

# DEVOPS PROJECT REPORT



**Project Title:** Web Application development using ci/cd pipelines with Jenkins, git, dockers and terraform

**Project Members:**

<b>Batch No</b>	11
22000333105	SAI RAGHAVENDRA
2200032740	NIKHIL VARDHAN REDDY
2200031490	TARISH
2200032535	RAMBABU

**Instructor Name:** Dr. Y. Prasanth

**Course Coordinator:** Ch. Lavanya Susanna

**Course Title:** Cloud DevOps

**Course Code:** 22SDCI05R

## Table of Contents

Sno	Content	Page No
1.	Introduction	3
2.	Project Overview	4
3.	Key Technology and Tools used in Project	5
4.	Software and Hardware Requirements	6
5.	Setting Up the Environment	7
6.	Workflow Overview	8
7.	Git – Storing Versions of the App	9
8.	Terraform–Provisioning Infrastructure as a Code	11
9.	Jenkins–Automating Builds and Continuous Delivery	13
10.	Docker–Switching between old and new versions	15
11.	Conclusion	17
12.	References	18

## Introduction

In modern software engineering, the demand for rapid delivery and high availability of applications has given rise to the adoption of Continuous Integration and Continuous Deployment (CI/CD) methodologies. Traditional software development practices often involve manual intervention, leading to increased chances of human error, slower release cycles, and inconsistent environments. CI/CD pipelines address these challenges by automating the entire software lifecycle—from code integration and testing to deployment and infrastructure provisioning.

This project centers around the development and deployment of a web application using CI/CD pipelines, employing widely used tools such as Jenkins, Git, Docker, and Terraform. By integrating these tools, the project demonstrates how automation can streamline development workflows, reduce time-to-market, and improve overall application quality.

The objective is to create an end-to-end automated pipeline where every code change is continuously integrated, tested, and deployed across multiple environments in a consistent and reproducible manner. Infrastructure provisioning is also automated using Infrastructure as Code (IaC) practices with Terraform, ensuring scalability and manageability in cloud-native environments.

This project not only provides a hands-on implementation of CI/CD principles but also showcases best practices in modern DevOps culture, enabling faster feedback, early bug detection, and reliable delivery.

The goal of this project is to demonstrate the automation of the entire software delivery lifecycle for a web application, from code commit to deployment and infrastructure setup. The core idea is to implement a CI/CD pipeline that not only handles application-level tasks (build, test, and deployment) but also manages cloud infrastructure using Terraform.

The web application serves as a practical use case for applying DevOps practices in a real-world scenario. Developers commit code to a Git repository, which is automatically picked up by Jenkins to initiate the pipeline. Jenkins then builds the application, creates Docker images, and triggers Terraform to provision or update the necessary infrastructure components on a cloud platform (e.g., AWS). Finally, the Docker containers are deployed onto the provisioned resources, followed by automated testing to validate the deployment.

This seamless integration of CI/CD and IaC practices helps in achieving consistency, reducing deployment errors, and promoting faster release cycles. The project not only simplifies the software deployment process but also ensures that both application and infrastructure configurations are version-controlled and repeatable. This provides a robust foundation for building scalable and resilient applications in cloud environments.

## **Key Technologies and Tools**

The successful implementation of this project relies on a well-integrated set of technologies, each serving a crucial role in the CI/CD pipeline:

Git is used for distributed version control, enabling collaboration and tracking of changes in the codebase. It helps maintain multiple branches and supports integration with continuous integration tools. Git also enables seamless rollback to previous versions, making the development process more robust and manageable.

Jenkins is the backbone of the automation process. It is configured to monitor code repositories, initiate pipeline executions on code changes, and manage stages such as build, test, deployment, and infrastructure provisioning. Jenkins supports extensive customization and scalability through plugins and declarative pipelines.

Docker plays a vital role in containerizing the application, ensuring that it runs uniformly across development, testing, and production environments. Docker images are built in Jenkins and stored in a registry for deployment. This allows for reproducible environments, faster startup times, and better resource utilization.

Terraform is used to define and provision infrastructure resources through code. It allows teams to automate the creation of compute instances, networking, storage, and other cloud services in a reliable and repeatable manner. It also supports version control of infrastructure, enabling teams to track and audit infrastructure changes over time.

## **Software Requirements**

### **1) Version Control System**

#### **a. Git (Latest Version)**

(1) Required for source code management and version control

(2) Enables tracking of code changes and collaboration among developers

(3) Can be installed from the Git Official Website

## 2) Infrastructure as Code (IaC) Tool

### a. Terraform (Latest Version)

(1) Automates the provisioning and management of cloud infrastructure

(2) Facilitates reproducible and version-controlled infrastructure

(3) Install from Terraform Official Website

## 3) CI/CD Automation Tool

### a. Jenkins

(1) Automates the build, test, and deployment processes

(2) Supports integration with various tools through plugins

(3) Installation available via Jenkins Official Website

(4) Essential Plugins: Git, Docker Pipeline, Terraform, etc.

## 4) Containerization Platform

### a. Docker (Latest Version)

(1) Used for building, packaging, and running applications in containers

(2) Ensures consistency across development, testing, and production environments

(3) Download from Docker Official Website

## **Setting Up the Environment**

To successfully deploy the application using Git, Jenkins, Docker, and Terraform, follow these steps to set up the environment:

### 1. Install Git

- Download and install Git from the official website.
- Use Git to clone the application repository:

- `git clone https://github.com/your-repo.git`

## 2. Set Up Jenkins

- Download and install Jenkins on your local machine or server using the Jenkins installation guide.
- Start Jenkins and unlock it using the administrator password (found in the Jenkins home directory).

### a. Install essential plugins such as:

- Git Plugin
- Docker Pipeline Plugin
- Terraform Plugin
- Create a new Jenkins job or pipeline for automating the deployment process.
- Integrate Jenkins with GitHub using webhooks for automatic pipeline triggers.
- Define pipeline stages including build, test, infrastructure provisioning, and deployment.

## 3. Install Docker

- Download and install Docker from the official Docker website.
- Verify installation with:
- `docker --version`
- Ensure the Docker service is running, and the user has permission to run Docker commands.

## 4. Install Terraform

- Download and install Terraform from the Terraform official website.
- Verify the installation with:
- `terraform -version`

## Workflow Overview

The deployment workflow follows a structured automation pipeline using Git, Jenkins, Docker, and Terraform to ensure seamless application deployment. The process begins with developers pushing code changes to the Git repository, which triggers Jenkins to start the CI/CD pipeline. Jenkins pulls the latest code and initiates the Terraform execution to provision the required cloud infrastructure dynamically. Once the infrastructure is set up, Jenkins proceeds to build and deploy the application using Docker containers, ensuring consistent and isolated environments.

Docker enables packaging the application with all its dependencies, eliminating the common “it works on my machine” problem. The containers are deployed to the cloud infrastructure provisioned by Terraform, creating a reliable and reproducible deployment process. Jenkins provides real-time monitoring, logging, and error reporting throughout the pipeline, which helps in identifying and resolving issues quickly. If any stage fails, Jenkins alerts the team and can halt or roll back the deployment as needed.

This streamlined workflow increases deployment speed, reduces human errors, and improves the reliability of application releases. It ensures consistency and repeatability across various environments—development, testing, and production—making it a scalable and efficient DevOps approach.

- Developers push code changes to the Git repository, triggering Jenkins to start the pipeline.
- Jenkins executes Terraform scripts to provision infrastructure on the chosen cloud platform.
- Jenkins builds Docker images of the application and pushes them to a container registry.
- Docker containers are deployed onto the provisioned infrastructure.
- Jenkins tracks the deployment process and provides logs and failure notifications.

## **Git – Storing Versions of the App**

Git is a distributed version control system used to manage the source code of the application. It allows developers to track changes, collaborate efficiently, and maintain a history of code updates. In this project, Git serves as the central repository where the application code is stored and managed. It integrates with Jenkins to automatically trigger the CI/CD pipeline when code changes are pushed. Git ensures that the latest version of the code is always used during the deployment process.

We use Git to enable version control, making it easier to manage and revert code changes when necessary. It supports team collaboration by allowing multiple developers to work on the same project simultaneously. Git integrates seamlessly with Jenkins, automating the build and deployment pipeline. It maintains a history of all changes, improving code transparency and traceability. Git ensures a reliable and consistent codebase

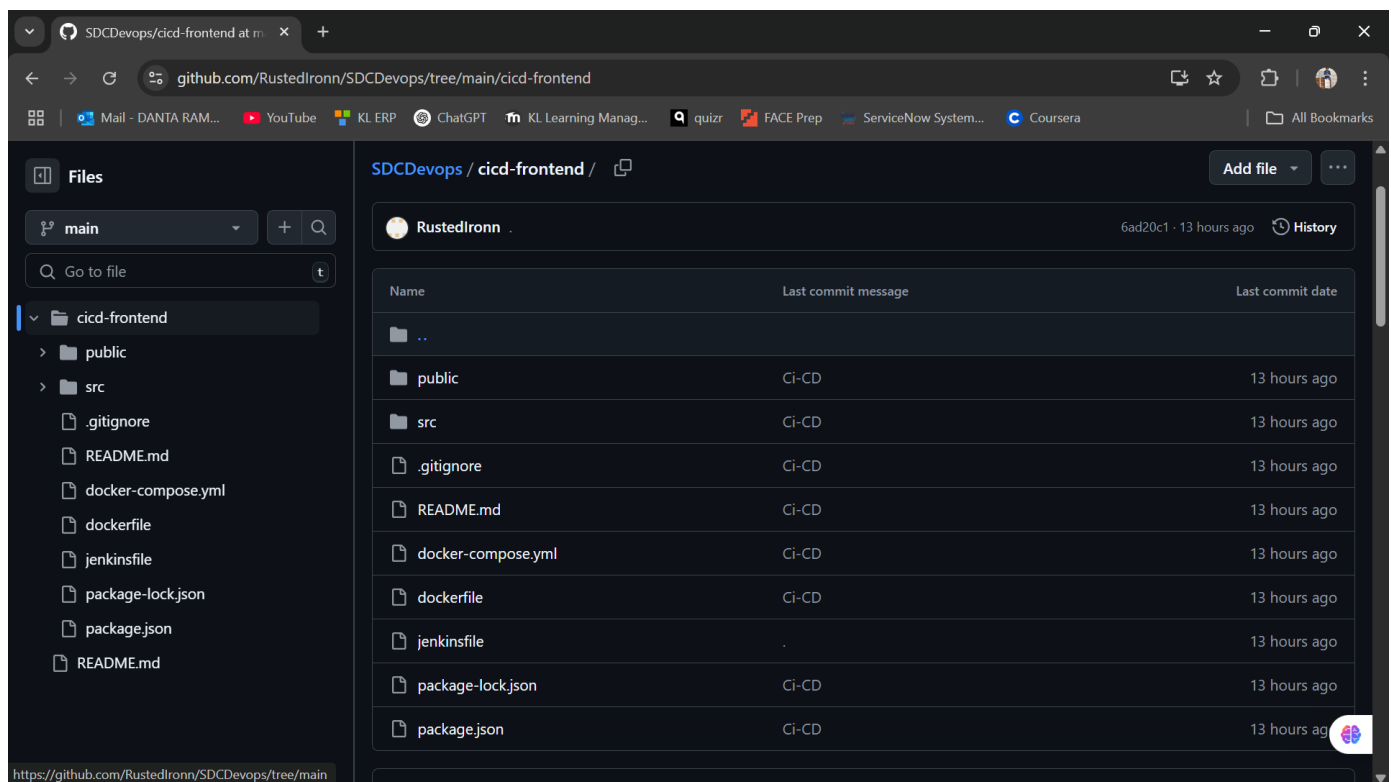


for every deployment.

### **Usage of git in this project**

First create a repository in github. After creation there exists https link which is used to clone the repository to local system. Open git bash in local system.

1. Create a directory in your system.
  - Command → `mkdir project`
2. Change to that directory.
  - Command → `cd project`
3. Initialize git here.
  - Command → `git init`
4. Clone the github repository here.
  - Command → `git clone https://github.com/RustedIronn/SDCDevops.git`
5. Change to that directory.
  - Command → `cd CloudDevops`
6. Here you upload the files or create the files.
7. After creating files add the files to git.
  - Command → `git add .`
8. Commit the changes which were made.
  - Command → `git commit -m "Version is added"`
9. Push the files to github repository
  - Command → `git push`



## Terraform – Provisioning Infrastructure as Code

Terraform is an open-source Infrastructure as Code (IaC) tool developed by Hashi Corp that allows users to define and provision data center infrastructure using a high-level configuration language called HCL (HashiCorp Configuration Language). With Terraform, infrastructure is described in code, making it easy to automate and manage resources across various cloud providers like AWS, Azure, and Google Cloud. One of Terraform's key features is its ability to create an execution plan showing what it will do before making changes, which helps avoid unexpected modifications.

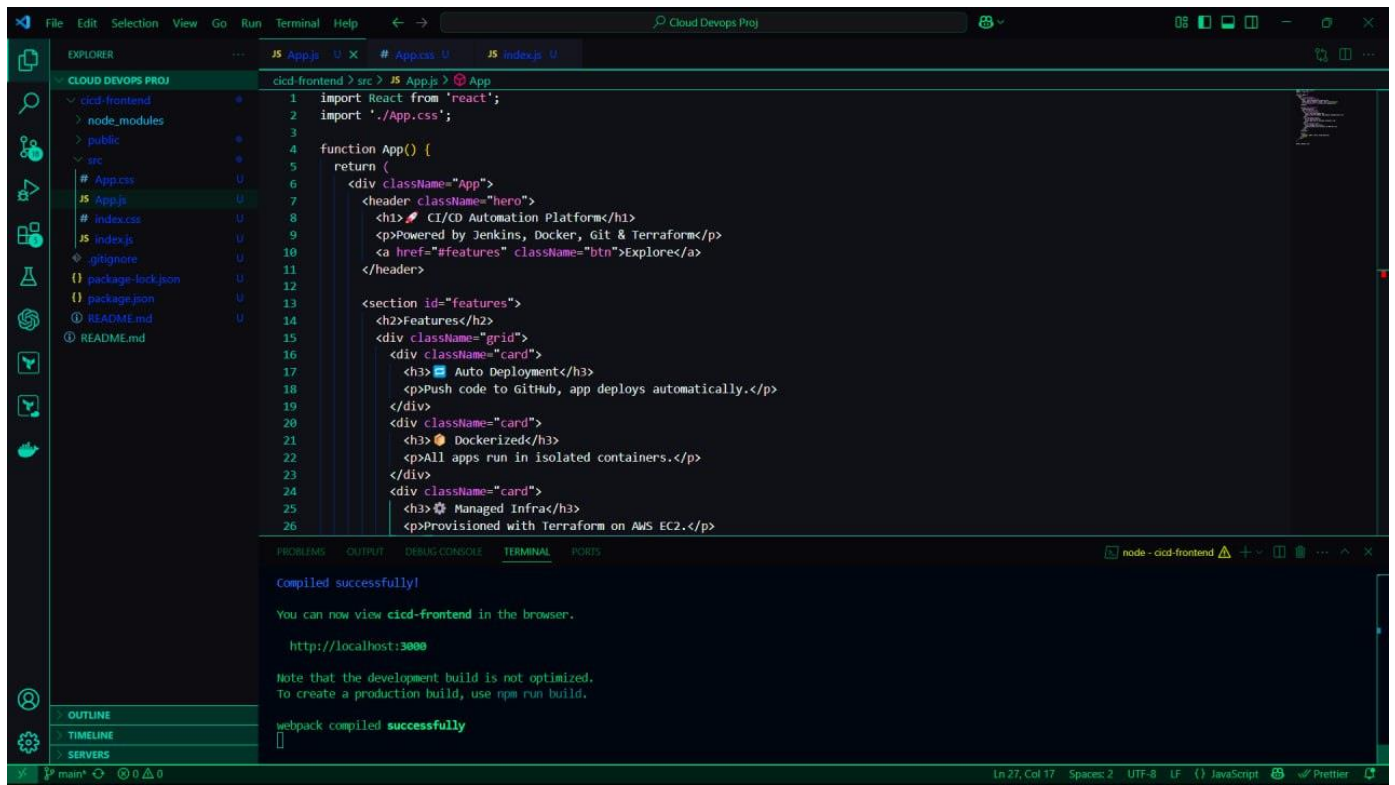
Terraform maintains a state file that tracks the current state of the infrastructure, allowing it to make incremental updates efficiently. It supports modularity, enabling users to reuse and organize code into smaller

components for better scalability. Using Terraform, teams can collaborate effectively and ensure consistency across development, staging, and production environments. Its provider-based architecture makes it extensible, meaning you can use it with not just cloud resources but also DNS providers, databases, and SaaS platforms. Terraforms declarative approach allows users to specify the desired end state, and it handles the execution to reach that state. It has become a standard in modern DevOps workflows due to its flexibility, portability, and ease of integration with CI/CD pipelines.

### **Usage of terraform in this project**

Created an EC2 instance with ubuntu as an virtual server using terraform. Using HCL, Developed a terraform file and launched an instance by following command.

- terraform init
- terraform validate
- terraform plan
- terraform apply
- terraform destroy



The screenshot shows the Visual Studio Code editor with a project named 'Cloud DevOps Proj'. The Explorer sidebar on the left shows the file structure, including 'cicd-frontend', 'node\_modules', 'public', 'src', and 'App.js'. The main editor area displays the content of 'App.js', which is a React component. The code defines an 'App' function that returns a JSX element. The JSX includes a header with a title 'CI/CD Automation Platform', a paragraph about the platform's capabilities, and a list of features: 'Auto Deployment', 'Dockerized', and 'Managed Infra'. The terminal at the bottom shows the output of a 'npm run build' command, indicating that the application was compiled successfully and is now available at 'http://localhost:3000'.

```
1 import React from 'react';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="hero">
8         <h1> CI/CD Automation Platform</h1>
9         <p>Powered by Jenkins, Docker, Git & Terraform</p>
10        <a href="#features" className="btn">Explore</a>
11      </header>
12
13      <section id="features">
14        <h2>Features</h2>
15        <div className="grid">
16          <div className="card">
17            <h3> Auto Deployment</h3>
18            <p>Push code to Github, app deploys automatically.</p>
19          </div>
20          <div className="card">
21            <h3> Dockerized</h3>
22            <p>All apps run in isolated containers.</p>
23          </div>
24          <div className="card">
25            <h3> Managed Infra</h3>
26            <p>Provisioned with Terraform on AWS EC2.</p>
27          </div>
28        </div>
29      </section>
30    </div>
31  );
32}
33
34export default App;
```

compiled successfully!

You can now view cicd-frontend in the browser.

http://localhost:3000

Note that the development build is not optimized.  
To create a production build, use `npm run build`.

webpack compiled successfully

## Jenkins – Automating Builds and Continuous Delivery

Jenkins is an open-source automation server that helps developers build, test, and deploy software efficiently. It is widely used for implementing Continuous Integration and Continuous Delivery (CI/CD) pipelines, making the development lifecycle faster and more reliable. Jenkins supports hundreds of plugins, allowing integration with version control systems like Git, build tools like Maven or Gradle, testing frameworks, and deployment platforms. It automates repetitive tasks such as compiling code, running unit tests, and deploying applications to staging or production environments. With Jenkins pipelines, developers can define the entire CI/CD flow using code, making it easier to maintain and version control. Jobs can be triggered manually, on

a schedule, or automatically in response to events like code pushes or pull requests.

Jenkins provides real-time feedback on builds and test results, enabling quick detection and resolution of issues. It also supports distributed builds, allowing multiple agents to run tasks in parallel to speed up the process. Jenkins is platform-independent and can run on various operating systems. It has a simple web interface for monitoring, configuring jobs, and checking logs. Security features like user authentication, authorization, and credential storage help ensure safe operations. Jenkins is commonly used in DevOps workflows, and its flexibility makes it suitable for projects of any size. Integration with tools like Docker, Kubernetes, and Terraform extends its functionality. By automating the entire software delivery pipeline, Jenkins helps teams release faster, with fewer bugs and greater confidence.

### **Usage of Jenkins in this project**

A pipeline is created using Jenkins. Web application was deployed in this pipeline. Launch Jenkins in local host.

1. To create a pipeline, Click on new item and give a name to it.
2. Select pipeline and provide git repo link.
3. Now click on build now.
4. Success message will be displayed on console.

## **Docker—Switching between old and new versions**

Docker is an open-source platform designed to automate the deployment, scaling, and management of applications using containerization. Containers allow developers to package applications and their dependencies into a single, portable unit that can run consistently across different environments. This eliminates the classic “it works on my machine” problem and accelerates development, testing, and deployment. Docker containers are lightweight and share the host operating system’s kernel, making them more efficient than traditional virtual machines.

When working with Docker in a real-world project—especially in CI/CD pipelines—it's common to switch between different versions of Docker or specific application images. This can be necessary when testing new features against previous stable environments or rolling back to a previous state due to bugs in newer

versions. Docker makes this easy through version-tagged images that you can pull and run using simple commands. You can also manage Docker CLI versions using package managers or version managers to align with specific project requirements.

### **Usage of Docker in this project:**

In this project, Docker plays a crucial role in containerizing the web application and ensuring a consistent runtime environment across the CI/CD pipeline.

#### **1. Launch Jenkins using Docker:**

- Pull and run Jenkins

#### **2. Create a New Pipeline:**

- Click on “New Item”, enter a name for the pipeline.
- Select "Pipeline" and click OK.

#### **3. Configure Git Repository:**

- In the Pipeline configuration section, provide the Git repository URL where the source code and Jenkinsfile are located.

#### **4. Build the Pipeline:**

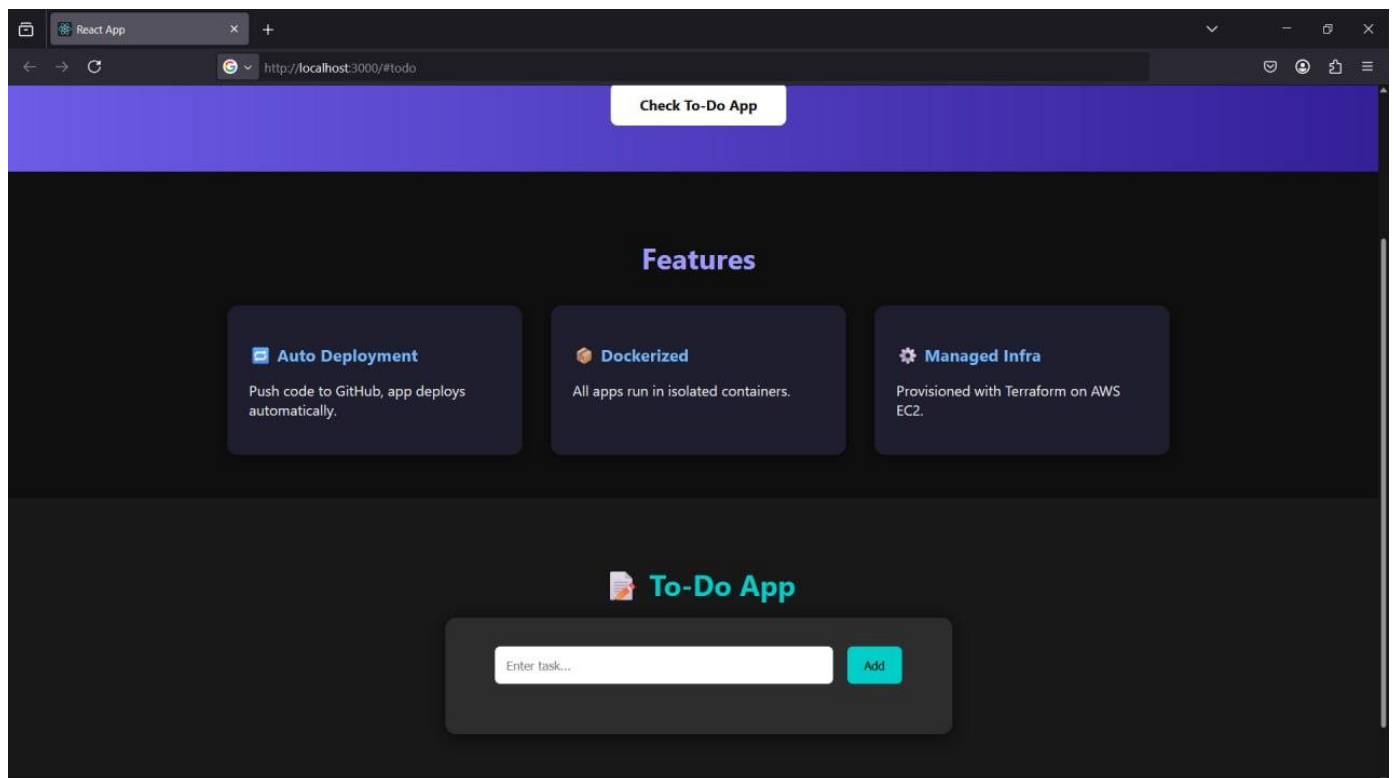
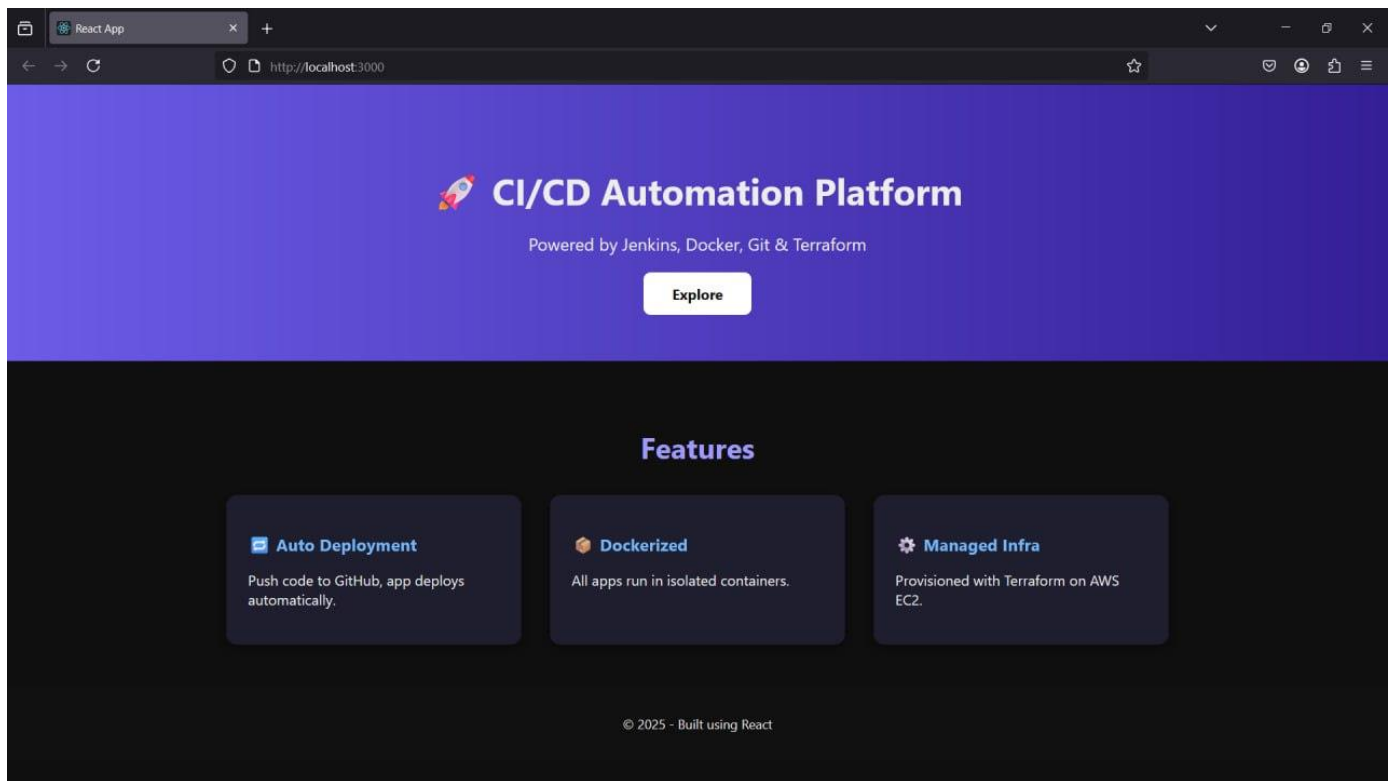
- Click “Build Now”.
- Jenkins will fetch the code, build the Docker image, and deploy the web application inside a container.

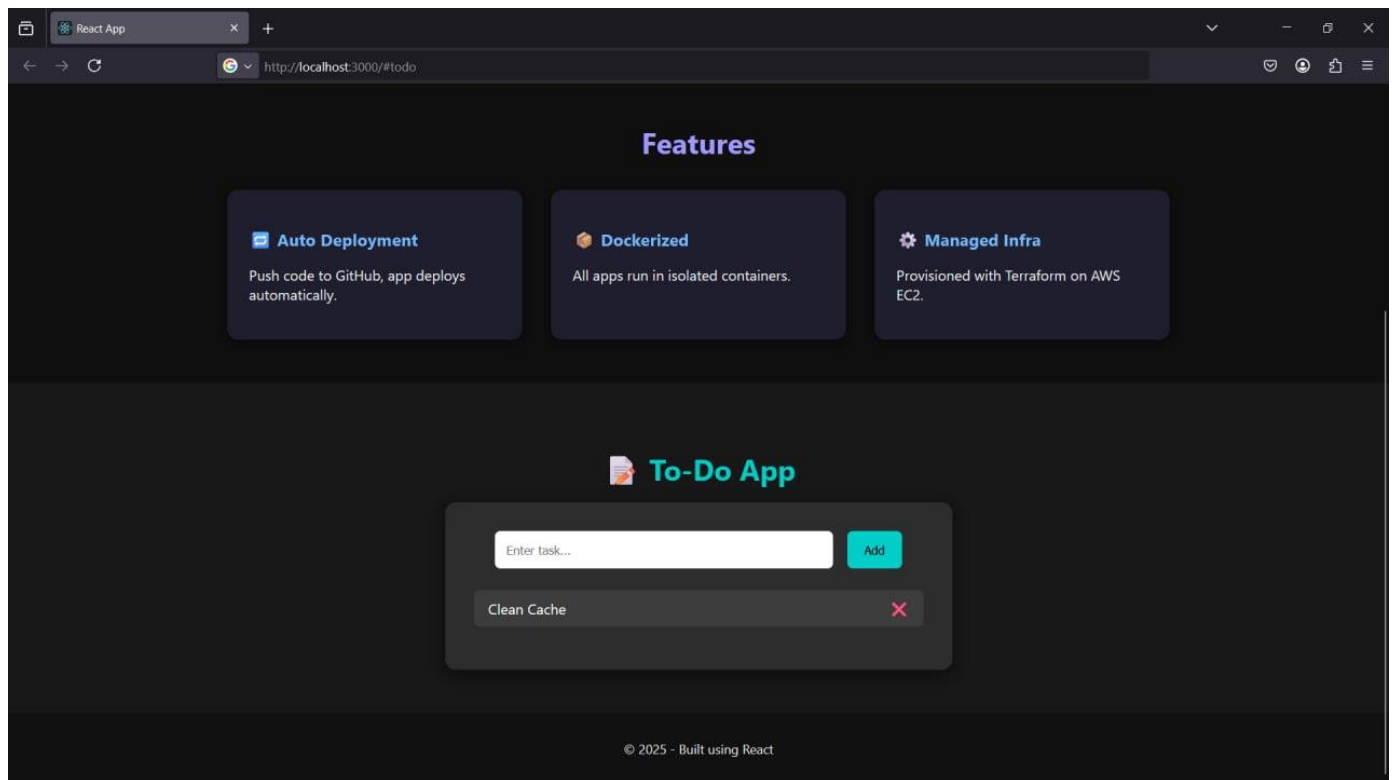
#### **5. Output:**

- Upon successful execution, a success message will be displayed in the Console

**Final Outputs**







## Conclusion

This project successfully demonstrates the implementation of a complete CI/CD pipeline for web application deployment using modern DevOps tools—Git, Jenkins, Docker, and Terraform. Each of these tools plays a critical role in automating the software delivery lifecycle and ensuring fast, reliable, and consistent

deployments.

Git serves as the version control system, allowing collaborative development, code tracking, and seamless integration with Jenkins. The source code and the Jenkins file are maintained in a Git repository, ensuring that the pipeline always runs with the most up-to-date codebase.

Jenkins acts as the automation server, pulling code from Git, building the application, running tests, and triggering Docker builds as defined in the pipeline configuration. It orchestrates every stage of the CI/CD process, providing visibility and control over the build and deployment workflow.

Docker ensures consistent environments across development, testing, and production by packaging the application and its dependencies into containers. This eliminates the "it works on my machine" problem and speeds up deployment by using lightweight and reproducible environments.

Terraform is used for infrastructure as code (IaC), provisioning the necessary cloud infrastructure for hosting the application. It enables the creation and management of servers, networking, and other infrastructure components in a declarative, automated, and scalable manner.

## References

1. Github: <https://docs.github.com/en>
2. Terraform: <https://developer.hashicorp.com/terraform/docs>

3. Jenkins: <https://www.jenkins.io/doc/>

4. Docker: <https://docs.docker.com/>