

---

# **CREATING A CI/CD PIPELINE USING DEVOPS**

*{Continuous Integration/Continuous Delivery}*

---

## **Parallel And Distributed Computing (UCS645) Project**

### **Submitted By:**

<b>Arpit Sagar</b>	<b>(102003130)</b>	<b>3CO6</b>
<b>Anshita</b>	<b>(102003182)</b>	<b>3CO8</b>
<b>Medhansh Singh Verma</b>	<b>(102003188)</b>	<b>3CO8</b>

### **Submitted To:**

**Dr.Manish Kumar**



THAPAR INSTITUTE  
OF ENGINEERING & TECHNOLOGY  
(Deemed to be University)

DEPARTMENT OF COMPUTER SCIENCE and ENGINEERING  
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY,  
(A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB  
INDIA

**April 2023**

## **ABSTRACT**

Fast growing automation and emerging need of new tools for faster response systems has paved the way for the organizational approach of DevOps , a software development methodology integrating development and operations using Continuous Integration and Continuous Delivery (CI/CD), real-time monitoring, incident response systems and collaboration platforms.

Building a stable CI/CD pipeline makes it possible for development teams to work on various pieces of code simultaneously but independently, enabling them to seamlessly integrate their contributions to the source code repository on a continuous basis.

In this project we create an automated CI/CD pipeline using DevOps tool “Jenkins” which includes the support for Kubernetes and Docker containers to establish best and simplified developments.

The CI/CD pipeline will automate the process of building, testing, and deploying code changes. As soon as developers push changes to the code repository, the pipeline will trigger a build and run automated tests to ensure that the code is of high quality. Running Jenkins jobs in a sequence resembling a pipeline to integrate changes made on one server reflect automatically on the desired target location which can then be pushed onto production.

## **INTRODUCTION**

The purpose of this project is to create an efficient and automated Continuous Integration and Continuous Deployment (CI/CD) pipeline using Jenkins, with an aim to streamline the software development process and reduce manual intervention, leading to faster and more efficient delivery of high-quality software.

This Pipeline is responsible for building codes, running tests, and deploying new software versions. The Pipeline executes the job in a defined manner by first coding it and then structuring it inside several blocks that may include several steps or tasks.

Continuous Integration is a practice that integrates code into a shared repository. It uses automated verifications for the early detection of problems.

CD (Continuous Delivery/Deployment) refers to the automated process of delivering software updates to production as quickly and efficiently as possible. The goal of CD is to minimize the time and effort required to deploy new features, bug fixes, and other changes to end-users. Continuous delivery is a practice that focuses on automating the delivery process from the development environment to the production environment. This involves building, testing, and deploying the code to a staging environment where it can be further tested before being deployed to production.

This means that any changes made to the codebase are automatically deployed to production without requiring any manual intervention.

Jenkins is the DevOps tool used for CI/CD pipelines. It is an open-source automation tool used to build and test software projects. The tool makes it more convenient for developers to integrate changes to the project. Jenkins achieves Continuous Integration with the help of plugins.

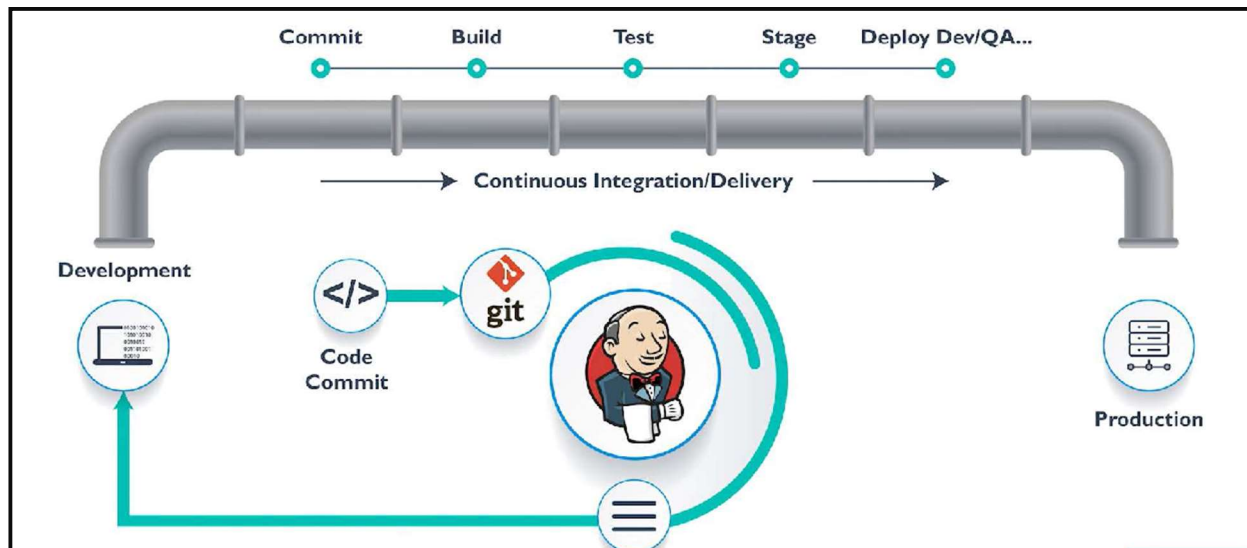
By creating this CI/CD pipeline, we aim to increase the speed and efficiency of software delivery, reduce the risk of manual errors, and improve the overall quality of the software. The pipeline will be continuously monitored and improved over time to ensure that it remains effective and efficient.

In addition to these core steps, the pipeline may also include other stages, such as static code analysis, security scanning, and performance testing. These stages help to ensure that the code meets the desired quality standards and is secure and performant.

A typical CI/CD pipeline works in 4 phases that are listed below:

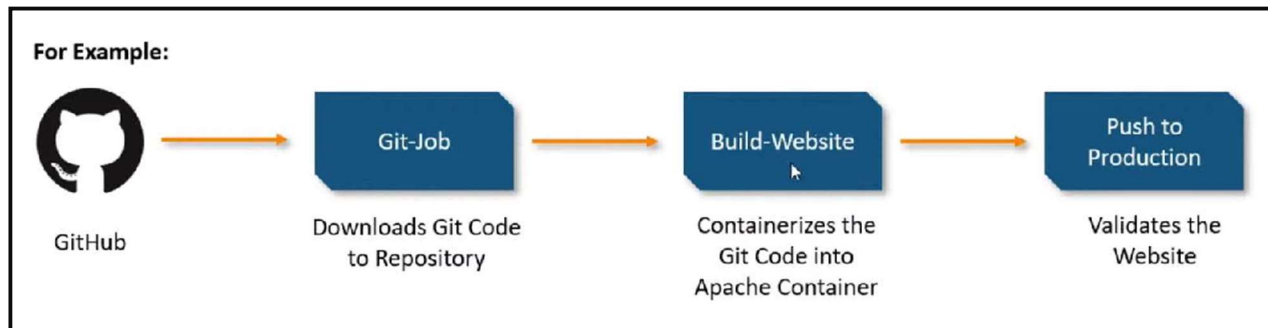
- **Phase 1: Commit** - This is the actual phase where developers commit changes to the code.
- **Phase 2: Build** – In this phase, the source code is integrated into the build from the repository.
- **Phase 3: Test Automation** - This step is an integral part of any CI/CD pipeline. The source code previously integrated into the build is subjected to a systematic cycle of testing.
- **Phase 4: Deploy** - The tested version is finally sent for deployment in this phase

The project begins with the installation and configuration of Jenkins on a server. Afterward, a series of build jobs are created to automate the building of the software applications. The build jobs are designed to trigger automatically when changes are committed to the source code repository.



The next stage of the project involves the implementation of automated testing to ensure that the software applications are functioning as expected. Jenkins is

used to execute unit tests, integration tests, and system tests as part of the build process.



Finally, the project concludes with the deployment of the software applications to production. Jenkins is used to automate the deployment process, ensuring that new changes are delivered seamlessly without causing any downtime.

### Benefits of CI/CD Pipeline:

**Faster release cycles:** A CI/CD pipeline automates the entire software delivery process, allowing teams to release new features and updates faster and more frequently.

**Higher quality software:** Automated testing and deployment processes can help catch bugs and issues early in the development cycle, leading to better quality software.

**Improved collaboration:** A CI/CD pipeline encourages collaboration between developers, testers, and operations teams, leading to better communication and coordination.

So we successfully integrate Jenkins with the GitHub repository. With each push, commit, or update, GitHub will trigger the Jenkins job, which in turn will execute the necessary steps to deploy the changes.

Overall, the project demonstrates the benefits of using Jenkins to create an efficient and automated CI/CD pipeline.


## **PROPOSED METHODOLOGY**

Proposed methodology for this project involves the following steps:

1. **Plan and design the CI/CD pipeline:** The first step is to identify the requirements of the project, including the type of software being developed, the programming languages used, and the testing frameworks required. Based on this, the CI/CD pipeline can be designed and planned out.
2. **Set up Jenkins:** The next step is to set up Jenkins on a server, configure it, and install any necessary plugins. This involves creating a new project, configuring the build settings, and setting up a GitHub webhook to trigger the build.
3. **Create and configure build jobs:** The build jobs will be created to automate the building of the software applications. These jobs will be designed to trigger automatically when changes are committed to the source code repository. The build jobs will be configured to compile the code, create a deployable artifact, and store it for testing.
4. **Implement automated testing:** Automated testing will be implemented to ensure that the software applications are functioning as expected. Jenkins will be used to execute unit tests, integration tests, and system tests as part of the build process. This will involve configuring the test framework, setting up the test environment, and defining the tests to be executed.
5. **Configure deployment:** The final step is to configure the deployment process. Jenkins will be used to automate the deployment process, ensuring that new changes are delivered seamlessly without causing any downtime. This will involve defining the deployment environment, configuring the deployment process, and setting up automatic rollback in case of any issues.
6. **Continuous monitoring and improvement:** Once the CI/CD pipeline is set up, it will be continuously monitored and improved over time to ensure that it remains effective and efficient. This will involve analyzing the performance metrics, identifying areas for improvement, and making necessary changes to optimize the pipeline.


## Working Screenshots


- Servers created:


Jenkins


Dashboard >

+ New Item

 People

 Build History

 Manage Jenkins

 My Views

Build Queue (2)

Git-job

build-website

Build Executor Status





1 idle

2 idle

Production





Staging

Search (CTRL+K)

   PDC Project  log out

All

+

S	W	Name	Last Success	Last Failure	Last Duration
		<a href="#">build-website</a>	N/A	N/A	N/A
		<a href="#">Git-job</a>	N/A	N/A	N/A


Icon: 


S


M

L

Icon legend

 Atom feed for all

 Atom feed for failures

 Atom feed for just latest builds

Add description

REST API

Jenkins 2.400

**Instances** (1/3) [Info](#)

Find instance by attribute or tag (case-sensitive)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6 IPs	Monitori
Jenkins	i-03f8a07b52b664fd6	Running	t2.micro	2/2 checks passed	No alarms	ap-south-1a	ec2-13-232-254-174.ap...	13.232.254.174	-	-	disabled
Staging	i-07f32b966b9ed8526	Running	t2.micro	2/2 checks passed	No alarms	ap-south-1a	ec2-43-205-130-107.ap...	43.205.130.107	-	-	disabled
Production	i-098276aafe5543496	Running	t2.micro	2/2 checks passed	No alarms	ap-south-1a	ec2-13-126-166-91.ap...	13.126.166.91	-	-	disabled

### Instance: i-098276aafe5543496 (Production)

- Details** | Security | Networking | Storage | Status checks | Monitoring | Tags

**▼ Instance summary** [Info](#)

Instance ID i-098276aafe5543496 (Production)	Public IPv4 address 13.126.166.91   <a href="#">open address</a>	Private IPv4 addresses 172.31.40.250
IPv6 address -	Instance state <span style="color: green;">● Running</span>	Public IPv4 DNS ec2-13-126-166-91.ap-south-1.compute.amazonaws.com   <a href="#">open address</a>
Hostname type IP name: ip-172-31-40-250.ap-south-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-40-250.ap-south-1.compute.internal	Elastic IP addresses -
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	AWS Compute Optimizer finding <a href="#">Opt-in to AWS Compute Optimizer for recommendations.</a>   <a href="#">Learn more</a>
Auto-assigned IP address 13.126.166.91 [Public IP]	VPC ID vpc-0bcf52135a1c7e5b	

- Changes notified on Github :

github.com/Arpit-Sagar/FRESH\_vegies/tree/master

Search or jump to... Pull requests Issues Codespaces Marketplace Explore

Arpit-Sagar / FRESH\_vegies Public

Pin Unwatch 1 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master had recent pushes 1 minute ago Compare & pull request

master 2 branches 0 tags Go to file Add file <> Code

This branch is 4 commits ahead, 1 commit behind main. Contribute

Arpit-Sagar Added Changes bca8fda2 2 minutes ago 4 commits

index1.html Added Changes 2 minutes ago

Help people interested in this repository understand your project by adding a README. Add a README

About No description, website, or topics provided. 0 stars 1 watching 0 forks

Releases No releases published Create a new release

Packages No packages published Publish your first package

- Commit Changes

github.com/Arpit-Sagar/FRESH\_vegies/compare/master?expand=1

base: main compare: master

There isn't anything to compare.  
main and master are entirely different commit histories.

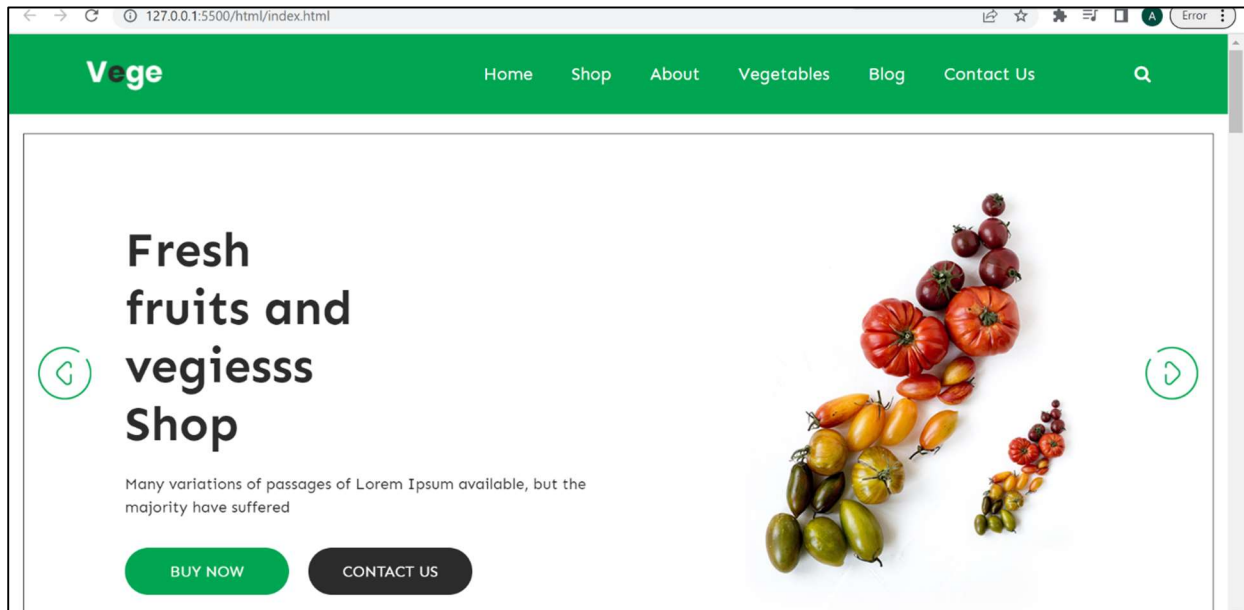
Showing 1 changed file with 1 addition and 1 deletion. Split Unified

index1.html

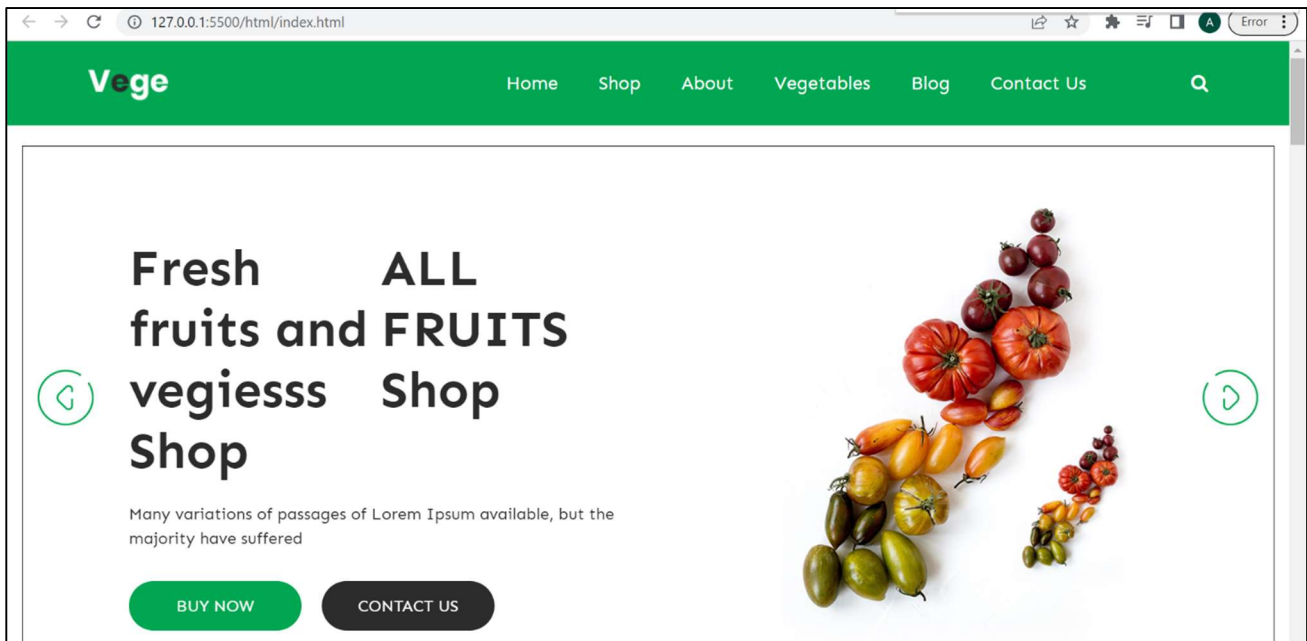
```
@@ -79,7 +79,7 @@  
79 79 <div class="row">  
80 80 <div class="col-sm-6">  
81 81 <div class="banner_taital_main">  
82 - <h1 class="banner_taital">Fresh fruits and vegiesss Shop</h1>  
82 + <h1 class="banner_taital">All FRUITS Shop</h1>  
83 83 <p class="banner_text">Many variations of passages of Lorem Ipsum available, but the majority have suffered</p>  
84 84 <div class="btn_main">  
85 85 <div class="started_text"><a href="#">Buy Now</a></div>
```



- Website:Before →



- Changes shown on website →



## **TOOLS AND TECHNOLOGY**

### **1. Jenkins:**



Jenkins will be the primary tool used to automate the continuous integration and continuous delivery pipeline. Jenkins offers a wide range of plugins that will help automate and manage the different stages of the pipeline, including building, testing, and deploying the software applications. Jenkins also offers a range of features that can help automate the different stages of the pipeline, such as pipeline scripting, job chaining, and parallelism. These features allow for the creation of complex and sophisticated pipelines that can handle different types of software applications and testing frameworks.

### **2. Git/GitHub:**



Git is a distributed version control system that is widely used for source code management. GitHub is a web-based platform that provides hosting for Git repositories and a wide range of tools to support collaboration and software development. In the context of our project, Git/GitHub can be used to manage the source code, collaborate with other developers, and track changes made to the codebase. Developers can create branches to work on new features or bug fixes, which can be merged back into the main branch once the changes are reviewed and tested. GitHub's pull request feature allows developers to review code changes and leave comments, which can be used to ensure code quality and maintain standards.

### 3. Docker:



Docker is a platform that uses containerization technology to enable developers to build, package, and deploy software applications as containers. Containers provide an isolated environment for running applications, with their own set of dependencies and runtime configurations. This makes it possible to run the same application on different environments without having to worry about the underlying infrastructure. In the context of a CI/CD pipeline, Docker can be used to build a containerized image of the application during the build process. This image can then be stored in a registry, such as Docker Hub, and pulled down during the deployment process. This ensures that the application runs consistently across different environments, from development to production.

## **CONCLUSION**

In this project, we have successfully implemented the automated CI/CD pipeline using Jenkins tool and Docker as the container for deployment.

The demonstration of this DevOps project provides an excellent example of effective collaboration amongst a team which can be achieved by automating the entire process of building, testing and deployment of a code segment wherein the need to restart the entire server is eliminated and all the changes and contributions from different team members are integrated and these changes can be viewed by a single refresh click.

Illustrating the application of parallel and distributed computing by streamlining the entire process of Software Development and providing efficient delivery, continuous monitoring and faster speed.

Thus, high quality software can be developed involving improved coordination, avoiding any downtime and meeting all quality standards.

Overall, this project has also helped us to have a better exposure of the latest technology of pipelining, Continuous Integration/Continuous Delivery using Jenkins and Docker.

In the future scenario, CI/CD has grown scope as the industry demands faster execution in deployment phase from multiple users.