A

Report

of

Industrial Training

On

CLOUD COMPUTING, DEVOPS & SRE

Submitted in partial fulfilment for the award of degree of **Bachelor of Technology**

in

Computer Science & Engineering

Submitted By

Vandan Jain
22EJCCS838

Guide

Ms. Charu Upadhyay
Assistant Professor

Department of Computer Science & Engineering

Jaipur Engineering College & Research Centre
Jaipur, Rajasthan
2024-25

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE	Jaipur Engineering College and Research Centre, Shri Ram ki Nangal, via Sitapura RIICO Jaipur- 302 022.	**Academic Year 2024-2025**

CERTIFICATE

This is to certify that the industrial training entitled "**CLOUD COMPUTING, DEVOPS & SRE"** is the bonafide work carried out by **"Vandan Jain",** student of B.Tech. in Computer Science & Engineering at Jaipur Engineering College and Research Centre, during the year 2025-24 in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering under my guidance.

Name of Guide: Ms. Charu Upadhyay

Designation: Assistant Professor

Place: Jaipur, Rajasthan

Date:

Upflairs
... because winning takes it all

CERTIFICATE
OF APPRECIATION

Proudly Presented to

Vandan Jain

B.Tech, II Sem | Department of Computer Science Engineering

from JECRC Foundation , Jaipur has successfully completed 15 days of Summer Internship & Training Program in **Python Programming** from 9th Aug 2023 to 23rd Aug 2023 organised by **Upflairs Pvt.Ltd.**

UPFLAIRS PRIVATE LIMITED

Director Technical
Upflairs Pvt. Ltd.

Enrollment No.
UF/0923/3084

UPFLAIRS PRIVATE LIMITED

Director Management
Upflairs Pvt. Ltd.

VISION OF CSE DEPARTMENT

To become a renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

we

MISSION OF CSE DEPARTMENT

1. To impart outcome-based education for emerging technologies in the field of computer science and engineering.
2. To provide opportunities for interaction between academia and industry.
3. To provide a platform for lifelong learning by accepting the change in technologies
4. To develop the aptitude of fulfilling social responsibilities.

PROGRAM OUTCOMES (POs)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialisation to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

The PEOs of the Btech (CSE) program are:

1. To produce graduates who are able to apply computer engineering knowledge to provide key IT solutions to national and international organisations.
2. To produce graduates with the necessary background and technical skills to work professionally in one or more of the areas like – IT solution design development and implementation consisting of system design, network design, software design and development, system implementation and management etc. Graduates would be able to provide solutions through logical and analytical thinking.
3. To enable graduates to design embedded systems for industrial applications.
4. To inculcate in graduates' effective communication skills and team work skills to enable them to work in a multidisciplinary environment.
5. To prepare graduates for personal and professional success with commitment to their ethical and social responsibilities.

PROGRAM SPECIFIC OUTCOMES (PSOs)

· PSO1: Ability to interpret and analyse network specific and cyber security issues in a real-world environment.

· PSO2: Ability to design and develop mobile and web-based applications under realistic constraints.

COURSE OUTCOMES (COs)

On completion of project Graduates will be able to-

- CO1: Generate the report based on the Projects carried out for demonstrating the ability to apply the knowledge of engineering field during training
- CO2: Demonstrate Competency in relevant engineering fields through problem identification, formulation and solution.

MAPPING: CO's & PO's

Subject Code	**Cos**	**Program Outcomes (POs)**											
		PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
5CS7-30 Industrial Training	**CO-1**	3	3	2	2	2	1	1	2	2	3	3	3
	CO-2	3	3	3	3	3	1	1	2	2	3	3	3

ACKNOWLEDGEMENT

It has been a great honour and privilege to undergo training at **Upflairs Pvt. Ltd., Jaipur**. I am very grateful to **Mr. Akash Gaur** for his valuable time and constructive guidance in preparing the report for training. It would not have been possible to complete this report in such a short period of time without their kind encouragement and valuable guidance.

I wish to express our deep sense of gratitude to our Guide **Ms. Charu Upadhyay**, Assistant Professor, Jaipur Engineering College and Research Centre, Jaipur for guiding us from the inception till the completion of the industrial training. We sincerely acknowledge him for giving his valuable guidance, support for literature survey, critical reviews and comments for our industrial training.

I would like to first of all express our thanks to **Mr. Arpit Agrawal,** Director of JECRC, for providing us such a great infrastructure and environment for our overall development.

I express sincere thanks to **Prof. Dr. V. K. Chandna**, Principal of JECRC, for his kind cooperation and extendible support towards the completion of our industrial training.

Words are inadequate in offering our thanks to **Dr. Vijeta Kumawat,** Deputy HOD of CSE department, for consistent encouragement and support for shaping our industrial training in the presentable form.

Also our warm thanks to **Jaipur Engineering College and Research Centre**, who provided us this opportunity to carryout, this prestigious industrial training and enhance our learning in various technical fields.

Vandan Jain

22EJCCS838

ABSTRACT

In an era marked by rapid technological advancements, our 45-day Summer Internship and Training Program in Cloud Computing, DevOps & SRE offers a comprehensive and immersive experience designed to empower participants with the knowledge and skills needed to navigate the dynamic landscape of cloud computing and devops engineering.

This program is meticulously crafted to provide a balance between theoretical foundations and practical applications. Participants will delve into fundamental concepts such as cloud servers, shell programming, continuous integration and deployment, and monitoring. Hands-on sessions using state-of-the-art tools and frameworks will complement theoretical learning, ensuring a holistic understanding of the subject matter.

The program structure is divided into modules, each focusing on a key aspect of cloud computing and devops. Participants will engage in real-world projects, fostering the development of critical problem-solving skills and the ability to apply theoretical knowledge to practical scenarios. Industry experts and seasoned professionals will lead interactive sessions, providing insights into current industry trends and best practices.

Key Highlights:

1. Comprehensive Curriculum: Covering essential topics in Cloud Computing & DevOps, including Cloud consoles, linux shell programming, process optimization, and continuous deployment.
2. Hands-on Projects: Participants will work on real-world projects, gaining practical experience and building a strong portfolio.
3. Industry Insights: Engage with experienced professionals through guest lectures, panel discussions, and networking opportunities.
4. Advanced Tools and Frameworks: Utilise cutting-edge tools and frameworks such as AWS console, Docker, Kubernetes, Ansible, Terraform and Jenkins to handle and automate backend servers.
5. Skill Assessment and Certification: Regular assessments and a final certification will validate the participants' proficiency in Cloud Computing & DevOps concepts.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1. INTRODUCTION

Cloud Computing and DevOps are revolutionising modern IT infrastructure and software delivery, enabling organisations to innovate, scale, and deploy with unparalleled agility. In today's competitive landscape, efficient and automated workflows are critical for success, minimising human error and accelerating time-to-market.

This project, titled **"End-to-End Deployment Automation Using Jenkins,"** explores the integration of **AWS, Jenkins, Docker, and GitHub** to create a seamless CI/CD pipeline. Jenkins serves as the central automation hub, orchestrating build, test, and deployment stages with minimal manual intervention. AWS provides a scalable cloud platform, while Docker containers ensure consistency across development, testing, and production environments. With source control on GitHub, this project demonstrates a robust framework for managing code, automating testing, and deploying applications rapidly and reliably.

Through this project, we demonstrate the synergy between cloud computing and DevOps tools, showcasing how automation can enhance productivity, improve software quality, and enable continuous delivery in modern development environments.

Purpose

- The purpose of the "**End-to-End Deployment Automation Using Jenkins**" project is to develop a robust CI/CD pipeline that automates the building, testing, and deployment processes within a cloud computing and DevOps environment. This project leverages AWS for scalable infrastructure, Jenkins as the primary automation server, Docker for containerization, and GitHub for version control. By integrating these tools, the system automates repetitive tasks, improves code quality, and enables faster delivery cycles.

- The CI/CD pipeline begins with source code management in GitHub, where each code change triggers Jenkins to initiate an automated build process. Jenkins fetches the updated code and builds the application in a Docker container, ensuring consistent environment configurations across all stages. This Dockerized approach reduces discrepancies between development, testing, and production, resulting in more reliable deployments.

- Upon building the application, Jenkins orchestrates automated testing to validate code functionality and quality. The results of each test cycle are logged, allowing for real-time feedback to developers. If tests pass, Jenkins progresses to the deployment phase, where it securely deploys the application to AWS. AWS provides a scalable and reliable cloud environment, ideal for hosting applications in production.

- This project demonstrates the benefits of cloud-based automation in a DevOps workflow, showcasing how end-to-end automation minimises human intervention, reduces deployment time, and enhances system reliability. Ultimately, this automated pipeline enables continuous integration and continuous delivery, ensuring that new features and improvements are deployed quickly, with reduced risk of errors in production.

Project Scope

· The project aims to showcase the benefits of cloud and DevOps automation, emphasising efficiency, consistency, and reliability in software development. The outcome will be a scalable, reusable, and easily manageable CI/CD pipeline applicable across various development projects. The primary focus areas include:

1. **Automated Build and Test Pipeline:** Setting up a fully automated pipeline in Jenkins, triggered by code updates on GitHub, to streamline build processes and ensure consistent application packaging with Docker containers. Automated testing stages will validate code quality, enabling quick feedback for developers.
2. **Cloud Deployment and Infrastructure as Code:** Deploying applications to AWS, utilising infrastructure as code (IaC) practices to manage and provision cloud resources. This allows for scalable and flexible deployments that align with DevOps best practices.
3. **Containerization for Consistent Environments:** Using Docker to create containerized environments that ensure consistency between development, testing, and production. This minimises deployment errors and supports scalability across various environments.
4. **Continuous Integration and Delivery (CI/CD):** Establishing continuous integration and delivery pipelines to automate repetitive tasks, reducing human intervention, and accelerating deployment times. This ensures a reliable and agile approach to releasing software updates.

Document Convention

When you read this manual, certain words are represented in different fonts, typefaces, sizes, and weights. This highlighting is systematic; different words are represented in the same style to indicate their inclusion in a specific category. The types of words that are represented this way include the following:

· One and a half spacing is used for typing the general text. The general text is justified and typed in the Font style 'Times New Roman' and Font size 12.

· Subheading is typed in the Font style 'Times New Roman' and Font size 12 and bold. · Heading is in the Font style 'Times New Roman' and Font size 14 and bold.

2. REQUIREMENT ANALYSIS

Hardware Requirements

The hardware requirements for the "End-to-End Deployment Automation Using Jenkins" project are largely cloud-based, as the system utilises AWS infrastructure. Using an AWS t2.micro instance with an Ubuntu image, this project leverages cloud resources to perform automated CI/CD tasks, minimising the need for extensive on-premises hardware. Below are the specific hardware requirements:

1. **AWS t2.micro Instance:**

- The t2.micro instance provides 1 vCPU and 1 GB of memory, suitable for lightweight automation tasks in DevOps pipelines.

- The instance is part of AWS's free tier, making it cost-effective for development and testing environments.

- For more demanding builds, testing, or deployments, instances with larger vCPU and memory capacities can be provisioned.

2. **Virtualized CPU (vCPU):**

- For basic CI/CD tasks, the 1 vCPU in the t2.micro instance is adequate.

- Scaling to larger instances with higher vCPU counts (e.g., t2.medium or m5.large) can improve performance for more complex automation workflows.

3. **Memory (RAM):**

- The 1 GB RAM in t2.micro is sufficient for lightweight Jenkins automation, Docker operations, and handling small to moderate pipelines.

- For intensive builds or parallel jobs, instances with more RAM may be required (e.g., t2.small or larger).

4. **Storage:**

- Elastic Block Store (EBS) is used for instance storage, with 8 GB allocated for the OS and software packages.

- SSD-backed EBS volumes provide fast access times, enhancing build and deployment speed.

5. **Networking:**

- High-speed networking on AWS supports efficient data transfer, critical for pushing/pulling code from GitHub and deploying to different environments.

- AWS Virtual Private Cloud (VPC) provides a secure, isolated network environment with options for configuring firewalls, security groups, and access controls.

6. **Docker Support**:

- Docker is installed on the Ubuntu image, allowing for containerized environments that ensure consistency across all stages of the CI/CD pipeline.

- The t2.micro instance supports Docker-based builds, though larger instances may be beneficial for complex or concurrent Docker tasks.

7. **Power and Cooling:**

- As AWS infrastructure is used, power and cooling requirements are managed by AWS, alleviating the need for dedicated on-site resources.

8. **Optional Scaling:**

- For larger workloads, AWS enables easy scaling by switching instance types or using auto-scaling groups.

Software Requirements

The software requirements for the project focus on enabling continuous integration and continuous deployment (CI/CD) workflows using cloud computing and DevOps tools. The primary components include Jenkins, Docker, AWS CLI, and Git for version control. Here are the key software requirements:

1. Operating System:

- The project is hosted on an AWS t2.micro Ubuntu instance. Ubuntu is chosen for its stability, compatibility with Jenkins, and ease of integration with DevOps tools.

- Ubuntu 20.04 LTS or higher is recommended for security updates and package compatibility.

2. Jenkins:

- Jenkins serves as the core automation server for managing the CI/CD pipeline.

- Install the latest stable version of Jenkins and configure it with required plugins for Git, Docker, and AWS integration.

- Plugins include: Git Plugin, Docker Plugin, Pipeline Plugin, and AWS CLI Plugin.

3. Docker:

- Docker enables containerization of applications, ensuring consistent environments across development, testing, and production stages.

- Install Docker and configure it on the Ubuntu instance to allow Jenkins to execute Docker commands for building and deploying containerized applications.

4. Git:

- Git is used for version control, tracking changes in code, and facilitating collaboration.

5. AWS Command Line Interface (CLI):

- AWS CLI is essential for interacting with AWS services directly from the command line, enabling Jenkins to deploy applications to the cloud.

- Configure AWS CLI with necessary IAM permissions to automate deployments, manage AWS resources, and integrate with other AWS services as needed.

6. Jenkins Pipeline Plugins:

- Jenkins Pipeline plugins allow defining build, test, and deployment workflows as code.

- Declarative Pipeline syntax is recommended for readability and ease of maintenance.

7. Build Automation Tools (Optional):

- Maven or Gradle may be used if the project involves Java-based applications, facilitating automated builds and dependency management.

8. Container Registry (Optional):

- For storing Docker images, consider integrating with a container registry such as Docker Hub or Amazon ECR (Elastic Container Registry) for secure storage and easy retrieval of images during deployment.

9. Monitoring and Logging Tools:

- CloudWatch or Prometheus (if needed) can be configured to monitor the performance and health of the CI/CD pipeline and AWS deployments.

10. Network and Security Tools:

- Security configurations include IAM roles and policies for granting permissions, VPC

Security Groups for managing access, and SSL/TLS certificates for secure data transmission if deploying web applications.

11. Version Control:

- GitHub or any Git-based repository is used to manage source code and track changes, integrating with Jenkins to enable triggered builds upon commits.

Functional Requirements

The functional requirements for the End-to-End Deployment Automation Using Jenkins project define the core capabilities and operations necessary to implement a fully automated CI/CD pipeline. These requirements ensure that the system meets its objectives of automating, streamlining, and securing the development and deployment processes within a DevOps environment.

1. Automated Build Triggering:

- The system must automatically trigger builds in Jenkins whenever there are code changes in the GitHub repository.
- It should support both scheduled and event-based triggers for flexibility in the build process.

2. Code Compilation and Dependency Management:

- The pipeline should compile code and manage dependencies where applicable. This may involve using tools like Maven or Gradle for Java projects.
- The system should verify that all dependencies are correctly installed and up-to-date.

3. Automated Testing and Validation:

- Jenkins should execute automated tests as part of the CI process to ensure code quality. Tests may include unit, integration, and end-to-end tests, depending on project requirements.

4. Containerization of Applications:

- The pipeline should containerize the application using Docker, creating Docker images that encapsulate the application and its dependencies.
- Docker images should be stored in a container registry (such as Docker Hub or AWS ECR) for easy access during deployment.

5. Automated Deployment to AWS:

- The system should support automated deployment of the application to AWS upon successful testing, utilising Jenkins' integration with AWS CLI.
- Deployment configurations should be adaptable, supporting different environments such as development, testing, and production.

6. Version Control and Rollback:

- The system must integrate with Git to maintain version control over all code changes.
- It should allow rollback to previous versions if issues are detected in the latest deployment, minimising downtime.

7. Pipeline Visualization and Status Reports:

- Jenkins should provide a visual representation of the CI/CD pipeline, showing each state's progress and outcome in real-time.
- It should generate reports summarising build, test, and deployment statuses for team members.

Non-Functional Requirements

1. Performance:

- The CI/CD pipeline should execute build, test, and deployment stages efficiently, with minimal latency, even as workloads increase.

2. Security:

- Secure access controls for Jenkins and AWS services, including role-based permissions and secure authentication methods for users.

3. Scalability:

- The system must be scalable, supporting additional instances or resources as workload demands increase, especially for large-scale deployments.

4. Reliability:

- Ensure high availability of Jenkins and the AWS infrastructure to avoid pipeline interruptions, with automated failure recovery mechanisms.

5. Usability:

- Provide a clear and accessible Jenkins dashboard with intuitive visualisations of the CI/CD pipeline stages for ease of use.

6. Maintainability:

- Follow best practices for pipeline scripts and configurations, ensuring modular, well-documented, and easily updatable code.

7. Cost Efficiency:

- Optimise AWS resource utilisation to minimise costs, with a focus on using cost-effective instance types and efficient storage solutions.

8. Monitoring and Logging:

- Implement logging for each pipeline stage and integrate monitoring tools like AWS CloudWatch to ensure system health and quickly detect issues.

9. Compliance:

- Adhere to industry standards for data protection and DevOps best practices to maintain compliance with relevant security and operational guidelines.

These requirements ensure the CI/CD pipeline is efficient, secure, and maintainable, supporting both operational and quality standards.

User classes and characteristics

1. DevOps Engineers:

- Characteristics: Skilled in continuous integration and continuous deployment (CI/CD) principles, infrastructure automation, and scripting languages. They set up, monitor, and troubleshoot the Jenkins pipeline, Docker configurations, and AWS deployments.

2. System Administrators:

- Characteristics: Experienced in server management and network configurations. Responsible for maintaining the health and security of the Jenkins server and AWS infrastructure, including permissions, backups, and resource management.

3. Project Managers:

- Characteristics: Oversee the project lifecycle, coordinate between teams, and ensure that CI/CD pipeline updates align with project timelines. They may require high-level status reports on the build and deployment stages.

4. Quality Assurance (QA) Engineers:

- Characteristics: Knowledgeable about automated testing tools and frameworks. They review and validate the testing stages in Jenkins, ensuring that code meets quality standards before deployment.

5. Security Engineers:

- Characteristics: Focused on safeguarding the CI/CD environment. They assess the pipeline for security vulnerabilities, enforce access controls, and monitor for compliance with security policies.

7. End Users (Non-Technical Stakeholders):

- Characteristics: Require insights into the project's progress without technical details. They use reports generated by Jenkins to understand build success, deployment status, and overall project health.

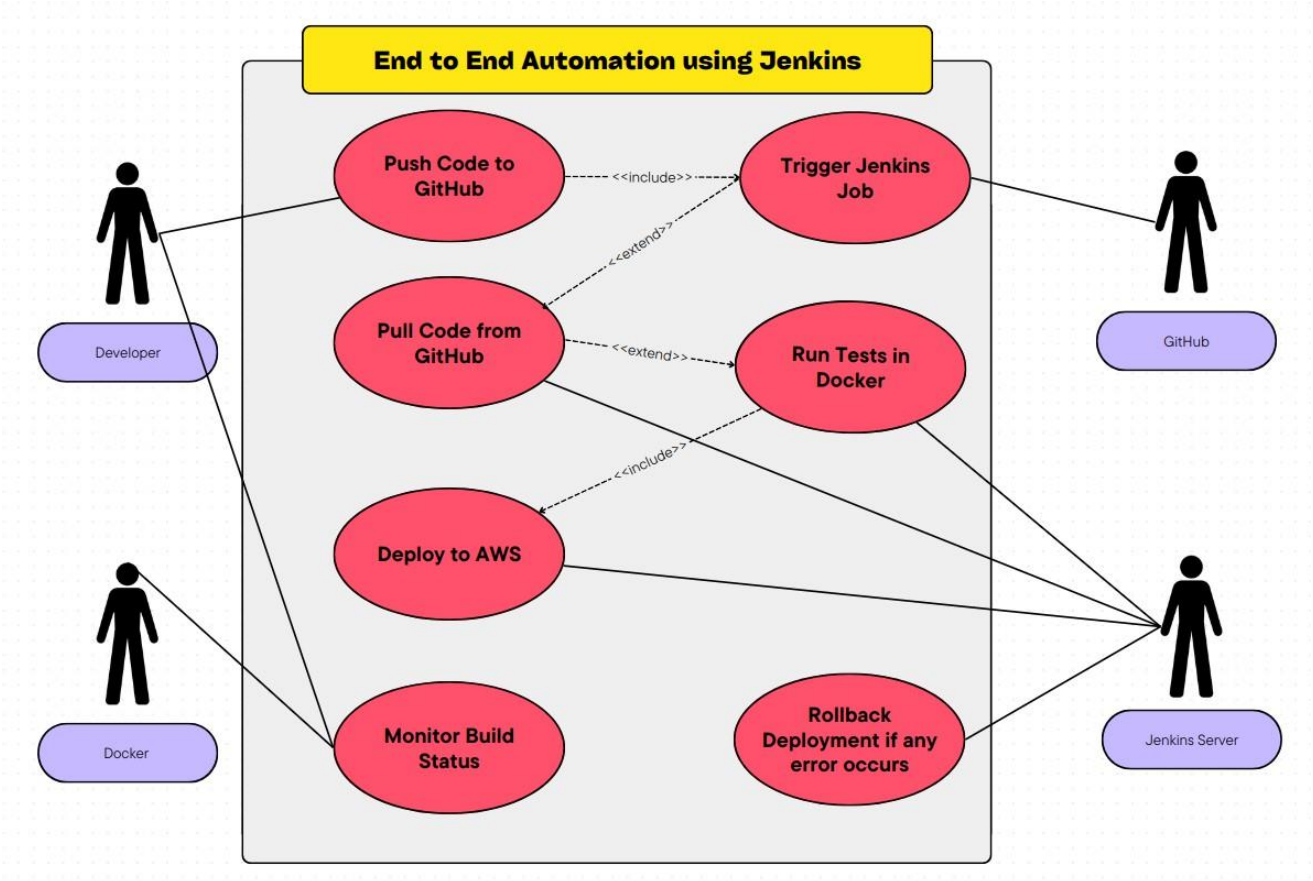
3. SYSTEM DESIGN

Use Case diagram

In the Unified Modeling Language (UML), a use case diagram can summarise the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialised symbols and connectors. An effective use case diagram can help in

- Scenarios in which your system or application interacts with people, organisations, or external systems.
- Goals that your system or application helps those entities (known as actors) achieve.
- The scope of your system.

Use-case diagram for End to End Deployment Automation using Jenkins:



Overview of Architecture

The project architecture is designed as a CI/CD pipeline using an AWS EC2 instance as the main compute environment. Jenkins, running on the EC2 instance, orchestrates the automation of the pipeline, Docker provides containerization for application deployment, and GitHub serves as the version control and source code repository.

- GitHub: GitHub functions as the source control for the project, storing the code and triggering Jenkins jobs when updates are pushed.
- Jenkins: Jenkins orchestrates the pipeline on the EC2 instance, initiating build and deployment steps as code changes are detected in the GitHub repository.
- Docker: Docker is responsible for building and running images of the application code, enabling containerized deployment.
- AWS EC2: AWS EC2 acts as the host environment, where Jenkins and Docker are installed, serving as the build server for continuous integration and deployment.

Workflow and Data Flow

The project workflow is structured to ensure an automated build and deployment process:

1. Code Commit: Developers push code changes to the GitHub repository.
2. Job Trigger: Jenkins, configured with a webhook to monitor the GitHub repository, detects the change and initiates the CI/CD job.
3. Build and Containerization: Within the Jenkins job, a shell command executes `docker build` and `docker run` to build the Docker image and run the application in a container.
4. Deployment: The application, once containerized by Docker, is deployed on the EC2 instance and accessible for use.

Security Layer with RSA Keys: Secure RSA keys establish an authenticated connection between the EC2 instance, Jenkins, and GitHub, ensuring secure and authorised automation processes.

Detailed Component Design

- Jenkins Configuration: Jenkins, running on the EC2 instance, is configured with a job that links directly to the GitHub repository. This job includes build steps that automate Docker commands to create and deploy the application.

- Docker: Docker enables isolated and consistent application environments. The use of `docker build` and `docker run` commands encapsulates the application within containers, making it easier to deploy and manage across different environments.

- AWS EC2 Instance: The EC2 instance serves as the main environment where Jenkins and Docker operate. This setup can scale as needed by either upgrading the instance type (vertical scaling) or adding additional EC2 instances to distribute workloads (horizontal scaling).

Scalability and Reliability Considerations

- Scalability: The current setup on an EC2 instance allows for vertical scaling by increasing instance size as needed. To handle larger workloads, horizontal scaling can be implemented by adding more EC2 instances, where Jenkins agents on separate instances could handle large-scale or parallel builds.

- Reliability: Reliability is enhanced by running Jenkins and Docker on a stable AWS environment. AWS CloudWatch can be used to monitor instance health, while automated backups and snapshots help maintain system continuity.

Sequence Diagram/Activity Diagram

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focused and they show the order of the interaction visually by using the vertical axis of the diagram to represent time, what messages are sent and when.

Sequence Diagrams captures:

· the interaction that takes place in a collaboration that either realises a use case or an operation (instance diagrams or generic diagrams).

· high-level interactions between users of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams).

Purpose of Sequence Diagram:

· Model high-level interaction between active objects in a system.

· Model the interaction between object instances within a collaboration that realises a use case.

· Model the interaction between objects within a collaboration that realises an operation.

Sequence Diagrams are organised according to object (horizontally) and time (vertically):

· **Object Dimension-** The horizontal axis shows the elements that are involved in the interaction. Conventionally, the objects involved in the operation are listed from left to right according to when they take part in the message sequence. However, the elements on the horizontal axis may appear in any order.

· **Time Dimension**- The vertical axis represents time proceedings (or progressing) down the page.

Deployment Diagram

A UML deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Deployment diagrams is a kind of structure diagram used in modelling the physical aspects of an object-oriented system. They are often used to model the static deployment view of a system (topology of the hardware).

When to Use Deployment Diagram

· What existing systems will the newly added system need to interact or integrate with?

· How robust does a system need to be (e.g., redundant hardware in case of a system failure)?

· What and who will connect to or interact with the system, and how will they do it?

· What middleware, including the operating system and communications approaches and protocols, will the system use?

· What hardware and software will users directly interact with (PCs, network computers, browsers, etc.)?

· How will you monitor the system once deployed?

· How secure does the system need to be (needs a firewall, physically secure hardware, etc.)?

Purpose of Deployment Diagrams

- They show the structure of the run-time system.
- They capture the hardware that will be used to implement the system and the links between different items of hardware.
- They model physical hardware elements and the communication paths between them.
- They can be used to plan the architecture of a system.
- They are also useful for Document the deployment of software components or nodes.

Deployment Diagram at a Glance

Deployment diagrams are important for visualising, specifying, and documenting embedded, client/server, and distributed systems and also for managing executable systems through forward and reverse engineering.

A deployment diagram is just a special kind of class diagram, which focuses on a system's nodes. Graphically, a deployment diagram is a collection of vertices and arcs. Deployment diagrams commonly contain:

Nodes

· 3-D box represents a node, either software or hardware

· HW node can be signified with <<stereotype>>

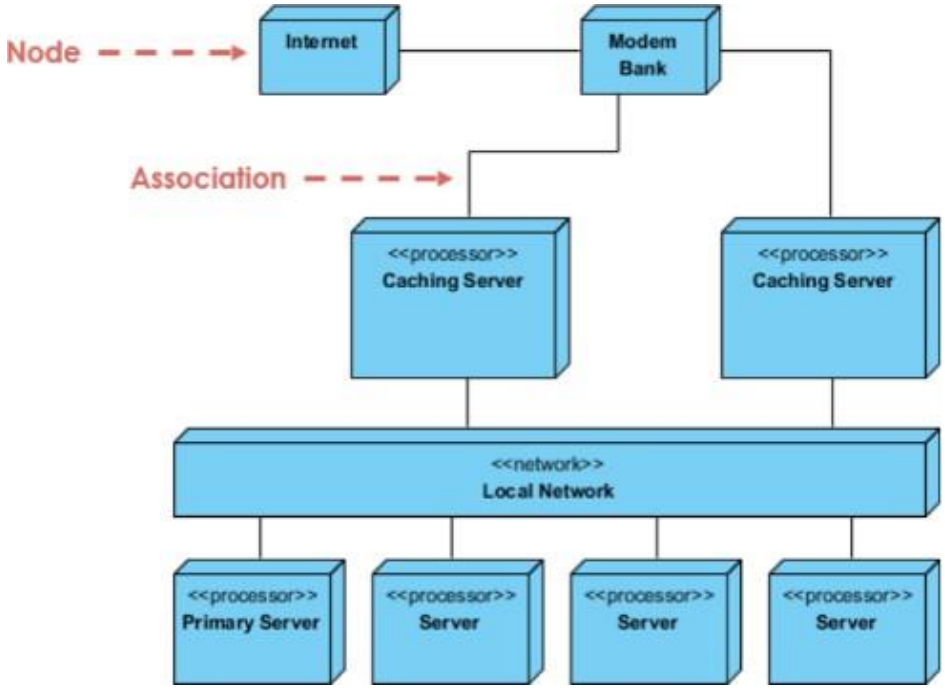
· Nodes can reside within a node

Other Notations

· Dependency

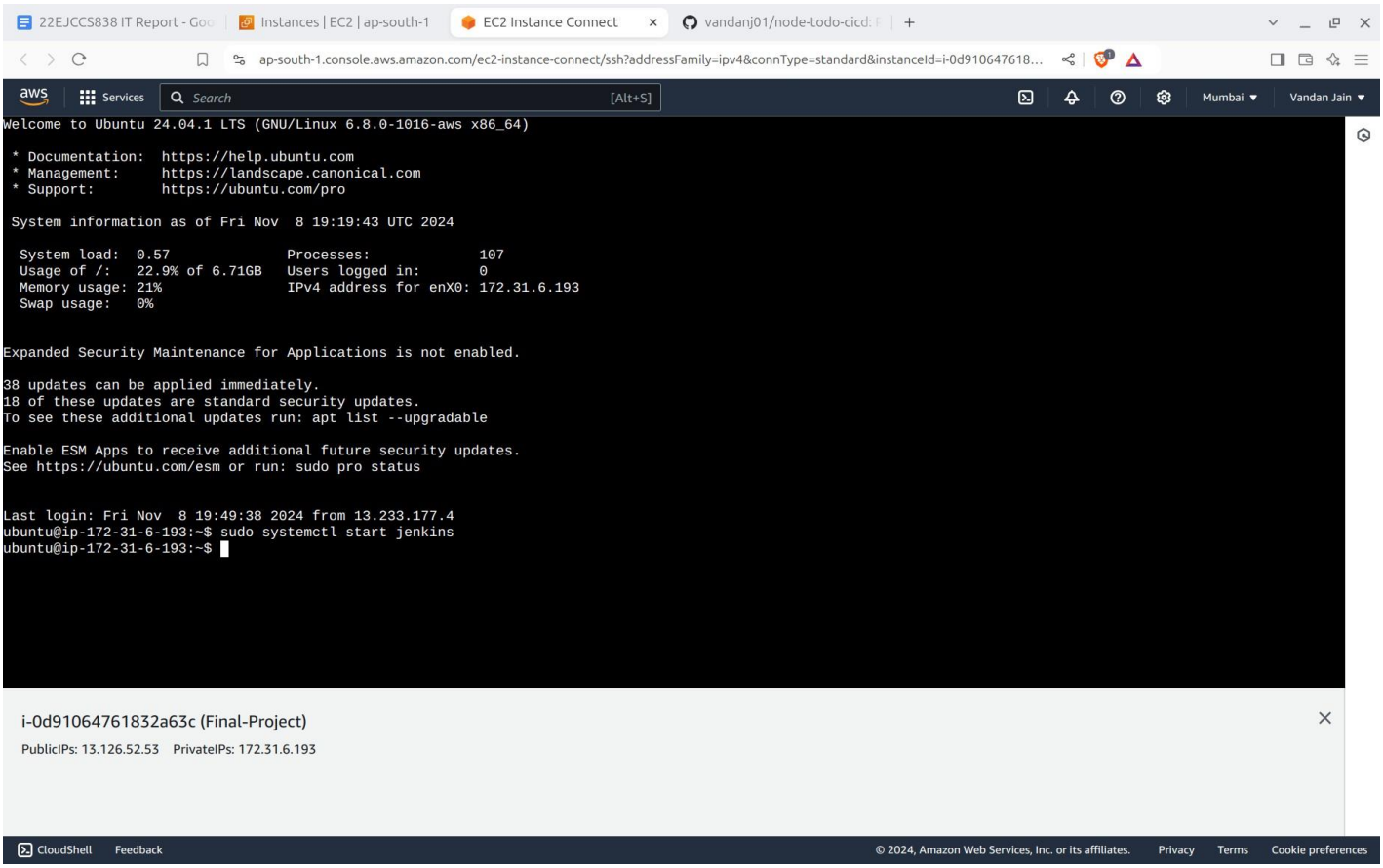
· Association relationships.

· May also contain notes and constraints.

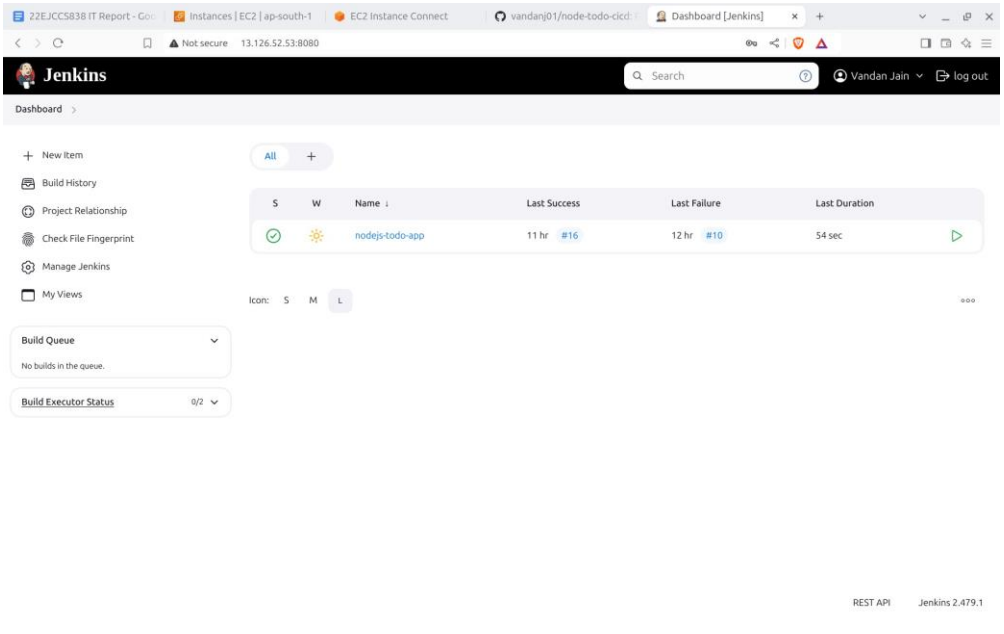

Node
Internet
Modem Bank
Association
<<processor>>
Caching Server
<<processor>>
Caching Server
<<network>>
Local Network
<<processor>>
Primary Server
<<processor>>
Server
<<processor>>
Server
<<processor>>
Server

4. SCREENSHOTS OF WORKING

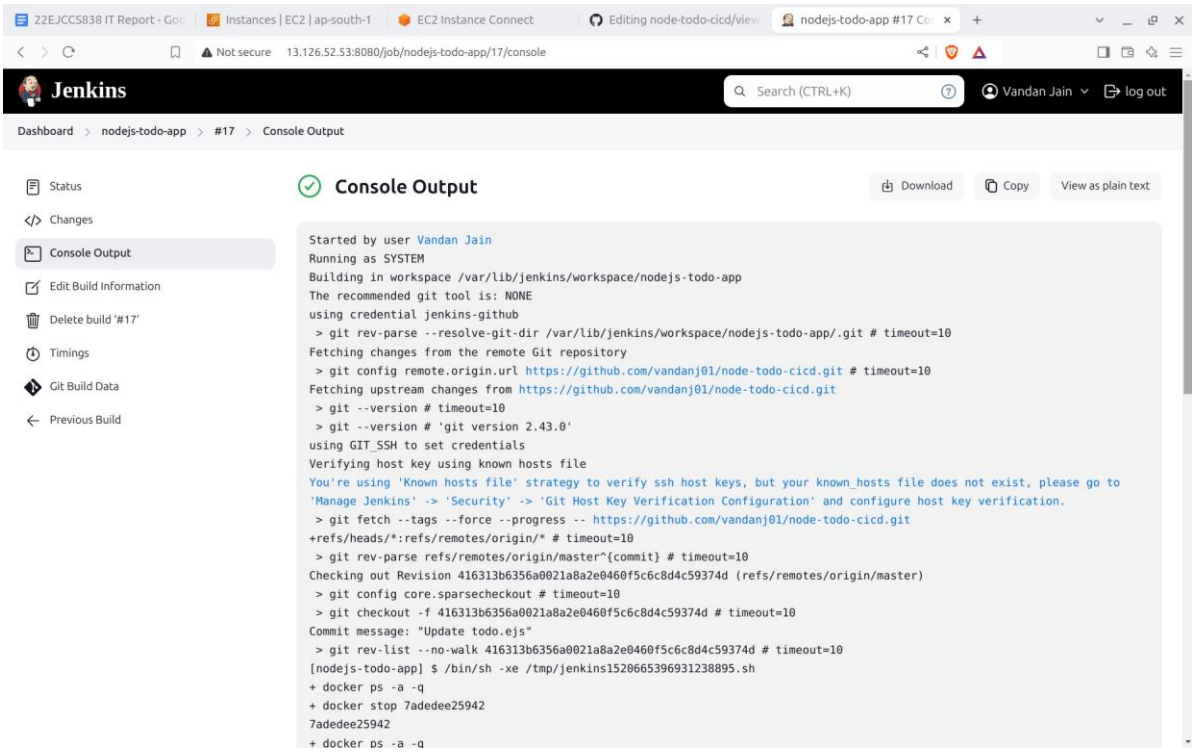
- **Start Jenkins from EC2 instance**



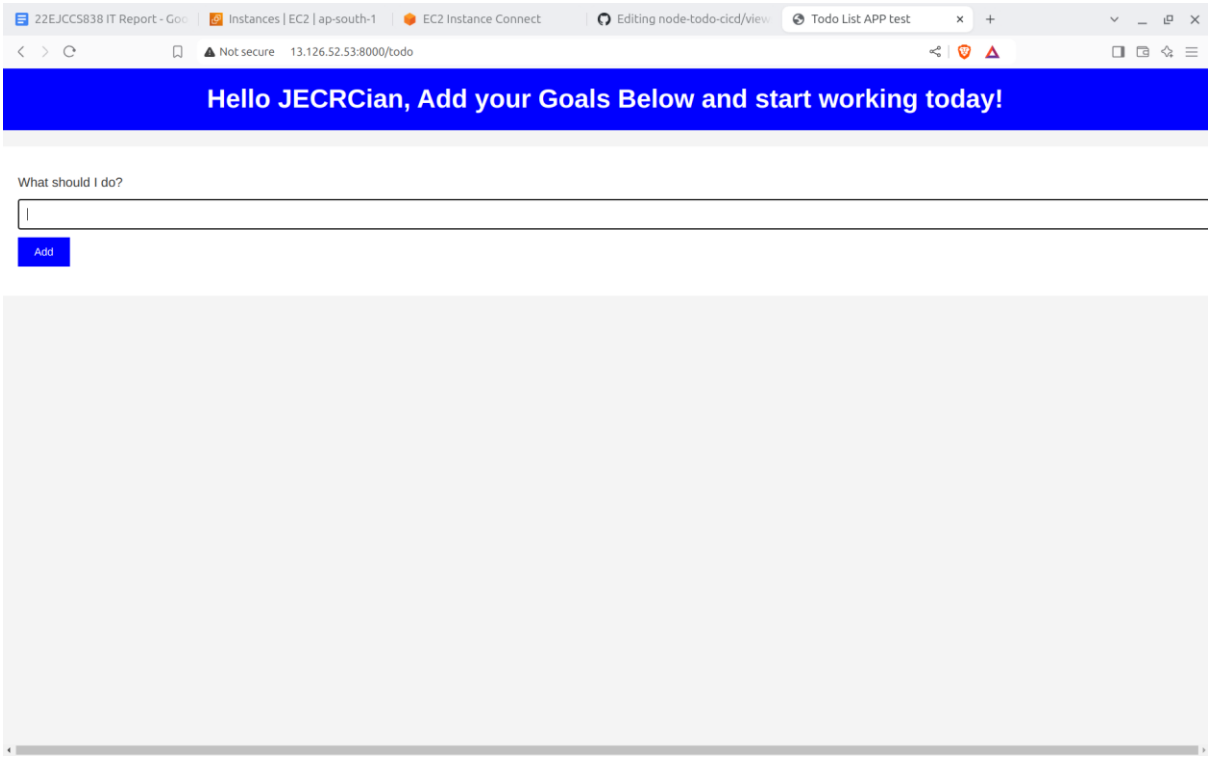
- **Jenkins Dashboard**



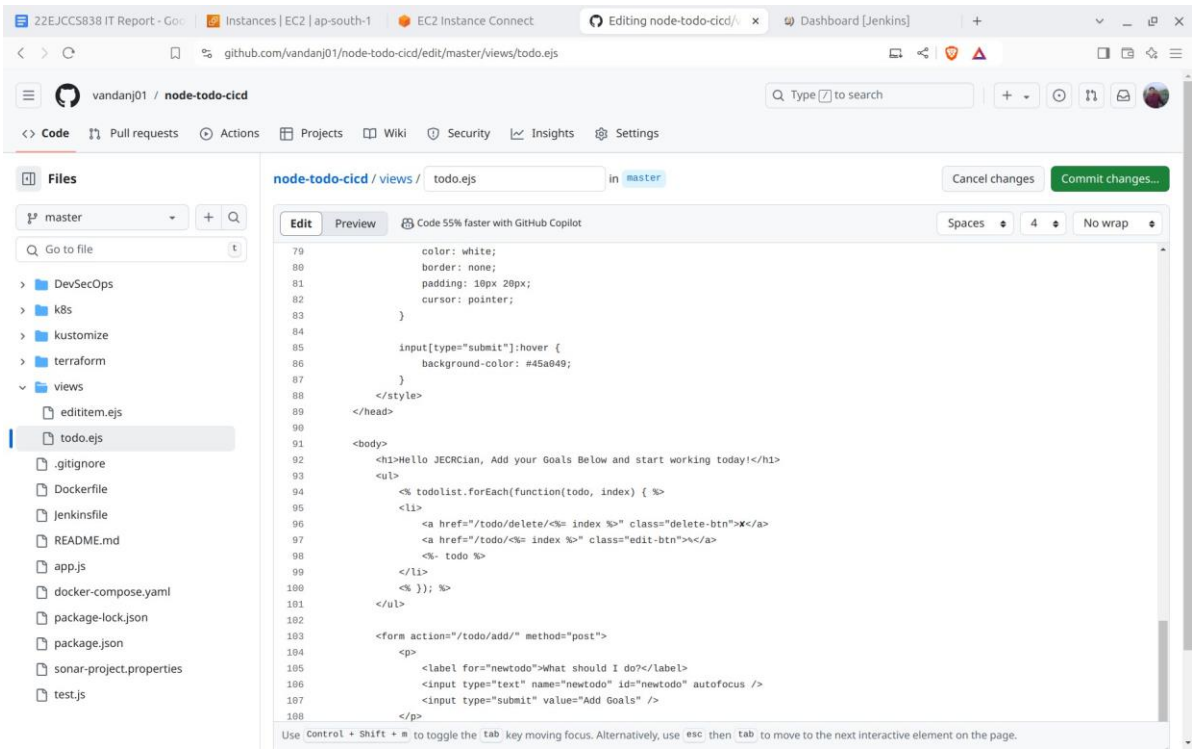
- **Successful Manual Building of a job**



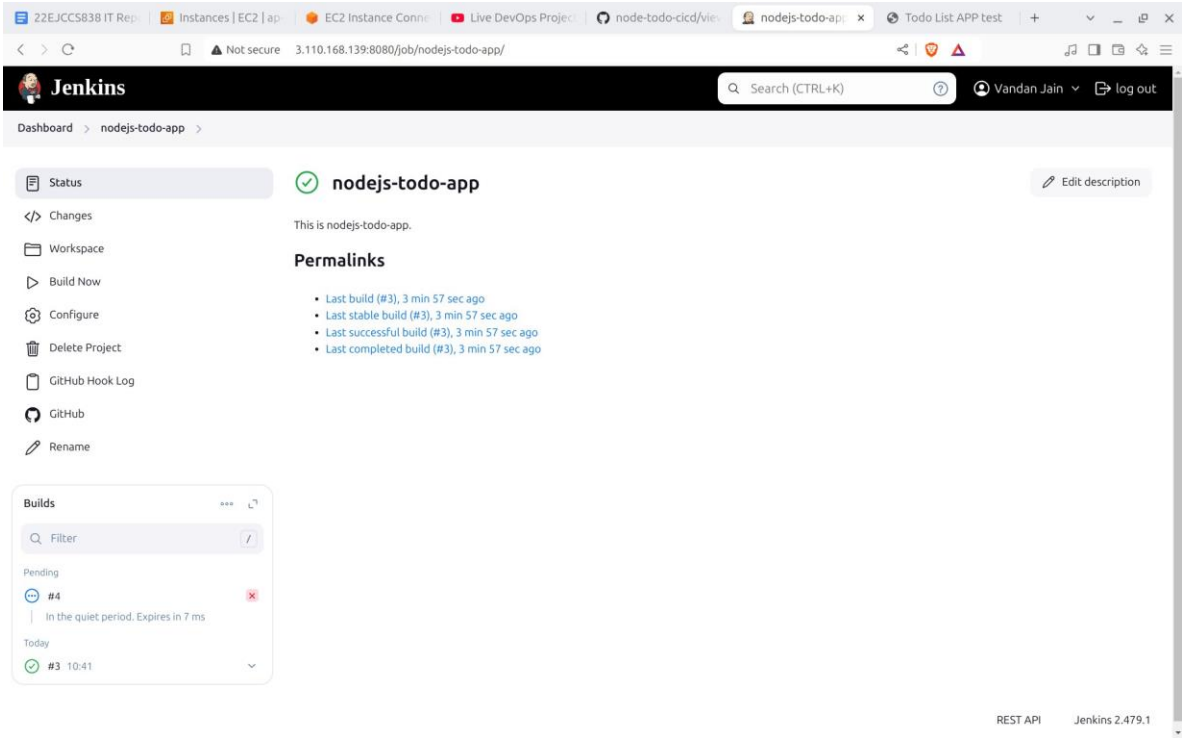
- **The Node.js application @ port 8000**



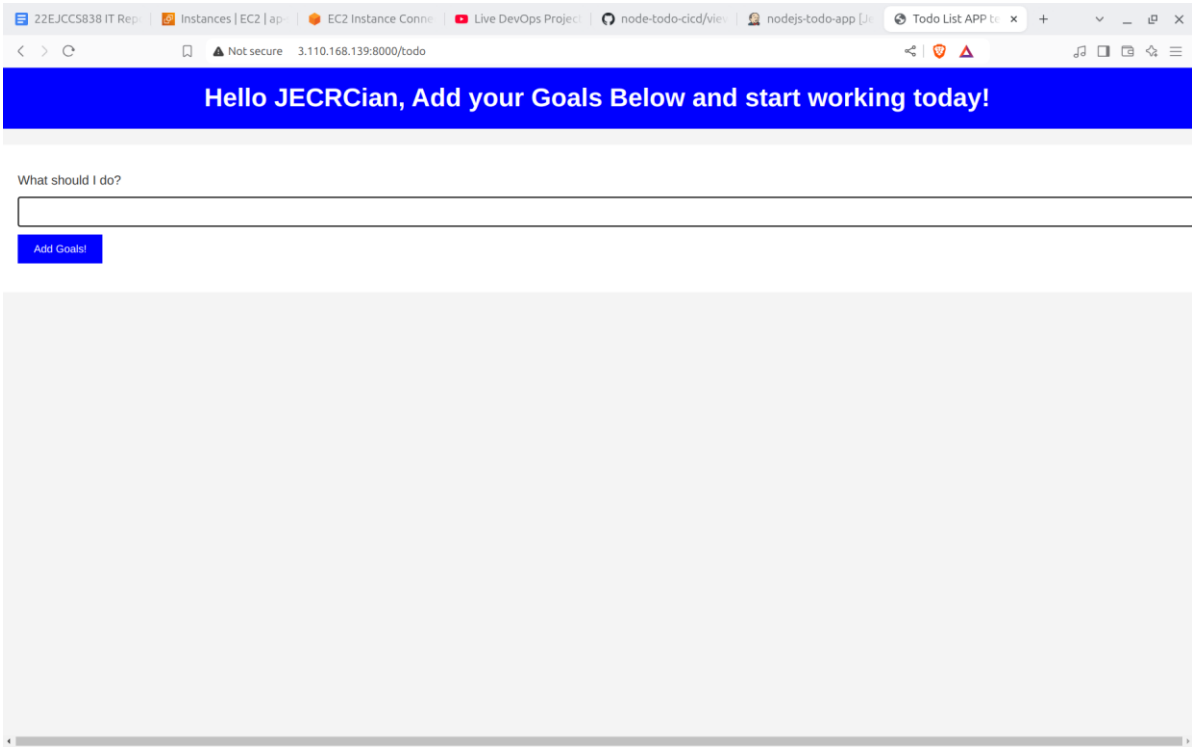
- **Committing changes to the GitHub repo**



- **Automatic Integration and Deployment at the Jenkins server**



- **Updated Website Application (Final Output)**



5. CONCLUSION

The **"End-to-End Deployment Automation Using Jenkins"** project successfully demonstrates the power of integrating cloud computing and DevOps tools to streamline and automate software deployment. By leveraging AWS for scalable infrastructure, Jenkins for CI/CD, Docker for containerization, and GitHub for version control, this project showcases a robust pipeline that automates the build, test, and deployment processes efficiently.

The project follows a structured workflow where Jenkins, configured on an EC2 instance, pulls code from GitHub, builds a Docker image, and runs the container automatically. This setup reduces manual intervention, minimises human error, and allows for faster, more consistent deployments. The use of RSA keys to enable secure communication between AWS, GitHub, and Jenkins ensures a smooth and secure automation pipeline.

Overall, this project highlights the advantages of using DevOps principles and cloud-based solutions for continuous integration and delivery, providing a foundation that can scale with future development needs. It serves as a practical example of how automation can enhance productivity, reduce downtime, and support agile practices in modern software development.

6. REFERENCES

- Upflairs Pvt. Ltd.
- jenkins.io
- stackoverflow.com
- youtube.com
- hub.docker.com