

# **Scientific Calculator with DevOps**

## **CS 816: Software Production Engineering**

*Guided By*

**B. Thangaraju**

**Professor**

**International Institute of Information Technology, Bangalore**

*Under the Assistance of*

**Jacob Mathew**

**Teaching Assistant**



**International Institute of Information Technology,  
Bangalore**

*Created by:*

**Vraj Jatin Naik**

**MT2023050**

**Vraj.Naik@iiitb.ac.in**

# ABSTRACT

This project centers on the development of a Java-based scientific calculator program, providing users with a comprehensive set of mathematical operations. Users can perform basic arithmetic operations such as addition, subtraction, multiplication, and division, alongside advanced functionalities including factorial calculation, exponentiation, and square root computation. Additionally, the calculator offers trigonometric operations such as sine, cosine, and tangent, enabling users to solve complex mathematical problems efficiently.

Throughout the software development lifecycle, robust DevOps practices are implemented to ensure efficiency and reliability. Git is utilized for version control, facilitating collaborative development and code management. Maven serves as the build tool, streamlining dependency management and project configuration.

Continuous Integration is achieved through Jenkins, automating the integration of code changes and executing comprehensive test suites to maintain code quality and reliability. Docker is employed for containerization, allowing for the encapsulation of the calculator program and its dependencies into portable containers.

Deployment on local hosts is orchestrated using Ansible, ensuring seamless configuration management and deployment processes. This project exemplifies the effective integration of DevOps principles in Java software development, providing users with a versatile and robust scientific calculator while streamlining development, testing, and deployment workflows.

# TABLE OF CONTENTS

ABSTRACT.....	I
TABLE OF CONTENTS.....	II
LIST OF FIGURES .....	III
<b>1. PROBLEM STATEMENT .....</b>	<b>1</b>
<b>2. DEVOPS INTRODUCTION .....</b>	<b>2</b>
2.1 WHAT IS DEVOPS?.....	2
2.2 NEED OF DEVOPS:.....	3
2.3 DEVOPS LIFE-CYCLE: .....	5
2.4 7 CS OF DEVOPS: .....	7
<b>3. DEVOPS TOOL CHAIN .....</b>	<b>11</b>
3.2.2 AUTOMATION PLUGINS OR TOOLS: .....	14
<b>4. EXECUTION FLOW .....</b>	<b>15</b>
<b>5. IMPLEMENTATION STEPS.....</b>	<b>16</b>
5.1 SOURCE CODE MANAGEMENT WITH GITHUB.....	16
5.2 CODE DEVELOPMENT WITH INTELLIJ .....	18
5.3 SOFTWARE BUILDING AND TESTING WITH MAVEN AND JUNIT ..	22
5.4 AUTOMATION WITH JENKINS AND PIPELINE .....	27
5.5 CONTAINERIZATION WITH DOCKER .....	36
5.6 CONTINUOUS DEPLOYMENT WITH ANSIBLE .....	38
5.7 EVENT TRIGGERED AUTOMATION .....	40
<b>6. CONCLUSION .....</b>	<b>43</b>
<b>7. REFERENCES .....</b>	<b>44</b>

## LIST OF FIGURES

Fig 2.1.1	DevOps Myths.....	2
Fig 2.2.1	Wall of Confusion.....	3
Fig 2.2.2	Development and Operation Wall of Confusion.....	4
Fig 2.2.3	DevOps Benefits.....	4
Fig 2.3.1	DevOps Life-Cycle.....	5
Fig 2.4.1	DevOps Steps.....	8
Fig 2.4.2	DevOps 7Cs.....	10
Fig 2.4.3	DevOps Tool chain.....	10
Fig 5.1.1	GitHub new Repository.....	16
Fig 5.1.2	Repository Page.....	17
Fig 5.1.3	Commit History.....	18
Fig 5.2.1	IntelliJ Project Creation - I.....	18
Fig 5.2.2	IntelliJ Project Creation - II.....	19
Fig 5.2.3	Project Directory Structure.....	21
Fig 5.3.1	Dependency Addition.....	22
Fig 5.3.2	JUnit Dependency Insertion.....	23
Fig 5.3.3	Java class libraries and Annotations.....	23
Fig 5.3.4	Factorial Function.....	24
Fig 5.3.5	Factorial Test Case.....	24
Fig 5.3.6	Clean Project using maven.....	25
Fig 5.3.7	Running CaculatorTest.....	26
Fig 5.3.8	Running Project in IntelliJ.....	26
Fig 5.4.1	Ansible Installation.....	29
Fig 5.4.2	Maven Installation.....	29
Fig 5.4.3	Docker Installation.....	29
Fig 5.4.4	Pipeline plugins Installation.....	30
Fig 5.4.5	Docker plugins Installation.....	30
Fig 5.4.6	GitHub plugins Installation.....	31
Fig 5.4.7	Ansible plugin Installation.....	31
Fig 5.4.8	Adding Admin Email Address.....	32
Fig 5.4.9	Adding GitHub Server.....	32
Fig 5.4.10	Selecting required Credentials.....	33
Fig 5.4.11	Running Pipeline in Jenkins.....	34
Fig 5.4.12	Jenkins Pipeline Script.....	35
Fig 5.5.1	Docker File.....	37
Fig 5.5.2	Containers on the System.....	37
Fig 5.6.1	deply.yml File.....	39
Fig 5.6.2	Inventory File.....	39
Fig 5.7.1	ngrok payload URL.....	40
Fig 5.7.2	GitHub Webhook.....	41
Fig 5.7.3	Add Webhook.....	41
Fig 5.7.4	Jenkins Configuration.....	42
Fig 5.7.5	Jenkins GitHub Trigger.....	42

# 1. PROBLEM STATEMENT

Develop a scientific calculator program with user menu-driven operations, including:

- square root function ( $\sqrt{x}$ )
- factorial function ( $x!$ )
- natural logarithm (base e) -  $\ln(x)$ , and
- power function ( $x^y$ ).

Follow the steps below to implement the calculator using a DevOps pipeline:

1. **Source Control Management:** Choose a source control management tool such as GitHub, GitLab, BitBucket, etc., to manage the project's source code.
2. **Testing:** Utilize testing frameworks like JUnit, Selenium, PyUnit, etc., to validate the correctness and reliability of the calculator program.
3. **Build:** Employ build automation tools like Maven, Gradle, Ant, etc., to compile the source code and generate executable artifacts.
4. **Continuous Integration (CI):** Implement Continuous Integration practices using tools like Jenkins, GitLab CLI, Travis CLI, etc. Configure CI pipelines to automatically integrate code changes and run tests upon each commit.
5. **Containerization:** Containerize the calculator program using Docker to ensure consistency and portability across different environments.
6. **Image Repository:** Push the Docker image containing the calculator program to Docker Hub or a similar container registry for versioning and distribution.
7. **Deployment:** Utilize configuration management and deployment tools like Chef, Puppet, Ansible, Rundeck, etc., to orchestrate the deployment process. This includes pulling the Docker image and running it on managed hosts.
8. **Target Environments:** Choose the appropriate deployment environment based on the requirements and infrastructure setup. Options include local machines, Kubernetes clusters, OpenStack cloud, Amazon AWS, Google Cloud, or other third-party clouds.

## 2. DEVOPS INTRODUCTION

### 2.1 What is DevOps?

The emergence of DevOps, a portmanteau of "development" and "operations," in 2009 by Patrick Debois marked a pivotal moment in the realm of software engineering. DevOps signifies a paradigm shift in how software is developed, deployed, and maintained, emphasizing collaboration, communication, and integration between development and operations teams.



*Fig. 2.1.1 DevOps Myths*

Historically, software development and operations were often siloed, leading to disjointed workflows, fragmented processes, and prolonged release cycles. This separation fostered a knowledge gap between teams and hindered the seamless delivery of software products. However, DevOps aims to break down these barriers, fostering a culture of cross-functional collaboration, shared responsibilities, and continuous improvement.

At its core, DevOps embodies a set of practices, principles, and cultural norms that promote agility, efficiency, and reliability in software delivery. It encourages organizations to adopt agile methodologies, lean principles, and automation tools to streamline development workflows, accelerate deployment cycles, and enhance overall productivity.

DevOps is not merely a methodology or a set of tools; it is a cultural transformation that transcends organizational boundaries. It encourages teams to work collaboratively towards common goals, share knowledge and expertise, and take ownership of the entire software lifecycle. By fostering a culture of collaboration and continuous learning, DevOps empowers teams to deliver high-quality software products faster and more efficiently than ever before.

This report delves into the fundamental principles of DevOps, exploring its origins, evolution, and impact on modern software development practices. Through case studies, best practices, and real-world examples, we will examine how organizations can leverage DevOps to drive innovation, improve operational efficiency, and deliver value to customers at scale. Additionally, we will explore the role of automation, continuous integration, and continuous delivery in enabling DevOps practices and driving organizational success.

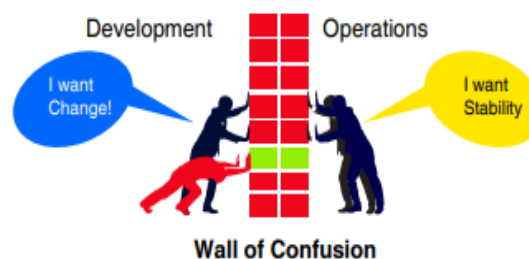
By embracing the principles of DevOps, organizations can unlock new levels of agility, resilience, and competitiveness in today's rapidly evolving digital landscape. This report serves as a comprehensive guide for organizations looking to embark on their DevOps journey, providing insights, strategies, and practical recommendations for achieving success in the age of continuous delivery and deployment.

The DevOps methodology is somewhat unique in that it presents a mixed iterative and infinite methodology. It alleviates many of these issues by providing software teams with a better educated atmosphere. It places all duties at the beginning of the process, giving everyone a stake in their work. The task is better distributed, more organized, and the communication/knowledge gap is reduced when both teams share responsibility.

This is based on the Agile methodology, which maintains fairly convincingly that iterative reevaluation of project objectives yields superior results.

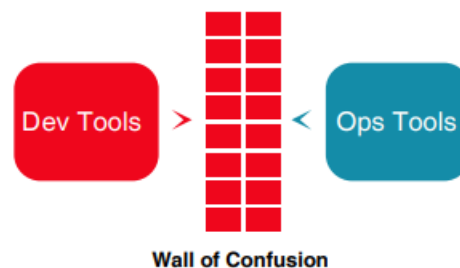
## 2.2 Need of DevOps:

The need for DevOps is rooted in the inherent tension between the priorities of development and operations teams. Developers are driven by the imperative to deliver changes swiftly, responding promptly to evolving business needs. Conversely, operations teams prioritize reliability and stability, aiming to ensure the smooth functioning of systems and minimize disruptions. This dichotomy often creates a "wall of confusion," as described by Lee Thomson, wherein the mindsets of the two teams diverge, compounded by the use of disparate tools for similar tasks.



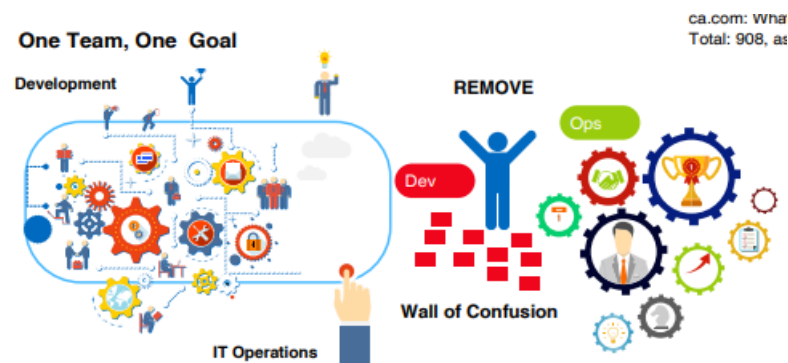
*Fig. 2.2.1 Wall Of confusion*

DevOps emerges as a transformative approach to dissolve these barriers, fostering collaboration and synergy between development and operations. By breaking down silos and unifying workflows, DevOps facilitates better and faster outcomes by aligning the objectives of both teams towards a shared goal: providing tangible business value. This integration not only harmonizes the agility-driven ethos of the development team with the stability-focused concerns of the operations team but also ensures that both aspects are directed towards delivering value to the organization.



*Fig 2.2.2 Development and Operation Wall of Confusion*

In essence, DevOps serves as a bridge between the drive for agility and responsiveness inherent in the development process and the imperative for quality and stability upheld by operations. By focusing on the ultimate goal of delivering business value, DevOps transcends the traditional divides and empowers teams to collaborate effectively, streamline processes, and accelerate delivery cycles. Thus, the need for DevOps lies in its ability to reconcile seemingly conflicting priorities, enabling organizations to thrive in a rapidly evolving digital landscape.



*Fig. 2.2.3 DevOps Benefits*



## Five Basics Principle of DevOps:

- Eliminate the blame game, Open post-mortems, Feedback, Rewarding failures
- Continuous Delivery, Monitoring, Configuration Management
- Business value for end user
- Performance Metrics, Logs, Business goals Metrics, People Integration Metrics, KPI
- Ideas, Plans, Goals, Metrics, Complications, Tools

## 2.3 DevOps Life-Cycle:

The DevOps lifecycle is a continuous software development process that employs DevOps best practices to plan, build, integrate, deploy, monitor, operate, and offer continuous feedback throughout the software's lifecycle.



*Fig. 2.3.1 DevOps Life-Cycle*

### 1. Plan:

The planning phase is exactly what it sounds like: planning the project's lifecycle. In contrast to conventional methods to the development lifecycle, this model assumes that each stage will be repeated as necessary. In this manner, the DevOps workflow is planned with the likelihood of future iterations and likely prior versions in mind.

This implies that we will likely have information from past iterations that will better inform the next iteration, and that the present iteration will likewise inform the next iteration. This stage often involves all teams to ensure that no area of the planning is ignored or forgotten.

## **2. Code:**

The developers will write the code and prepare it for the next phase during the coding stage. Developers will write code in accordance with the specifications outlined in the planning phase and will ensure that the code is created with the project's operations in mind.

## **3. Build:**

Code will be introduced to the project during the construction phase, and if necessary, the project will be rebuilt to accommodate the new code. This can be accomplished in a variety of ways, although GitHub or a comparable version control site is frequently used.

The developer will request the addition of the code, which will then be reviewed as necessary. The request will be approved if the code is ready to be uploaded, and the code will be added to the project. Even when adding new features and addressing bugs, this method is effective.

## **4. Test:**

Throughout the testing phase, teams will do any necessary testing to ensure the project performs as planned. Teams will also test for edge and corner case issues at this stage. An “edge case” is a bug or issue that only manifests during an extreme operating event, whereas a “corner case” occurs when many circumstances are met.

## **5. Release:**

The release phase occurs when the code has been verified as ready for deployment and a last check for production readiness has been performed. The project will subsequently enter the deployment phase if it satisfies all requirements and has been thoroughly inspected for bugs and other problems.

## **6. Deploy:**

In the deploy phase, the project is prepared for the production environment and is operating as planned in that environment. This would be the responsibility of the operations team; in DevOps, it is a shared responsibility. This shared duty pushes team members to collaborate to guarantee a successful deployment.

## **7. Operate:**

In the operating phase, teams test the project in a production environment, and end users utilize the product. This crucial stage is by no means the final step. Rather, it informs future development cycles and manages the configuration of the production environment and the implementation of any runtime requirements.

## **8. Monitor:**

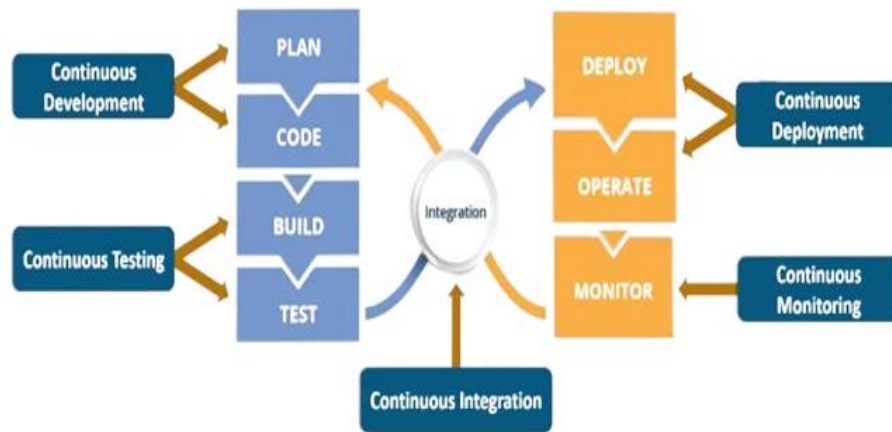
During the monitoring phase, product usage, as well as any feedback, issues, or possibilities for improvement, are recognized and documented. This information is then conveyed to the subsequent iteration to aid in the development process. This phase is essential for planning the next iteration and streamlines the pipeline's development process.

## **2.4 7 Cs of DevOps:**

### **1. Continuous Development:**

This step is crucial in defining the vision for the entire software development process. It focuses mostly on project planning and coding. At this phase, stakeholders and project needs are gathered and discussed. In addition, the product backlog is maintained based on customer feedback and is divided down into smaller releases and milestones to facilitate continuous software development.

Once the team reaches consensus on the business requirements, the development team begins coding to meet those objectives. It is an ongoing procedure in which developers are obliged to code whenever there are modifications to the project requirements or performance difficulties.



*Fig 2.4.1 DevOps Steps*

## 2. Continuous Integration:

Continuous integration is the most important stage of the DevOps lifecycle. At this phase, updated code or new functionality and features are developed and incorporated into the existing code. In addition, defects are spotted and recognized in the code at each level of unit testing during this phase, and the source code is updated accordingly. This stage transforms integration into a continuous process in which code is tested before each commit. In addition, the necessary tests are planned during this period.

## 3. Continuous Testing:

Some teams conduct the continuous testing phase prior to integration, whereas others conduct it after integration. Using Docker containers, quality analysts regularly test the software for defects and issues during this phase. In the event of a bug or error, the code is returned to the integration phase for correction. Moreover, automation testing minimizes the time and effort required to get reliable findings. During this stage, teams use technologies like as Selenium. In addition, continuous testing improves the test assessment report and reduces the cost of delivering and maintaining test environments.

#### **4. Continuous Deployment:**

This is the most important and active step of the DevOps lifecycle, during which the finished code is released to production servers. Continuous deployment involves configuration management to ensure the proper and smooth deployment of code on servers. Throughout the production phase, development teams deliver code to servers and schedule upgrades for servers, maintaining consistent configurations.

In addition to facilitating deployment, containerization tools ensure consistency throughout the development, testing, production, and staging environments. This methodology enabled the constant release of new features in production.

#### **5. Continuous Feedback:**

Constant feedback was implemented to assess and enhance the application's source code. During this phase, client behavior is routinely examined for each release in an effort to enhance future releases and deployments. Companies can collect feedback using either a structured or unstructured strategy.

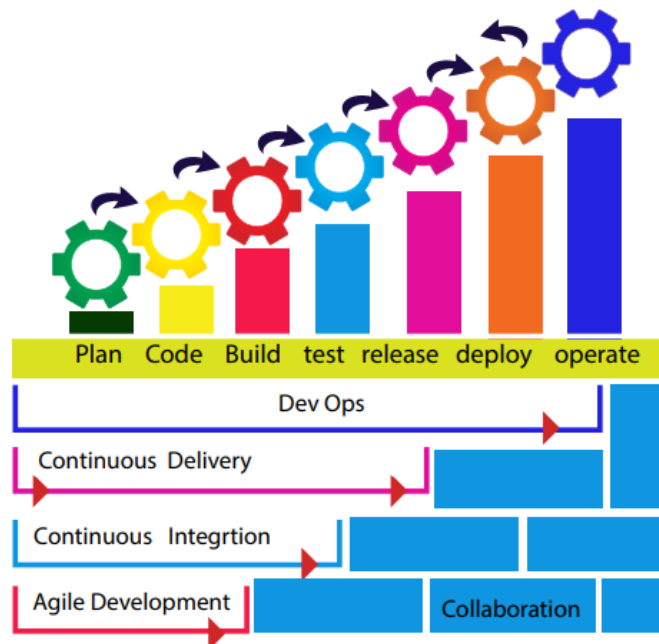
Under the structural method, input is gathered using questionnaires and surveys. In contrast, feedback is received in an unstructured manner via social media platforms. This phase is critical for making continuous delivery possible in order to release a better version of the program.

#### **6. Continuous Monitoring:**

During this phase, the functioning and features of the application are regularly monitored to detect system faults such as low memory or a non-reachable server. This procedure enables the IT staff to swiftly detect app performance issues and their underlying causes. Whenever IT teams discover a serious issue, the application goes through the complete DevOps cycle again to determine a solution. During this phase, however, security vulnerabilities can be recognized and corrected automatically.

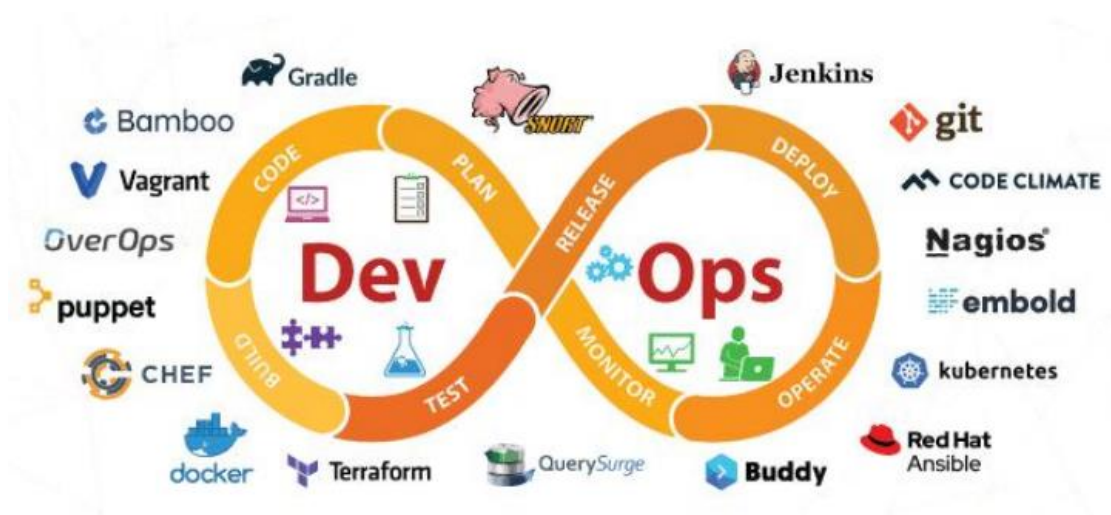
## 7. Continuous Operations:

The final phase of the DevOps lifecycle is essential for minimizing scheduled maintenance and other planned downtime. Typically, developers are forced to take the server offline in order to perform updates, which increases the downtime and could cost the organization a large amount of money. Eventually, continuous operation automates the app's startup and subsequent upgrades. It eliminates downtime using container management platforms such as Kubernetes and Docker.



Without automation there is no DevOps.

*Fig. 2.4.2 DevOps 7Cs*



*Fig. 2.4.3 DevOps Tool Chain*

### 3. DEVOPS TOOL CHAIN

#### 1. Maven



**Description:**

Maven is a powerful build automation tool primarily used for Java projects. It simplifies the build process by managing project dependencies and providing a standardized way to build, package, and deploy Java applications.

**Key Features:**

**Dependency Management:** Maven uses a project object model (POM) file to define project structure, dependencies, and configurations, allowing developers to easily manage dependencies and ensure consistency across different environments.

**Build Lifecycle:** It offers a set of predefined build phases such as compile, test, package, and deploy, simplifying the build process and ensuring that necessary tasks are executed in the correct order.

**Plugin Ecosystem:** Maven boasts a rich ecosystem of plugins that extend its functionality, allowing developers to integrate with other tools and perform custom tasks as needed.

#### 2. GitHub



**Description:**

GitHub is a leading platform for version control and collaborative software development. It offers a range of features that support DevOps practices such as versioning, collaboration, automated testing, and deployment.

**Key Features:**

**Version Control:** GitHub provides Git-based version control, enabling teams to track changes, manage branches, and collaborate on code development efficiently.

**Continuous Integration/Continuous Deployment (CI/CD):** Through integrations with tools like GitHub Actions, it enables automated workflows for building, testing, and deploying applications, fostering a culture of automation and rapid delivery.

**Code Review:** GitHub facilitates code review processes, allowing developers to comment on code changes, suggest improvements, and ensure code quality before merging changes into the main codebase.

**Issue Tracking:** Its built-in issue tracking system helps teams manage and prioritize tasks, track bugs, and address feature requests effectively.

### 3. Jenkins



**Jenkins**

#### **Description:**

Jenkins is a popular open-source automation server used for automating various aspects of software development, including building, testing, and deploying code.

#### **Key Features:**

**Continuous Integration/Continuous Deployment (CI/CD):** Jenkins provides robust support for CI/CD pipelines, allowing teams to automate the build, test, and deployment processes, thereby accelerating software delivery and ensuring quality.

**Plugin Ecosystem:** With a vast array of plugins, Jenkins offers extensibility and flexibility, enabling integration with a wide range of tools and technologies to support diverse development workflows.

**Scalability:** Jenkins is highly scalable and suitable for organizations of all sizes. It can be easily configured to meet specific requirements and can scale to accommodate complex build and deployment scenarios.

**Community Support:** Being an open-source project, Jenkins enjoys strong community support, with regular updates, contributions, and a wealth of resources available for users.

### 4. Docker



#### **Description:**

Docker is a containerization platform that simplifies the process of building, deploying, and managing applications by encapsulating them within lightweight, portable containers.

#### **Key Features:**

**Containerization:** Docker containers provide a consistent and isolated environment for running applications, ensuring consistency across different computing environments and simplifying deployment.

**Portability:** Docker containers are portable and can run on any system that supports Docker, making it easier to deploy applications across different environments, from development to production.

**Resource Efficiency:** Docker containers consume fewer resources compared to traditional virtual machines, enabling efficient resource utilization and faster application startup times.

**Microservices Architecture:** Docker is well-suited for microservices architecture, allowing developers to modularize applications into smaller,



manageable components that can be independently developed, deployed, and scaled.

## 5. Ansible



### Description:

Ansible is an open-source automation tool that simplifies IT tasks such as configuration management, application deployment, and orchestration through declarative YAML-based automation.

### Key Features:

**Agentless Architecture:** Ansible operates over SSH, making it agentless and easy to deploy without requiring additional software on managed hosts.

**Infrastructure as Code (IaC):** Ansible allows infrastructure to be defined as code, enabling automation and version control of infrastructure configurations, leading to improved consistency and reliability.

**Playbook-based Automation:** Ansible automation is defined using playbooks, which are written in YAML and describe a set of tasks to be executed on remote hosts, making automation easy to read, write, and understand.

**Integration:** Ansible integrates seamlessly with other DevOps tools and technologies, enabling end-to-end automation of the software delivery pipeline.

## 6. ELK Stack

### Description:

The ELK stack is a collection of open-source tools (Elasticsearch, Logstash, Kibana) used for log management and analysis, providing real-time insights into large volumes of log data.

### Key Features:

**Elasticsearch:** A distributed search and analytics engine that enables indexing, querying, and analyzing large volumes of data in real-time.

**Logstash:** A data processing pipeline that collects, filters, and transforms log data from various sources before sending it to Elasticsearch for indexing.

**Kibana:** A data visualization tool that provides interactive dashboards and visualizations for analyzing and interpreting log data stored in Elasticsearch.

**Centralized Logging:** The ELK stack facilitates centralized logging, allowing organizations to aggregate logs from multiple sources, monitor application performance, and troubleshoot issues effectively.

### 3.2.2 Automation Plugins or tools:

#### 1. Webhooks



##### **Description:**

Webhooks are user-defined HTTP callbacks triggered by specific events or actions on a source application. They facilitate real-time communication between different web applications, allowing them to exchange data and automate workflows.

##### **Key Features:**

**Real-time Integration:** Webhooks enable real-time integration between different web services, allowing them to react to events and trigger actions without polling or manual intervention.

**Customizable:** Users can define webhooks to trigger actions based on specific events or conditions, making them highly customizable and adaptable to various use cases. Webhooks follow an event-driven architecture, enabling loosely coupled integration between applications and promoting agility and scalability.

**Easy Configuration:** Webhooks can be easily configured within applications using simple HTTP endpoints, making them accessible and straightforward to implement.

#### 2. Ngrok



##### **Description:**

Ngrok is a tool for creating secure tunnels from a public endpoint to a locally running web server, allowing developers to expose local web services to the public internet securely.

##### **Key Features:**

**Secure Tunnelling:** Ngrok creates secure tunnels from a public endpoint to a local server, encrypting traffic and ensuring data security during transit.

**Local Development:** It simplifies local development by eliminating the need to deploy applications to a public server or expose the local development environment to the internet.

**Protocol Support:** Ngrok supports HTTP, HTTPS, and TCP protocols, allowing developers to expose a wide range of local services to the internet securely.

**User-Friendly Interface:** Ngrok provides a user-friendly interface for managing tunnels and inspecting traffic, making it easy to use for testing, debugging, and integrating with third-party services.

## **4. EXECUTION FLOW**

### **1. Establish Git Repository and Version Control:**

- Set up a Git repository and configure it for version control.
- Define branching strategies and release processes.
- Utilize pull requests for code review and collaboration.

### **2. Deploy Jenkins for Continuous Integration:**

- Install Jenkins on a designated server.
- Integrate Jenkins with the Git repository.
- Configure Jenkins jobs for automated building, testing, and deployment.
- Employ plugins for enhanced automation capabilities.
- Automate as many steps as possible using Jenkins plugins.

### **3. Utilize Maven for Dependency Management and Build Automation:**

- Install IntelliJ IDE
- Configure Maven within IntelliJ IDEA.
- Manage project dependencies effectively.
- Create a comprehensive build pipeline covering code compilation and testing, including JUnit testing, and artifact packaging.

### **4. Implement Docker for Containerization:**

- Set up a Docker registry.
- Utilize Docker containers for application encapsulation.
- Ensure consistency and portability across environments with Docker images.

### **5. Automate Infrastructure with Ansible:**

- Use Ansible for automating infrastructure provisioning and configuration.
- Develop Ansible playbooks for consistent deployment across environments.

### **6. Testing and Deployment:**

- Execute testing, including JUnit testing, and deployment processes.
- Ensure thorough testing before deployment to guarantee the reliability of the application.

## 5. IMPLEMENTATION STEPS

### 5.1 SOURCE CODE MANAGEMENT WITH GITHUB

#### 1. Create a new repository on GitHub:

- Log in to your GitHub account.
- Click on the "+" icon on the page and select "New repository".
- Fill in the repository name (e.g., "miniProject"), add an optional description, choose visibility (public or private), and initialize with a README if needed.
- Click on "Create repository".

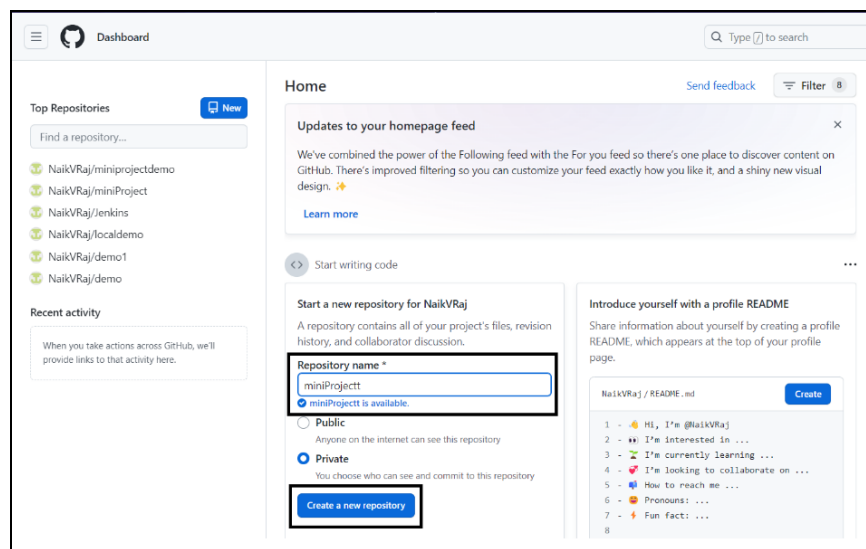


Fig. 5.1.1 GitHub new Repository

#### 2. Set up local repository and link it to the remote repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to create your local repository. (Where you are going to create calculator miniProject)
- Execute the following commands:

```
echo "# miniProject" >> README.md
git init
git add README.md
git commit -m "Initial commit"
git branch -M main
git remote add origin https://github.com/your - username/mini
- Project.git
git push -u origin main
```

### 3. Develop code and manage source code:

- Write your code according to project requirements within your local repository.
- After making changes, stage the changes for commit.

***git add .***

- Commit the changes along with a descriptive message:

***git commit -m "Your descriptive message here"***

- Push the changes to the remote repository:

***git push origin main***

### 4. Viewing commit history:

- To view commit history, you can use the following command:

***git log***

- This will display a list of commits along with their commit IDs, authors, dates, and commit messages.
- You can also use various options with git log to customize the output as needed.

### 5. Checking out to a previous commit:

- To go back to a previous commit, you can use the following command:

***git checkout <commit\_id>***

- Replace <commit\_id> with the commit ID you want to revert to.
- After checking out to a previous commit, you can create a new branch if you want to make changes from that point onwards without affecting the current branch.

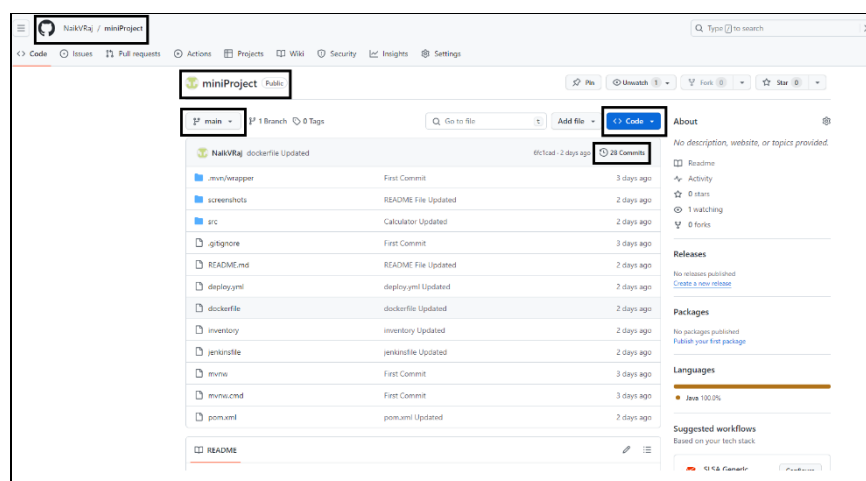


Fig. 5.1.2 Repository Page

Commit History Commit-id Commit by whom? When? Commit message

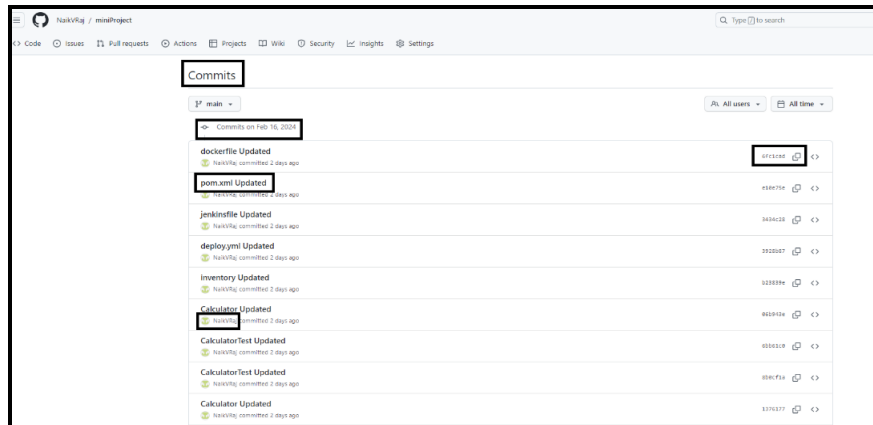


Fig. 5.1.3 Commit History

## 5.2 CODE DEVELOPMENT WITH IntelliJ

### 1. Launch IntelliJ IDEA:

- Launch IntelliJ IDEA from your desktop or applications menu.

### 2. Create a New Maven Project:

- Click on "File" in the top menu.
- Select "New" and then "Project".
- On the window that appears, click "Maven" on the left side.
- Choose "Create from archetype" on the right side.

### 3. Choose the Maven Archetype:

- Explore the list of available archetypes.
- Pick one that best fits your project needs.
- Click "Next" to proceed.

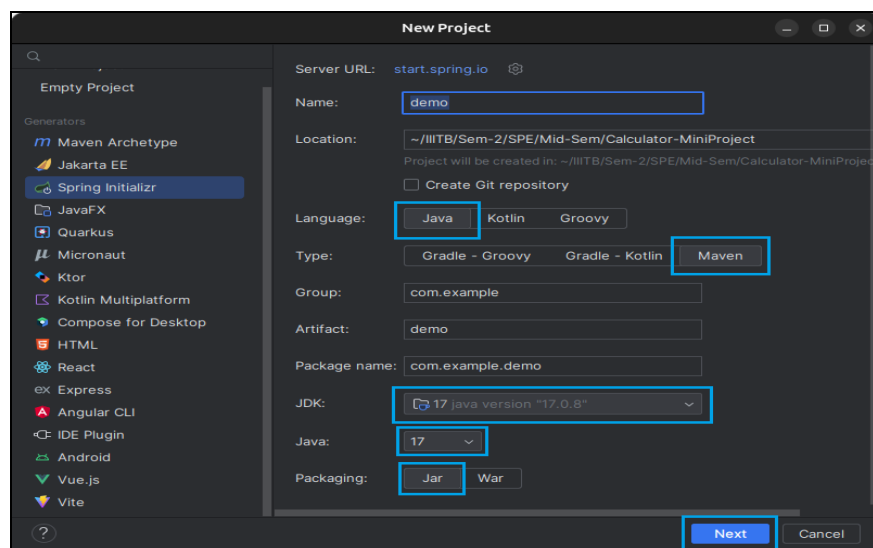
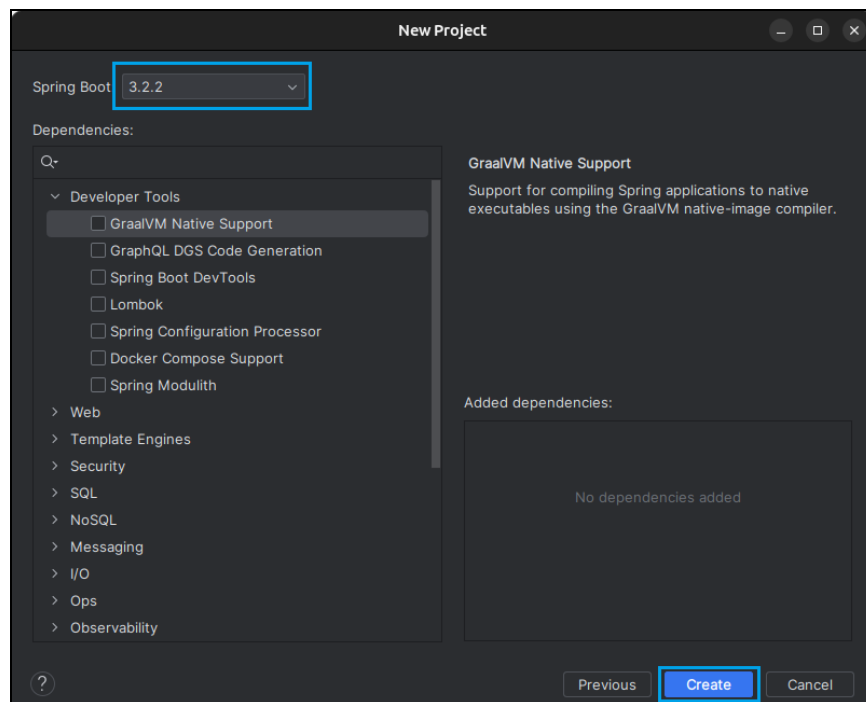


Fig. 5.2.1 IntelliJ Project Creation - I

#### 4. Set up the Maven Project:

- Provide the necessary project details:
- Group Id: Similar to a family name for your project.
- Artifact Id: A specific name for your project, like a personal name.
- Version: Your project's version number, often starting with 1.0.
- Package: Organize your files into folders.
- Click "Finish" to complete the setup.



*Fig. 5.2.2 IntelliJ Project Creation - II*

#### 5. Configure the POM File:

- Locate the pom.xml file in your project.
- Right-click on it and choose "Maven" → "Add Dependency".
- Here, you can search for and add any extra tools or libraries your project requires.

#### 6. Build the Maven Project:

- Right-click on your project name in the project explorer.
- Select "Maven" → "Reimport" to ensure proper setup.

## **7. Configure DevOps Tools:**

- IntelliJ IDEA seamlessly integrates with various DevOps tools.
- You can set up these tools later as you become more familiar with them.

## **8. Write Your Code:**

- Create a Java file named Calculator.java within the src/main/java directory (you can choose any name for your class).
- Write your application logic for the Scientific Calculator, including functionalities such as addition, subtraction, multiplication, division, power, square root, pi, logarithm, and factorial.

## **9. Write Unit Tests:**

- Create a Java test file named CalculatorTest.java within the src/test/java directory (you can choose any name for your test class).
- Write unit tests for your Java class in the test file to validate the functionalities of your Scientific Calculator.

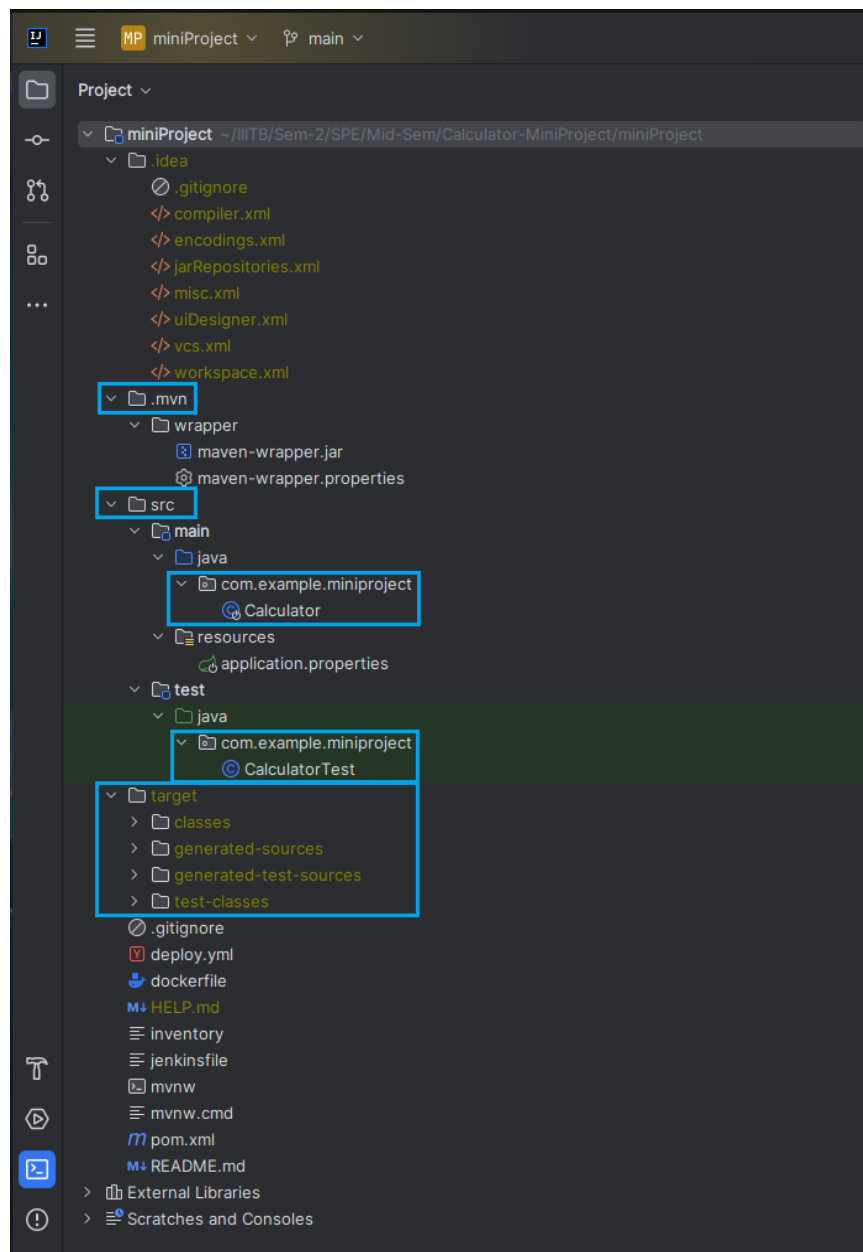
## **10. Run Tests:**

- Use IntelliJ IDEA's built-in test runner to execute your unit tests.
- Verify that all tests pass successfully, indicating that your code behaves as expected.

## **11. Deploy the Maven Project:**

- Once your project is ready, you can deploy it to a server or any other deployment environment.
- If you've configured DevOps tools, you can automate the deployment process for seamless deployment.





*Fig. 5.2.3 Project Directory Structure*

The structure can be customized to fit the specific needs of the project, but following the convention allows for easier understanding and maintenance by other developers.

## 5.3 SOFTWARE BUILDING AND TESTING WITH MAVEN AND JUNIT

### 1. Setting Up Maven and Java Version:

- First, make sure you have Maven installed on your system. It's like a handy assistant for managing your Java projects.
- Since we're working with Java 17, we need to tell Maven about it. We do this in the pom.xml file, which is like a special note where we jot down important project details. We make sure to mention that we're using Java version 17.
- Add JUnit dependency to the pom.xml file under the <dependencies> section. This ensures that JUnit is available for unit testing. Since we are using JUnit for the testing purpose.



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.2</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>miniProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>miniProject</name>
  <description>miniProject</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
      <version>3.2.2</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <version>3.2.2</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <version>3.2.2</version>
    </dependency>
  </dependencies>
</project>
```

Fig. 5.3.1 Dependency Addition

```

43
44
45     <dependency>
46         <groupId>org.junit.jupiter</groupId>
47         <artifactId>junit-jupiter-api</artifactId>
48         <version>5.10.1</version>
49         <scope>test</scope>
50     </dependency>
51 </dependencies>
52
53     <build>
54         <plugins>
55             <plugin>
56                 <groupId>org.springframework.boot</groupId>
57                 <artifactId>spring-boot-maven-plugin</artifactId>
58             </plugin>
59         </plugins>
60         <finalName>miniproject</finalName>
61     </build>

```

*Fig. 5.3.2 JUnit Dependency Insertion*

## 2. Creating a Java Class:

- Now, let's start by writing some code. We're going to make a super smart calculator in Java. We'll create a class called Calculator and give it methods for all sorts of math operations like addition, subtraction, multiplication, division, finding powers, logarithms, square roots, and trigonometric functions like sine, cosine, and tangent.
- As an example, here factorial function.

```

package com.example.miniproject;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RestController;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

@SpringBootApplication
@RestController
public class Calculator {

    public static void main(String[] args) throws IOException, InterruptedException {
        SpringApplication.run(Calculator.class, args);
    }
}

```

*Fig. 5.3.3 Java Class libraries and Annotation*

```

public static double performFactorial(int n) {
    if (n < 0) {
        System.out.println("Error: Factorial of a negative number is undefined!");
        return Double.NaN;
    } else {
        long factorial = 1;
        for (int i = 1; i <= n; i++) {
            factorial *= i;
        }
        return factorial;
    }
}

```

*Fig. 5.3.4 Factorial Function*

### 3. Writing Unit Tests:

- It's important to make sure our calculator works perfectly. We're going to write tests for each math operation to check if everything is doing what it's supposed to. We do this using JUnit, which is like a set of tools for testing Java code.
- We'll create a new Java class called CalculatorTest. Inside, we'll write test methods for each operation our calculator can do.

```

@Test
public void testPerformFactorial() {
    double result = Calculator.performFactorial(5);
    assertEquals("expected: 120, result, message: \"Testing factorial function\"", result, 120);
}

```

*Fig. 5.3.5 Factorial Test Case*

### 4. Building the Project:

- Once we're confident in our code and tests, we need to get everything ready for action. This process is called "building" the project. It's like putting all the pieces together and making sure they work well.
- We click on "Build → Build Project" in our IDE to do this.



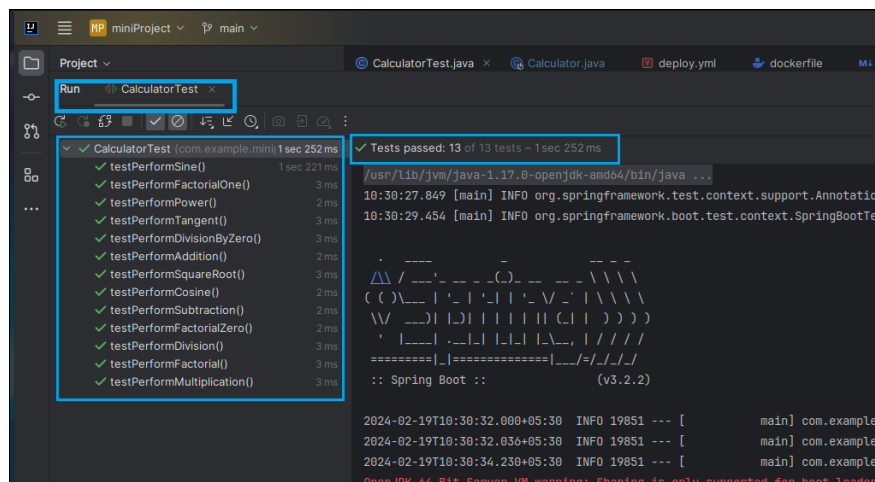


Fig 5.3.7 Running CalculatorTest

## 8. Running the Project:

- Now that everything is built and ready to go, we can run our project. We find the special JAR file that Maven made for us in the target folder and double-click it. Alternatively, we can deploy our project to another place for more testing or to use it in the real world.

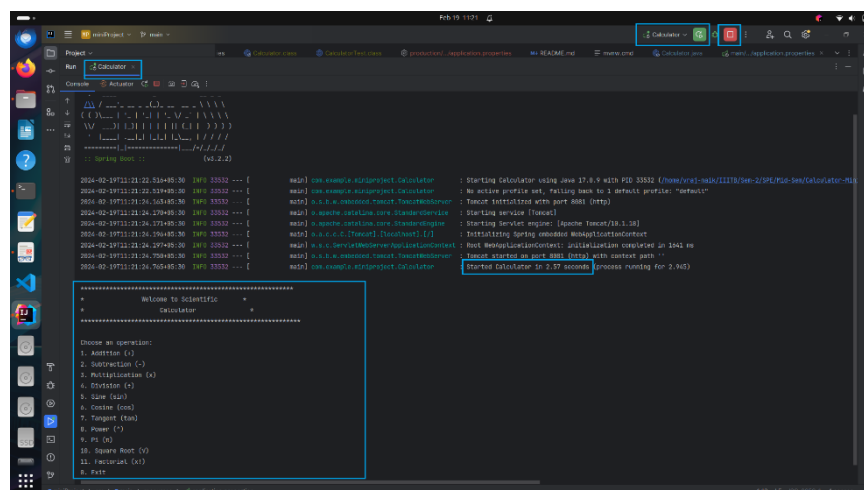


Fig. 5.3.8 Running Project in IntelliJ

## 5.4 AUTOMATION WITH JENKINS AND PIPELINE

### 1. Update Package List and Install Java:

- Update Package List: This ensures that your system has the latest information about available packages and their versions. It's crucial before installing any new software.
- Install Java: Jenkins requires Java to run. Installing OpenJDK 17 ensures that the Java runtime environment is available on your system, which Jenkins will utilize.

***sudo apt update***

***sudo apt install openjdk – 17 – jdk***

### 2. Add Jenkins Repository Key and Repository:

- Add Jenkins Repository Key: This step adds the Jenkins repository key to your system, allowing your package manager to verify the integrity of Jenkins packages during installation.
- Add Jenkins Repository: By adding the Jenkins repository to your system's package sources, you enable your package manager to find and install Jenkins packages.

***wget – q – O – https://pkg.jenkins.io/debian  
– stable/jenkins.io.key | sudo apt – key add –***

***sudo sh – c 'echo deb https://pkg.jenkins.io/debian  
– stable binary/  
> /etc/apt/sources.list.d/jenkins.list'***

### 3. Install Jenkins:

- Update Package List Again: Refreshes the package list to include the newly added Jenkins repository.
- Install Jenkins: This command installs the Jenkins continuous integration server on your system, allowing you to use Jenkins for automation and building projects.

***sudo apt update***

***sudo apt install jenkins***

### 4. Start and Check Jenkins Service:

- Start Jenkins Service: Initiates the Jenkins service, allowing it to run in the background and serve web requests.
- Check Jenkins Service: Verifies whether the Jenkins service has started successfully and is currently running.

***sudo service jenkins start***

***sudo service jenkins status***

### 5. Access and Unlock Jenkins:

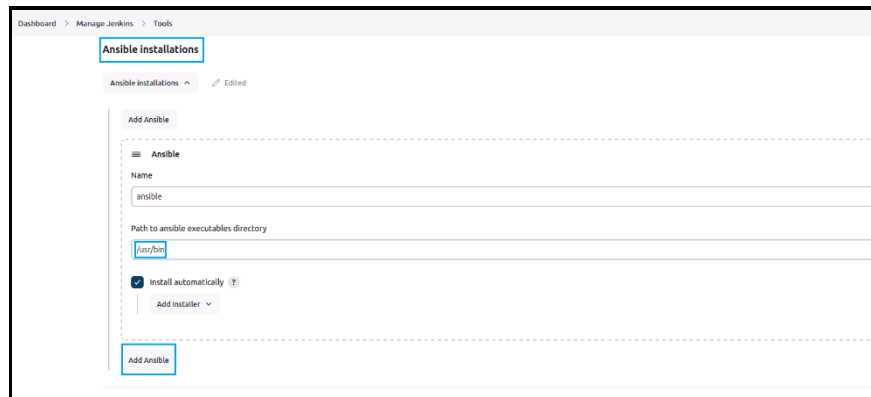
- Access Jenkins: Opens Jenkins in your web browser, allowing you to interact with its web-based interface.
- Unlock Jenkins: Jenkins is initially locked to prevent unauthorized access. Finding and entering the initial administrator password unlocks Jenkins, enabling you to configure it.

***sudo cat /var/lib/jenkins/secrets/initialAdminPassword***



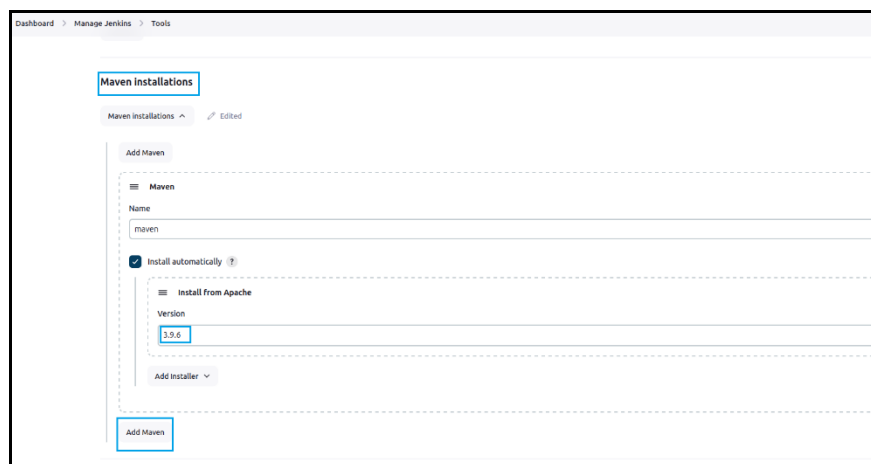
## 6. Setup Jenkins and Install Essential Tools:

- Follow Setup Wizard: The setup wizard guides you through the initial configuration of Jenkins, such as setting up the admin user and selecting plugins.
- Install Required Plugins: Installing essential plugins like Maven, Git, Ansible, and Docker ensures that Jenkins has the necessary functionality for building and deploying projects.



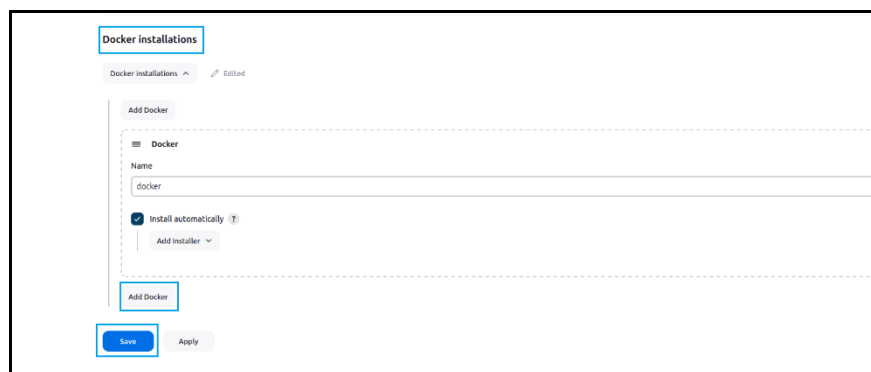
The screenshot shows the 'Ansible Installations' page in the Jenkins 'Tools' section. It features a form for adding a new Ansible installation. The 'Name' field is set to 'ansible'. The 'Path to ansible executables directory' field is set to '/usr/bin'. The 'Install automatically' checkbox is checked. There are 'Add Ansible' buttons at the top and bottom of the form, and an 'Add installer' dropdown menu.

*Fig. 5.4.1 Ansible Installation*



The screenshot shows the 'Maven Installations' page in the Jenkins 'Tools' section. It features a form for adding a new Maven installation. The 'Name' field is set to 'maven'. The 'Install automatically' checkbox is checked. Under the 'Install from Apache' section, the 'Version' field is set to '3.9.6'. There are 'Add Maven' buttons at the top and bottom of the form, and an 'Add installer' dropdown menu.

*Fig. 5.4.2 Maven Installation*



The screenshot shows the 'Docker installations' page in the Jenkins 'Tools' section. It features a form for adding a new Docker installation. The 'Name' field is set to 'docker'. The 'Install automatically' checkbox is checked. There are 'Add Docker' buttons at the top and bottom of the form, and an 'Add installer' dropdown menu. At the bottom of the page, there are 'Save' and 'Apply' buttons.

*Fig. 5.4.3 Docker Installation*

## 7. Install Additional Required Plugins:

- Navigate to Jenkins dashboard and go to "Manage Jenkins" → "Manage Plugins".
- Select the "Available" tab and search for the following plugins: Maven, Git, Ansible, Docker.
- Install these plugins to enable their functionality within Jenkins.

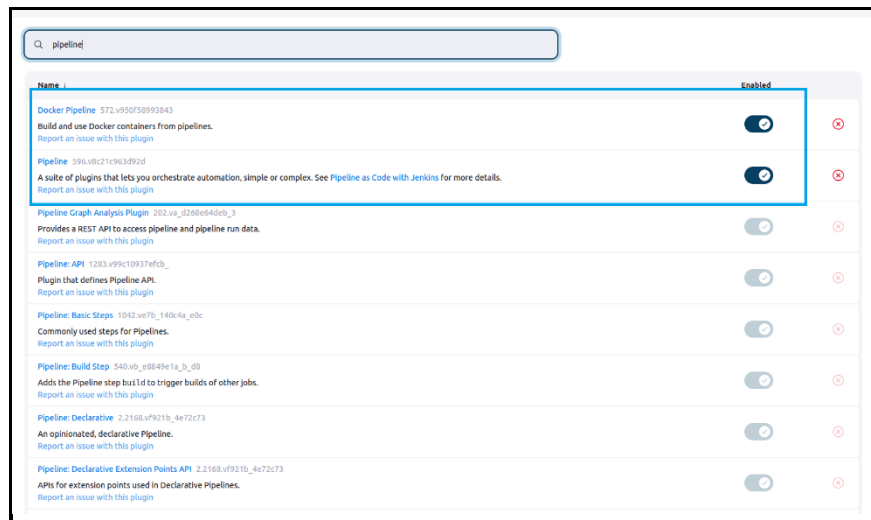


Fig. 5.4.4 Pipeline plugins installation

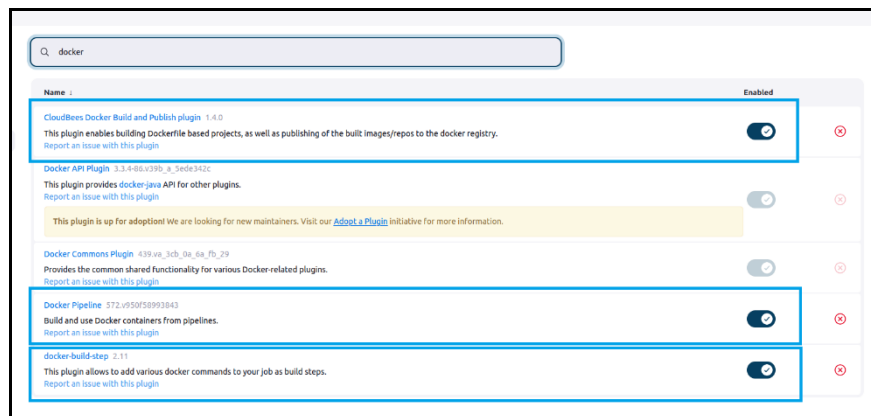
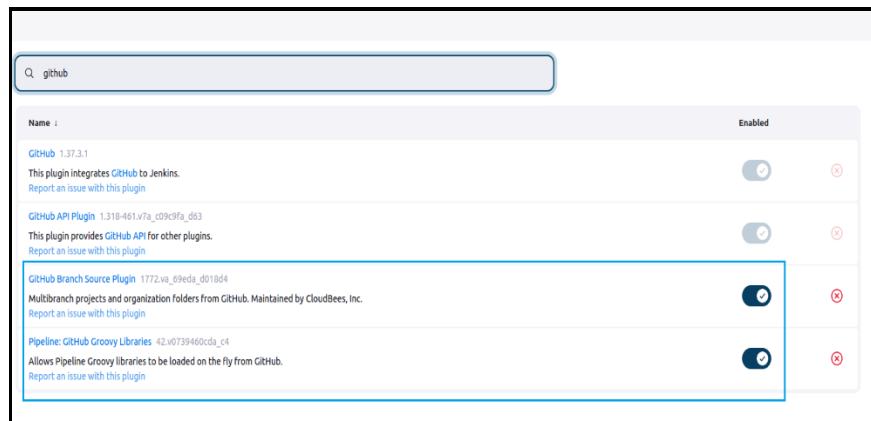
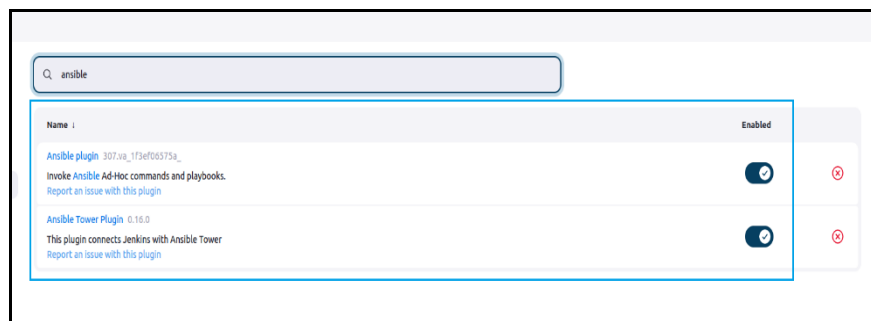


Fig. 5.4.5 Docker plugins installation



*Fig. 5.4.6 GitHub plugins installation*



*Fig. 5.4.7 Ansible plugins installation*

## 8. Add Global Tool Configuration from Manage Jenkins:

- Go to “Manage Jenkins” → “Global Tool Configuration”.
- Configure the tools as follows:
  - For Maven:
    - Set the name as “maven”.
    - Specify the MAVEN\_HOME directory.
  - For Git:
    - Add the Git executable path.
  - For Ansible and Docker:
    - If applicable, specify the executable paths for Ansible and Docker.

## 9. Configure Jenkins location and GitHub server:

- For Jenkins Location:
  - Set the system administrator email address to a Gmail account.

- For GitHub Server:
  - Go to “Manage Jenkins” → “Configure System”.
  - Scroll down to the "GitHub" section.
  - Click on "Add GitHub Server" and provide credentials for accessing GitHub repositories.
  - Configure additional settings as necessary, such as webhook management and advanced connection settings.

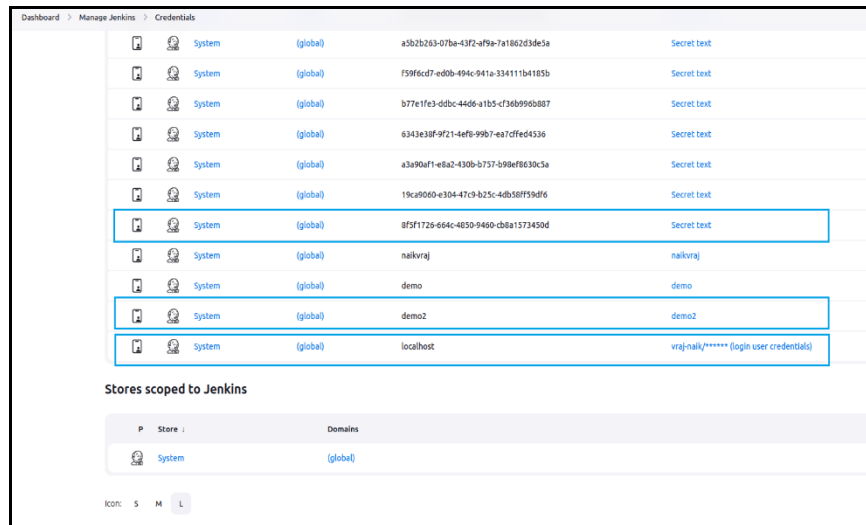
*Fig. 5.4.8 Adding Admin e-mail Address*

*Fig. 5.4.9 Adding GitHub Server Credentials*

## 10. Add Github and Dockerhub Credentials for Jenkins Access:

- Go to “Manage Jenkins” → “Manage Credentials”.
- Under “System”, click on “Global credentials”.
- Add credentials for GitHub and Docker Hub:
- For GitHub:
  - Add a username/password credential or SSH key credential with access to the repository.

- For Docker Hub:
  - Add a username/password credential with access to Docker Hub.



*Fig. 5.4.10 Selecting Required Credentials*

## 11. Integrate Pipeline Script:

- Create a new Jenkins pipeline job.
- Paste the provided pipeline script into the Jenkinsfile section of the pipeline job configuration.
- Or also specify file name containing pipeline script.
- Make sure to replace placeholders like the GitHub repository URL and Docker Hub credentials with your actual values.
- Replace placeholders like scmGit with the appropriate SCM configuration for your Git repository.
- Verify that Maven commands, Docker build/push commands, and Ansible playbook paths match your project structure and requirements.
- Ensure that environment variables and credentials are properly defined and utilized throughout the pipeline.
- The pipeline script utilizes the Jenkins pipeline syntax, allowing for the definition of a sequence of stages to be executed one after the other.
- The agent directive specifies that the pipeline can run on any available agent.
- The tools directive specifies the Maven tool to be used in the pipeline.
- The stages directive defines a list of stages to be executed.

- Each stage contains a list of steps to be executed within that stage.
- The checkout step pulls the latest changes from the specified Git repository branch.
- The sh step runs Maven commands to build the project and execute JUnit tests.
- The docker build step builds a Docker image from the packaged application.
- The docker login step logs in to Docker Hub using the specified credentials.
- The docker push step pushes the built Docker image to Docker Hub.
- The docker rm step stops and removes any existing Docker container named 'miniProject'.
- The ansiblePlaybook step runs an Ansible playbook for deployment, specifying the playbook and inventory file.

## 12. Save and Run Pipeline:

- Save the pipeline job configuration.
- Trigger a build to execute the pipeline and observe the execution progress and any potential issues.
- Monitor the build console output for any errors or warnings that may require troubleshooting

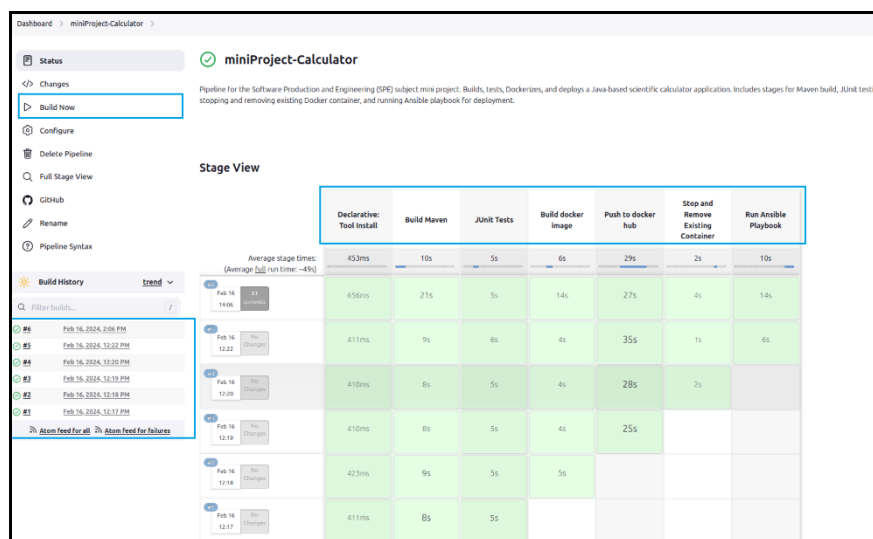


Fig. 5.4.11 Running Pipeline in Jenkins

```

pipeline {
    agent any
    tools {
        maven 'maven'
    }
    stages {

        // Stage to checkout source code and build with Maven
        stage('Build Maven') {
            steps {
                // Checkout source code from Git repository
                checkout scmGit(branches: [[name: '*/main']], extensions: [],
userRemoteConfigs: [[url: 'https://github.com/NaikVRaj/miniProject/']])
                // Build project with Maven
                sh 'mvn clean install'
            }
        }

        // Stage to run JUnit tests
        stage('JUnit Tests') {
            steps {
                // Run JUnit tests with Maven
                sh 'mvn test'
            }
        }

        // Stage to build Docker image
        stage('Build docker image') {
            steps {
                script {
                    // Build Docker image
                    sh 'docker build -t naikvraj/miniProject .'
                }
            }
        }

        // Stage to push Docker image to Docker Hub
        stage('Push to docker hub') {
            steps {
                script {
                    // Login to Docker Hub with credentials
                    withCredentials([string(credentialsId: 'demo2', variable:
'speminiproject')]) {
                        sh 'docker login -u naikvraj -p ${speminiproject}'
                    }
                    // Push Docker image to Docker Hub
                    sh 'docker push naikvraj/miniProject'
                }
            }
        }

        // Stage to stop and remove existing Docker container
        stage('Stop and Remove Existing Container') {
            steps {
                script {
                    // Stop and remove existing Docker container named miniProject if
it exists then
                    sh 'docker rm -f miniProject || true'
                }
            }
        }

        // Stage to run Ansible playbook for deployment
        stage('Run Ansible Playbook') {
            steps {
                script {
                    // Run Ansible playbook for deployment
                    ansiblePlaybook(
                        playbook: 'deploy.yml',
                        inventory: 'inventory'
                    )
                }
            }
        }
    }
}

```

*Fig. 5.4.12 Jenkins Pipeline Script*

## 5.5 CONTAINERIZATION WITH DOCKER

### 1. Create Dockerfile:

- A Dockerfile is a text file containing instructions to build a Docker image.
- It specifies the base image, environment variables, dependencies, and configuration details needed to run the application in a container.
- Common instructions in a Dockerfile include:
  - FROM: Specifies the base image for building the new image.
  - RUN: Executes commands during the image build process.
  - COPY or ADD: Copies files from the host machine to the image's file system.
  - WORKDIR: Sets the working directory inside the image.
  - EXPOSE: Declares network ports that the container will listen on.
  - CMD or ENTRYPOINT: Specifies the command to run when the container starts

### 2. Build a Docker Image:

- Use the Dockerfile to build a Docker image by running the docker build command.
- This process creates a new image based on the instructions in the Dockerfile.

### 3. Run a Docker Container:

- Start a Docker container using the docker run command.
- This creates an instance of the Docker image as a container and starts the application.

### 4. Test the Docker container:

- Ensure the containerized application functions correctly.
- Access the application via the container's IP address or using Docker CLI commands.



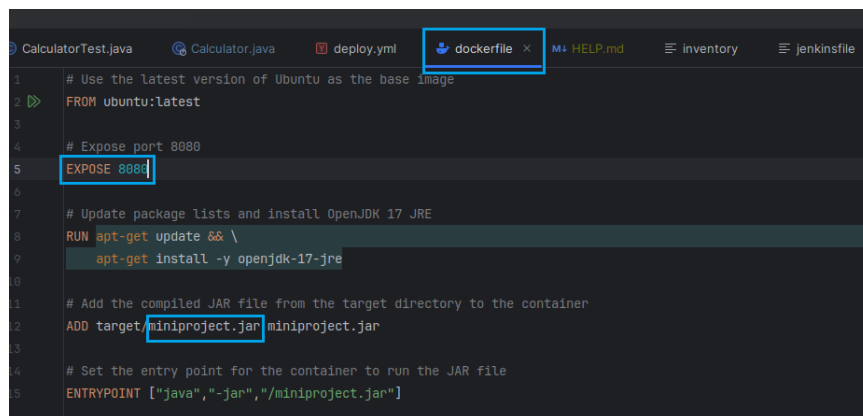
## 5. Push the Docker image to a registry:

- After testing, push the Docker image to a registry like Docker Hub or a private registry.
- This allows others to download and use the Docker image for deployment.

## 6. Deploy the Docker container:

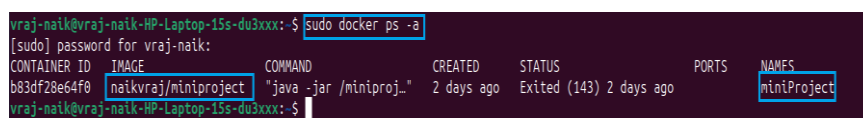
- Deploy the Docker container to a production environment (not explicitly done in this project, runs locally):
  - Cloud service provider
  - Virtual machine
  - Kubernetes cluster

Note: Deployment to a production environment involves additional steps like configuring networking, setting up scaling, monitoring, and ensuring high availability. In this project, deployment is assumed to be on a local machine for testing purposes.



```
1 # Use the latest version of Ubuntu as the base image
2 FROM ubuntu:latest
3
4 # Expose port 8080
5 EXPOSE 8080
6
7 # Update package lists and install OpenJDK 17 JRE
8 RUN apt-get update && \
9     apt-get install -y openjdk-17-jre
10
11 # Add the compiled JAR file from the target directory to the container
12 ADD target/miniproject.jar miniproject.jar
13
14 # Set the entry point for the container to run the JAR file
15 ENTRYPOINT ["java", "-jar", "/miniproject.jar"]
```

Fig. 5.5.1 DockerFile



```
vraj-naik@vraj-naik-HP-Laptop-15s-du3xxx:~$ sudo docker ps -a
[sudo] password for vraj-naik:
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS   NAMES
b83df28e64f0   naikvraj/miniproject   "java -jar /miniproj.."   2 days ago    Exited (143) 2 days ago           miniProject
vraj-naik@vraj-naik-HP-Laptop-15s-du3xxx:~$
```

Fig.5.5.2 Containers on the system

## 5.6 CONTINUOUS DEPLOYMENT WITH ANSIBLE

### 1. Create an Inventory File:

- Define a list of servers to manage.
- Specify server details such as IP addresses or hostnames.
- Organize servers into groups if necessary.

### 2. Write a Playbook:

- Define tasks to be executed on the servers.
- Use YAML syntax to structure the playbook.
- Include modules and parameters to describe desired configurations.
- Implement conditionals, loops, and error handling if needed.

### 3. Build the Docker Image:

- Write a Dockerfile specifying Ansible installation.
- Copy the playbook and inventory file into the Docker image.
- Install any dependencies required for the playbook.
- Build the Docker image using the Docker build command.
- Tag the image appropriately and push it to Dockerhub.

### 4. Pull the Docker Image:

- Use the Docker pull command to retrieve the image from Dockerhub.
- Ensure proper authentication if the repository is private.

### 5. Run the Docker Container:

- Start a Docker container based on the pulled image.
- Mount the directory containing the playbook and inventory file as a volume inside the container.
- Specify any additional runtime configurations as needed.

## 6. Execute the Playbook:

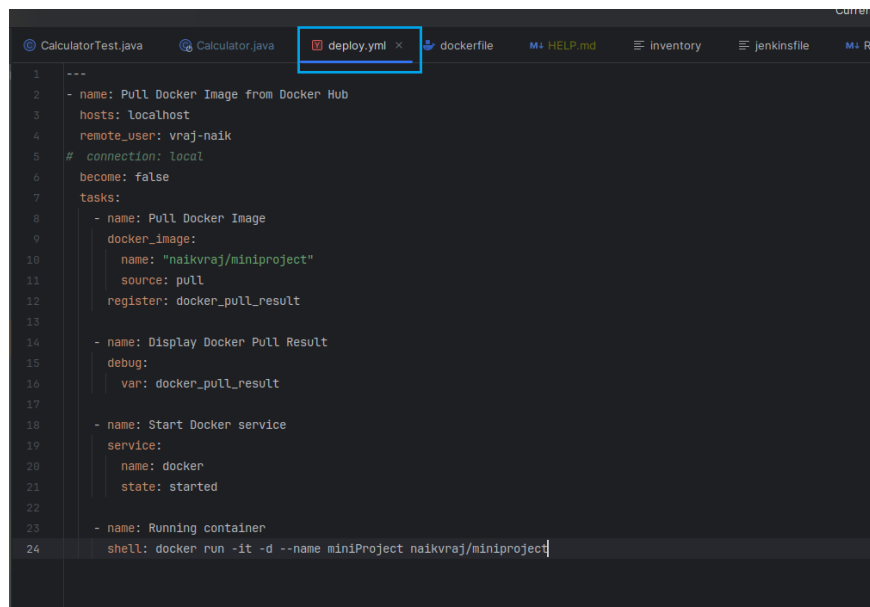
- Use the docker exec command to run Ansible inside the Docker container.
- Provide the necessary parameters such as the playbook name and inventory file path.
- Monitor the execution for any errors or warnings.

***docker exec <container\_name> ansible – playbook  
<playbook\_name> –i <inventory\_file>***

- Replace **<container\_name>** with the name of your Docker container, **<playbook\_name>** with the name of your playbook, and **<inventory\_file>** with the path to your inventory file.

## 7. Verify the Results:

- Check the output of the playbook execution for success or failure indicators.
- Perform any additional validation tests to ensure the integrity of the infrastructure.



```
1 ---
2 - name: Pull Docker Image from Docker Hub
3   hosts: localhost
4   remote_user: vnaik
5   # connection: local
6   become: false
7   tasks:
8     - name: Pull Docker Image
9       docker_image:
10         name: "naikvraj/miniproject"
11         source: pull
12         register: docker_pull_result
13
14     - name: Display Docker Pull Result
15       debug:
16         var: docker_pull_result
17
18     - name: Start Docker service
19       service:
20         name: docker
21         state: started
22
23     - name: Running container
24       shell: docker run -it -d --name miniProject naikvraj/miniproject
```

*Fig. 5.6.1 deploy.yml File*

```
ansible_host = localhost ansible_user=vnaik ansible_ssh_pass=Vnaik@5421
ansible_ssh_common_args = '-o StrictHostKeyChecking=no'
```

*Fig. 5.6.2 Inventory File*

## 5.7 EVENT TRIGGERED AUTOMATION

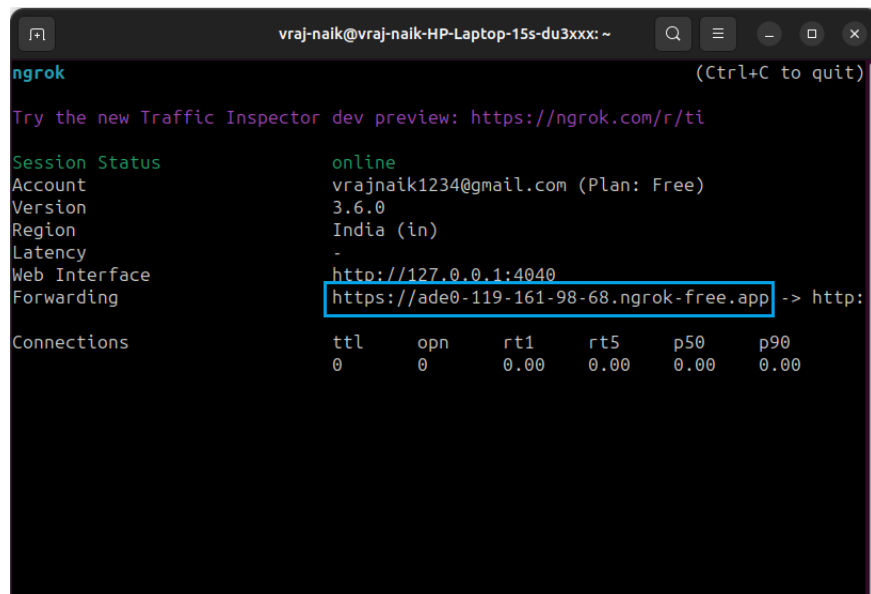
### 1. Setting up ngrok:

- Download and install ngrok:

Download ngrok from <https://ngrok.com/download> and install it on your system.

- Expose Jenkins Server:

Run ngrok http 8080 in your terminal to expose your local Jenkins server to the internet.



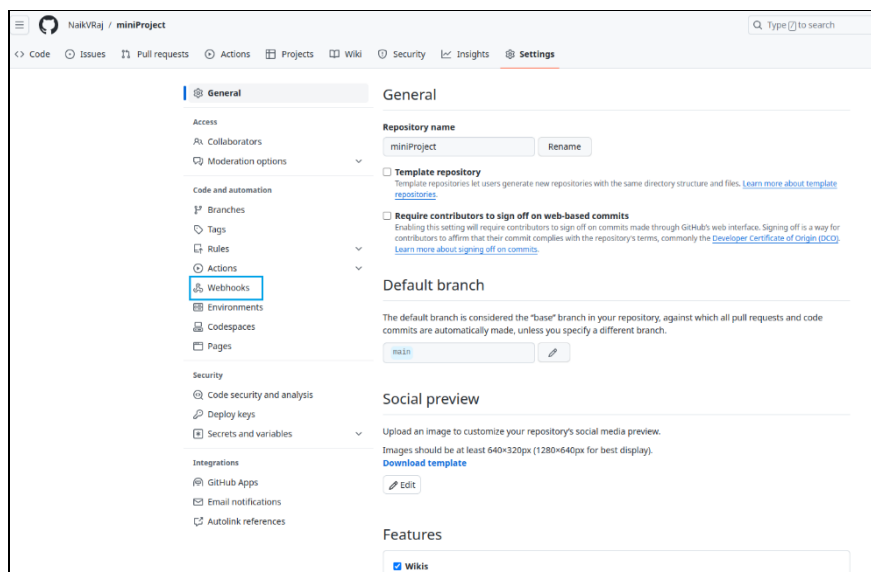
```
vraj-naik@vraj-naik-HP-Laptop-15s-du3xxx: ~  
ngrok (Ctrl+C to quit)  
Try the new Traffic Inspector dev preview: https://ngrok.com/r/ti  
Session Status      online  
Account             vrajnaik1234@gmail.com (Plan: Free)  
Version             3.6.0  
Region              India (in)  
Latency              -  
Web Interface       http://127.0.0.1:4040  
Forwarding           https://ade0-119-161-98-68.ngrok-free.app -> http:  
Connections  
ttn    opn    rt1    rt5    p50    p90  
0      0      0.00  0.00  0.00  0.00
```

#### 5.7.1 ngrok payload URL

### 2. Setting up GitHub Webhook:

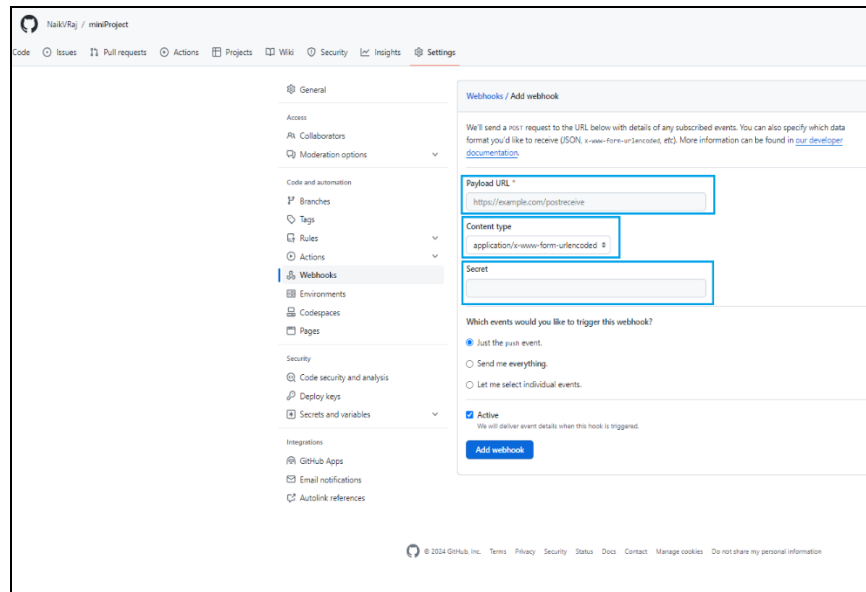
- Create a Webhook:

Go to your GitHub repository's settings and navigate to the "Webhooks" section.



#### 5.7.2 GitHub Webhook

- Click on "Add webhook" or "Create webhook".  
Provide the ngrok HTTPS URL as the Payload URL, followed by /github-webhook/.
- Provide public URL as specified in above ss of ngrok.
- Configure the webhook to trigger on events like "Pushes" or "Pull requests".  
Save the webhook configuration.



### 5.7.3 Add Webhook

## 3. Setting up Jenkins:

- Install Required Plugins:  
In Jenkins, install the "GitHub Integration Plugin" and any other necessary plugins.
- These plugins already installed along with pipeline plugins in earlier steps.
- Configure Jenkins Job:  
Create or select the Jenkins job you want to trigger.  
Enable the option for "GitHub hook trigger for GITScm polling" in the job configuration.

Quiet period [?](#)

5

SCM checkout retry count

0

☐ Restrict project naming

Jenkins Location

Jenkins URL [?](#)

https://ae71-119-161-98-68.ngrok-free.app/

System Admin e-mail address [?](#)

Jenkins-Master <vrajnalk1234@gmail.com>

Serve resource files from another domain

Resource Root URL [?](#)

Without a resource root URL, resources will be served from the Jenkins URL with Content-Security-Policy set.

Global properties

☐ Disable deferred wipeout on this node [?](#)

☐ Environment variables

Save

Apply

5.7.4 Jenkins Configuration

Dashboard > myProject-Calculator > Configuration

Configure

General

Advanced Project Options

Pipeline

☐ Pipeline speed/availability override [?](#)

☐ Preserve statuses from completed builds [?](#)

☐ This project is parameterized [?](#)

☐ Throttle builds [?](#)

Build Triggers

☐ Build after other projects are built [?](#)

☐ Build periodically [?](#)

☒ GitHub hook trigger for GIT/GITLAB [?](#)

☐ Poll SCM [?](#)

☐ Quiet period [?](#)

☐ Trigger builds remotely (e.g., from scripts) [?](#)

Advanced Project Options

Advanced [v](#)

Pipeline

Definition

Pipeline script [v](#)

Script [?](#)

```
1 // @pipeline {
2   // name: my-project
3   // description: 'A simple pipeline'
4   // author: 'me'
5   // triggers: ['poll-scm']
6   //
7   // Stage 1: Clone the repository
8   stage('Clone') {
9     // checkout source code from git repository
10    checkout('https://github.com:myProject.git')
11    // checkout source code from git repository
12    checkout('https://github.com:myProject.git')
13    // checkout source code from git repository
14    checkout('https://github.com:myProject.git')
15    // checkout source code from git repository
16    checkout('https://github.com:myProject.git')
17  }
18  // Stage 2: Build the project
19  stage('Build') {
20    // Build the project
21    build()
22  }
23 }
```

☒ Use Groovy Sandbox [?](#)

[Pipeline Syntax](#)

Save

Apply

5.7.5 Jenkins Github Trigger

42 | Page

## 6. CONCLUSION

The development journey of the scientific calculator program, coupled with the integration of a DevOps toolchain, illuminates the power of automation and collaboration in software development. Through the structured pipeline encompassing version control, testing, building, continuous integration, containerization, and deployment, the project exemplifies how DevOps practices streamline workflows and enhance productivity.

Git and GitHub have served as reliable companions for version control management, facilitating seamless collaboration and version tracking. The utilization of JUnit for testing ensures the robustness and reliability of the calculator program, validating its functionalities with rigor.

Maven's role in building the application has been instrumental, simplifying the compilation and packaging processes and enhancing efficiency. By integrating Jenkins for continuous integration, the project automates build and deployment processes, reducing manual intervention and promoting agility.

The adoption of Docker for containerization has empowered the team to achieve consistency and portability in deploying the application across various environments. Ansible's automation capabilities have streamlined configuration management and deployment, ensuring reliability and consistency.

Overall, the project highlights the transformative impact of DevOps methodologies in enhancing collaboration, accelerating development cycles, and improving deployment reliability. By embracing automation and integration, developers can focus more on innovation, thereby delivering high-quality software solutions efficiently and effectively.

## 7. REFERENCES

- [1] <https://www.geeksforgeeks.org/introduction-to-devops/>
- [2] <https://www.javatpoint.com/devops>
- [3] <https://www.codecademy.com/learn/introduction-to-dev-ops>
- [4] <https://www.happiestminds.com/whitepapers/devops.pdf>
- [5] [https://athena.ecs.csus.edu/~buckley/CSc233/DevOps\\_for\\_Dummies.PDF](https://athena.ecs.csus.edu/~buckley/CSc233/DevOps_for_Dummies.PDF)
- [6] <https://www.jenkins.io/doc/book/installing/linux/>
- [7] [https://www.tutorialspoint.com/docker/docker\\_tutorial.pdf](https://www.tutorialspoint.com/docker/docker_tutorial.pdf)
- [8] <https://cs50.harvard.edu/hls/2020/winter/seminars/Git%20and%20GitHub.pdf>
- [9] <https://opensource.com/resources/what-ansible>
- [10] <https://codetoimage.com/>

\*\*\*\*\*

**GitHub Repository:** <https://github.com/NaikVRaj/miniProject>

**Docker Hub Repository:** <https://hub.docker.com/r/naikvraj/miniproject>