

"Application Hosting Using DevOps Tools"

A PROJECT REPORT SUBMITTED TO
THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(An Autonomous Institute under VTU, Belagavi)



In partial fulfillment of the requirements for Project work (Minor Project CS6C06),
sixth semester

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by

Karuturi Chandini (4NI18CS030)

Sanjita V Nayak (4NI18CS076)

Sinchana K P (4NI18CS090)

Under the Guidance of

Sumana K M
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE NATIONAL INSTITUTE OF ENGINEERING

Mysuru-570 008

2020-2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE NATIONAL INSTITUTE OF ENGINEERING



CERTIFICATE

This is to certify that the project work entitled “**Application Hosting Using DevOps Tools**” is a work carried out by **Karuturi Chandini (4NI18CS030)**, **Sanjita V Nayak (4NI18CS076)** and **Sinchana K P (4NI18CS090)** in partial fulfillment for the project work (Minor Project -CS6C06), sixth semester, Computer Science & Engineering, The National Institute of Engineering (Autonomous Institution under Visvesvaraya Technological University, Belagavi) during the academic year 2020-2021. It is certified that all corrections and suggestions indicated for the Internal Assessment have been incorporated in the report deposited in the department library. The project work report has been approved in partial fulfillment as per academic regulations of The National Institute of Engineering, Mysuru.

Signature of the Guide

Sumana K M
Assistant Professor
Dept. of CS&E
NIE, Mysuru

Signature of the HoD

Dr. V K Annapurna
Professor and Head
Dept. of CS&E
NIE, Mysuru

ACKNOWLEDGEMENT

We would like to take this opportunity to express our profound gratitude to all those people who were directly or indirectly involved in the completion of this project. I thank each and every one who encouraged us in every possible way.

We would like to thank our Principal Dr. N V Raghavendra for letting us be a part of this prestigious institution and letting us explore our abilities to the fullest. We would like to extend our sincere gratitude to Dr. V K Annapurna, our H.O.D for being the source of inspiration and instilling an enthusiastic spirit in us throughout the process of project making.

We would like to express heartfelt gratitude towards our project guide Sumana KM for their constant guidance, valuable knowledge and experience. We are thankful to our parents, family and friends for their constant support, inspiration and encouragement in the academic process.

Thanking you

-Karuturi Chandini

-Sanjita V Nayak

-Sinchana K P

Table of Contents

<u>Sl.no</u>	<u>Contents</u>	<u>P.no</u>
1	Introduction	1
2	System Analysis The Evolution of DevOps Culture What is DevOps? DevOps tools used DevOps and Cloud : Symbiotic relationship	2-12
3	Application Frameworks Frontend design Backend design	13
4	System Requirements Hardware requirements Software requirements Storage requirements	14-16
5	Project Workflow Building application code Working with Git Working with Docker Working with Jenkins Hosted application	17-25
6	System Testing	26-27
7	Conclusion and Future Enhancements	28
8	References	29

List of Figures

<u>Figure No.</u>	<u>Description</u>	<u>Page</u>
2.1	DevOps	2
2.2	Without DevOps	3
2.3	With DevOps	3
2.4	DevOps with CI/CD	3
2.5	DevOps tools	4
2.6	GitHub	4
2.7	Functioning of Git	5
2.8	Docker Architecture	6
2.9	Building Docker image	7
2.10	Docker container vs virtual machines	7
2.11	Jenkins	8
2.12	Build and publish Docker image with Jenkins	9
2.13	DevOps and Cloud	10
2.14	AWS code pipeline	12
4.1	Configuring storage	14
4.2	Configuring security groups	14
4.3	Configuring Git	15
4.4	Configuring Docker	15
4.5	Configuring Jenkins	15
4.6	Configuring RDS	16
4.7	MySQL Workbench	16
5.1	File and Folders in local	17
5.2	Creating a repository in GitHub	17
5.3	Pushing the code to central repository	18
5.4	Central repository after pushing the code	18
5.5	Cloning the code into instance	19
5.6	Docker-compose.yml file	19
5.7	Building and running backend image	20

<u>Figure No.</u>	<u>Description</u>	<u>Page</u>
5.8	Building and running frontend image	20
5.9	Pull the code from GitHub central repository	21
5.10	Build and Publish the image to Docker Hub repository	21
5.11	Automatically schedule a job whenever changes are committed in the GitHub repository	22
5.12	Scheduled jobs	22
5.13	Home page 1	23
5.14	Home page 2	23
5.15	List of clubs at NIE	24
5.16	Registration form	24
5.17	Response after submitting form	25
5.18	Confirmation mail received by the student	25
5.19	Membership request mail sent to the core of the club	25
6.1	Continuous Testing	26
6.2	Continuous Testing using Jenkins	27
6.3	Console of a failed testcase	27

CHAPTER 1

INTRODUCTION

The main objective of project is to host an application using various DevOps tools. This system helps to reduce the problems faced by developers and IT professionals. It helps to manage the source code in easier and efficient way. Customers need continuous engagement with the project so that they can provide the feedback continuously. Over the years the organizations have adopted the agile transformation for optimization, but the evolution takes place to change the technology to DevOps. It is important to keep all the phases in pace so that the software delivery lifecycle will not be delayed. DevOps is the mechanism which bridges the gap between developer-operations and not only limited to developer-operations but also for the continuous development, continuous testing and continuous integration.

DevOps is the collaboration of development and deployment of software. DevOps is the portmanteau of development and operations. It is a software development method that escalates to the amalgamation between software development team and operations team. This is the time to change the old technology to new technology like DevOps. - “Time to stop wasting money, time to start delivering great software and building systems that scale and last” –Patrick Debois

DevOps main goal is to deliver the software rapidly with continuous development, continuous integration, continuous feedback and communication with development and operations team. DevOps is the extension of agile principles in software delivery pipeline. DevOps principles place an important role in complete Systems Development Life Cycle, but it makes sense if both the development team and operations team work in a same pace

CHAPTER 2

SYSTEM ANALYSIS

2.1 The Evolution of DevOps Culture

There are many available resources dedicated to delivering better business value faster. Based on this advice, many organizations and teams are adopting common software delivery practices — even outside the usual use case of providing software (take, for example, SpaceX teams using agile, or leadership teams that use scrum practices and have daily stand-up calls). This progression into faster, better, cheaper and easier — no matter what you're delivering — is what makes incorporating DevOps into business practices critical to delivering unmatched value for end users.

Then and Now

A great perspective to consider when approaching the subject of DevOps is how technology has changed just about everything — how we build, connect, share and educate. Up until the 2000s, for example, the standard turnaround time for taking and sharing pictures was one to three weeks, if not longer. This process used to involve film cameras, darkrooms to process the photographs, photo finishing labs and hard copy, physical prints. Today, our mobile devices allow us to snap hundreds of professional-quality images and share them immediately online.

These earth-shattering changes don't apply to photography alone, but to many other industries including financial services, banking and IT. Consistently, technology is finding new ways to disrupt industries, priming them to be more efficient to fit into our fast-moving lifestyles.

2.2 What is DevOps?

DevOps is a culture that allows the development and operations teams to work together. With this type of working environment, developers continuously develop and test codes, and there is continuous integration taking place throughout the lifecycle. The operations team continuously deploys the code to the production environment. DevOps allows for better collaboration, increased trust, and faster software releases.

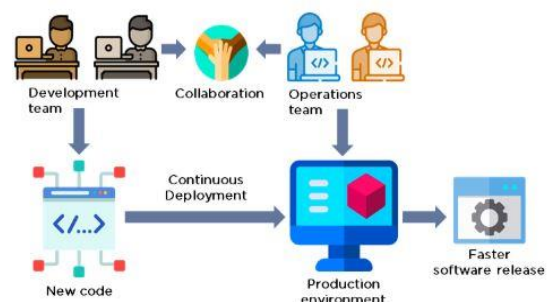


Fig 2.1 DevOps

The DevOps life cycle

In the past, the process of traditional software delivery was linear and sequential; waterfall methodologies offered little flexibility and did not succumb to the nature of change and demand for technology services we see today. What we've experienced with software delivery now uses DevOps to leverage new insights through people, processes and technology, giving organizations the ability to deliver continuous business value. It means that businesses are adding value to our lives all the time, whether we see it or not. For technology organizations, this process for delivering business value is modeled through what we call the DevOps life cycle.

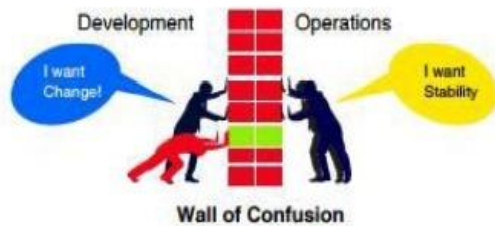


Fig 2.2 Without DevOps

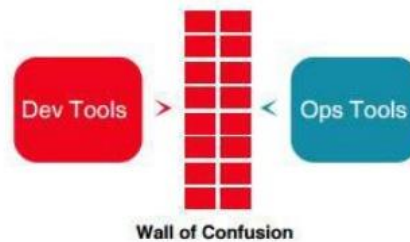


Fig 2.3 With DevOps

One of the main responsibilities of DevOps managers is to support the creation of a release pipeline, which is the route a piece of code or feature must travel from a developer's local computer into production.

There are different best practices to apply in the building of the pipeline: Continuous Delivery, Continuous Deployment, Continuous Integration, and more. Nevertheless, when creating the ideal release pipeline, you should make sure that it will:

- Increase product quality
- Increase the agility of the product's development
- Reduce the stress and last-minute panic associated with product releases



Fig 2.4 DevOps with CI/CD

2.3 DevOps Tools Used

DevOps tools ensure transparency, automation, and collaboration stay at the forefront of your value stream. These tools facilitate ways for effective sharing and exchange of information and technical know-how between all stakeholders be it development, operations, security or business teams for effective product output.

DevOps tools help firms to resolve some of the challenges that come with the implementation of DevOps practices. However, there is no “one-size-fits-all” solution available out there. As a result, there is a wide variety of DevOps tools for every requirement.

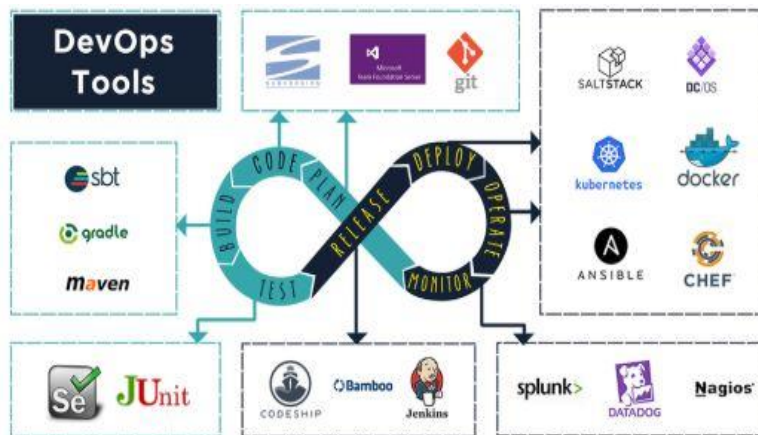


Fig 2.5 DevOps Tools

2.3.1 Version Control Tools: GitHub

Github is considered as one of the largest and most advanced development platforms in the world. Millions of developers and companies build, ship, and maintain their software on GitHub. Some of its' salient features are:

- Collaborative Coding
- Automation / CI & CD
- Security including additional features for enterprise customers
- Project Management

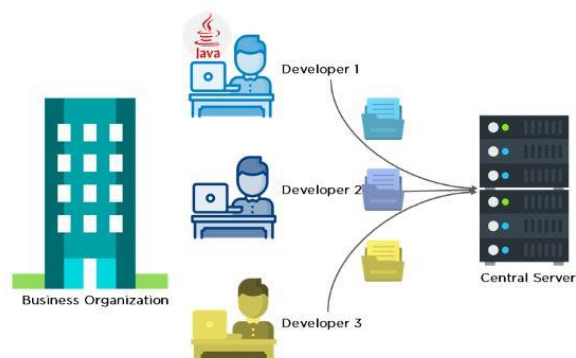


Fig 2.6 GitHub

Scenario before Git:

- Developers used to submit their codes to the central server without having copies of their own
- Any changes made to the source code were unknown to the other developers
- There was no communication between any of the developers

Scenario after Git:

- Every developer has an entire copy of the code on their local systems
- Any changes made to the source code can be tracked by others
- There is regular communication between the developers

Features of Git

- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development

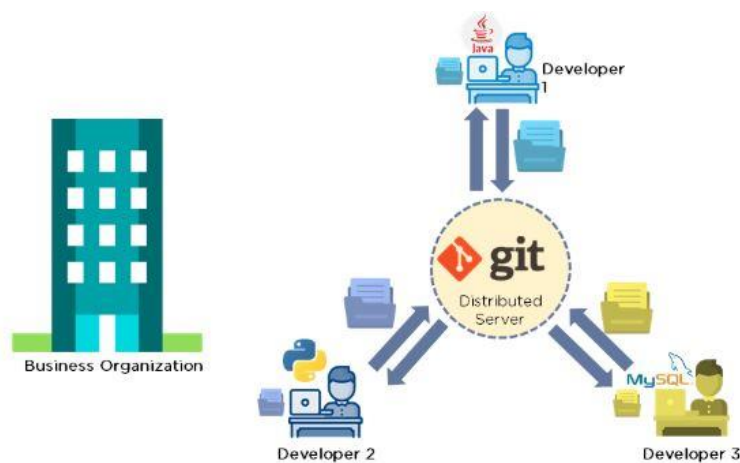


Fig 2.7 Functioning of Git

Git Workflow

The Git workflow is divided into three states:

- Working directory - Modify files in your working directory
- Staging area (Index) - Stage the files and add snapshots of them to your staging area
- Git directory (Repository) - Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version, make changes, stage them and commit.

2.3.2 Container Management tool: Docker

Docker is a light-weight tool that aims to simplify and accelerate various workflows in your SDLC with an integrated approach. A docker container image is a standalone, executable package that includes everything you need to run an application. Some of its primary features are that helped it become indispensable among DevOps tools are:

- Standardized Packaging format for diverse applications.
- Container runtime that runs on various Linux and Windows Server OSs.
- Developers uses Docker for build, test and collaborate.
- Docker Hub to explore millions of images from community and verified publishers.
- Package, Execute and Manage distributed applications with Docker App.

Docker Architecture

There are four main internal components in Docker ,which include Docker Client and Server, Docker Images,Docker Registries and Docker Containers.

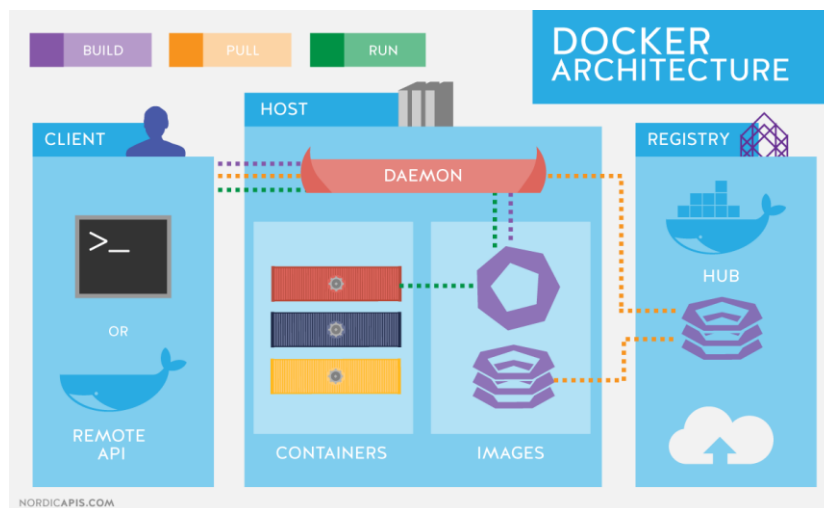


Fig 2.8 Docker Architecture

1. Docker Client and Server

The docker server gets the request from the docker client and then process it accordingly. The complete RESTful API and a command line client binary are shipped by docker. Docker daemon/server and docker client can be run on the same machine or a local docker client can be connected with a remote server or daemon, which is running on another machine.

2. Docker Images

There are two methods to build an image. The first one is to build an image by using a read-only template. The foundation of every image is a base image. Operating system images are basically the base images, such as Ubuntu 14.04 LTS, or Fedora 20. The images of operating system create a container with an ability of complete running OS. Base image can also be created from the scratch. Required applications can be added to the base image by modifying it, but it is necessary to build a new image. The process of building a new image is called “committing a change”. The second method

is to create a docker file. The docker file contains a list of instructions when “Docker build” command is run from the bash terminal it follows all the instructions given in the docker file and builds an image. This is an automated way of building an image.

3. Docker Registries

Docker images are placed in docker registries. It works correspondingly to source code repositories where images can be pushed or pulled from a single source. There are two types of registries, public and private. Docker Hub is called a public registry where everyone can pull available images and push their own images without creating an image from the scratch. Images can be distributed to a particular area (public or private) by using docker hub feature.

4. Docker Containers

Docker image creates a docker container. Containers hold the whole kit required for an application, so the application can be run in an isolated way. For example, suppose there is an image of Ubuntu OS with SQL SERVER, when this image is run with docker run command, then a container will be created and SQL SERVER will be running on Ubuntu OS.

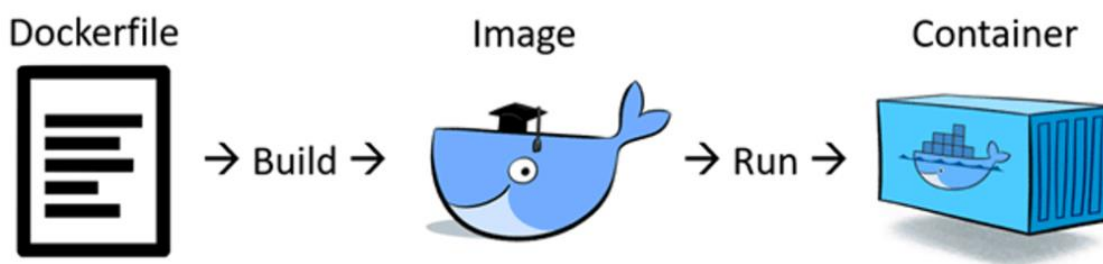


Fig 2.9 Building Docker image

Docker V/s Virtual Machines

The operating system support of Virtual machine and Docker container is very different. From the image above, you can see each virtual machine has its guest operating system above the host operating system, which makes virtual machines heavy. While on the other hand, Docker containers share the host operating system, and that is why they are lightweight. Sharing the host operating system between the containers make them very light and helps them to boot up in just a few seconds. Hence, the over

Containers vs. VMs

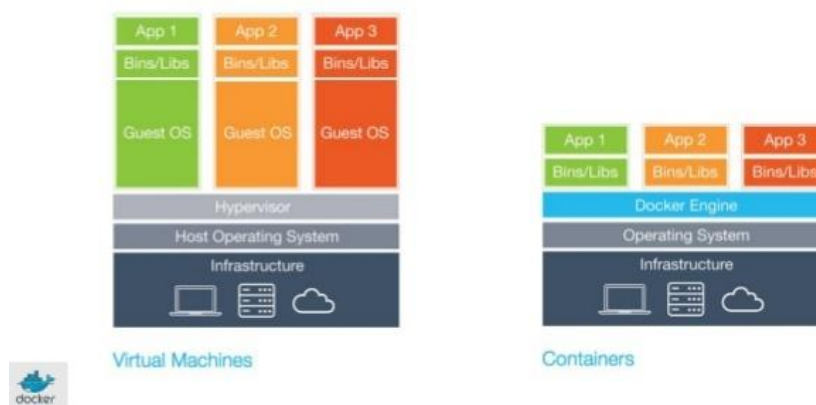


Fig 2.10 Docker Containers v/s Virtual Machines

2.3.3 CI / Deployment Automation tool: Jenkins

Written in Java, Jenkins is an open-source platform for continuous integration and continuous delivery that is used to automate your end-end release management lifecycle. Jenkins has emerged as one of the essential DevOps Tools because of its features:

- Used as a simple CI server or turned into the CD hub for any project.
- Easily set up and configured via its web interface, which includes on-the-fly error checks and built-in help.
- Easily distribute work across multiple machines, helping drive builds, tests and deployments across multiple platforms faster.

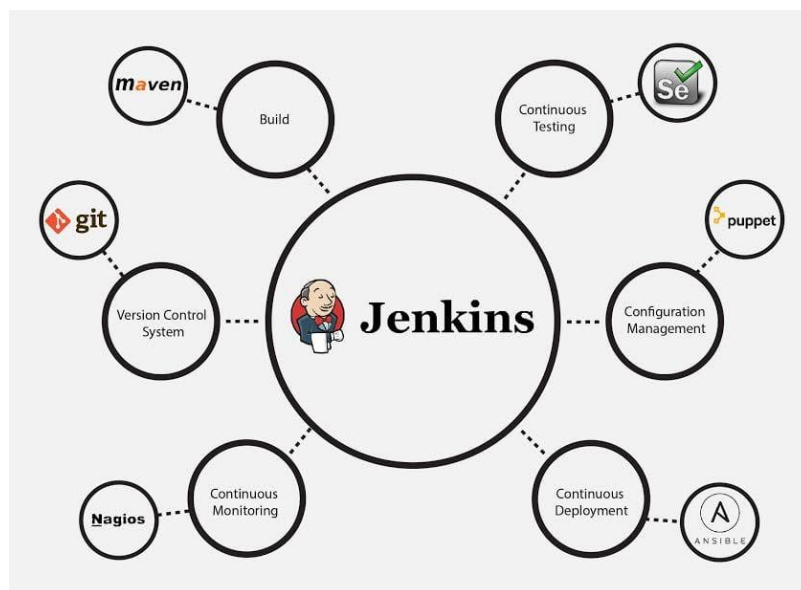


Fig 2.11 Jenkins

What is Continuous Integration?

Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early. Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results, etc.

Continuous Integration is the most important part of DevOps that is used to integrate various DevOps stages. Jenkins is the most famous Continuous Integration tool.

Jenkins achieves Continuous Integration with the help of plugins. Plugins allow the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example Git, Maven 2 project, Amazon EC2, HTML publisher etc.

Before and After Jenkins

Before Jenkins	After Jenkins
The entire source code was built and then tested. Locating and fixing bugs in the event of build and test failure was difficult and time-consuming, which in turn slows the software delivery process.	Every commit made in the source code is built and tested. So, instead of checking the entire source code developers only need to focus on a particular commit. This leads to frequent new software releases.
Developers have to wait for test results	Developers know the test result of every commit made in the source code on the run.
The whole process is manual	We only need to commit changes to the source code and Jenkins will automate the rest of the process for you.

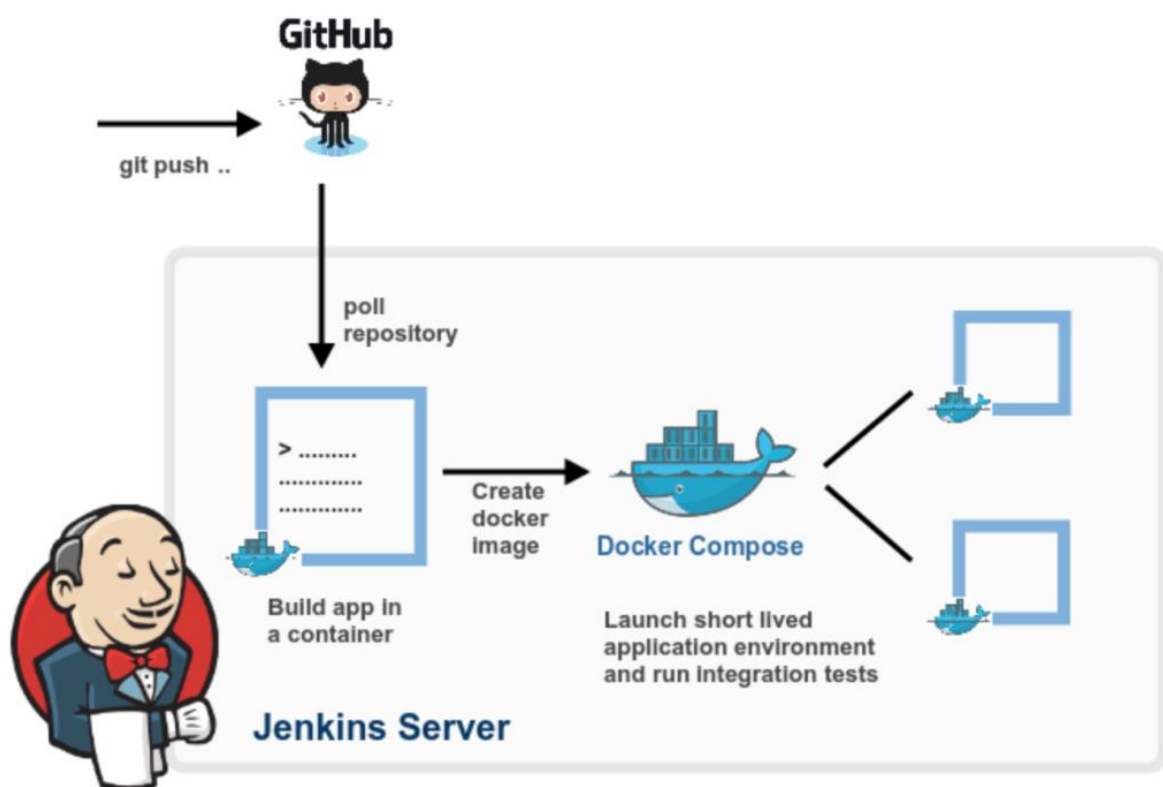


Fig 2.12 Build and Publish Docker image with Jenkins

2.4 DevOps and Cloud: A symbiotic Relationship

Most companies want to increase their competitiveness in today's swiftly changing world, and so they cannot ignore digital transformation. DevOps and cloud computing have become two of the ways companies can achieve this needed transformation, though the relationship between the two is not easily reconciled—DevOps is about the process and process improvement, while cloud computing is about technology and services. It's important to understand how the cloud and DevOps work together to help businesses achieve their transformation goals.

DevOps definitions generally fall into two terms:

1. In organizations it is defined as developer-friendly operations—IT operations are run separately yet in a way that is much more friendly to developers (e.g., self-service catalogs are provided to developers for stipulating infrastructure or providing technology-enabled pipelines for deploying new code).
2. DevOps as a single consolidated team is habituated in organizations—developers take on operations responsibilities and vice versa.

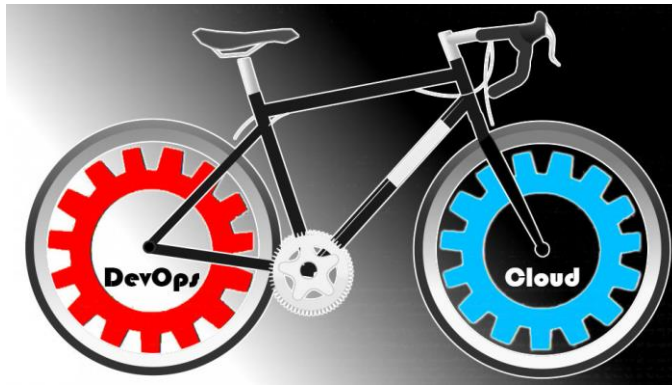


Fig 2.13 Devops and Cloud

The aesthetics of cloud technologies and services become so easily amalgamated in DevOps vocabulary because they complement DevOps processes, regardless of how your organization defines them—or, regardless of which DevOps routes you travel to digital transformation with the cloud integration. Companies that focus on developers for operations often use cloud computing to speed developer productivity and efficiency. Cloud computing permits developers more control over their own components, resulting in smaller wait times. This application-specific architecture makes it easy for developers to own more components. By using cloud tools and services to automate the process of building, managing and provisioning through the code, service teams speed up the development process, eliminate possible human error and establish repeatability. Developers are able to quickly try new things, fail fast and just easily succeed in getting new products to market faster, without having to wait for IT operations to provision services for them.

DevOps as a single term is really a mixture of these approaches, in which developers and operations work together using the cloud as a single common language. In this way, both the DevOps and cloud are able to work together, as everyone is learning new definitions and approaches at the same time. Developers and operations are equally cushy with the new language of the cloud, as developers often teach operations about the code aspect and operations can teach developers about infrastructure and security, developing a meeting point that leads to strong team dynamics.

What is AWS?

Amazon web service is an online platform that provides scalable and cost-effective cloud computing solutions. AWS is a broadly adopted cloud platform that offers several on-demand operations like compute power, database storage, content delivery, etc., to help corporates scale and grow.

Why AWS in DevOps?

There is a much more evident connection between AWS and DevOps. The introductions of AWS and DevOps into the technological world took place almost at the same time (DevOps came a few years later than AWS). Since then, both of them has fueled each other for their expansion and growth. AWS has rolled out many services on its platform that are DevOps friendly and have encouraged the DevOps culture. On the other hand, DevOps tools released for the industries are compatible with the cloud platform and hence facilitating the growth of AWS. This is the reason why several IT companies looking for engineers having the knowledge of DevOps also ask if they are aware of the cloud computing services which also includes AWS. DevOps instructs to take into consideration the importance of release time. It says that to reduce the risk of failure of a product or service, the release time should be reduced. Continuous development and frequent release events lower risk significantly. With the growth of Cloud Computing, AWS also grew and so did DevOps.

AWS Services

Amazon has many services for cloud applications

- Compute Services
- Storage
- Database
- Networking and delivery of content
- Security, Developer and Management tools

Services Used in project:

1. Compute Service: AWS EC2

These services help developers build, deploy, and scale an application in the cloud platform. Amazon Elastic Compute Cloud provides scalable computing capacity in the AWS Cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. It offers various instance types to developers so that they can choose required resources such as CPU, memory, storage, and networking capacity based on their application requirements.

2. Database: RDS

AWS database domain service offers cost-efficient, highly secure, and scalable database instances in the cloud. It is a managed distributed relational database cloud service that helps developers to operate

and scale a database in a simple manner. We launched it to simplify the setup, operation, and scaling process for developers while accessing a relational database.

3. Elastic IP Address

An Elastic IP address is a static IPv4 address designed for dynamic cloud computing. An Elastic IP address is allocated to your AWS account, and is yours until you release it. By using an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account. Alternatively, you can specify the Elastic IP address in a DNS record for your domain, so that your domain points to your instance.

Advantages of AWS Cloud

The power of AWS services lies in the fact that it enables businesses to reach the marketplaces with little initial investment. Here are some advantages of AWS services:

1. Security

There is a false misconception that data stored in a public cloud is not secure. On the contrary, not only does AWS offer security tools that are cheaper than other alternatives, but it is one of the most secure, extensive, and reliable cloud platforms.

2. Global Availability

AWS has 80 Availability Zones across 25 geographic regions global data centers.

3. Scalability and Flexibility

AWS offers unlimited flexibility and scalability on demand. This enables organizations to plan their infrastructure roadmap on a subscription basis without full commitment.

4. Little Investment

AWS cloud services enable companies to save expenditures on extra software and hardware. There is no physical data required, which ultimately lowers down operating costs.

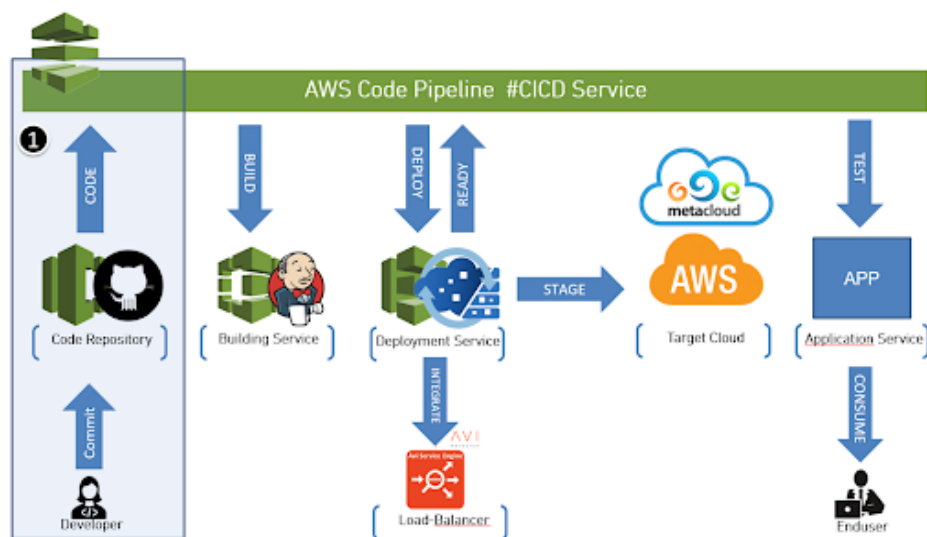


Fig 2.14 AWS Code Pipeline

CHAPTER 3

APPLICATION FRAMEWORKS

3.1 Frontend Design

The hosted application is a static webpage designed using tools like HTML, CSS and Bootstrap for frontend.

HTML: HTML or HYPERTEXT MARKUP LANGUAGE is the standard markup language used to create WebPages. The purpose of the web browser is to read HTML documents and compose them into visible or audible web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page. HTML describes the structure of a website semantically along with cues for presentation, making it a markup language rather than a programming language.

CSS: Cascading style sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging web pages, user interface for web applications, and user interface for many mobile applications. CSS is designed primarily to the separation of document content from document presentation, including aspects such as the layout, colors and fonts .

BOOTSTRAP: Bootstrap is a powerful toolkit - a collection of HTML, CSS, and JavaScript tools for creating and building web pages and web applications. It is a potent front-end framework used to create modern websites and web apps. It's open-source and free to use, yet features numerous HTML and CSS templates for UI interface elements such as buttons and forms. Bootstrap also supports JavaScript extensions.

3.2 Backend Design

Flask: Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

MySQL: MySQL is an open-source relational database management system . A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

CHAPTER 4

SYSTEM REQUIREMENTS

The software requirement specification (SRS) and hardware specification forms the basis of software development. A main purpose of software requirement specification is the clear definition and specification of functionality and of the software product. It allows the developer to be carried out, performance level to be obtained and corresponding interface to be established.

4.1 Hardware requirements

Setting up EC2 instance

OS: Ubuntu 20.4.0

Storage: 8 GB

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encryption
Root	/dev/sda1	snap-0c063602c11839b7c	8	General Purpose SSD (gp2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

[Add New Volume](#)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. [Learn more](#) about free usage tier eligibility and usage restrictions.

Fig 4.1 Configuring storage

Security groups:

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

Security group name:

Description:

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Anywhere 0.0.0.0/0	e.g. SSH for Admin Desktop
HTTP	TCP	80	Anywhere 0.0.0.0/0	e.g. SSH for Admin Desktop
HTTPS	TCP	443	Anywhere 0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP	TCP	8080	Anywhere 0.0.0.0/0	e.g. SSH for Admin Desktop

[Add Rule](#)

Warning
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Fig 4.2 Configuring security groups

4.2 Software requirements

Setting up git in the launched instance

```
ubuntu@ip-172-31-38-66: ~/minor-project
Fetchd 19.4 MB in 6s (3446 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-38-66:~$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.25.1-1ubuntu3.1).
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 81 not upgraded.
ubuntu@ip-172-31-38-66:~$ git --version
git version 2.25.1
```

Fig 4.3 Configuring Git

Setting up Docker

```
Select root@ip-172-31-38-66: /home/ubuntu
40 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Sat Jul 10 05:34:42 2021 from 157.50.27.244
ubuntu@ip-172-31-38-66:~$ sudo su
root@ip-172-31-38-66: /home/ubuntu# ls
minor-project
root@ip-172-31-38-66: /home/ubuntu# apt-get install docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base libidn11 pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io libidn11 pigz runc ubuntu-fan
0 upgraded, 9 newly installed, 0 to remove and 81 not upgraded.
Need to get 72.6 MB of archives.
After this operation, 352 MB of additional disk space will be used.
Do you want to continue? [Y/n] yes
```

Fig 4.4 Configuring Docker

Setting up Jenkins

```
Select ubuntu@ip-172-31-38-66:~
ubuntu@ip-172-31-38-66:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
OK
ubuntu@ip-172-31-38-66:~$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.l
ist'
ubuntu@ip-172-31-38-66:~$ sudo apt install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package jenkins is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source
```

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

Fig 4.5 Configuring Jenkins

4.3 Storage requirements

Setting up RDS in AWS

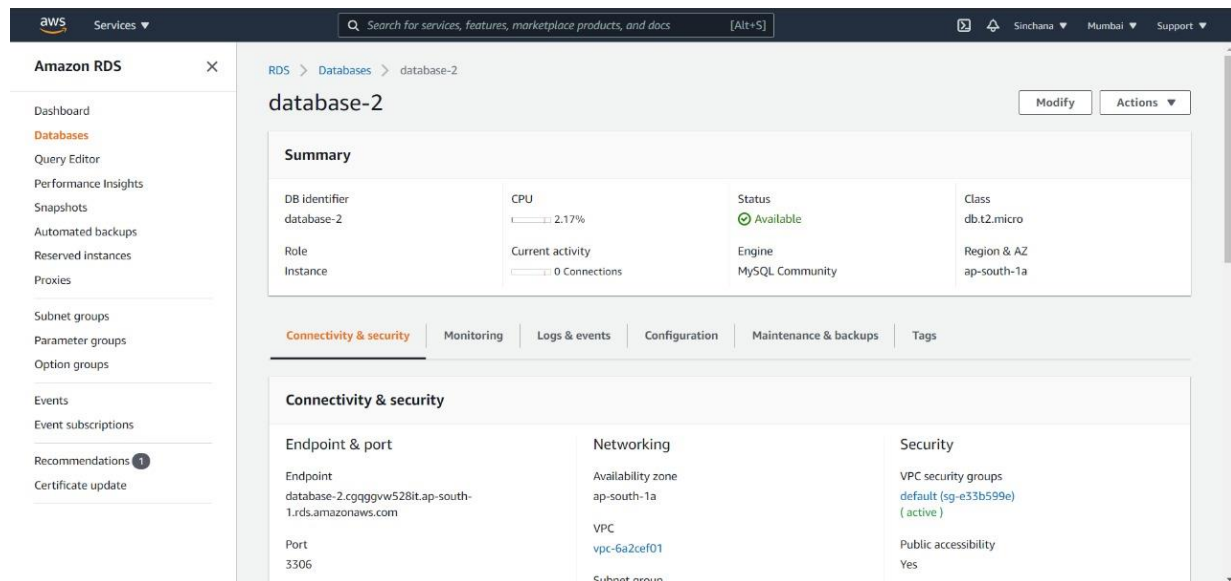


Fig 4.6 Configuring RDS

Accessing Database through MySQL Workbench

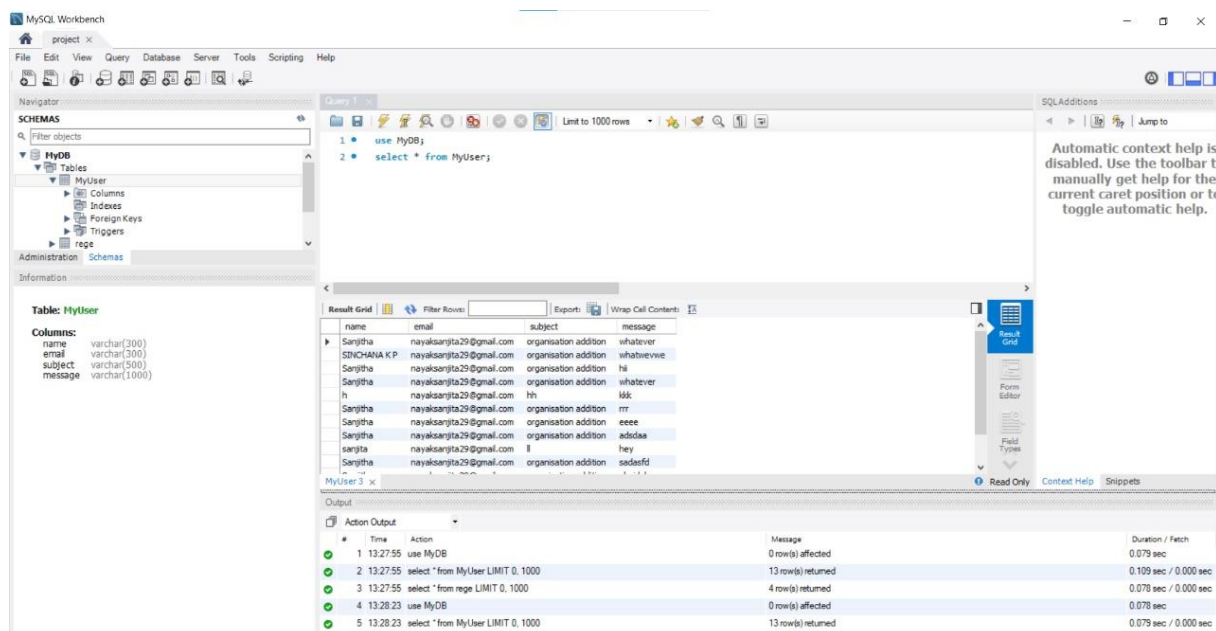


Fig 4.7 MySQL Workbench

CHAPTER 5

PROJECT WORKFLOW

5.1 Building Application code

The aim to build this web application is to help freshers with all the information regarding various student clubs at NIE. Once the website is opened the user is directed to the home page which contains information about the college. The student can go through all the information about the clubs, if interested can request for the registration by filling the form. This request will help the cores of respective clubs to contact the interested student by sending a mail. The whole application code is built to support all these features.

The frontend code lies in the templates and static folder. The backend code is scripted in python using flask framework.

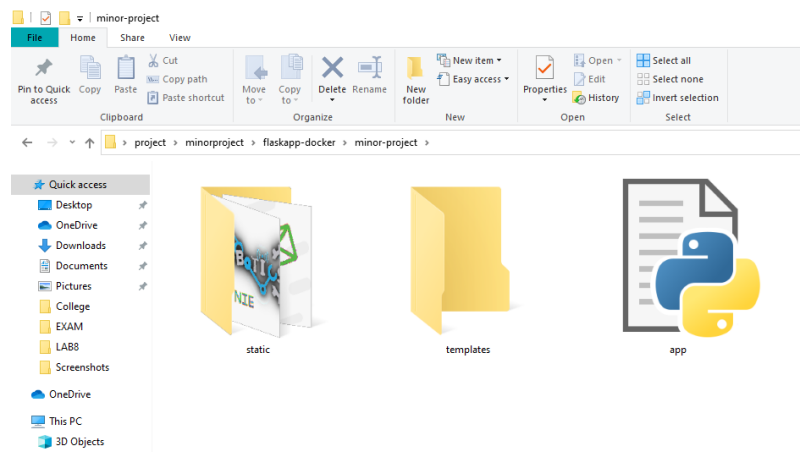


Fig 5.1 File and folders in local

5.2 Working with Git

We created a repository in GitHub for centralizing the code.

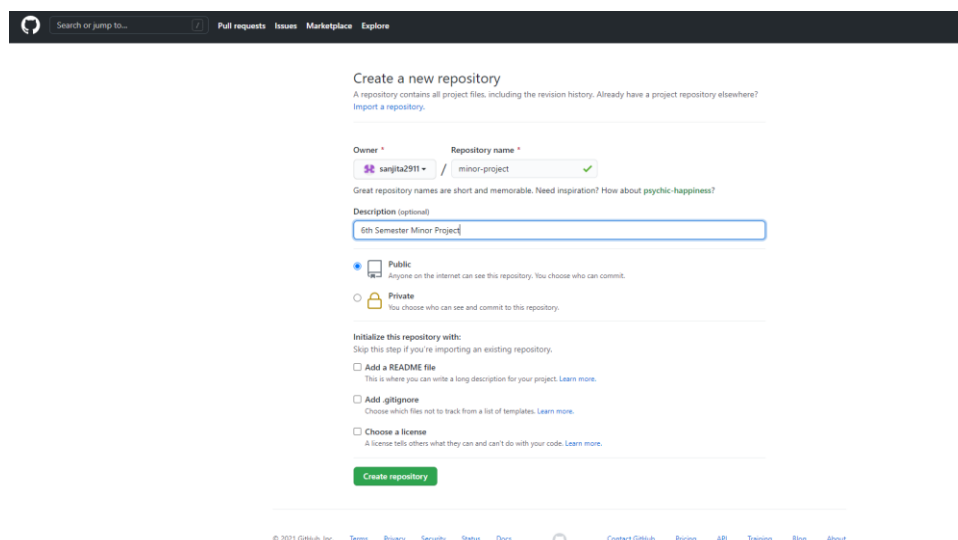


Fig 5.2 Creating a repository in GitHub

Pushing the code from local to GitHub repository

```
MINGW64 /d/NAYAK/Desktop/project/minorproject
Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 /d/NAYAK/Desktop/project/minorproject (master)
$ git remote rm origin

Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 /d/NAYAK/Desktop/project/minorproject (master)
$ git remote add origin "https://github.com/sanjita2911/minor-project.git"

Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 /d/NAYAK/Desktop/project/minorproject (master)
$ cd

Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 ~ (b2)
$ cd /d/NAYAK/Desktop/project/minorproject

Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 /d/NAYAK/Desktop/project/minorproject (master)
$ dir
docker-compose.yml flaskapp-docker nginx

Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 /d/NAYAK/Desktop/project/minorproject (master)
$ git add docker-compose.yml

Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 /d/NAYAK/Desktop/project/minorproject (master)
$ git commit -m "yml file"
On branch master
nothing to commit, working tree clean

Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 /d/NAYAK/Desktop/project/minorproject (master)
$ git push origin master
Enumerating objects: 166, done.
Counting objects: 100% (166/166), done.
Delta compression using up to 8 threads
Compressing objects: 100% (143/143), done.
Writing objects: 100% (166/166), 3.62 MiB | 2.18 MiB/s, done.
Total 166 (delta 16), reused 166 (delta 16), pack-reused 0
remote: Resolving deltas: 100% (16/16), done.
To https://github.com/sanjita2911/minor-project.git
 * [new branch] master -> master

Sanjita Nayak@DESKTOP-IBCA4PL MINGW64 /d/NAYAK/Desktop/project/minorproject (master)
$ !
```

Fig 5.3 Pushing the code to Central repository

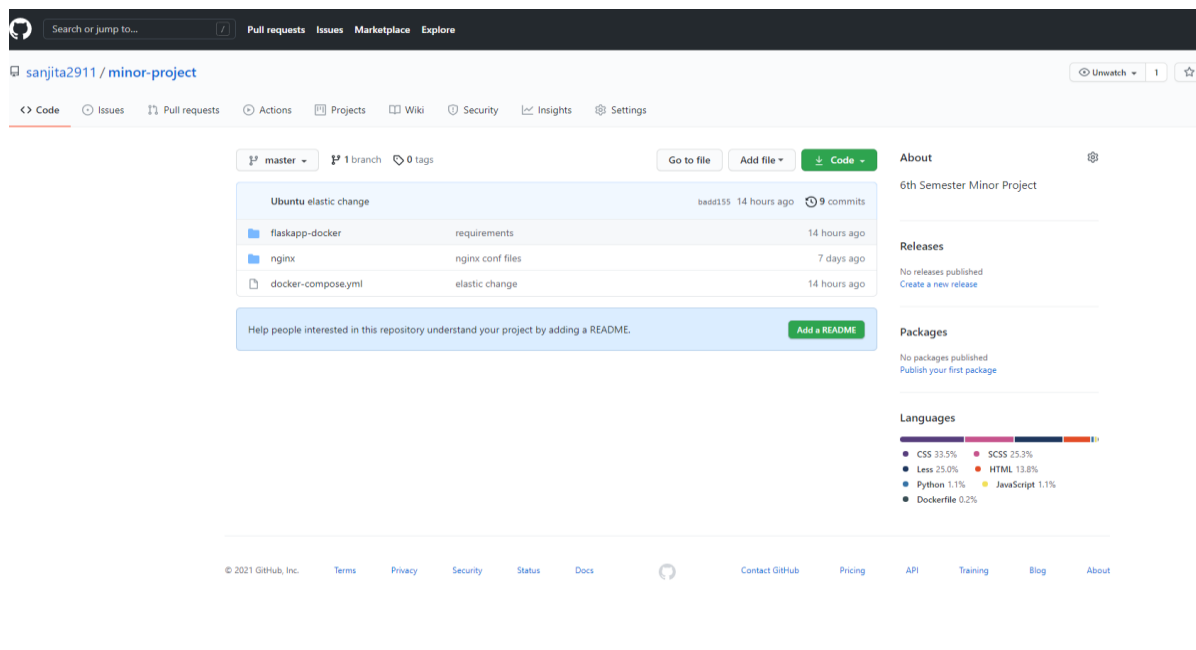


Fig 5.4 Central Repository after pushing the code

Cloning the code from central repository to the launched instance

```
ubuntu@ip-172-31-38-66: ~/minor-project
Fetched 19.4 MB in 6s (3446 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-38-66:~$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.25.1-1ubuntu3.1).
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 81 not upgraded.
ubuntu@ip-172-31-38-66:~$ git --version
git version 2.25.1
ubuntu@ip-172-31-38-66:~$ git init
Initialized empty Git repository in /home/ubuntu/.git/
ubuntu@ip-172-31-38-66:~$ git clone "https://github.com/sanjita2911/minor-project.git"
Cloning into 'minor-project'...
remote: Enumerating objects: 166, done.
remote: Counting objects: 100% (166/166), done.
remote: Compressing objects: 100% (143/143), done.
remote: Total 166 (delta 16), reused 166 (delta 16), pack-reused 0
Receiving objects: 100% (166/166), 3.62 MiB | 6.20 MiB/s, done.
Resolving deltas: 100% (16/16), done.
ubuntu@ip-172-31-38-66:~$ ls
minor-project
ubuntu@ip-172-31-38-66:~$ cd minor-project/
ubuntu@ip-172-31-38-66:~/minor-project$ ls
docker-compose.yml  flaskapp-docker  nginx
ubuntu@ip-172-31-38-66:~/minor-project$
```

Fig 5.5 Cloning the code into the instance

5.3 Working with Docker

Configuring docker-compose.yml file to build two docker images simultaneously

```
root@ip-172-31-38-66: /home/ubuntu/minor-project
version: "3"

services:
  minor-project:
    build: ./flaskapp-docker
    ports:
      - "5000:5000"
    network_mode: bridge
  nginx:
    build: ./nginx
    container_name: nginx
    environment:
      - SERVER_NAME=13.233.106.195
    restart: always
    network_mode: bridge
```

Fig 5.6 Docker-compose.yml file

Build and running the docker images

```
Select root@ip-172-31-38-66: /home/ubuntu/minor-project
Stored in directory: /root/.cache/pip/wheels/3a/c1/c3/5a19639a551c921c2c2b39468f4278ce5aa27b4e386a4158e4
Successfully built blinker docopt Flask-Email Flask-Mail Flask-MySQLdb mysqlclient
ERROR: requests 2.22.0 has requirement chardet<3.1.0,>=3.0.2, but you'll have chardet 4.0.0 which is incompatible.
ERROR: requests 2.22.0 has requirement idna<2.9,>=2.5, but you'll have idna 2.10 which is incompatible.
ERROR: requests 2.22.0 has requirement urllib3<1.25.0,>=1.25.1,<1.26,>=1.21.1, but you'll have urllib3 1.26.5 which is incompatible.
Installing collected packages: appdirs, blinker, certifi, chardet, click, colorama, distlib, docopt, filelock, Werkzeug, itsdangerous, MarkupSafe, Jinja2, Flask-MySQLdb, idna, urllib3, requests, yarg, pipreqs, requests-unixsocket, six, virtualenv
Successfully installed Flask-2.0.1 Flask-Email-1.4.4 Flask-Mail-0.9.1 Flask-MySQLdb-0.2.0 Jinja2-3.0.1 MarkupSafe-2.0.1 Werkzeug-2.0.1 appdirs-1.4.4 blinker-1.4.4 distlib-0.3.2 docopt-0.6.2 filelock-3.0.12 idna-2.10 itsdangerous-2.0.1 mysqlclient-2.0.3 pipreqs-0.4.10 requests-2.22.0 requests-unixsocket-0.2.0 six-1.16.0
Removing intermediate container 6c342d7d6db9
--> 4c967e4e359b
Step 11/13 : WORKDIR /opt/
--> Running in 50c6d9de7303
Removing intermediate container 50c6d9de7303
--> 6e532d466c7c
Step 12/13 : EXPOSE 8000
--> Running in 17ab02385e05
Removing intermediate container 17ab02385e05
--> 08dd51ed0c45
Step 13/13 : CMD ["gunicorn", "-b", "0.0.0.0:8000", "app:app", "--workers=5"]
--> Running in dde6c377f485
Removing intermediate container dde6c377f485
--> 1142534a9450
Successfully built 1142534a9450
Successfully tagged minor-project:latest
Building nginx
Step 1/3 : FROM nginx
latest: Pulling from library/nginx
b4d181a07f80: Pull complete
66b1c490df3f: Pull complete
d0f91ae9b44c: Pull complete
baf987068537: Pull complete
6bbc76cbebeb: Pull complete
32b766478bc2: Pull complete
Digest: sha256:8df46d7414eda82c2a8c9c50926545293811ae59f977825845dda7d558b4125b
Status: Downloaded newer image for nginx:latest
--> 4cdc5dd7eaad
Step 2/3 : RUN rm /etc/nginx/conf.d/default.conf
--> Running in f63489a444aa
Removing intermediate container f63489a444aa
--> c59c23427cc3
Step 3/3 : COPY flaskapp.conf /etc/nginx/conf.d
--> d415dedb3ae5
Successfully built d415dedb3ae5
Successfully tagged minor-project_nginx:latest
Creating nginx ... done
Creating minor-project_minor-project_1 ... done
Attaching to nginx, minor-project_minor-project_1
nginx | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf is not a file or does not exist
nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx | /docker-entrypoint.sh: Configuration complete; ready for start up
nginx | 2021/07/10 05:59:09 [notice] 1#1: using the "epoll" event method
nginx | 2021/07/10 05:59:09 [notice] 1#1: nginx/1.21.1
nginx | 2021/07/10 05:59:09 [notice] 1#1: built by gcc 8.3.0 (Debian 8.3.0-6)
```

Fig 5.7 Building and running backend image

```
Select root@ip-172-31-38-66: /home/ubuntu/minor-project
--> Running in dde6c377f485
Removing intermediate container dde6c377f485
--> 1142534a9450
Successfully built 1142534a9450
Successfully tagged minor-project:latest
Building nginx
Step 1/3 : FROM nginx
latest: Pulling from library/nginx
b4d181a07f80: Pull complete
66b1c490df3f: Pull complete
d0f91ae9b44c: Pull complete
baf987068537: Pull complete
6bbc76cbebeb: Pull complete
32b766478bc2: Pull complete
Digest: sha256:8df46d7414eda82c2a8c9c50926545293811ae59f977825845dda7d558b4125b
Status: Downloaded newer image for nginx:latest
--> 4cdc5dd7eaad
Step 2/3 : RUN rm /etc/nginx/conf.d/default.conf
--> Running in f63489a444aa
Removing intermediate container f63489a444aa
--> c59c23427cc3
Step 3/3 : COPY flaskapp.conf /etc/nginx/conf.d
--> d415dedb3ae5
Successfully built d415dedb3ae5
Successfully tagged minor-project_nginx:latest
Creating nginx ... done
Creating minor-project_minor-project_1 ... done
Attaching to nginx, minor-project_minor-project_1
nginx | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf is not a file or does not exist
nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx | /docker-entrypoint.sh: Configuration complete; ready for start up
nginx | 2021/07/10 05:59:09 [notice] 1#1: using the "epoll" event method
nginx | 2021/07/10 05:59:09 [notice] 1#1: nginx/1.21.1
nginx | 2021/07/10 05:59:09 [notice] 1#1: built by gcc 8.3.0 (Debian 8.3.0-6)
```

Fig 5.8 Building and running frontend image

Once we get the active response, we can check the functioning of the web application by entering the IP address of the instance along with the port number 8000.

5.4 Working with Jenkins

Creating Job: Automate, Build and Publish Docker image

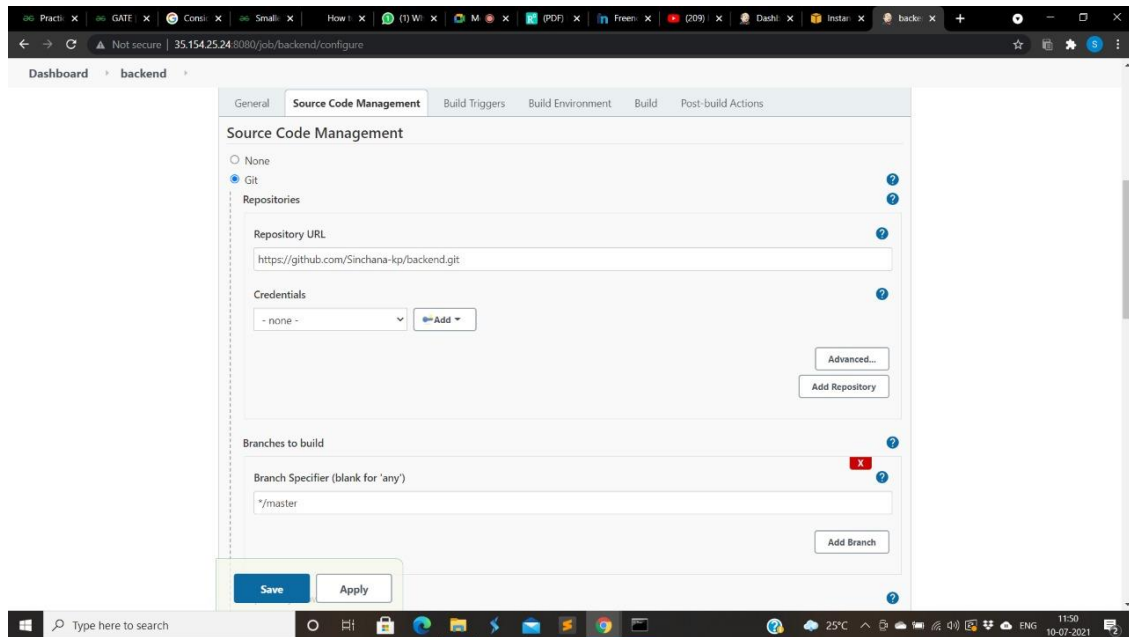


Fig 5.9 Pull the code from GitHub central repository

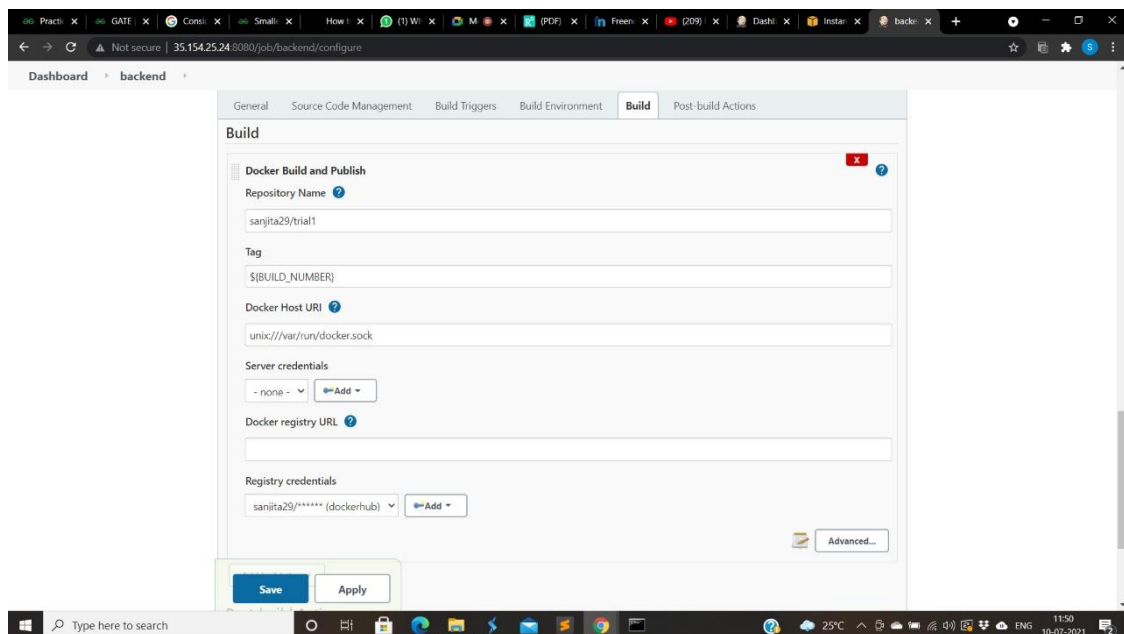


Fig 5.10 Build and publish the image to Docker Hub Repository

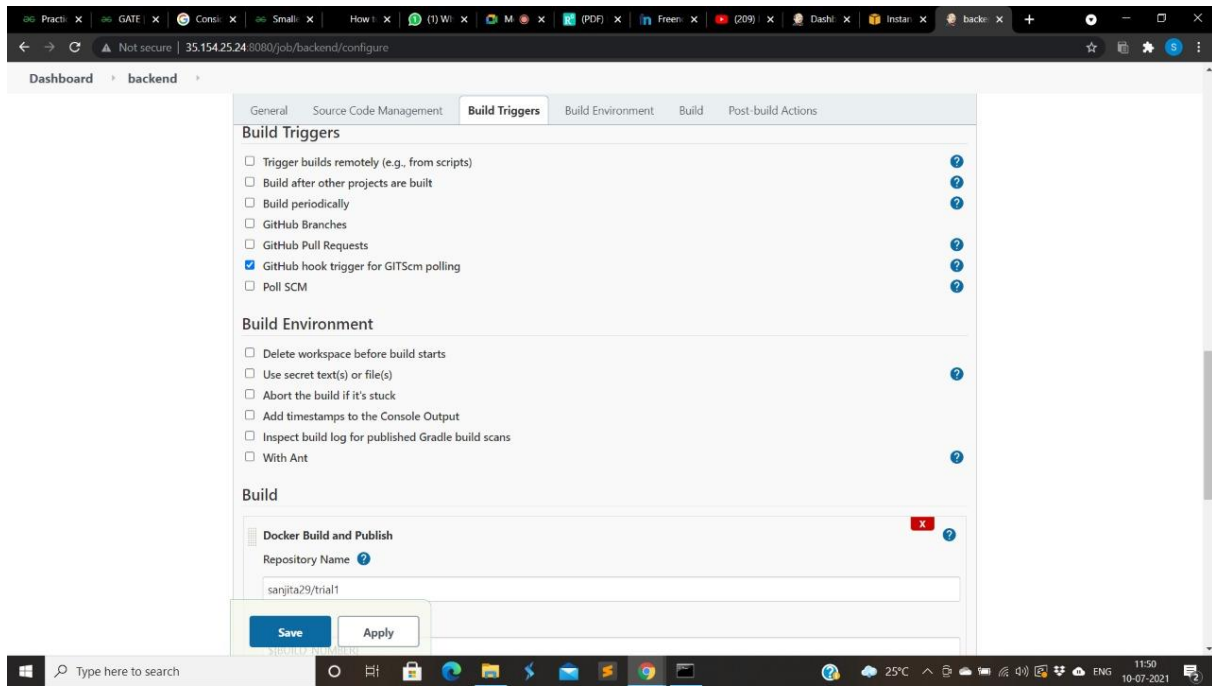


Fig 5.11 Automatically schedule a job whenever changes are committed in the GitHub repository

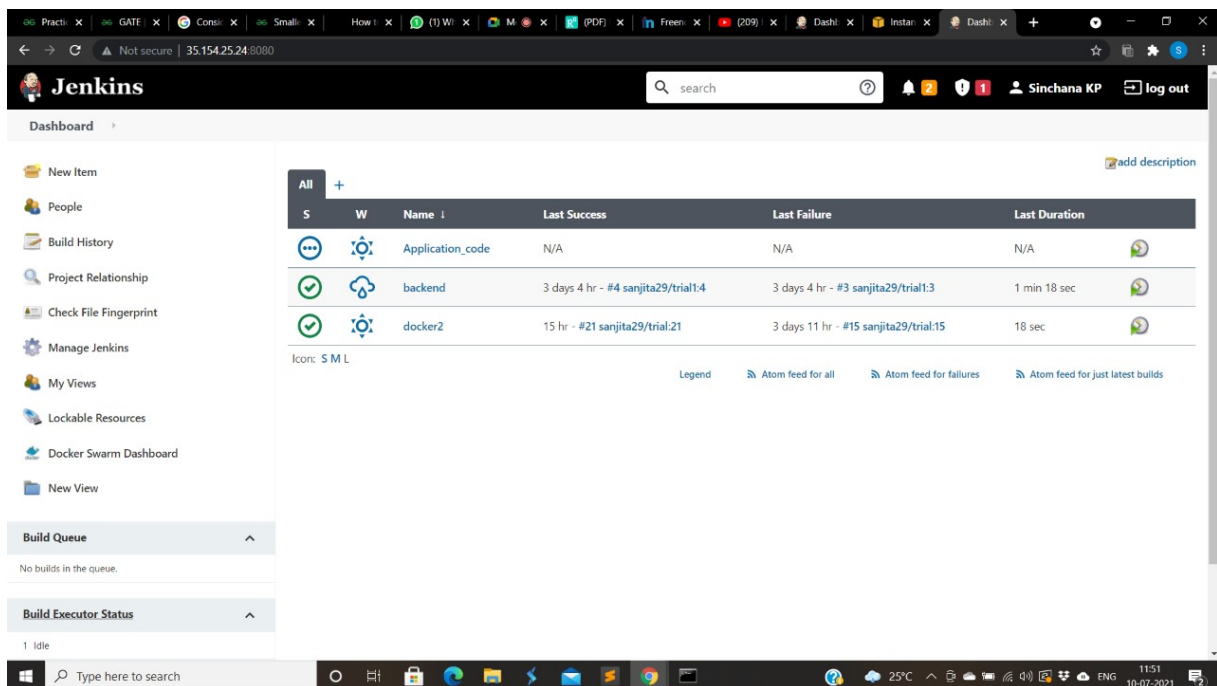


Fig 5.12 Scheduled Jobs

5.5 Hosted Application

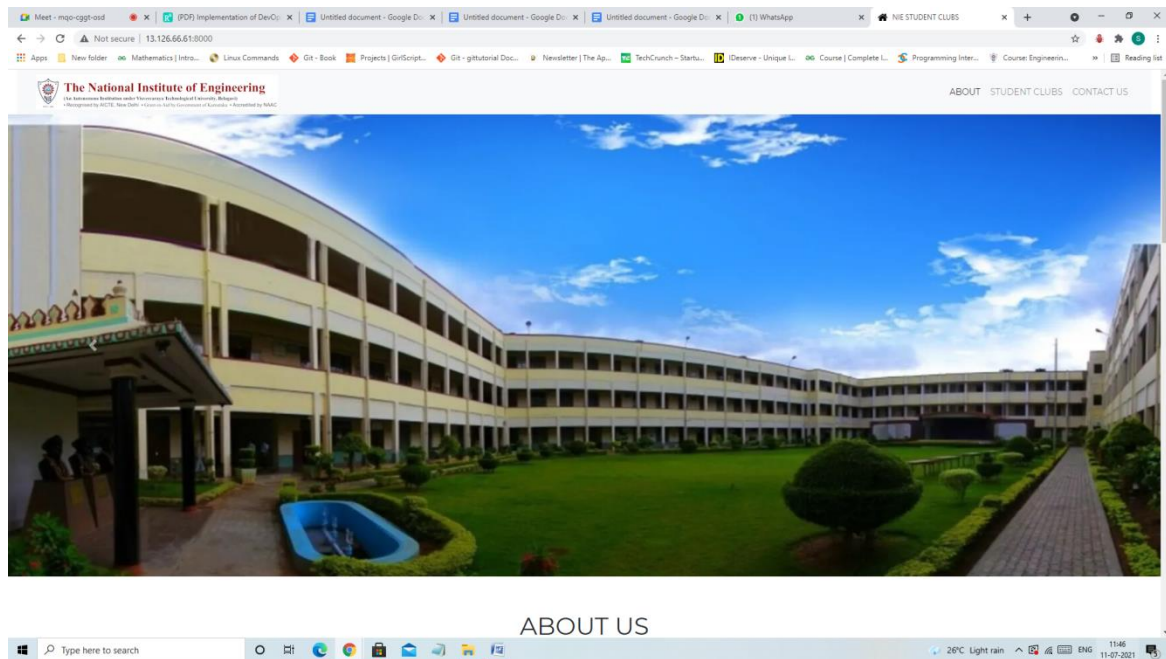


Fig 5.13 Home page 1

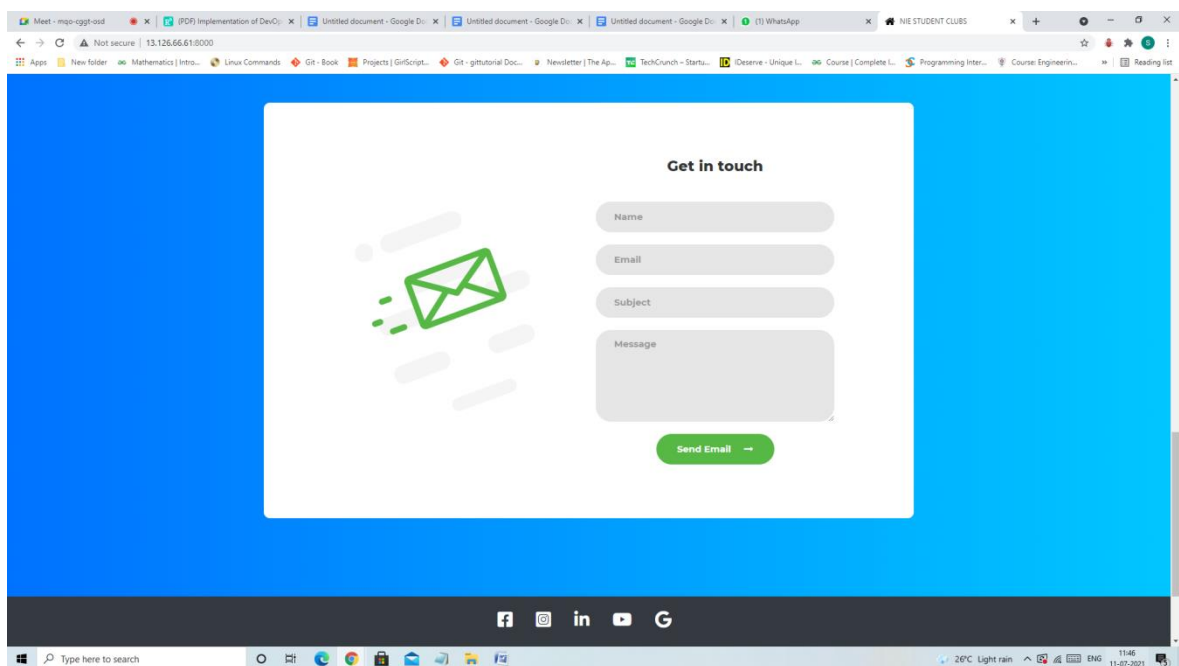


Fig 5.14 Home page 2

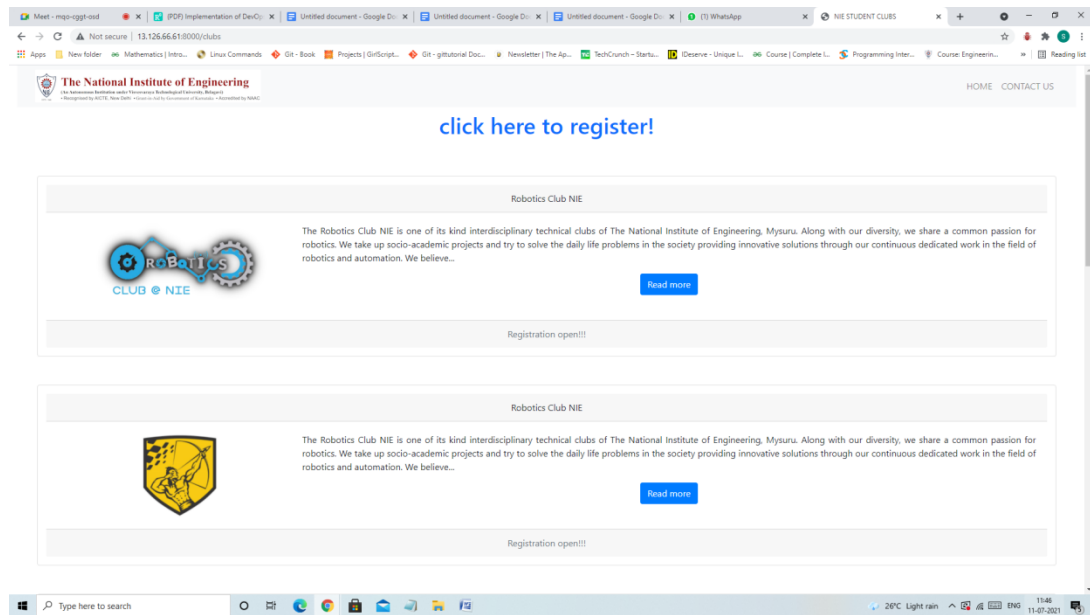


Fig 5.15 List of clubs at NIE

The screenshot displays a 'Registration Form' on a teal background. The form fields are as follows:

- First Name:** Sanjita
- Last Name:** Nayak
- USN:** 4NIBCS090
- Gender:** ☐ Male ☒ Female
- Email:** nayaksanjita29@gmail.com
- Phone Number:** 09035678478
- Branch:** Computer Science (dropdown menu)
- Semester:** VI (dropdown menu)
- Clubs:** ONYX (dropdown menu)

A blue 'Submit' button is located at the bottom of the form. The browser's taskbar at the bottom shows the system clock as 11:47 on 11-07-2021.

Fig 5.16 Registration Form

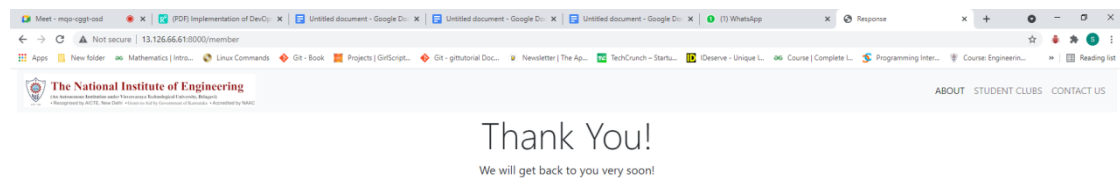


Fig 5.17 Response after submitting form

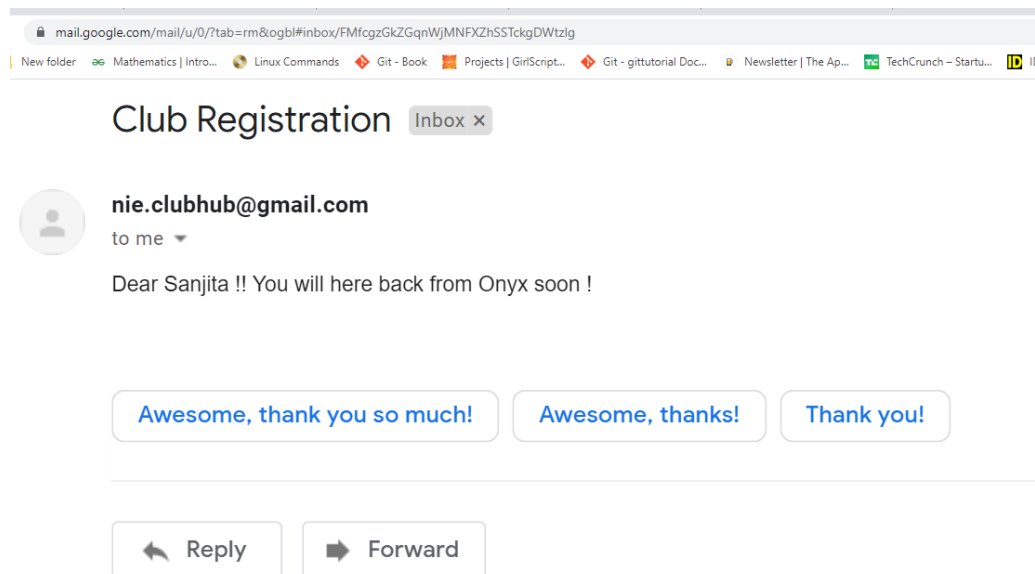


Fig 5.18 Confirmation mail received by the student

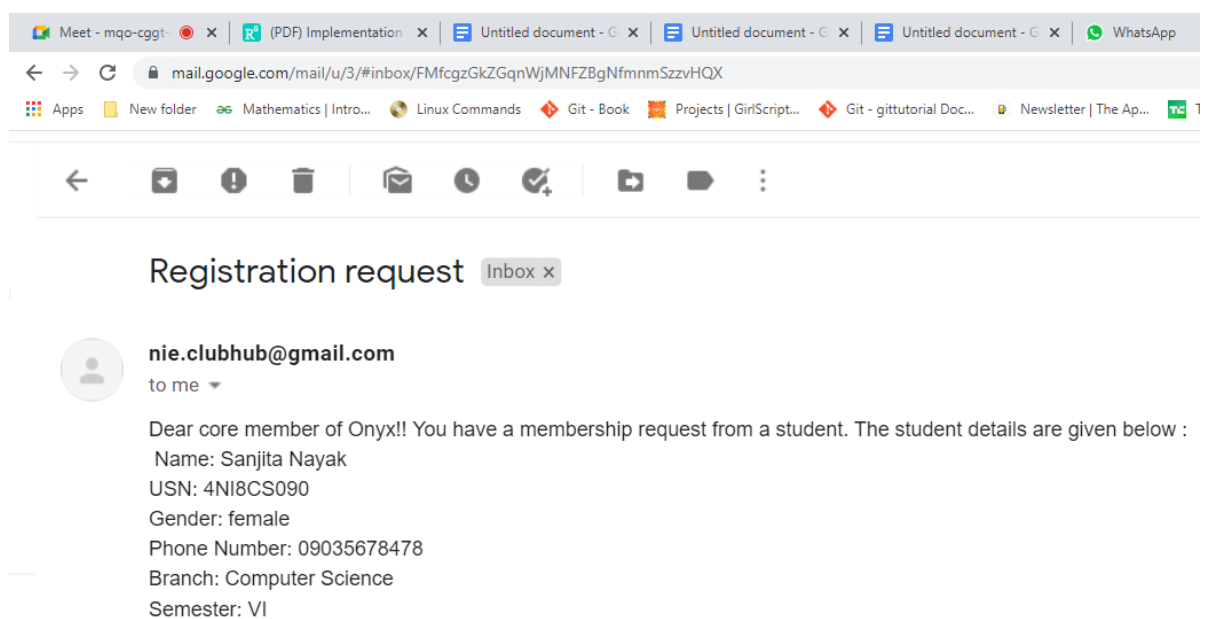


Fig 5.19 Membership request mail sent to the core of the club

CHAPTER 6

SYSTEM TESTING

The old way of testing was hand off centric. The software was handed off from one team to another. A project would have definite Development and QA phases. QA teams always wanted more time to ensure quality. The goal was that the quality should prevail over project schedule. However, business wants faster delivery of software to the end user. The newer is the software, the better it can be marketed and increase revenue potential of the company. Hence, a new way of testing was evolved.

Continuous Testing

Continuous means uninterrupted testing done on a continuous basis. In a Continuous DevOps process, a software change (release candidate) is continuously moving from Development to Testing to Deployment.

Continuous Testing in DevOps is a software testing type that involves testing the software at every stage of the software development life cycle. The goal of Continuous testing is evaluating the quality of software at every step of the Continuous Delivery Process by testing early and testing often. The Continuous Testing process in DevOps involves stakeholders like Developer, DevOps, QA and Operational system.

The code is continuously developed, delivered, tested and deployed. For Example, whenever a developer checks the code in the Source Code Server like Jenkins automated set of unit tests are executed in the continuous process. If the tests fail, the build is rejected, and the developer is notified. If the build passes the test, it is deployed to performance, QA servers for exhaustive functional and load tests. The tests are run in parallel.

If the tests pass, the software is deployed in production. Continuous Testing is a small cog in the Continuous Development, Integration and Deployment Cycle.

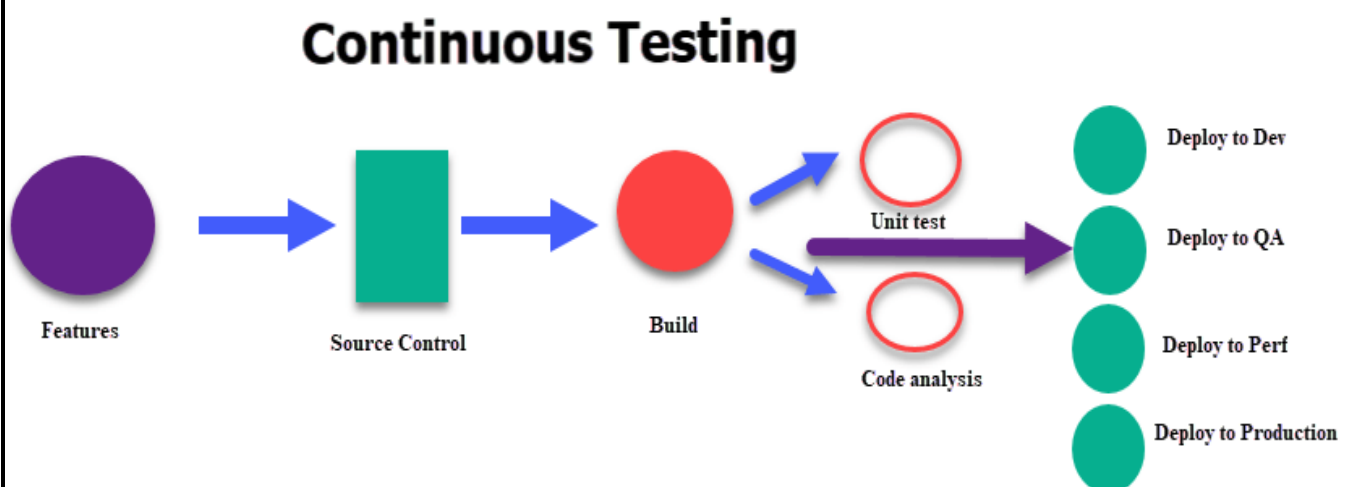


Fig 6.1 Continuous Testing

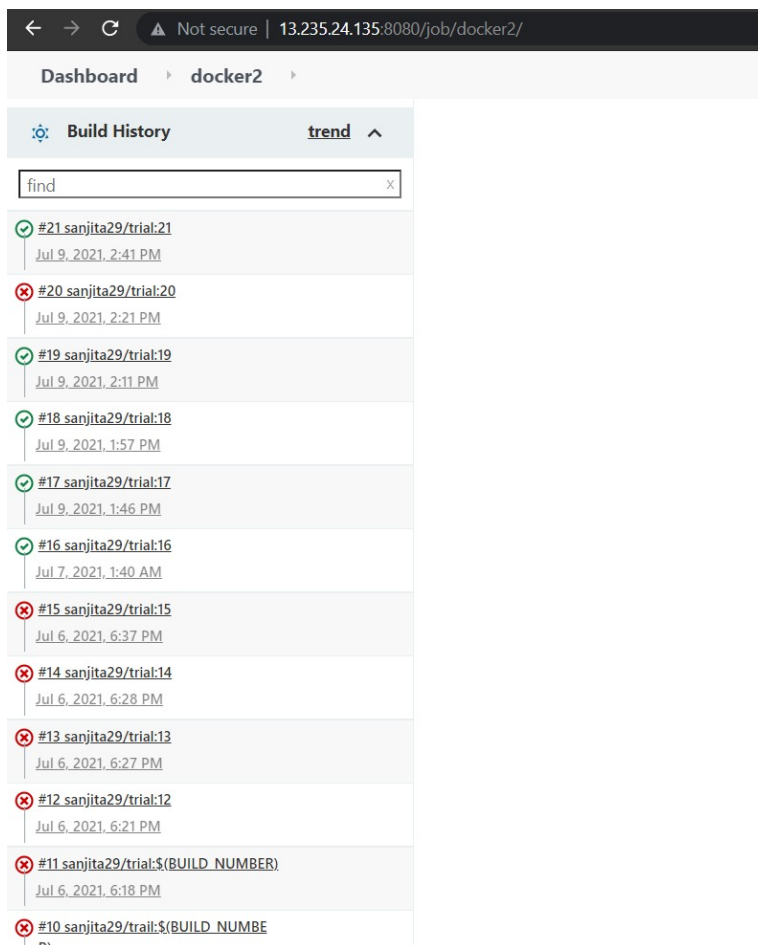


Fig 6.2 Continuous testing using Jenkins

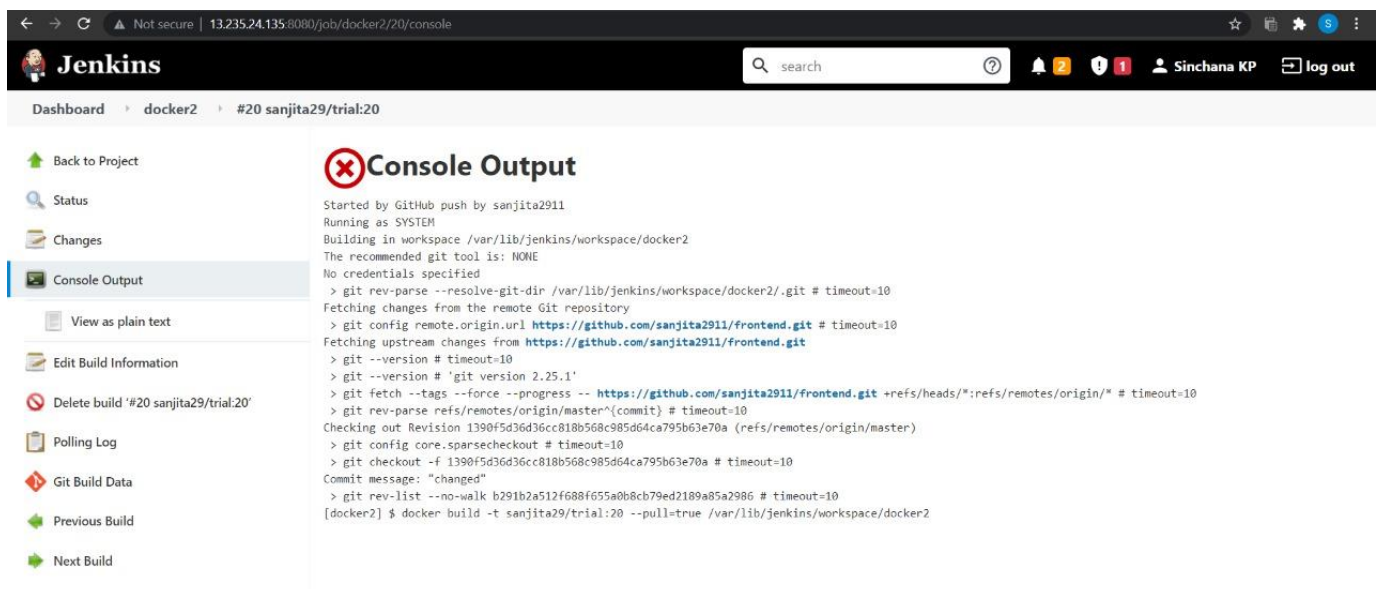


Fig 6.3 Console of a failed testcase

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

This proposed work is successfully designed, implemented and tested. DevOps is a software development methodology that escalates to the amalgamation between software developers and information technology operation professionals. It focuses mainly on delivering software products faster and reducing the failure rate of releases to make the product efficient. This system will be helpful for the developers or testers who need to fix the bugs rapidly and want to add extra features to the existing product according to the client requirement. At present, DevOps is the most advanced approach in IT industry than Waterfall and Agile model. This system is extended by on boarding the complete project into cloud service AWS, to improve the efficiency. At its core, DevOps is a culture, a movement, a philosophy. It is not a single person job to begin with.

This project achieves continuous integration and continuous testing. In future, it can be further extended to achieve continuous deployment. Currently, the web application can be accessed using the public IP address (13.126.66.61:8000). We also plan to map the IP address with suitable domain name using DNS service in future. This can also be extended by generating the review reports of the project through data visualization tools like Power BI or Tableau for better understanding of the project to the client.

REFERENCES

1. DEVOPS AND AWS:

<https://aws.amazon.com/devops/>

2. AWS, RDS AND MYSQL WORKBENCH WITH FLASK:

<https://www.youtube.com/watch?v=uTNjradClr8&t=228s>

3. SETTING EC2 INSTANCE FOR DOCKER-FLASK APPLICATION:

https://youtu.be/2tQ_Yn6O3f4

4. FLASK TUTORIALS

<https://flask.palletsprojects.com/en/2.0.x/>

5. DOCKER TUTORIALS

<https://www.youtube.com/watch?v=pTFZFxd4hOI>

6. BUILDING DOCKER IMAGE WITH JENKINS PIPELINE

<https://www.youtube.com/watch?v=mszE-OCI2V4&t=422s>