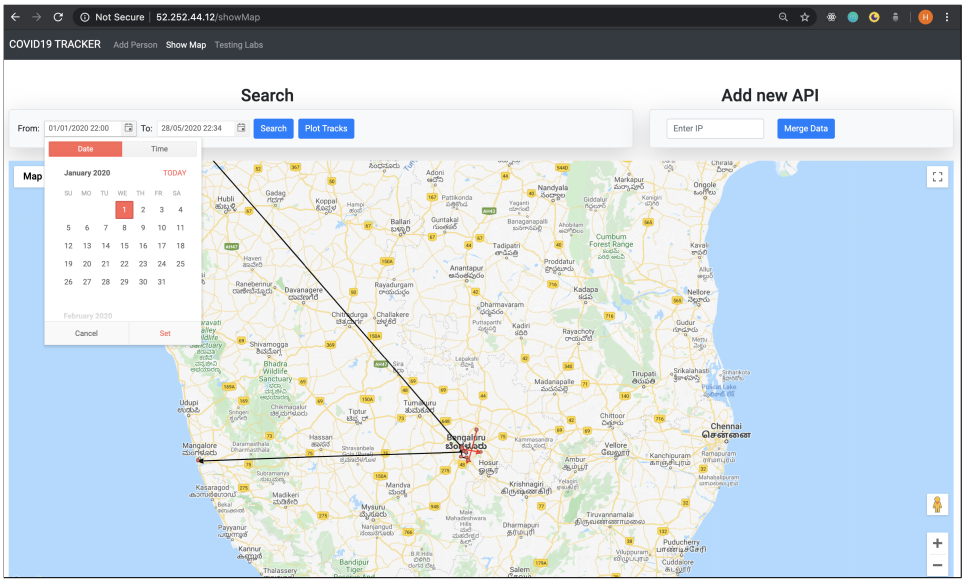


Figure 55: Map view with information about the Patient - I

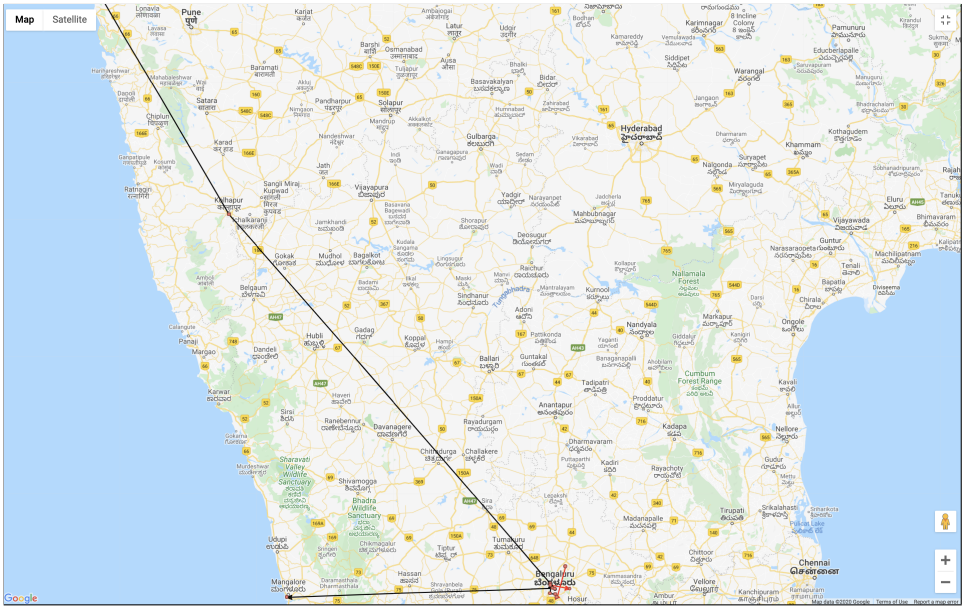


Figure 56: Map view with information about the Patient - II

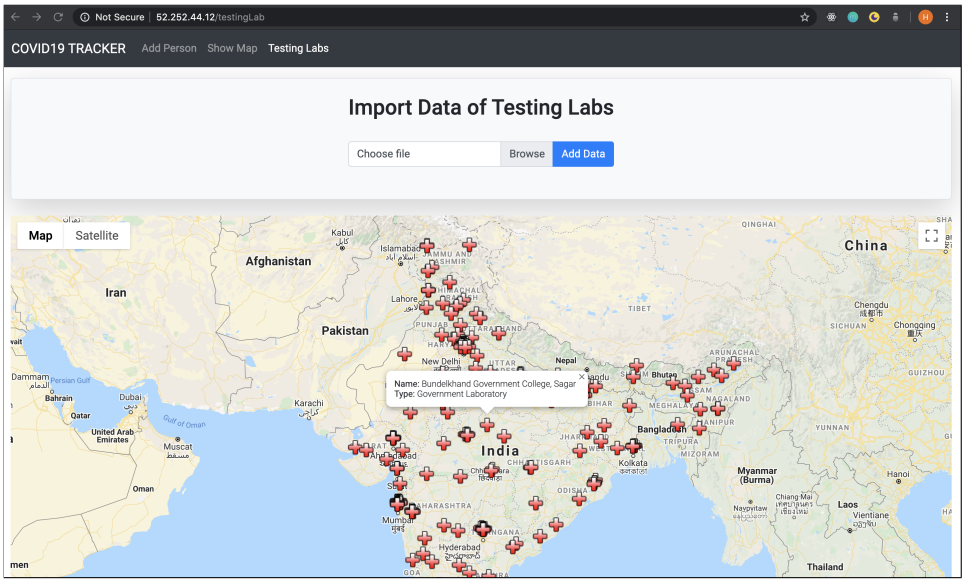


Figure 57: Map view showing Testing Labs across India

7 Future Work

Currently the application track the travel history of only COVID-19 infected patients. In future, we will extend this project to track patients infected with various diseases. Along with this we also expect to develop this on state level tracking so that each state can use this application according to their own necessities.

8 Conclusion

Due to recent rise of the pandemic COVID-19, it is necessary for each person to stay aware about its neighborhood. Having information about the people, also helps the authority to take necessary decisions. With the help of such tracking system, task of keeping eye on the people becomes easy and efficient. Hence this application can prove to be handy for the government to take necessary decisions to keep people in the country safe.

9 Links of the code and docker image

1. Github : https://github.com/VipinRaiP/spe-covid19.git
2. Docker Hub : https://hub.docker.com/repository/docker/speminipro/covid-repo

10 References

1. Docker Basic Hands-on Guide : https://github.com/Alakazam03/demo-app-nodejs
2. Angular testing tools : https://www.concretepage.com/angular/angular-test-input-text
3. Angular testing guide : https://angular.io/guide/testing
4. ELK Tutorial : https://github.com/Alakazam03/ELK-Tutorial
5. Angular logging : https://onthecode.co.uk/angular-logging-made-easy-with-ngx-logger/
6. Angular logging with ELK : https://blog.donkeycode.com/superpower-angular-logging-with-elasticsearch-logstash-and-kibana-part-1-cfb612da4b87
7. Elastic Stack installation on Ubuntu : https://www.howtoforge.com/tutorial/how-to-install-elastic-stack-ubuntu-1804/
8. Installing jenkins on aws : https://medium.com/@Marklon/how-to-install-jenkins-on-ubuntu-16-04-on-aws-e584c45c2684
9. Enabling Continuous Deployment Through Jenkins : https://medium.com/vishweshvinchurkar/enabling-continuous-deployment-through-jenkins-471adbb68314
10. Maps JavaScript API : https://developers.google.com/maps/documentation/javascript/tutorial
11. Google Maps JavaScript API Tutorial : https://www.youtube.com/watch?v=Zxf1mnP5zcw
12. Google Maps API Integration with Angular : https://medium.com/@jkeung/integrating-google-maps-api-w-angular-7-e7672396ce2d
13. Data source for testing Lab : https://www.kaggle.com/sudalairajkumar/covid19-in-india