Software Production Engineering (CS816)

Major Project
Buttercrust

Under the guidance of Professor B. Thangaraju and Vaibhav Tandon

Group Members

- Harshit Agrawal (MT2021051)
- Samrat Seth (MT2021113)

1. Abstract

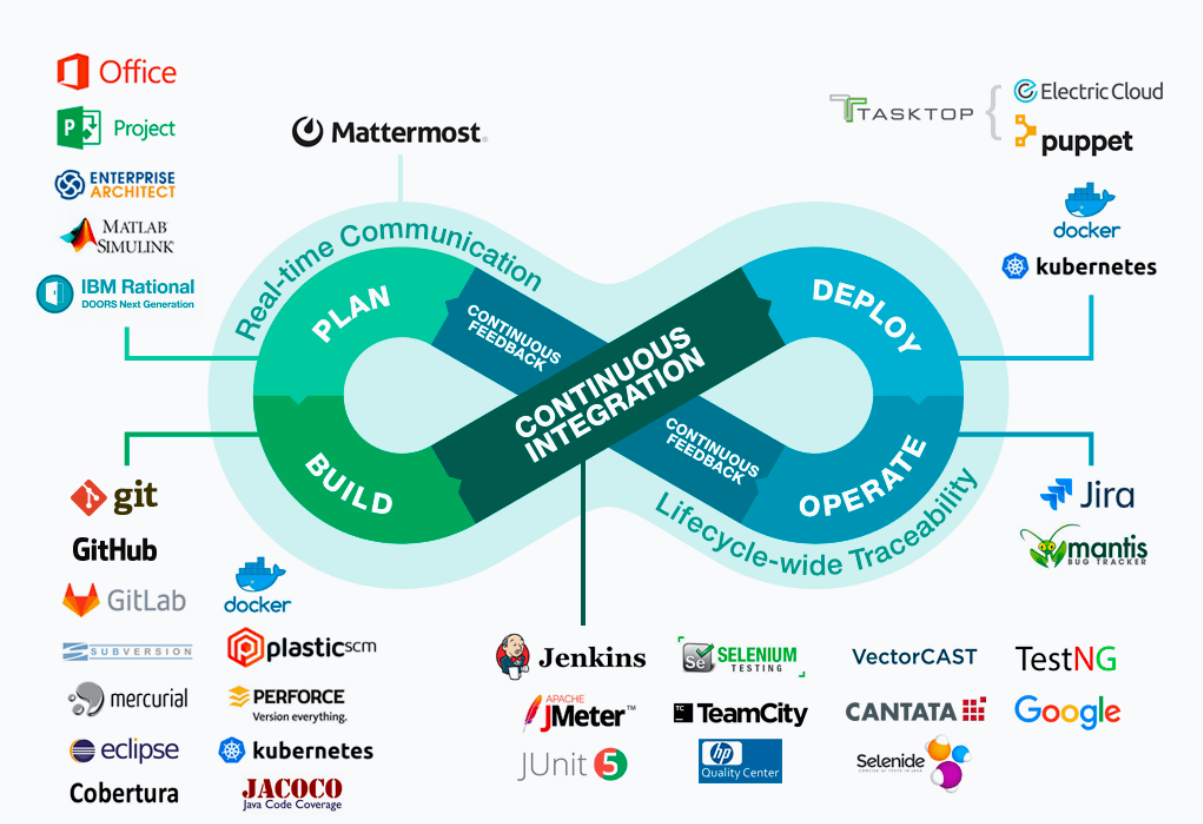
The online Pizza ordering system is a web-based application that enables customers to order their pizzas online for home delivery. Each country has its own kind of dishes to offer. But if we pick a food item that is loved by all the people on this planet, then pizza will be a clear winner in it. The whole world is in love with pizzas. The billions of dollars earned by different pizzerias across the globe just prove this. The love of pizzas has enabled the rise of large pizza companies like Pizza Hut, Domino’s, Papa John’s, and much more.

As the internet users are increasing exponentially, these companies have introduced an Online Pizza ordering system for taking orders from customers. This system not only improves the customer experience but also eases the workload on the staff of pizzerias.

This is a Full Stack (MERN) Pizza Delivery Application developed using React for Front End, Redux-Thunk for Asynchronous operations, Node JS for Runtime environment, Express JS for Backend Routing, and Mongo DB for Database.

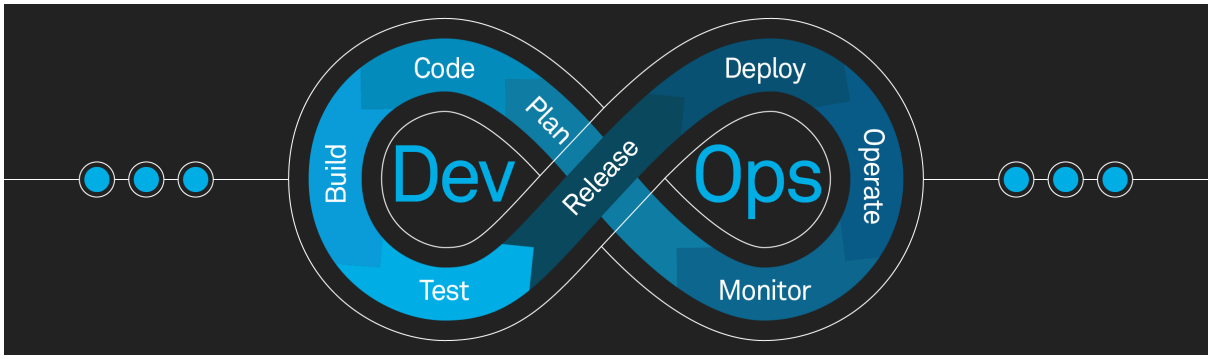
2. What is DevOps?

1. DevOps is the practice of operations and development engineers participating together in
   the entire service lifecycle, from design through the development process to production support.
2. DevOps is also characterized by operations staff making use of many of the same techniques/tools as developers for their systems work.
3. DevOps is the combination of cultural philosophies, practices, and tools that increases an
   organization’s ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations
   to better serve their customers and compete more effectively in the market.



3. Why DevOps?

We plan to build this project in a growing way. The design has ideas and services that work independently. Given the complexity of the project, it is impossible for any of us to create and
test the entire code manually every time we make a small change. And since the three of us work from different locations, the automatic pipeline will not only make our job easier, and make it more efficient. The amount of communication that must take place between us will decrease. DevOps helps us focus on key aspects of the project, improving efficiency, stability, and security. There is a small range of manual errors as well. And since we plan to build this
product at some point, continuous delivery makes it easier. Also, monitoring allows us to better
understand usage and help us improve the application.



4. System Configuration

4.1 Operation System

- Ubuntu 20.04.4 LTS (Focal Fossa)

4.2 CPU and RAM

- Ryzen 9 CPU and 16 GB Ram

4.3 Frameworks

- React JS
- Node JS

4.4 Database

- MongoDB

4.5 Building Tools

- npm (npm is a package manager for the JavaScript programming language)

4.6 AWS EC2 Instance

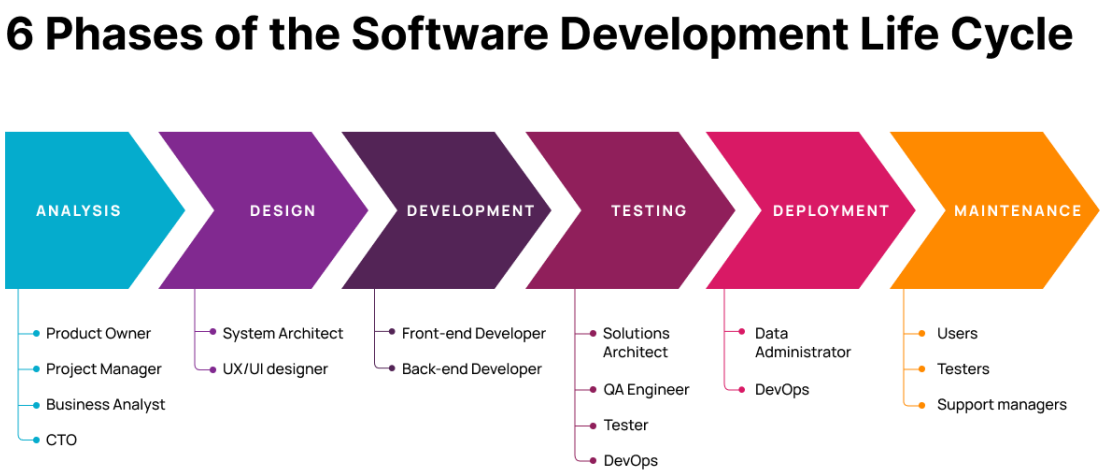
- Type - T2.medium
- OS - Ubuntu 20.04
- Ram - 2 GiB
- Storage - 15 GiB

4.6 DevOps Tools

- GitHub: Version control system
- Jenkins: CI/CD pipeline
- Ansible: Configuration management and infrastructure as code

- ELK: Monitoring
- Docker: Deployment/Containerization

5. Software Development Life Cycle (SDLC)



5.1. Source Code Management (SCM):
(Link to the repository: https://github.com/sethsamrat/Buttercrust-App)

- SCMs are used to give versions/revisions to the program. Each type is given a timestamp and
  includes the person responsible for the change. Even different versions can be compared and
  integrated with other types. That is why SCM is also called Version Control, Revision Control or
  Source Control.
- In order to achieve SCM, we need to create a GitHub repository on github.com by specifying
  the name and description of the project. This creates an empty repository on GitHub.
- We can also add a readme file in the repository that contains some information about the
  project. After creating an empty repository on GitHub we need to clone an empty project to the local system. This would create a directory in the name of the project in which we can add the files of our project. In this directory, add all the files of the project.
- Now add these files to the staging area.

```
$ git add *
```

Then commit those changes so that the files would get added to the local repo.

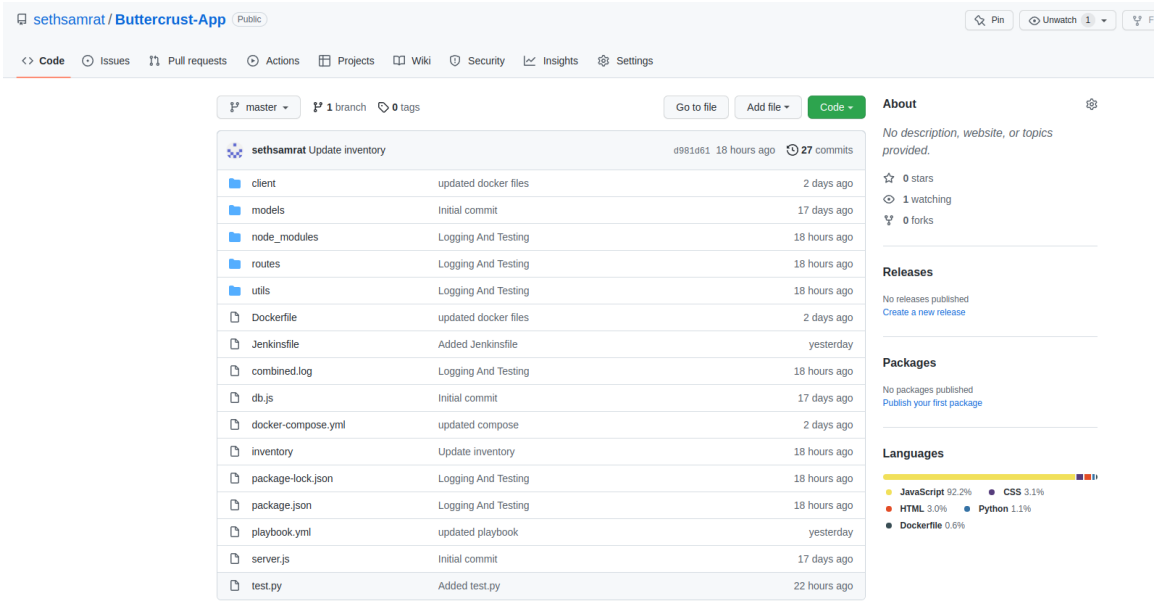
```
$ git commit -m “message”
```

Now in order to add these files to the GitHub repo, we need to push the files to the repo.

```
$ git push origin master
```

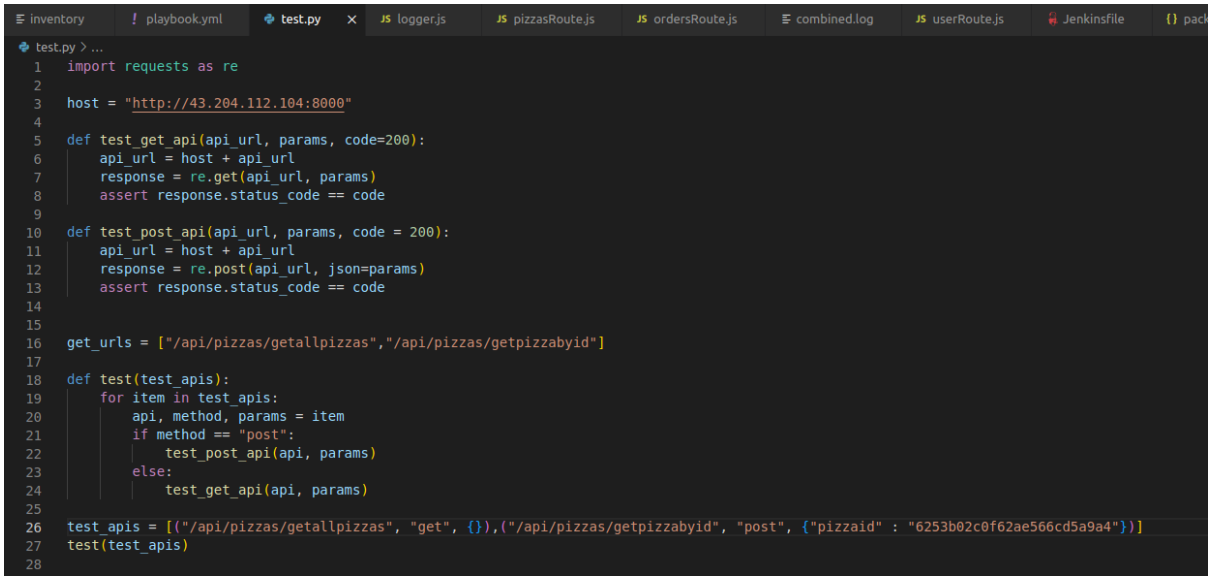
We have now successfully added our project to the GitHub which enables the other users to use
the same project and made required modifications to the project by git pull.
After pushing the files to GitHub the GitHub would look as below.

Repository



5.2 Testing

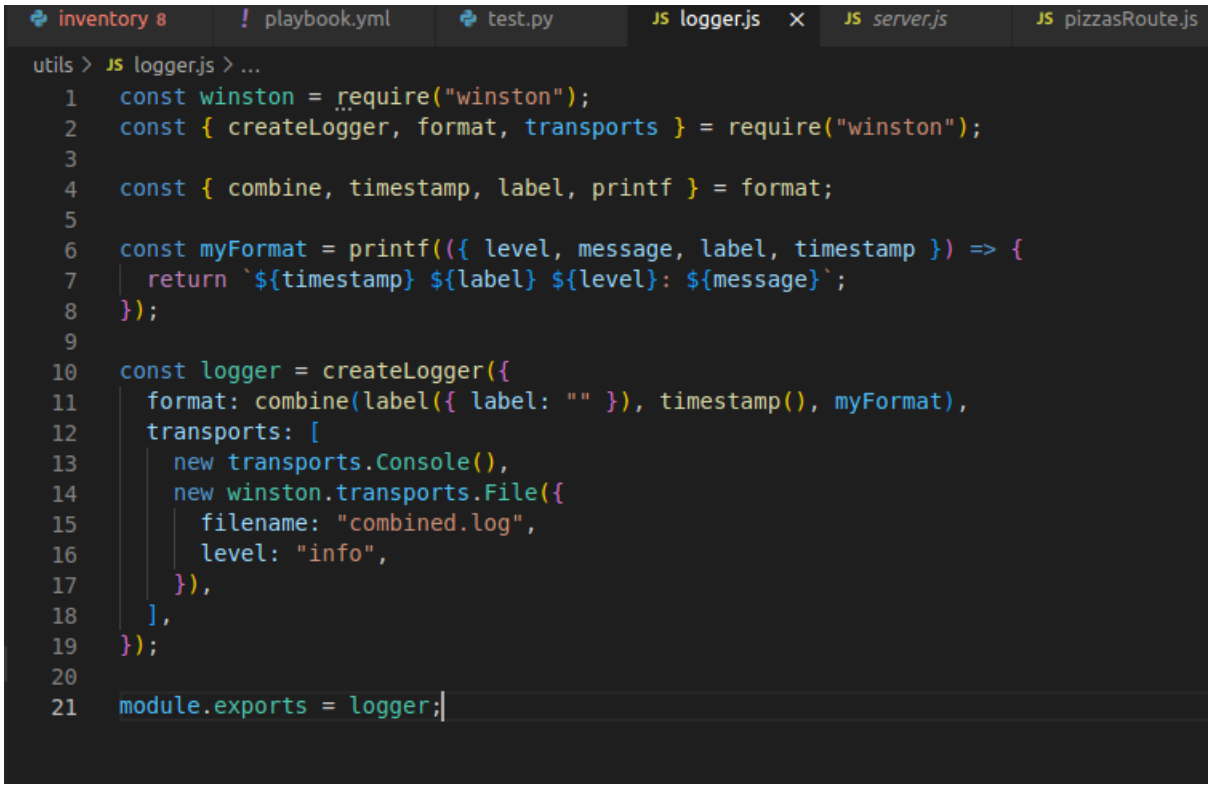
- We used python for testing our APIs
- Below is the modular code to test our project’s APIs
- We just need to pass the API in the function with the request method and it tests the API



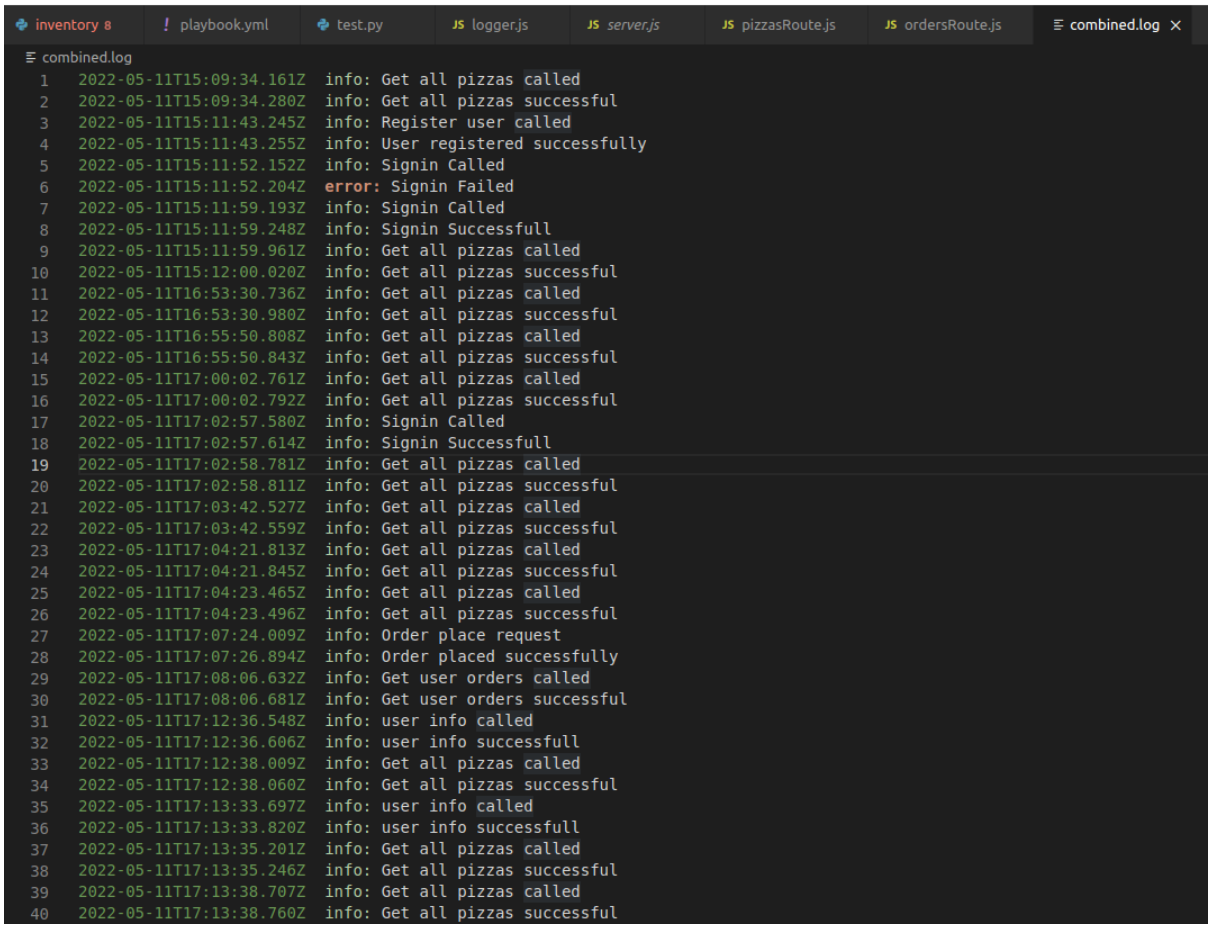
5.3 Logger

- **Winston Logger**

  `winston` is designed to be a simple and universal logging library with support for multiple transports. A transport is essentially a storage device for your logs. Each `winston` logger can have multiple transports configured at different levels. For example, one may want error logs to be stored in a persistent remote location (like a database), but all logs output to the console or a local file.

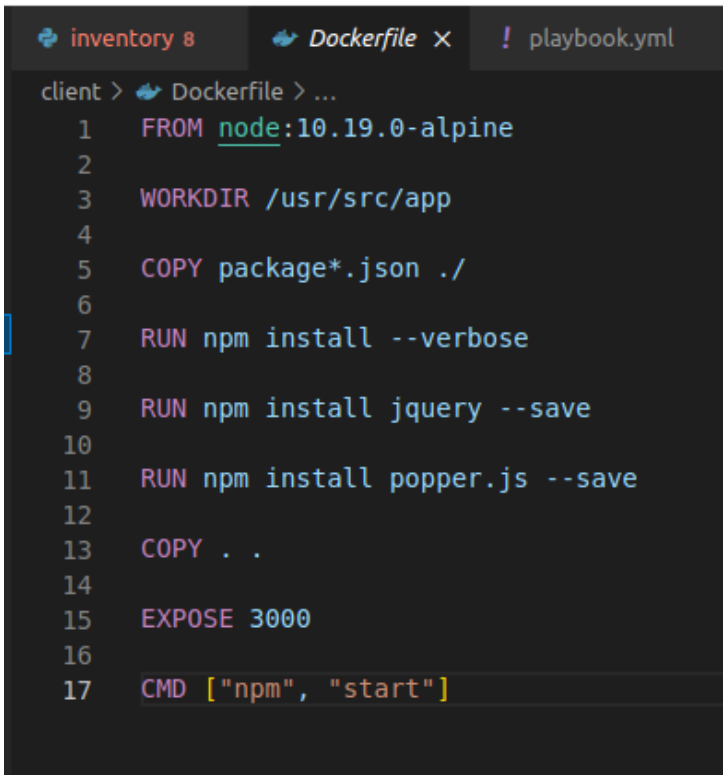


- **Logfile**

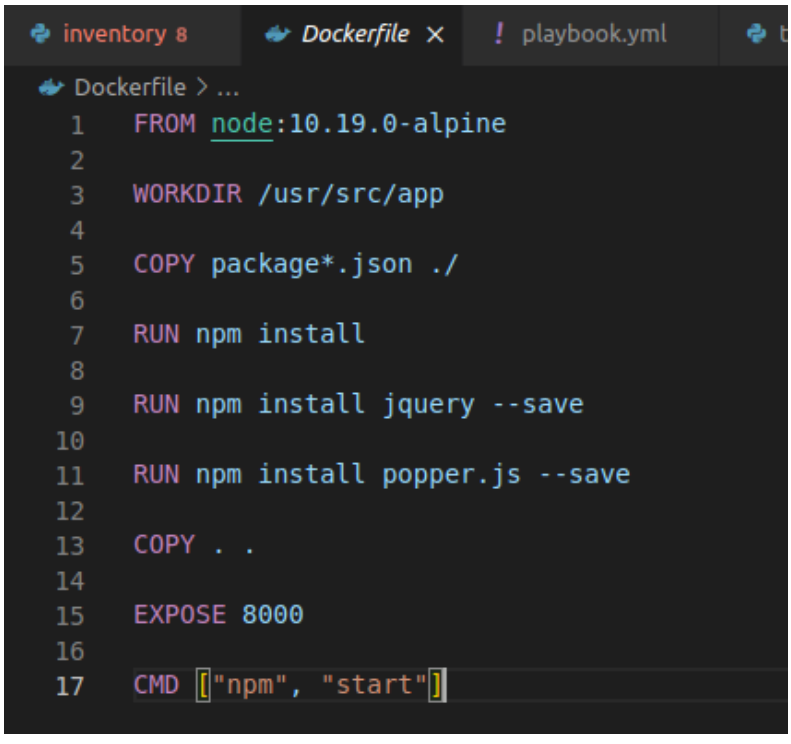


5.4 Docker

- Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere. As Bottomley told me, containers give you instant application portability.
- Containers do this by enabling developers to isolate code into a single container. This makes it
  easier to modify and update the program. It also lends itself, as Docker points out, for enterprises to break up big development projects among multiple smaller, Agile teams using Jenkins, an open-source CI/CD program, to automate the delivery of new software in containers.
- **Dockerfile for Client Image**



- **Dockerfile for Server Image**



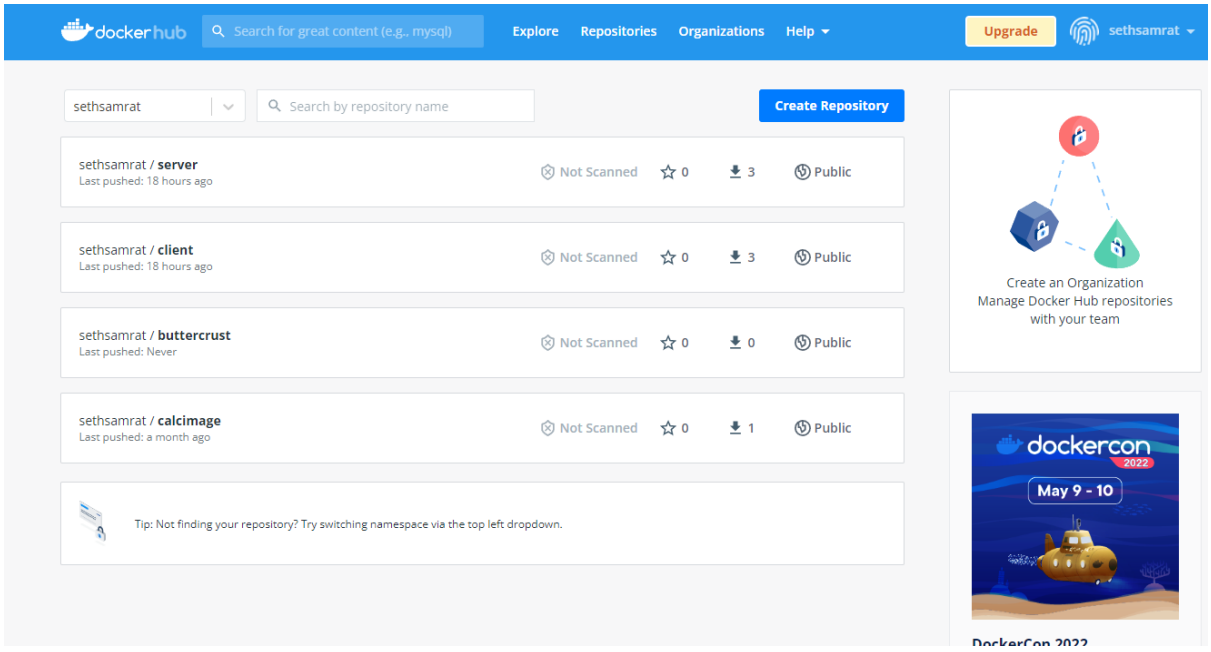
- **Docker-Compose**

inventory 8 | docker-compose.yml | playbo

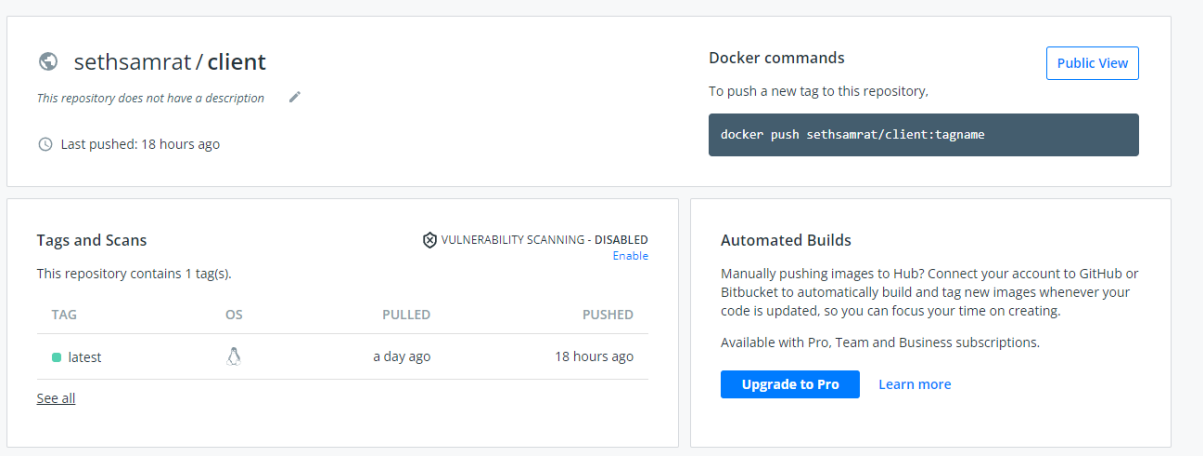
docker-compose.yml

```yaml
version: '3'
services:
  api-server:
    image: sethsamrat/server
    ports:
      - "8000:8000"
    networks:
      - mern-app
    container_name: server
    command: npm start
  react-app:
    image: sethsamrat/client
    stdin_open: true
    ports:
      - "3000:3000"
    networks:
      - mern-app
    command: npm start
networks:
  mern-app:
    driver: bridge
```

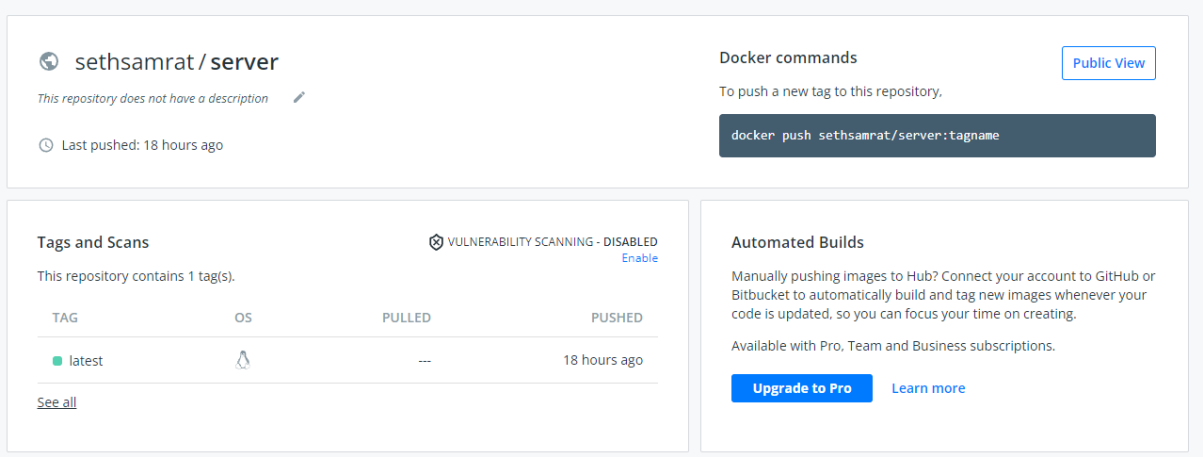
- **Repositories in DockerHub**



- **Client Repository**



- **Server Repository**

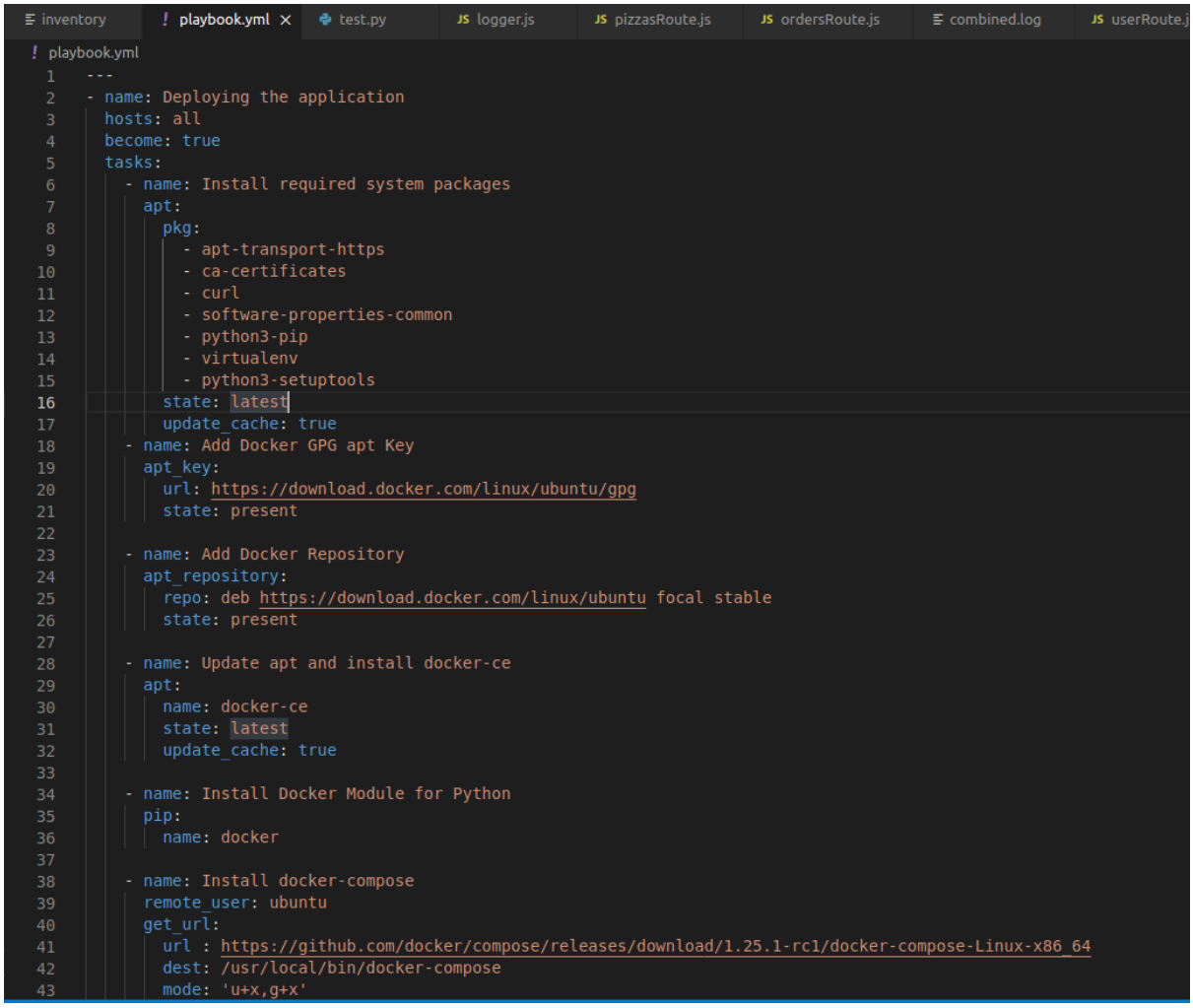


5.5 Deployment using Ansible (Deployed on AWS)

- Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs
- Designed for multi-tier deployments since day one, Ansible models your IT infrastructure
  by describing how all of your systems interrelate, rather than just managing one system at a time.
- It uses no agents and no additional custom security infrastructure, so it's easy to deploy -
  and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.
- Ansible works by connecting to your nodes and pushing out small programs, called "Ansible modules" to them. These programs are written to be resource models of the

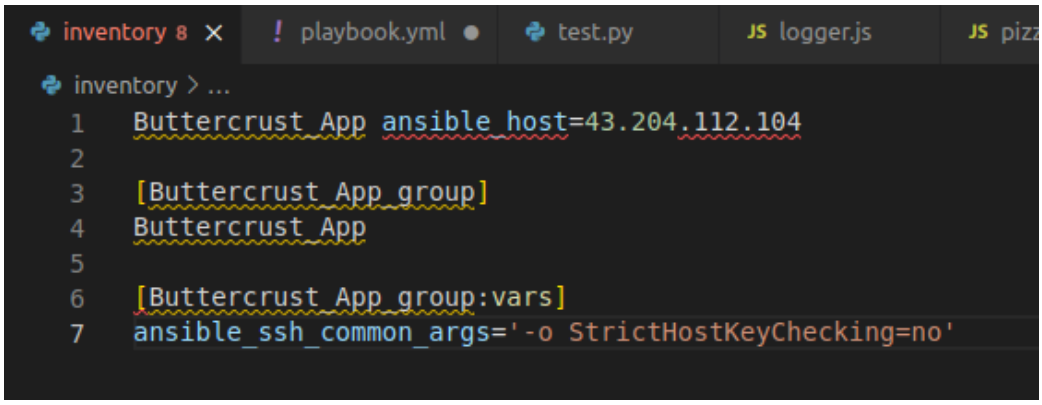
desired state of the system. Ansible then executes these modules (over SSH by default) and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required

- Typically you'll work with your favorite terminal program, a text editor, and probably a version control system to keep track of changes to your content.
- Passwords are supported, but SSH keys with ssh-agent are one of the best ways to use Ansible. Though if you want to use Kerberos, that's good too.
- p**laybook.yml**



```
- name: Starting the docker service
  service:
    name: docker
    state: started

- name: Copying the docker compose file
  copy:
    src: ./docker-compose.yml
    dest: ./

- name: Starting the application
  shell: docker-compose up -d
```

- **Inventory**

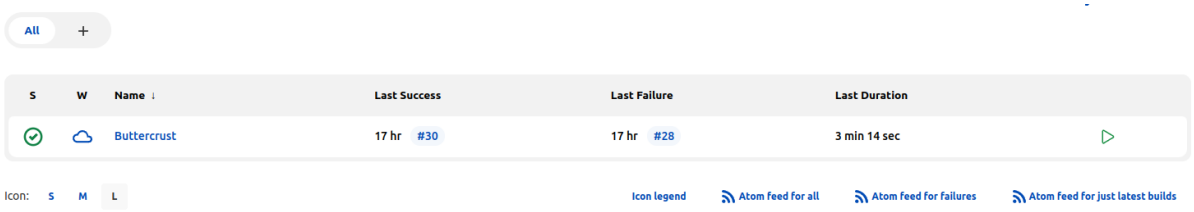


5.6 Continuous Integration using Jenkins

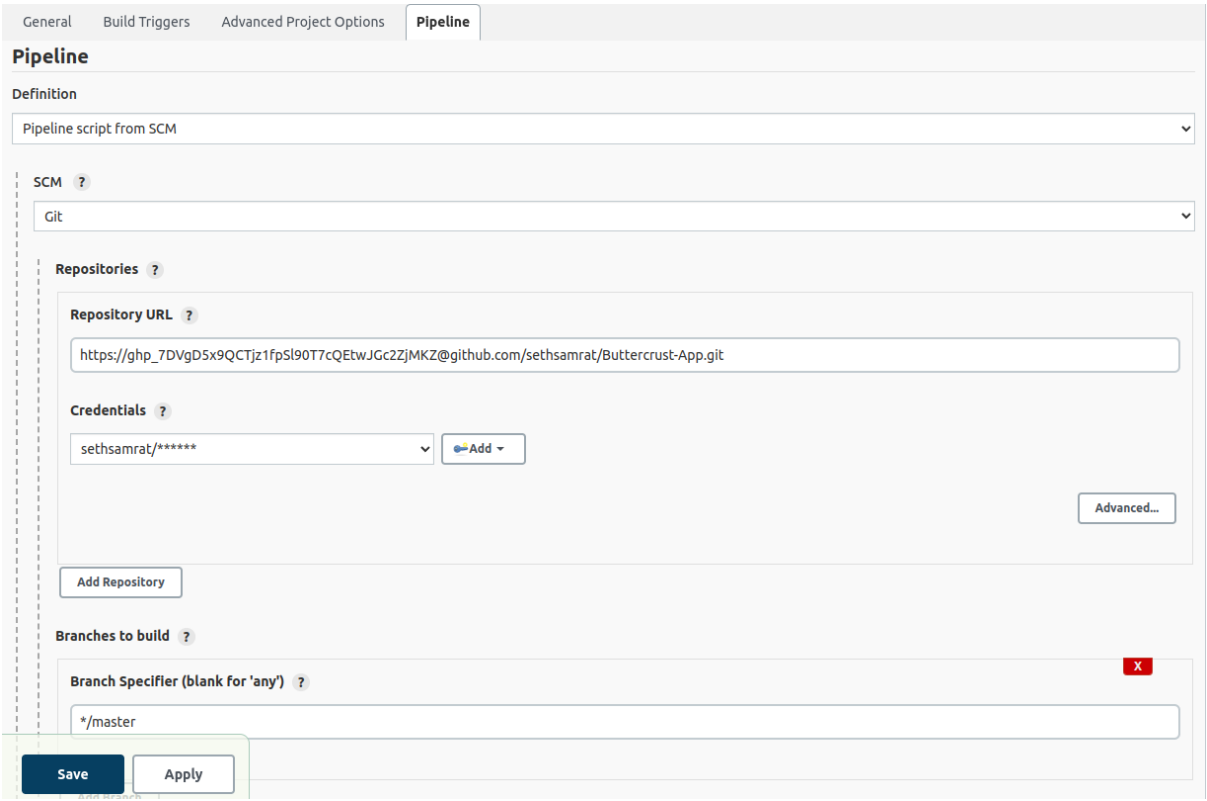
- Jenkins is an open-source automation tool written in Java with plugins built for Continuous
  Integration purposes. Jenkins is used to building and testing your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.
- The following steps were followed to install Jenkins in our localhost

```
$ wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
$ sudo apt-get update
$ sudo apt-get install jenkins
```

- **Summary of builds**



- **Pipeline Script from Git SCM**



- **Pipeline Code**
  - **Stage 1 : Declarative: Checkout SCM**
    It pulls code from the GitHub repo for the Jenkins pipeline.
  - **Stage 2: Git Clone**
    It pulls the remote repository from GitHub using Jenkins.
  - **Stage 3: Frontend prerequisite installations**
    This step is for building our react app.
  - **Stage 4: Backend prerequisite installations**
    This step is for building our server.
  - **Stage 5: Building the images**
    It is used to create images in our local system of the frontend and backend separately.

- **Stage 6: Pushing the images to DockerHub**
  The build images are pushed into the public DockerHub repository so that they can be pulled by anyone later on or during docker-compose by us.
- **Stage 7: Ansible Deploy**
  This is the deployment stage were using the already build images and the concept of containerization we can now execute our app on any platform using the Ansible inventory file and the playbook files.

master Buttercrust-App / Jenkinsfile Go to file ···

sethsamrat Added Jenkinsfile Latest commit 8e857ca yesterday History

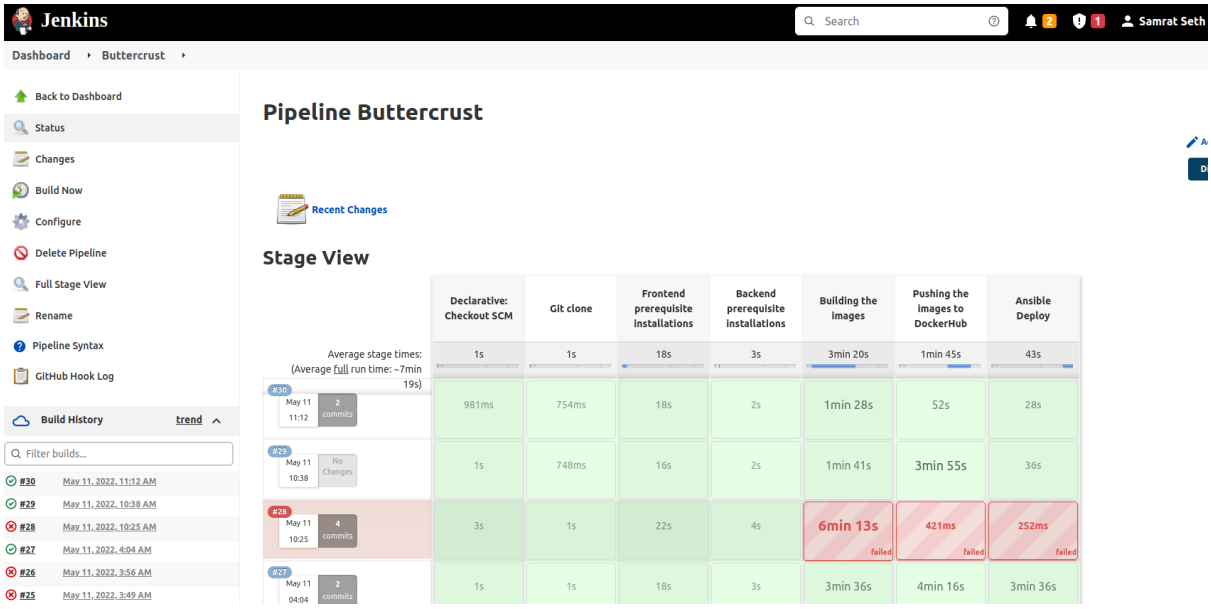
1 contributor

63 lines (55 sloc) | 1.74 KB Raw Blame

```
pipeline {
    agent any
            environment {
                registry1 = "sethsamrat/client"
                registry2 = "sethsamrat/server"
                registryCredential = 'docker_cred'
                dockerImage = ''
            }

    stages {
        stage("Git clone") {
            steps {
                git url: 'https://github.com/sethsamrat/Buttercrust-App.git', branch: 'master'
            }
        }
        stage("Frontend prerequisite installations") {
            steps{
                dir('client'){
                    sh 'npm install'
                }
            }
        }

        stage('Backend prerequisite installations'){
            steps {
                dir(''){

                    sh 'npm install'

                }
            }
        }
```

```
        stage('Backend prerequisite installations'){
            steps {
                dir(''){

                    sh 'npm install'

                }
            }
        }

        stage('Building the images'){
            steps {
                dir('client'){
                    sh 'docker build -t sethsamrat/client .'
                }
                dir(''){
                    sh 'docker build -t sethsamrat/server .'
                }
            }
        }

        stage('Pushing the images to DockerHub'){
            steps{
                script {
                    withDockerRegistry([ credentialsId: registryCredential, url: "" ])
                    {sh 'docker push $registry1'}

                    withDockerRegistry([ credentialsId: registryCredential, url: "" ])
                    {sh 'docker push $registry2'}
                }
            }
        }

        stage('Ansible Deploy') {
             steps {
                  ansiblePlaybook colorized: true,credentialsId: "container_access_key", disableHostKeyChecking: true, installation: 'Ansible', inventory: 'inve
             }
        }
    }
}
```

- **Stage View**



5.7 Amazon Web Services

- AWS (Amazon Web Services) is a comprehensive, evolving cloud computing the platform provided by Amazon that includes a mixture of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings. AWS services can offer an organization tools such as computing power, database storage and content delivery services

- AWS launched in 2006 from the internal infrastructure that Amazon.com built to handle its online retail operations. AWS was one of the first companies to introduce a pay-as-you-go cloud computing model that scales to provide users with computing, storage, or throughput as needed.

5.7.1 EC2

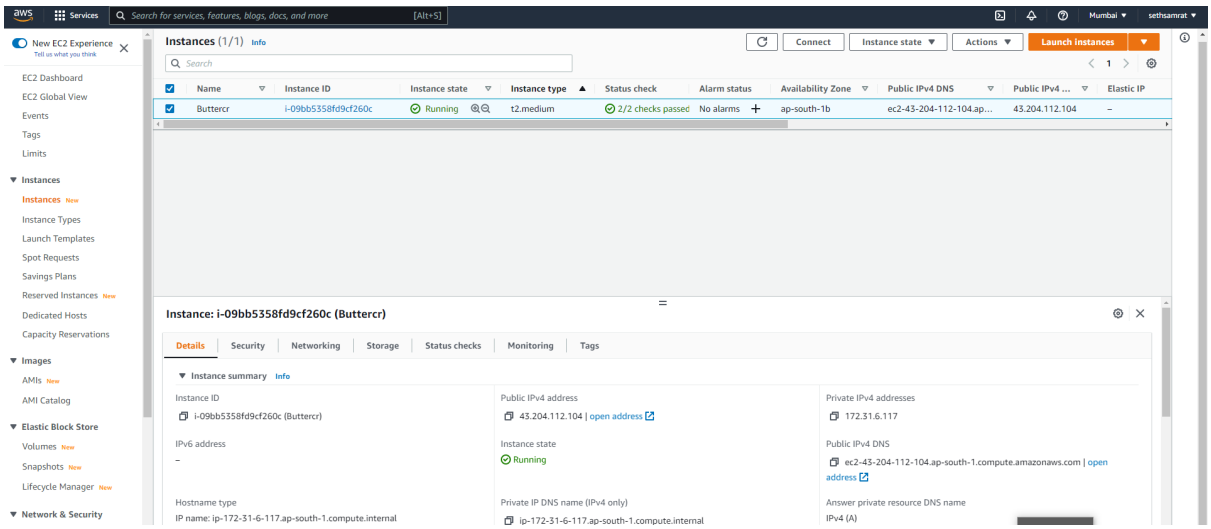
- Amazon EC2 (Elastic Compute Cloud) is a web service interface that provides resizable
  compute capacity in the AWS cloud. It is designed for developers to have complete control
  over web-scaling and computing resources.
- EC2 instances can be resized and the number of instances scaled up or down as per our
  requirement. These instances can be launched in one or more geographical locations or
  regions, and Availability Zones (AZs). Each region comprises several AZs at distinct locations, connected by low latency networks in the same region.

5.7.2 Features of EC2

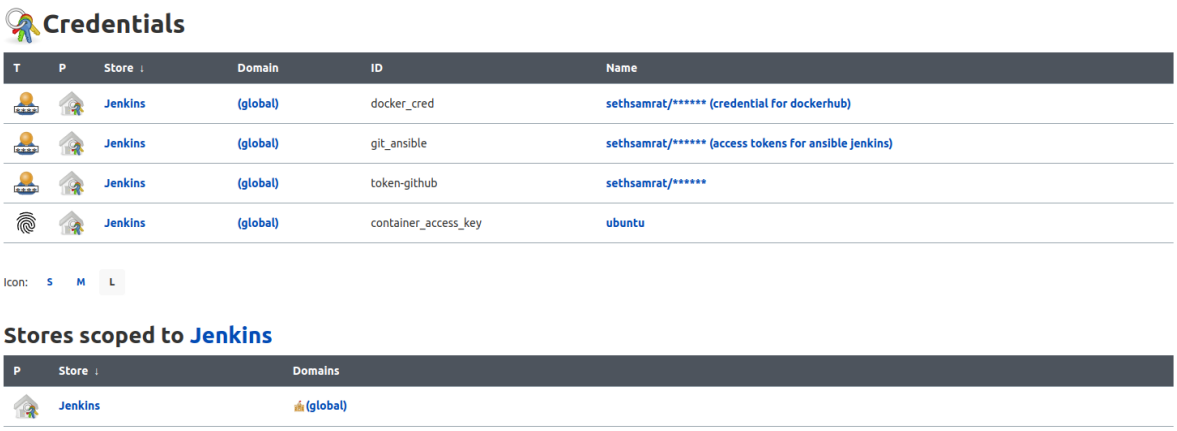
- **Reliable** − Amazon EC2 offers a highly reliable environment where the replacement of
  instances is rapidly possible. Service Level Agreement commitment is 99.9% availability for
  each Amazon EC2 region.
- **Designed for Amazon Web Services** − Amazon EC2 works fine with Amazon services like
  Amazon S3, Amazon RDS, Amazon DynamoDB, and Amazon SQS. It provides a complete
  solution for computing, query processing, and storage across a wide range of applications.
- **Secure** − Amazon EC2 works in Amazon Virtual Private Cloud to provide a secure and
  robust network of resources.
- **Flexible Tools** − Amazon EC2 provides the tools for developers and system administrators
  to build failure applications and isolate themselves from common failure situations.
- **Inexpensive** − Amazon EC2 wants us to pay only for the resources that we use. It includes
  multiple purchase plans such as On-Demand Instances, Reserved Instances, Spot

Instances,
etc. which we can choose as per our requirement.

- The instance of the AWS can be accessed from a local computer with the key provided by the AWS using the command:
  “sudo ssh -i "sethsamrat.pem" ubuntu@ec2-43-204-112-104.ap-south-1.compute.amazonaws.com”
- **AWS Instance**



- **EC2 instance credentials in Jenkins**



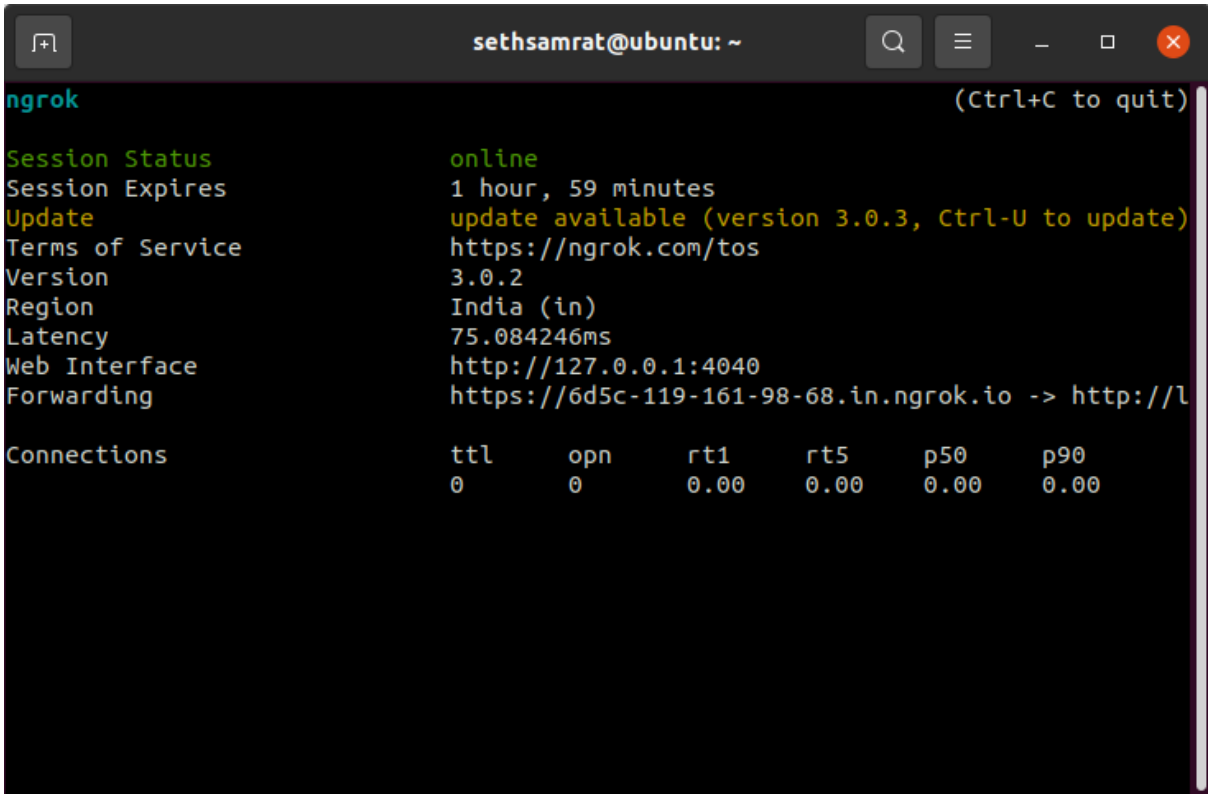
5.8 Logs and Monitoring

When the application is deployed and running properly on the managed node, we also want to check whether there are any problems in the run time or not. To do that we can implement a monitoring system using the ELK stack.

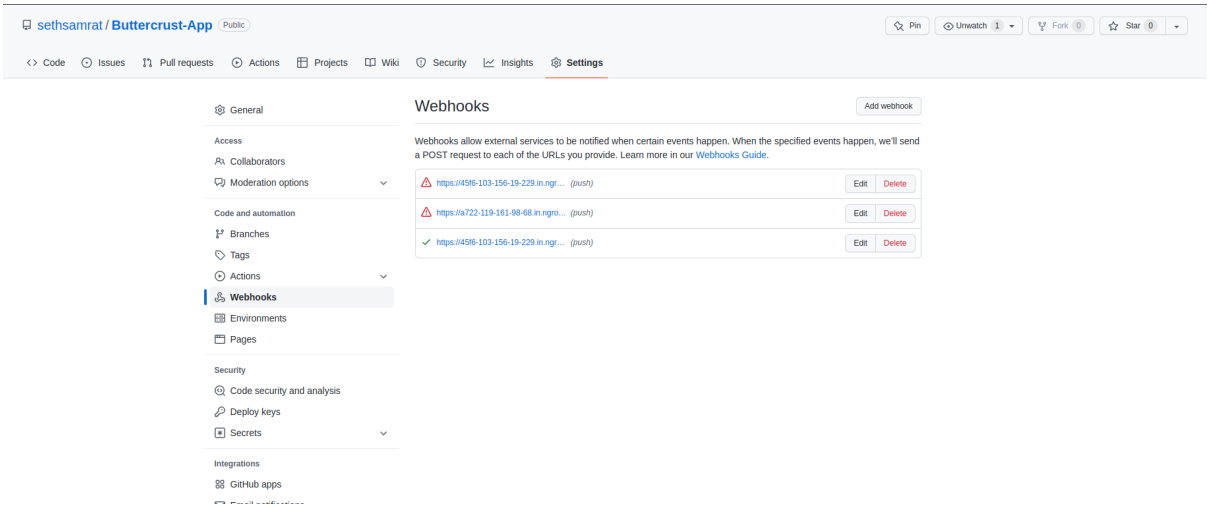
5.9 WebHooks For Triggering the Pipeline

Webhooks are automated messages that are sent whenever any changes are made. In our case when we make any changes to the GitHub repo, the webhook will automatically start the Jenkins pipeline.
Ngrok exposes local servers behind NATs (Network Address Translation) and firewalls to the public internet over secure tunnels. It provides a real-time web UI where you can introspect all HTTP traffic running over your tunnels. It allows you to expose a web server running on your local machine to the internet. Just tell ngrok what port your web server is listening on.

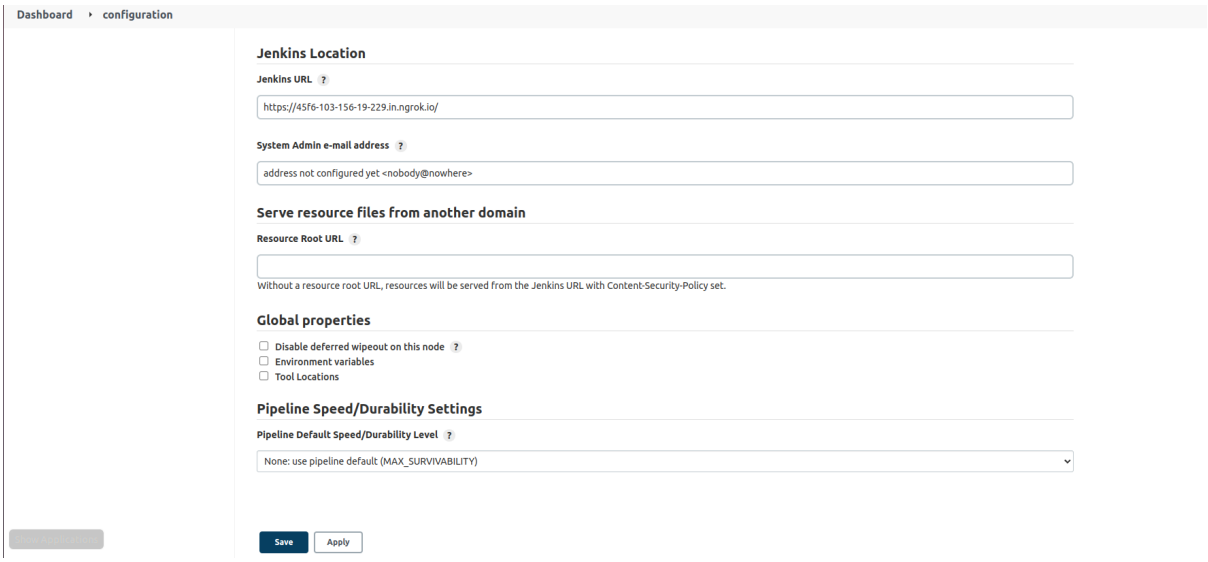
- **NGROK**



- **Setting up webhooks for Buttercrust repository**



- **Adding webhooks to Jenkins Location**

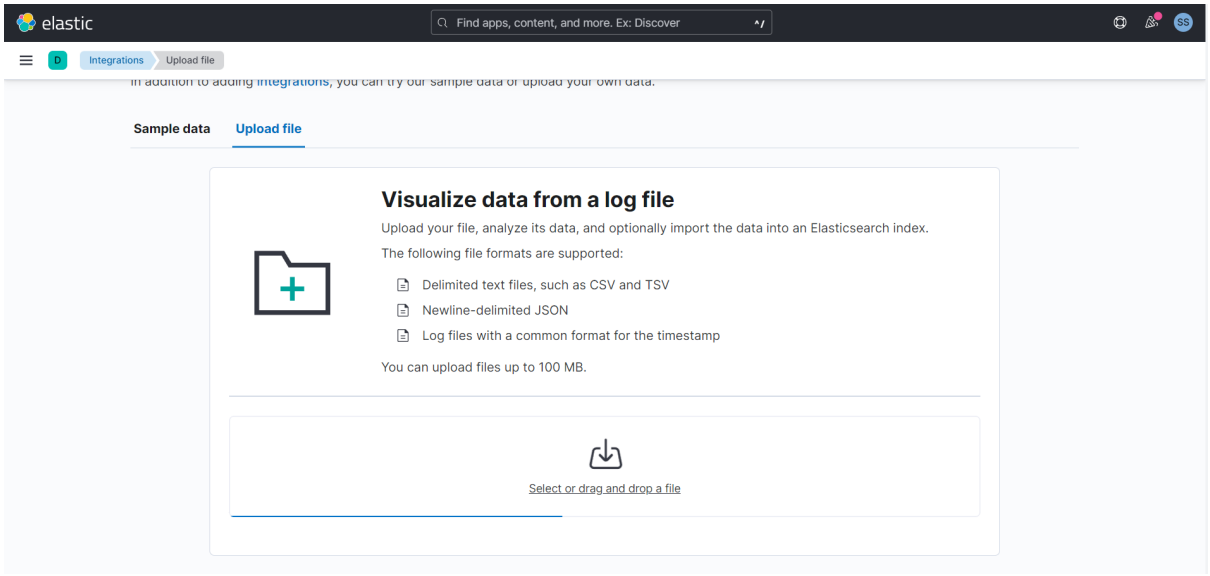


5.10 ELK for continous monitoring

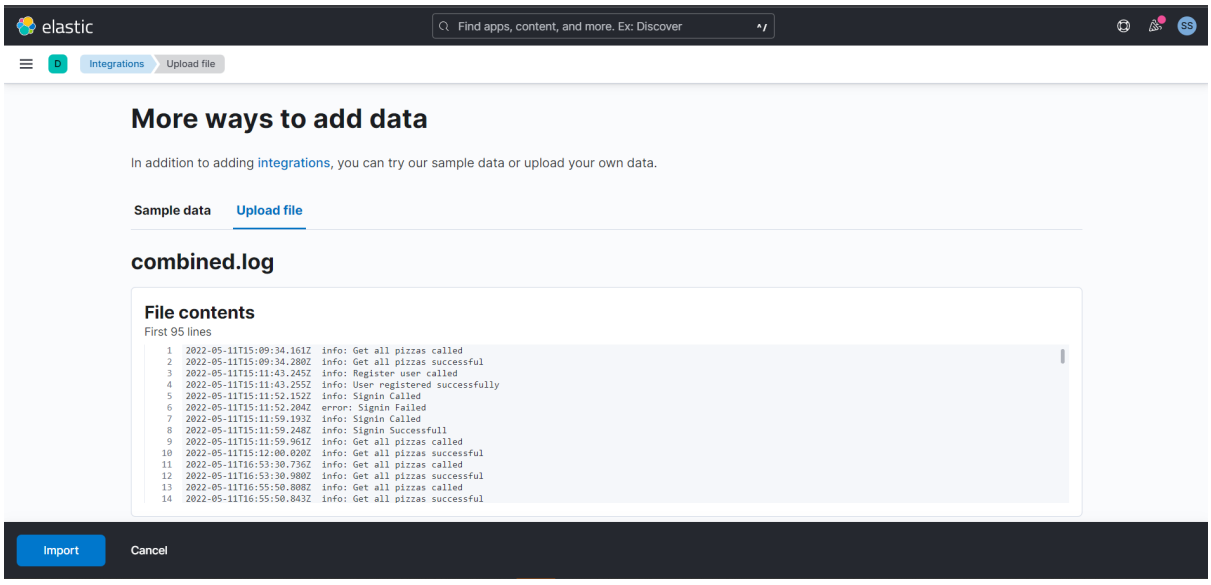
It is a technology and process that IT organizations use to get near-immediate feedback and insight into performance and interactions across the network, which helps drive operational, security, and business performance. After doing deployment the next major thing which needs to be done is monitoring. Monitoring involves checking whether the software is behaving as it is meant to be or not.
ELK stack makes the monitoring tool for any deployed software, it analyzes the logs and the same analysis can then be viewed on the kibana dashboard. ELK stack comprises 3 independent components: Elasticsearch, Logstash, and Kibana.

- Elasticsearch is an open-source, full-text search and analysis engine.
- Logstash is a log aggregator that collects data from various input sources and then ships the data to various supported output destinations.

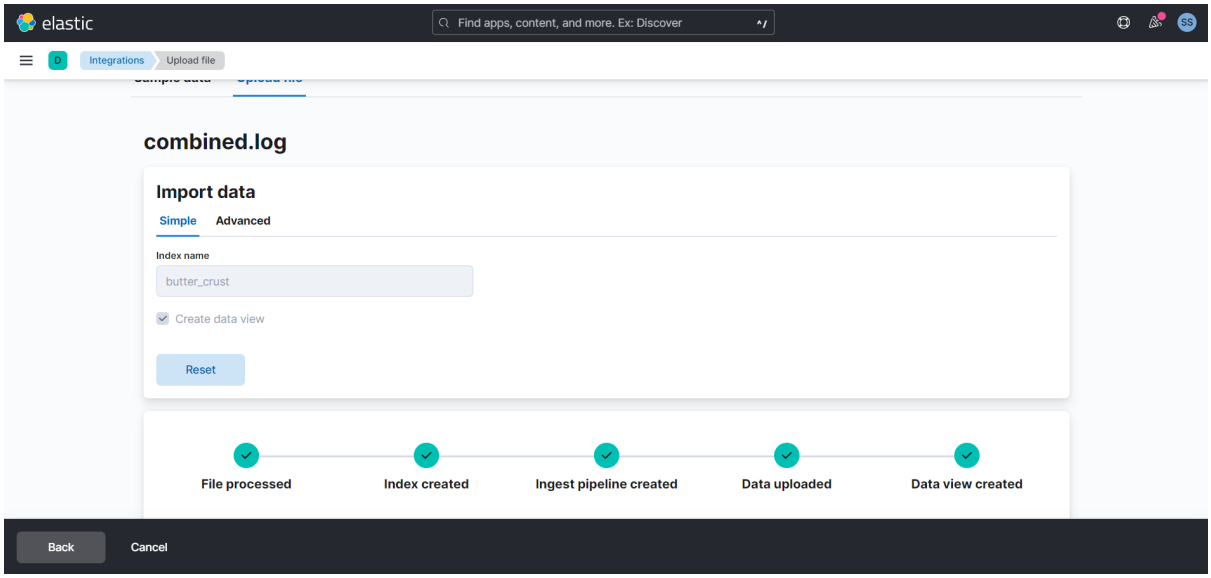
- Kibana is a visualization layer that works on top of Elasticsearch, providing users with the ability to analyze and visualize the data.
- **Uploading the log file**



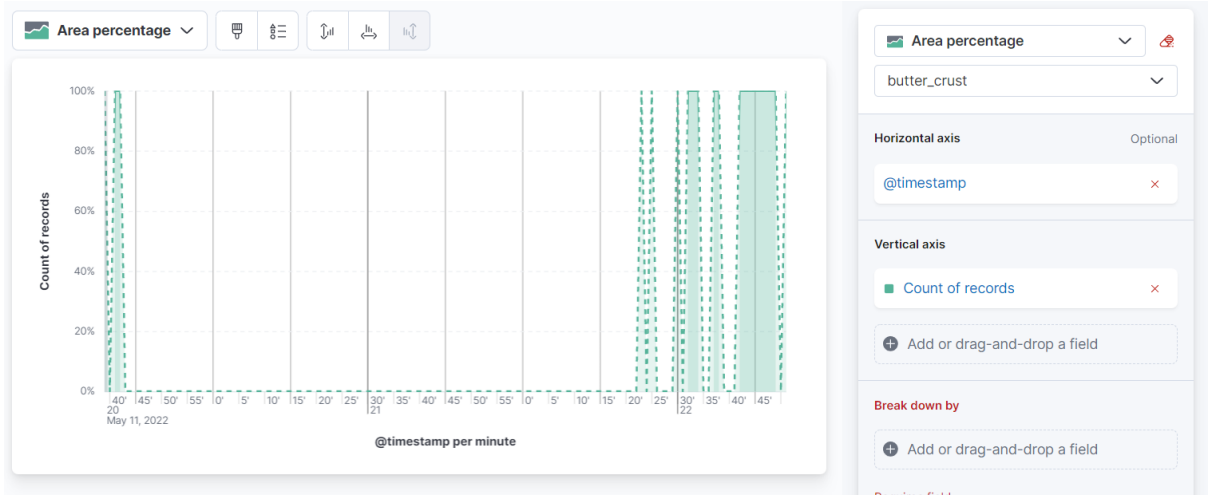
- **Importing the log file**



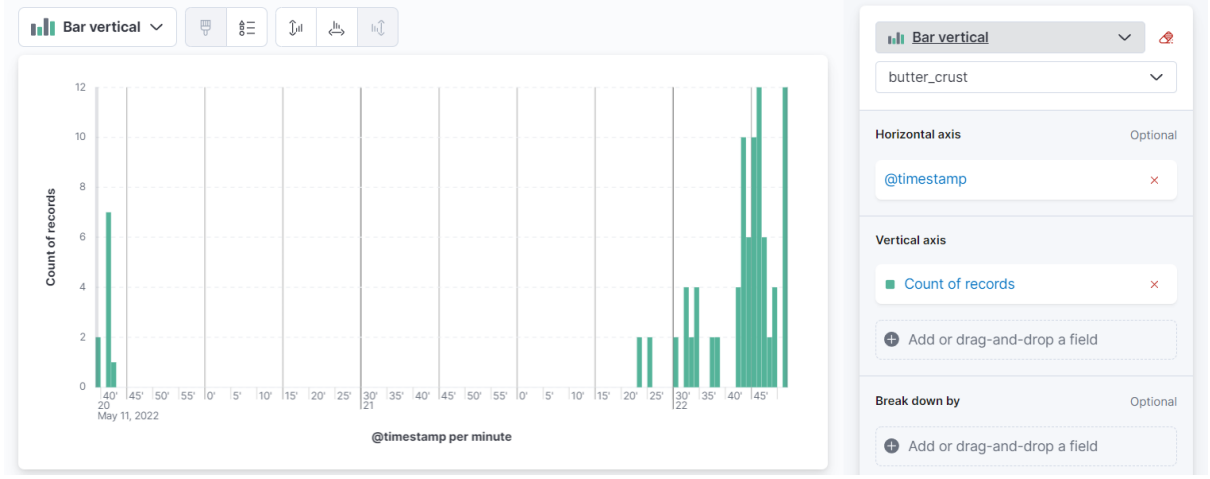
- **Giving an index name and adding the file**



- **Area Percentage**



- **Bar Vertical**



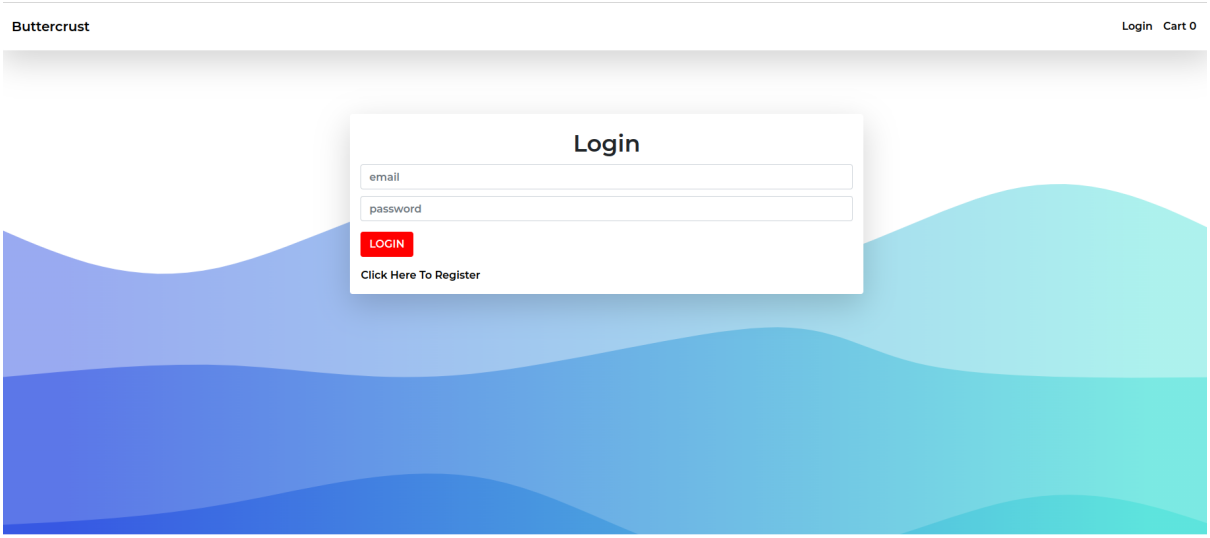
- **Line Graph**

6. Features and API

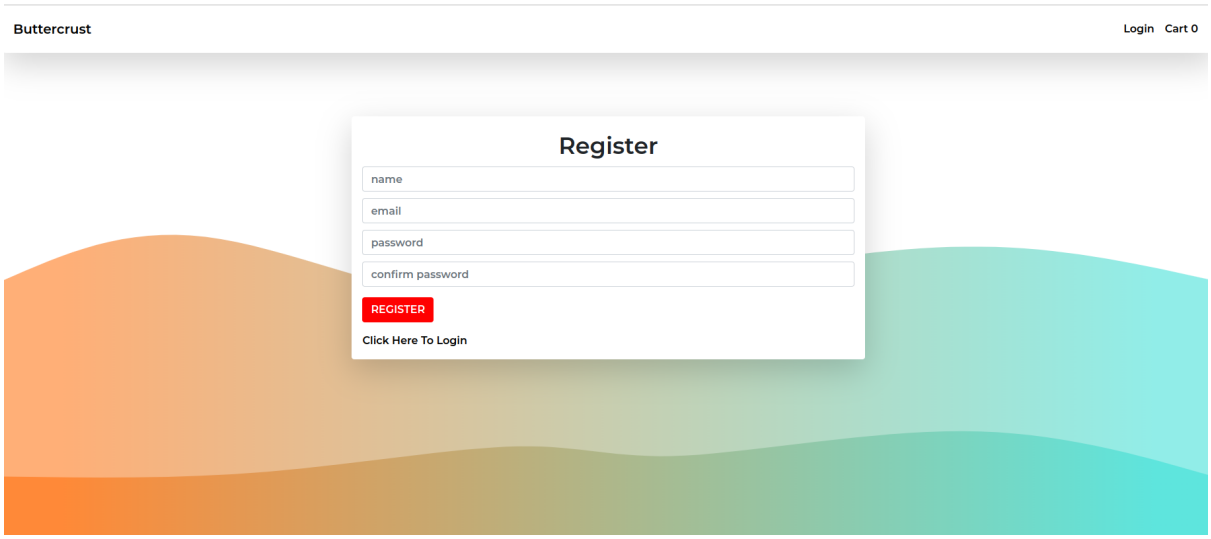
6.1 Features

This application has the following functionalities:

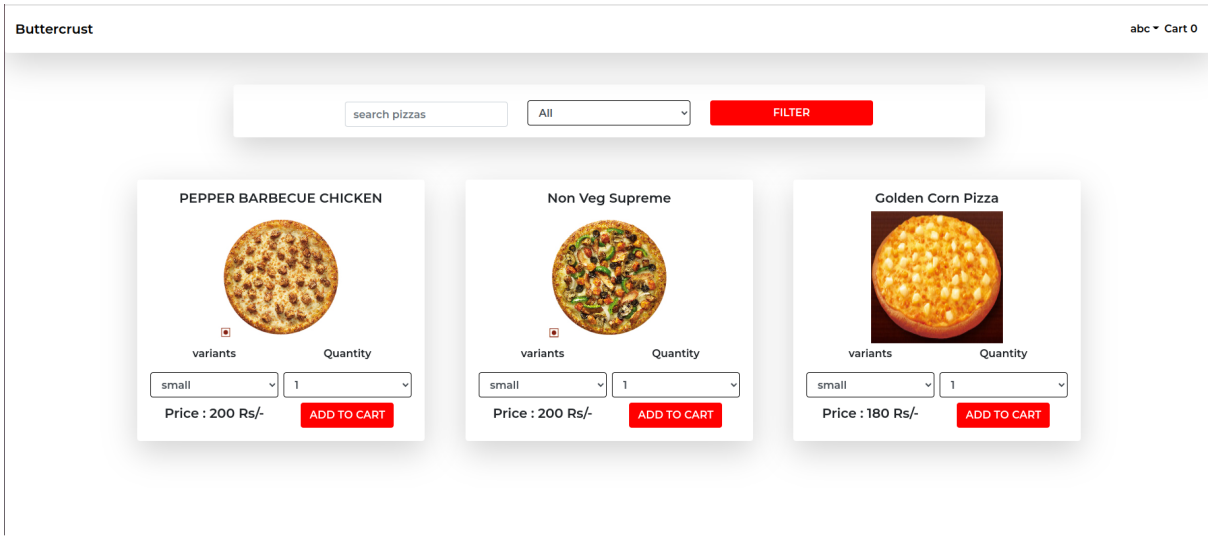
- **Login Page**



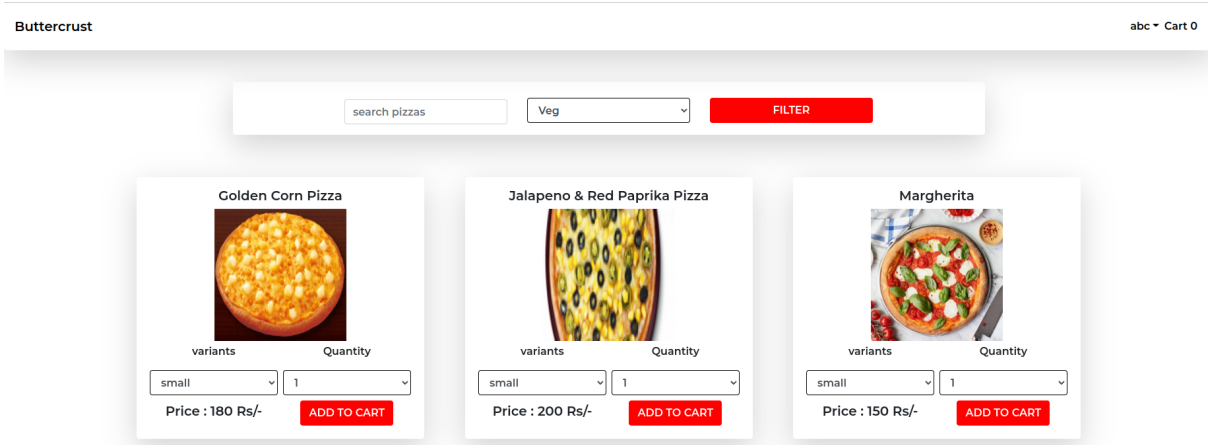
- **Update quantity in cart**



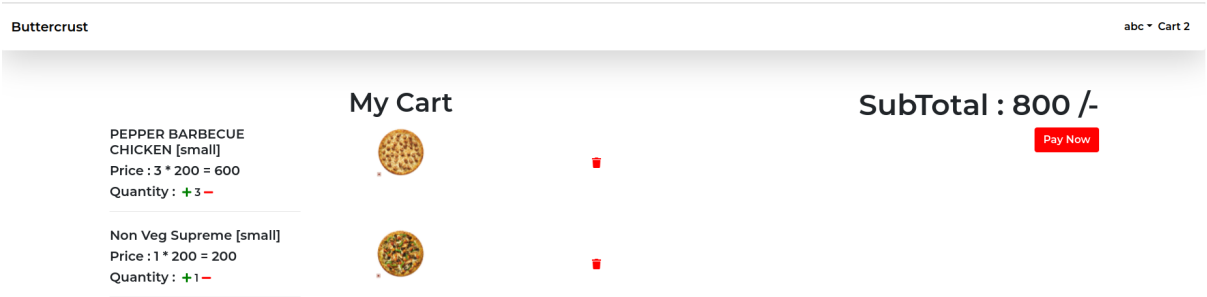
- **User Dashboard**



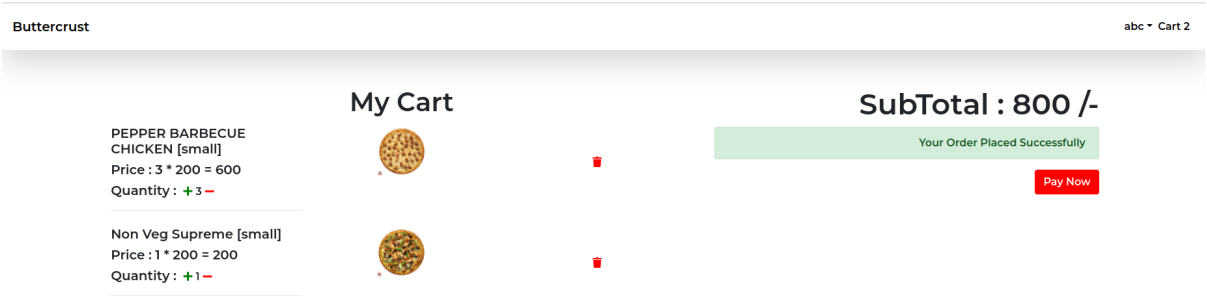
- **Pizza Filter**



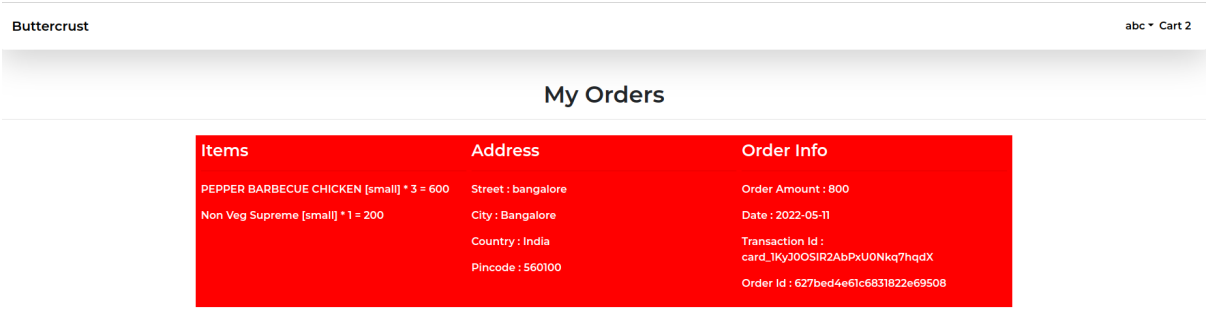
- **Cart Page**



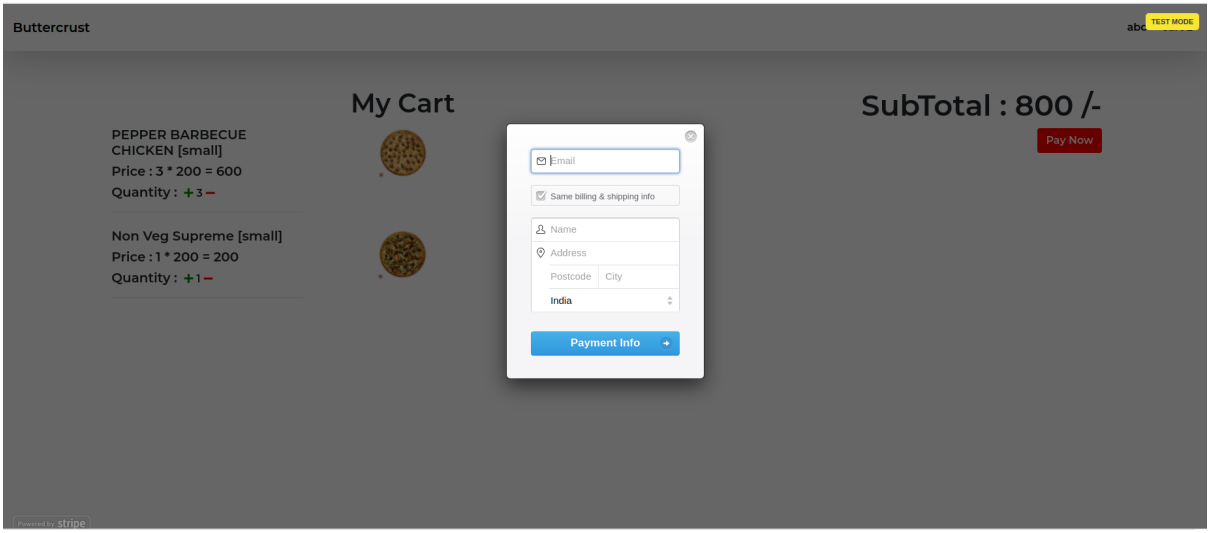
- **Order Placed**



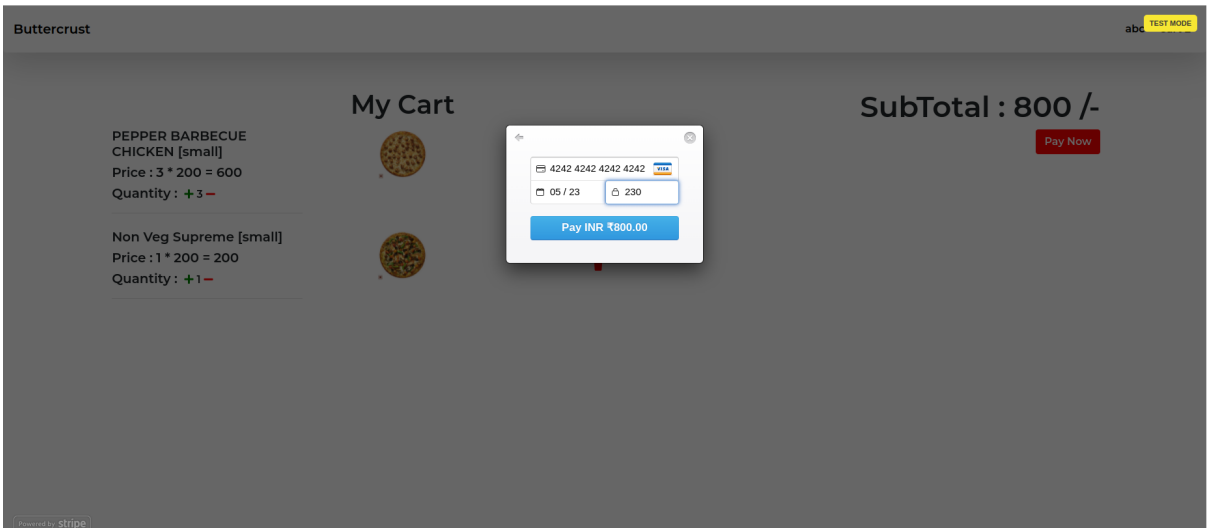
- **Orders Page**



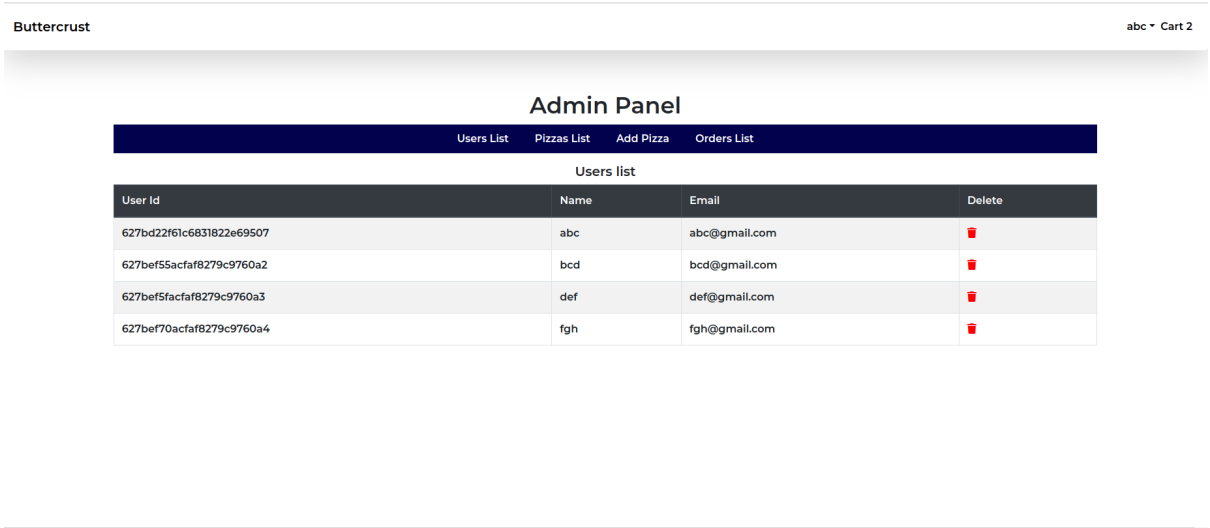
- **Stripe Payment Gateway**



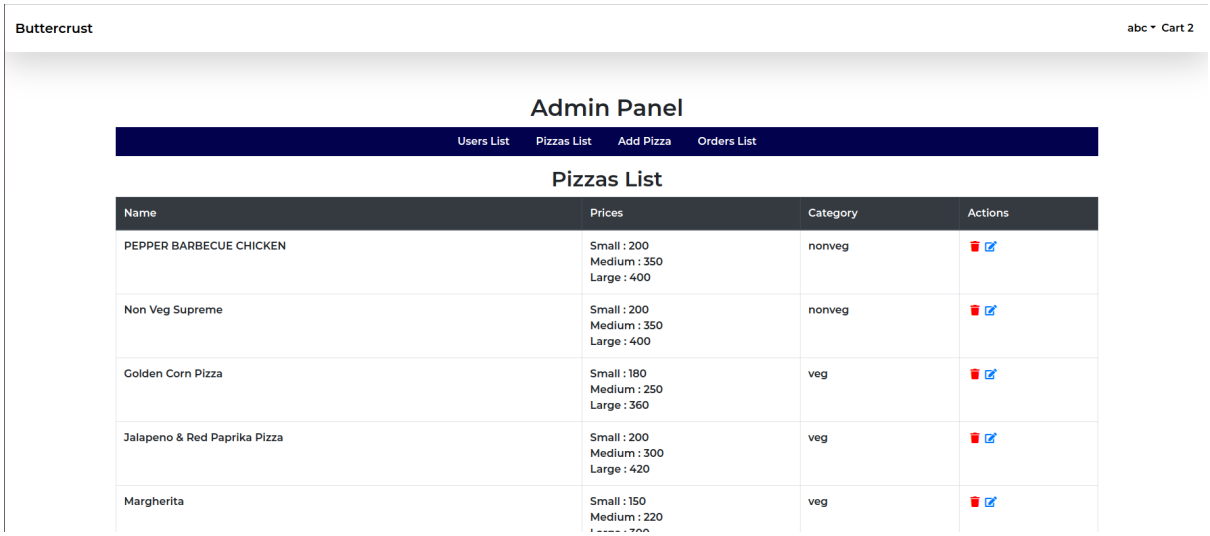
- **Card Details**



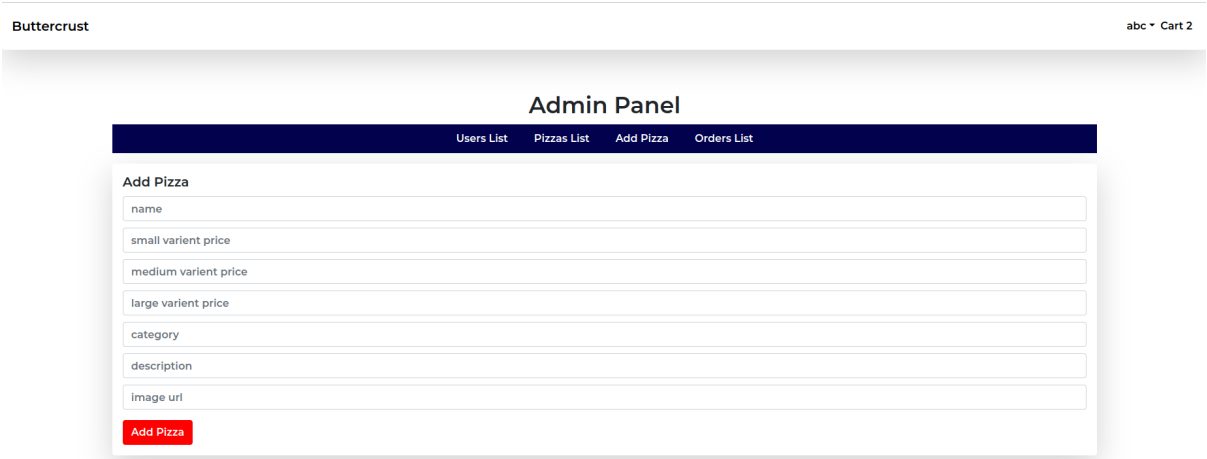
- **Admin Panel - Users List**



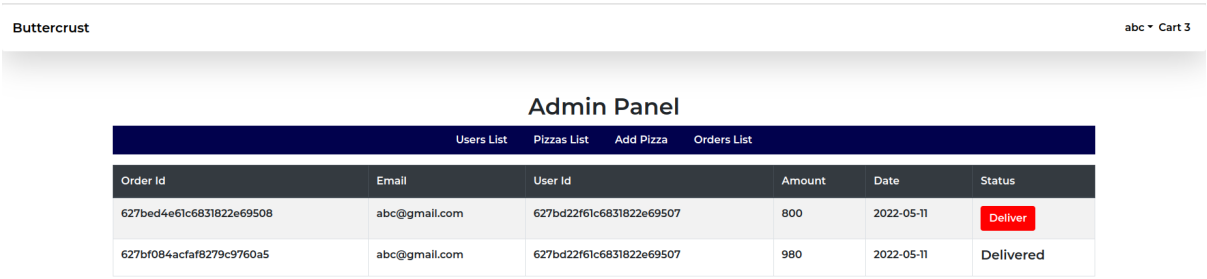
- **Admin Panel - Pizzas List**



- **Admin Panel - Adding New Pizzas to the Menu**



- **Admin Panel - Orders List**



Show Applications

6.2 APIs

Serial no.	Use Case	API	Request Method	End-user
1.	User Registration	/api/users/register	POST	User
2.	User Login	/api/users/login	POST	User
3.	Get a list of all registered users	/api/users/getallusers	GET	Admin
4.	Delete a user	/api/users/deleteuser	POST	Admin
5.	Get a list of all pizzas	/api/pizzas/getallpizzas	GET	User
6.	Add new pizza to the menu	/api/pizzas/addpizza	POST	Admin

7.	Get a specific pizza	/api/pizzas/getpizzabyid	POST	User
8.	Update pizza description	/api/pizzas/editpizza	POST	Admin
9.	Remove a pizza from the menu	/api/pizzas/deletepizza	POST	Admin
10.	Place order	/api/orders/placeorder	POST	User
11.	Get the orders of a user	/api/orders/getuserorders	POST	User
12.	Get a list of all orders	/api/orders/getallorders	GET	Admin
13.	Update the delivery status	/api/orders/deliverorder	POST	Admin

6.3 Code Snapshots

- **Server.js**

JS server.js > ...

```
const express = require("express");

const Pizza = require('./models/pizzaModel')

const app = express();
const db = require("./db.js")
app.use(express.json());
const path = require('path')
const pizzasRoute = require('./routes/pizzasRoute')
const userRoute = require('./routes/userRoute')
const ordersRoute = require('./routes/ordersRoute')









app.use('/api/pizzas/', pizzasRoute)
app.use('/api/users/' , userRoute)
app.use('/api/orders/' , ordersRoute)


if(process.env.NODE_ENV ==='production')
{
    app.use('/' , express.static('client/build'))

    app.get('*' , (req , res)=>{

        res.sendFile(path.resolve(__dirname  , 'client/build/index.html'))

    })
}









const port = process.env.PORT || 8000;

app.listen(port, () => `Server running on port port 🔥`)
```

- **ordersRoute.js**

routes > JS ordersRoute.js > ...

```
const express = require("express");
const router = express.Router();
const { v4: uuidv4 } = require('uuid');
const stripe = require("stripe")("sk_test_51IYnC0SIR2AbPxU0EiMx1fTwzbZXLbkaOcbc2cXx49528d9TGkQVjUINJfUDAnQMVaBFfBDP5xtcHCkZG1n1V3E800U7qXFmGf")
const Order = require('../models/orderModel');
const logger = require("../utils/logger");

router.post("/placeorder", async (req, res) => {


    logger.log({
        level: "info",
        message: "Order place request",
    });
    const { token, subtotal, currentUser, cartItems } = req.body

    try {
        const customer = await stripe.customers.create({
            email: token.email,
            source: token.id
        })

        const payment = await stripe.charges.create({
            amount: subtotal * 100,
            currency: 'inr',
            customer: customer.id,
            receipt_email: token.email
        }, {
            idempotencyKey: uuidv4()
        })

        if (payment) {

            const neworder = new Order({
                name: currentUser.name,
                email: currentUser.email,
                userid: currentUser._id,
                orderItems: cartItems,
                orderAmount: subtotal,
                shippingAddress: {
                    street: token.card.address_line1,
                    city: token.card.address_city,
                    country: token.card.address_country,
```

7. Key challenges

- The challenges we faced initially were related to docker-compose. We were able to create the images properly and both the containers were running as well but the frontend and backend were unable to communicate with each other. The reason being we did not provide the container name in the docker-compose file. The fix was to put the container name in the docker-compose file and then use the same “*container name*” as URI in the API instead of “*localhost*”

8. Key learnings

8.1 Technical

- React.js
- Node.js
- Jenkins
- Docker
- Ansible
- Amazon Web Services

8.2 Experience

- Designing this application has taught us the power of collaborating and the importance of being a team player.

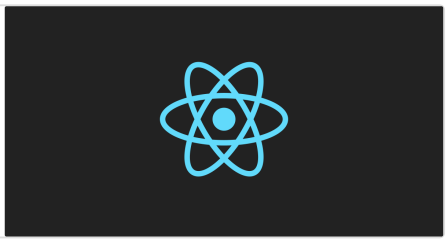
- Creating the application from scratch and aligning it with the team's vision and choosing the correct people for the appropriate task have taught us to believe and support each other.

9. References

Getting Started - React

A JavaScript library for building user interfaces

https://reactjs.org/docs/getting-started.html



https://docs.docker.com/get-started/

User Guide - Ansible Documentation

Welcome to the Ansible User Guide! This guide covers how to work with Ansible, including using the command line, working with inventory, interacting with data, writing tasks, plays, and playbooks; executing

https://docs.ansible.com/ansible/latest/user_guide/index.html#getting-started

Tutorials overview

The following tutorials show how to use Jenkins to cover the basics of CI/CD concepts based on specific technology stacks. Choose the tutorial that's relevant to your technology stack or one that you're most

https://www.jenkins.io/doc/tutorials/#pipeline/

Index | Node.js v18.1.0 Documentation

https://nodejs.org/docs/latest-v10.x/api/

What is Amazon EC2?

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (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,

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html