DevOps Project

MoneyMetrics – The Financial Calculator

Name : Dhayanidhi S

Reg no : 23BIT0214

Email : dhayaisro09@gmail.com

Project Statement :

'Create a complete DevOps CI/CD Pipeline with Jenkins for a Use Case (different from the Calculator application) by using GIT, Maven, Puppet/Ansible/Terraform, Docker, JUnit, Graphite, and Grafana. You can add more tools if you want.'

Overview :

This project automates the complete software delivery lifecycle—including build, test, dockerization, deployment, monitoring, and rollback—for a CLI-based Java Financial Calculator. The pipeline is triggered automatically using SCM polling in Jenkins, which checks for new commits every 2 minutes and initiates a build only if changes are detected. Jenkins orchestrates the entire process, while Ansible executes modular roles defined in a centralized playbook to carry out tasks such as Maven-based compilation, JUnit testing, Docker image creation, and deployment to a Kubernetes (Minikube) cluster. Real-time monitoring is implemented using Collectd (for data collection), Graphite (for time-series storage), and Grafana (for visualization dashboards). Additionally, an automatic rollback mechanism is built into the pipeline to revert to the last successful version if any stage fails. This tightly integrated setup ensures fast, reliable, and fully automated software delivery with minimal manual intervention.

📚 Table of Contents

🛠️ Project Components

1. 🔢 Java Financial Calculator

- A CLI-based application developed in Java.
- Accepts user input via terminal.
- Computes interest-related financial values.

2. ✅ JUnit Test Cases

- Unit tests verify all financial functions.
- Test failure is **intentionally triggered** to simulate rollback.

3. 💉 Maven Build & Test

- Jenkins runs

```
mvn clean package
```

- If any JUnit test fails, the pipeline does **not stop immediately**. Instead, Ansible marks a flag for rollback.

🐳 Docker Structure

- The Dockerfile builds a Java app image:

Dockerfile

- Two image tags maintained:
  - latest
  - previous (used for rollback)

📜 Jenkins CI/CD Pipeline Stages

1. **SCM Checkout**
   - Pulls the latest code from your Git repo (local path or GitHub).
   - SCM Polling or Webhook triggers pipeline automatically on code push.
2. **Build**
   - Compiles code using mvn package.
3. **Test**
   - Runs JUnit tests.
   - If any test fails, Ansible creates a rollback trigger file.

4. **Dockerize**
    - Copies Dockerfile to workspace.
    - Builds a Docker image tagged with build number (e.g., financial-calc:6) and latest.
    - Tags previous image as previous for rollback.
5. **Deploy**
    - Removes any running financial-calc container.
    - Runs the new image.
    - Waits for 5 seconds to allow startup.
6. **Monitor**
    - Starts Grafana + Graphite stack using Docker Compose.
    - Visualization of CPU, memory, app logs, etc.
7. **Rollback (if needed)**
    - If rollback_trigger file is detected, rollback role is executed.
    - Removes broken container.
    - Replaces with image tagged as previous.

📦 Ansible Roles

The project has modular Ansible roles:

Role	Purpose
build	Maven build and packaging
test	Run JUnit tests and create rollback trigger on failure
dockerize	Build Docker image and tag it
deploy	Remove old container and deploy new one
monitor	Start Grafana and Graphite with Docker Compose
rollback	Check for test failure and perform image rollback

All roles are included via site.yml master playbook with tag-based execution.

📊 Monitoring Stack

Tools:

- **Collectd** – Gathers metrics.

- **Graphite** – Time-series database.
- **Grafana** – Visual dashboards.

Setup:

- Runs via Docker Compose (monitor/docker-compose.yml).
- Started through Ansible during the monitor role.

🔁 Rollback Workflow

Condition	Action
Tests and Build pass	Continue with deployment
Tests or Build fail	- Create rollback_trigger file - Rollback role triggers
Rollback	Remove new image run previously working image

Rollback is fully automated and **resilient**.

🔐 Permissions & Ownership

- Git repo is made a safe directory using:

  ```
  git config --global --add safe.directory /path/to/repo
  ```

- Ownership is managed so both jenkins and dhayanidhi-s users can access files and Docker socket.

⏱️ SCM Polling vs Webhook

- SCM Polling is configured with * * * * * to check every minute.
- You can eliminate polling delay by using a **GitHub webhook** to notify Jenkins instantly on push.

✅ Success Criteria

- Jenkins automatically detects code push.
- If build or test fails, rollback is successful.
- Docker container always reflects last known working build.
- Metrics are viewable in Grafana dashboard.

📦 Directory Structure (Simplified)

```
project_root/
├── ansible_project/
│   ├── roles/
│   │   ├── build/
│   │   ├── test/
│   │   ├── dockerize/
│   │   ├── deploy/
│   │   ├── monitor/
│   │   └── rollback/
│   ├── inventory
│   ├── site.yml
├── Dockerfile
├── src/
├── target/
├── Jenkinsfile
├── Jar File
└── pom.xml
```

📈 Learning Outcomes

- Understand CI/CD fundamentals using Jenkins.
- Build Ansible playbooks and roles for automation.
- Learn Docker image creation and container deployment.
- Automate rollback mechanisms using Ansible.
- Integrate monitoring into a DevOps pipeline using Grafana.
- Hands-on experience in connecting all DevOps stages into a seamless workflow.

🧠 Possible Future Enhancements

- Convert Java CLI app into a REST API using Spring Boot.
- Use GitHub Webhooks instead of SCM polling for real-time trigger.
- Integrate Slack/Email notifications for pipeline events.
- Add database integration for persistent data storage.
- Deploy in Kubernetes (Minikube or cloud-managed) for scalability

---

💡 USE CASE

The Financial Calculator Project

The **Financial Calculator** is a comprehensive Java-based application designed to perform a wide range of financial computations that are commonly needed by individuals, accountants, businesses, students, and financial analysts. It serves as a powerful command-line tool that simplifies complex mathematical financial operations by accepting user inputs and providing accurate results within seconds. The calculator is built with clean, modular Java code and offers a reliable, easy-to-use interface, making it suitable for both educational purposes and real-world financial decision-making.

🔢 Key Functionalities (Performs 10+ Operations):

1. **Simple Interest (SI):** Calculates interest based on principal, rate, and time.
2. **Compound Interest (CI):** Computes interest on principal + accumulated interest over time.
3. **EMI (Equated Monthly Installment):** Calculates fixed monthly payment for a loan.
4. **Future Value (FV):** Determines the value of an investment after a period at a fixed interest rate.
5. **Present Value (PV):** Calculates today's value of a future amount of money.
6. **Loan Affordability Check:** Computes maximum loan based on EMI and interest rate.
7. **Savings Growth Estimation:** Predicts savings over time with periodic deposits.
8. **Investment Return Analysis:** Calculates returns from various investment scenarios.
9. **Break-even Point Estimation:** Determines when an investment will break even.
10. **Interest Rate Conversion:** Converts annual interest rates to monthly or daily and vice versa.

🎯 Real-World Use Cases & Benefits

This calculator is especially useful for **bank employees**, **loan officers**, **investors**, **financial advisors**, **students studying commerce or finance**, and **individuals managing personal finances**. It allows quick assessments of loans, savings, and investments, making budgeting and financial planning more efficient. For example, a student can estimate how much they'll owe after taking a student loan, or a home buyer can compute EMI options before approaching a bank. Businesses can evaluate investment proposals or compare loan options to select the most cost-effective one.

The Financial Calculator eliminates manual errors, saves time, and promotes financial literacy by helping users understand how financial parameters interact — making it a vital tool in day-to-day financial decision-making.

📦 Creating project repository

Install GIT client and server on the same host .

ssh git@localhost – Establishes a secure SSH connection to the local Git server for performing Git operations.

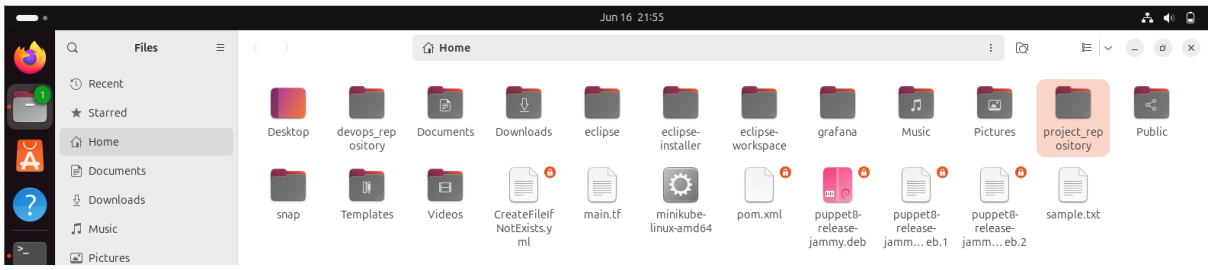
git init --bare project_repository.git – Creates a bare repository to act as the central remote for code versioning and CI/CD integration.

Bare repositories store only version history and are ideal for collaborative or automated workflows (no working directory).

git clone git@localhost:project_repository.git – Clones the bare repo into a working directory for development or Jenkins builds.

This setup is essential for pushing code, polling for changes, and triggering the DevOps pipeline automatically

```
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ ssh git@localhost
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.11.0-26-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

12 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Mon Jun 16 21:50:28 2025 from 127.0.0.1
git@dhayanidhi-s-VirtualBox:~$ git init --bare project_repository.git
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/git/project_repository.git/
git@dhayanidhi-s-VirtualBox:~$ exit
logout
Connection to localhost closed.
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ git clone git@admin:devops_repository.git
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ git clone git@localhost:project_repository.git
Cloning into 'project_repository'...
warning: You appear to have cloned an empty repository.
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$
```



Project_repository created .

🚀 Setting Up Maven Project

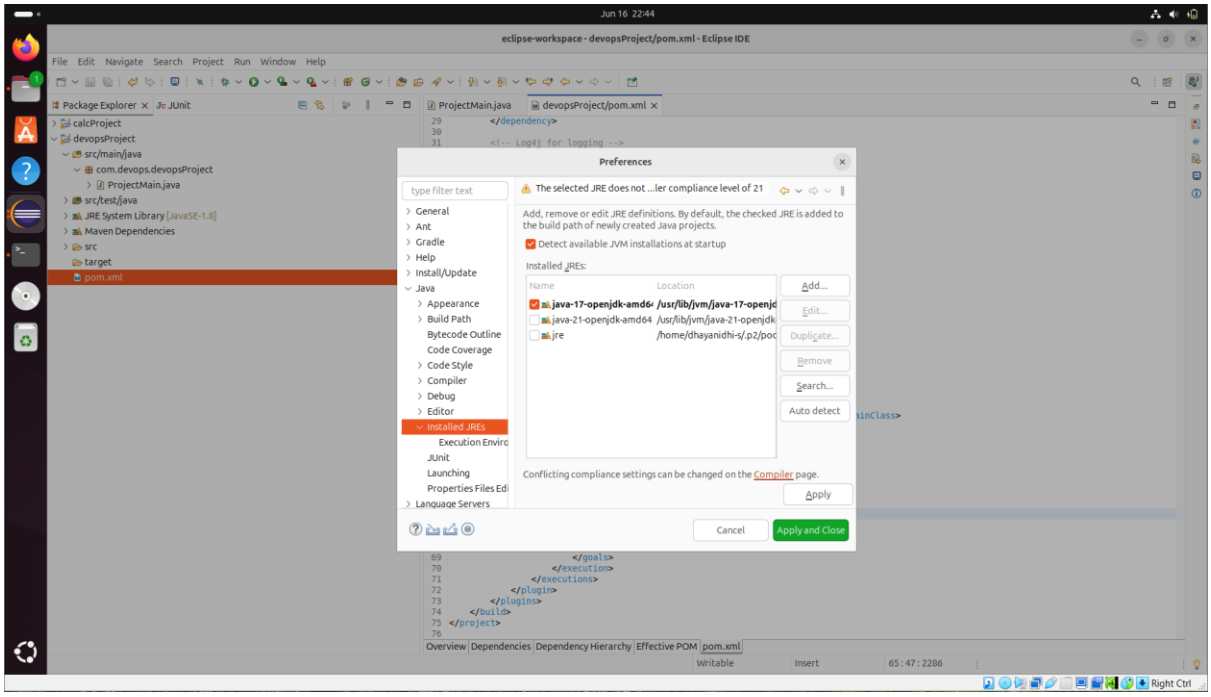
Creating a modular Maven project that will later be used in the CI/CD pipeline.

🛠️ 1. Installed Java JDK (Prerequisite for Eclipse)

📥 2. Downloaded and Installed Eclipse IDE

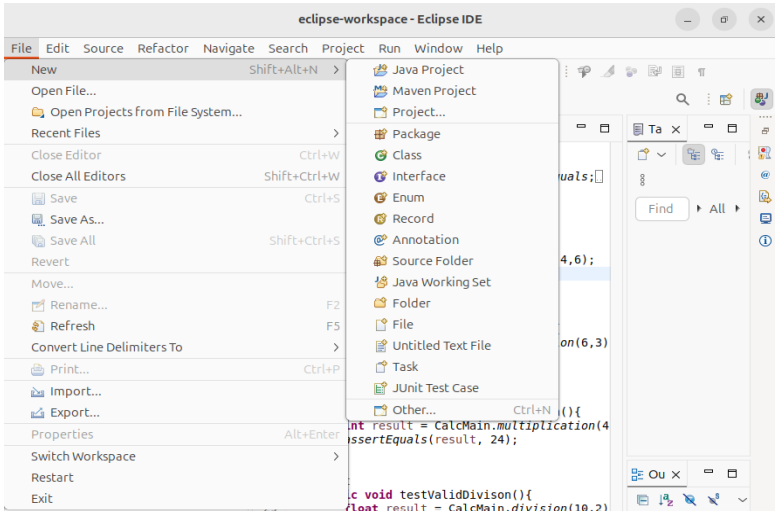
⚙️ 3. Configuring Eclipse Environment

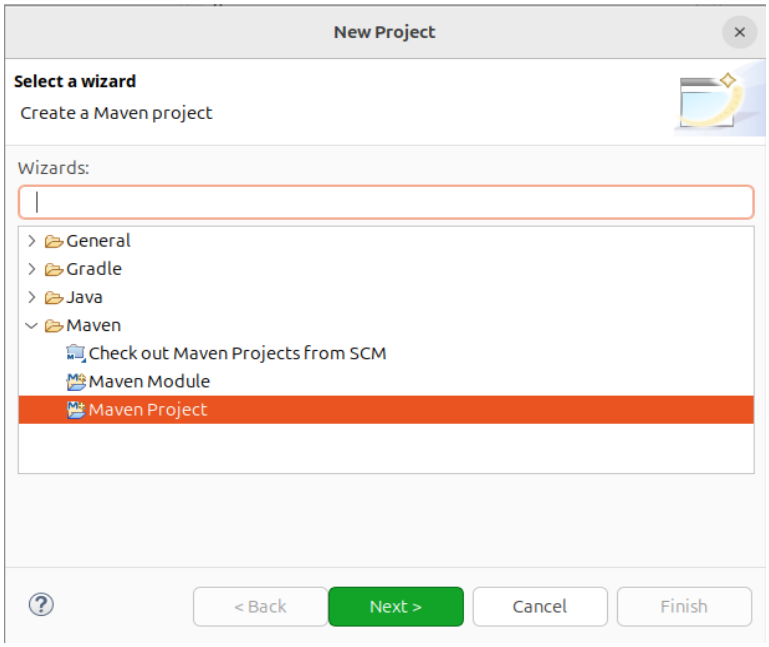
- Ensuring Eclipse is using the correct JDK via:
  Window → Preferences → Java → Installed JREs
- Selecting the path to **OpenJDK 17**.

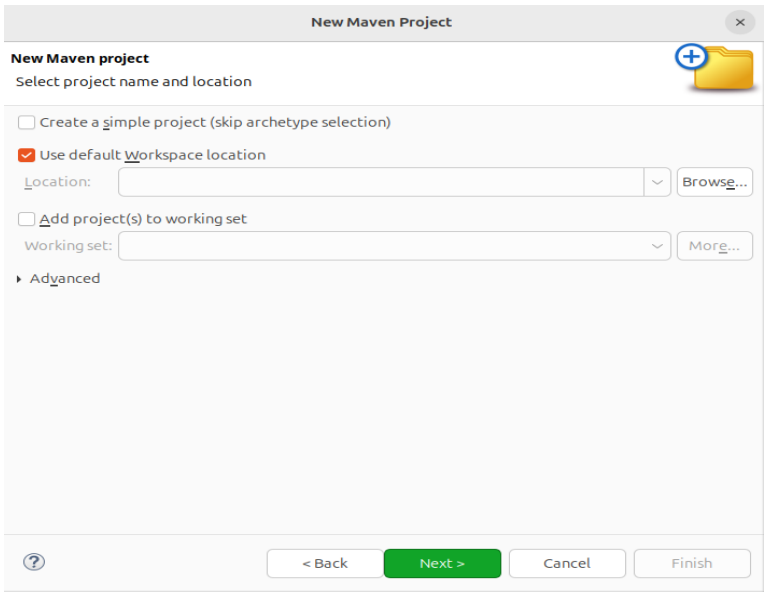


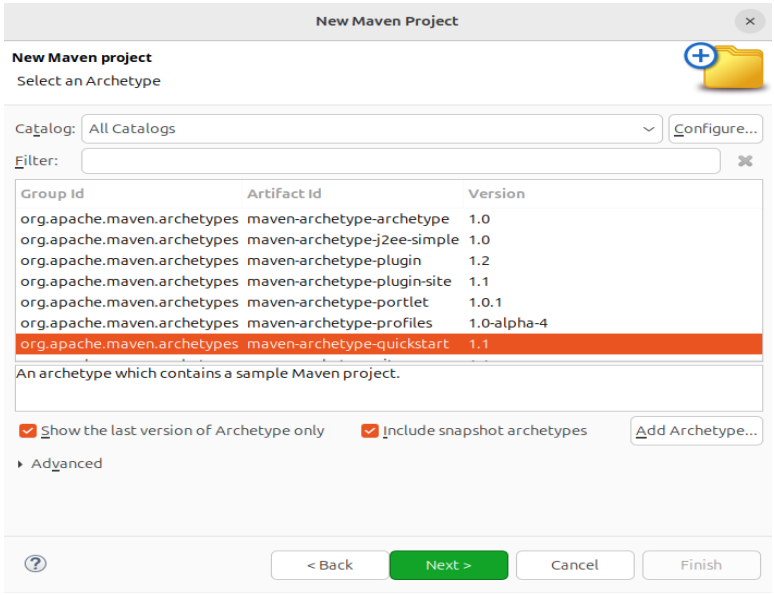
📦 4. Creating a Maven Project in Eclipse

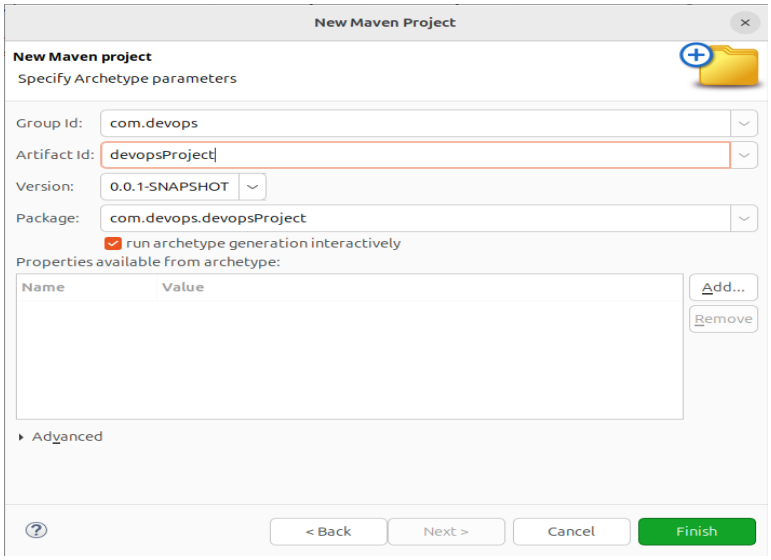
- File → New → Project → Maven → Maven Project
- Choose **archetype**, like maven-archetype-quickstart
- Fill in:
  - **Group ID**: com.devops
  - **Artifact ID**: devopsProject
- Click **Finish** to create the project structure.

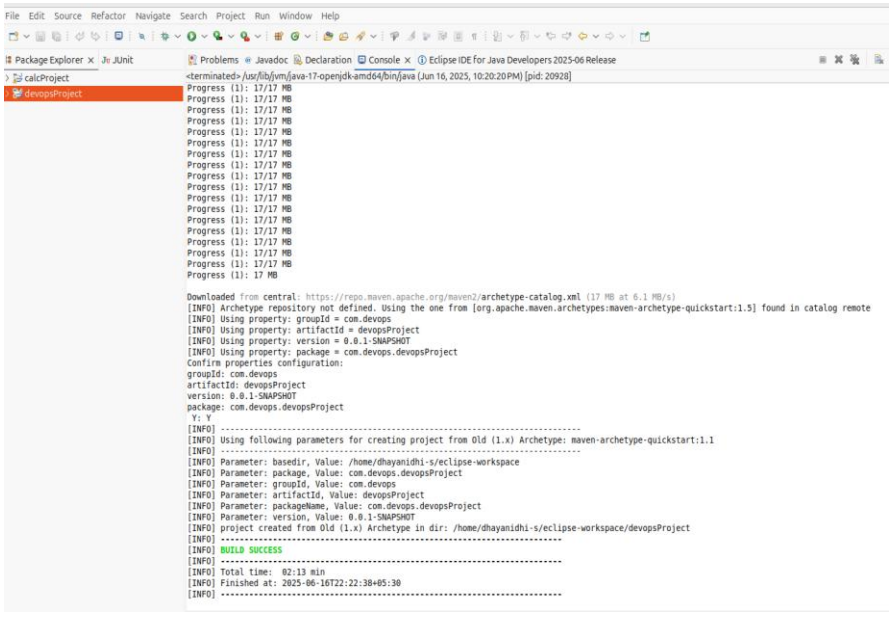

eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
New Shift+Alt+N
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Close All Editors Shift+Ctrl+W
Save Ctrl+S
Save As...
Save All Shift+Ctrl+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
Switch Workspace
Restart
Exit
Java Project
Maven Project
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
Task
JUnit Test Case
Other... Ctrl+N
Find
All


New Project
Select a wizard
Create a Maven project
Wizards:
General
Gradle
Java
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
< Back
Next >
Cancel
Finish


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Cancel
Finish

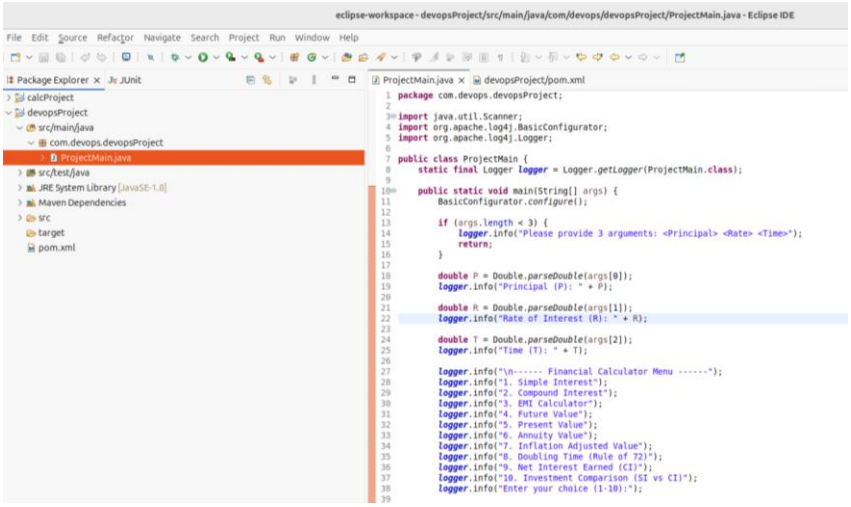

New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter:
Group Id
Artifact Id
Version
org.apache.maven.archetypes maven-archetype-archetype 1.0
org.apache.maven.archetypes maven-archetype-j2ee-simple 1.0
org.apache.maven.archetypes maven-archetype-plugin 1.2
org.apache.maven.archetypes maven-archetype-plugin-site 1.1
org.apache.maven.archetypes maven-archetype-portlet 1.0.1
org.apache.maven.archetypes maven-archetype-profiles 1.0-alpha-4
org.apache.maven.archetypes maven-archetype-quickstart 1.1
An archetype which contains a sample Maven project.
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced
< Back
Next >
Cancel
Finish


New Maven Project
New Maven project
Specify Archetype parameters
Group Id: com.devops
Artifact Id: devopsProject
Version: 0.0.1-SNAPSHOT
Package: com.devops.devopsProject
run archetype generation interactively
Properties available from archetype:
Name
Value
Add...
Remove
Advanced
< Back
Next >
Cancel
Finish


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
calcProject
devopsProject
Problems
Javadoc
Declaration
Console
Eclipse IDE for Java Developers 2025-06 Release
<terminated> /usr/lib/jvm/java-17-openjdk-amd64/bin/java (Jun 16, 2025, 10:20:20 PM) [pid: 20928]
Progress (1): 17/17 MB
Progress (1): 17 MB
Downloaded from central: https://repo.maven.apache.org/maven2/archetype-catalog.xml (17 MB at 6.1 MB/s)
[INFO] Archetype repository not defined. Using the one from [org.apache.maven.archetypes:maven-archetype-quickstart:1.5] found in catalog remote
[INFO] Using property: groupId = com.devops
[INFO] Using property: artifactId = devopsProject
[INFO] Using property: version = 0.0.1-SNAPSHOT
[INFO] Using property: package = com.devops.devopsProject
Confirm properties configuration:
groupId: com.devops
artifactId: devopsProject
version: 0.0.1-SNAPSHOT
package: com.devops.devopsProject
Y: Y
[INFO] --
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.1
[INFO] --
[INFO] Parameter: basedir, Value: /home/dhayanidhi-s/eclipse-workspace
[INFO] Parameter: package, Value: com.devops.devopsProject
[INFO] Parameter: groupId, Value: com.devops
[INFO] Parameter: artifactId, Value: devopsProject
[INFO] Parameter: packageName, Value: com.devops.devopsProject
[INFO] Parameter: version, Value: 0.0.1-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/dhayanidhi-s/eclipse-workspace/devopsProject
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 02:13 min
[INFO] Finished at: 2025-06-16T22:22:38+05:30
[INFO] --

5. Developing Financial Calculator App

- Inside src/main/java/com.devops.devopsProject, create ProjectMain.java with logic for:
    - Simple Interest , Compound Interest , EMI , Net Interest Earned and more as needed.
    - Accept command-line arguments and log output using **Log4j**
    - **Used java as language**

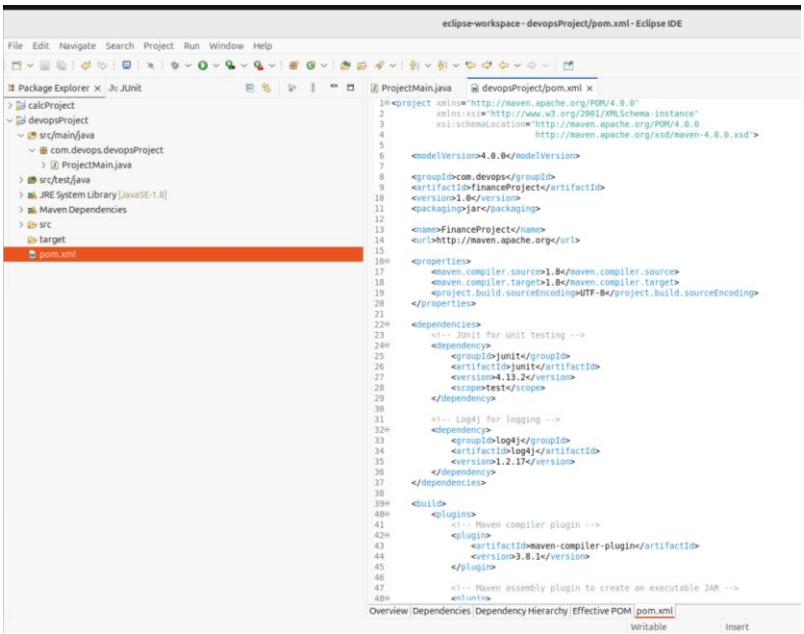


---

6. Configuring pom.xml

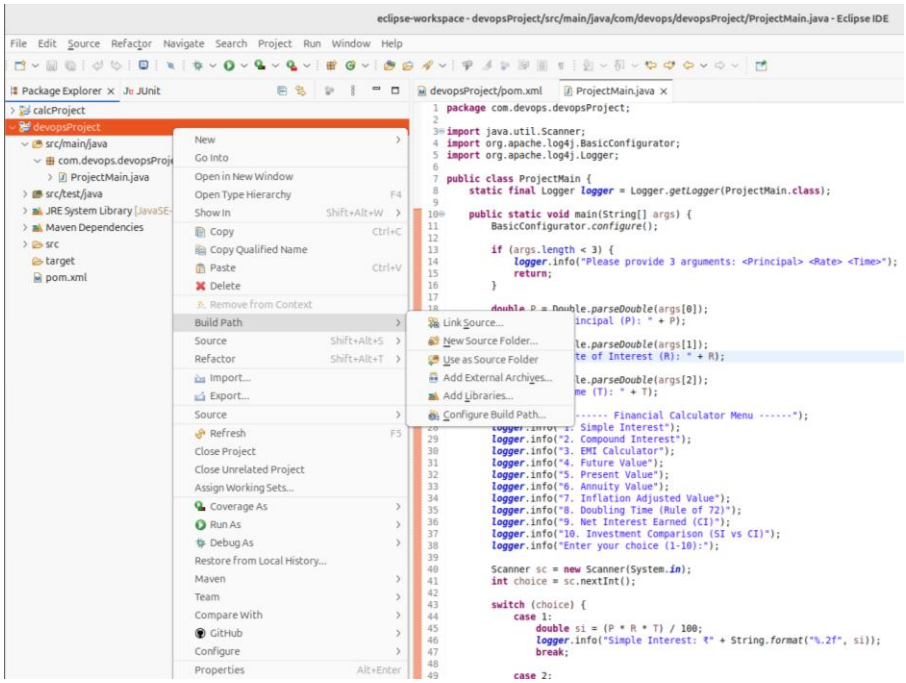
Add Required Maven Dependencies which Include:

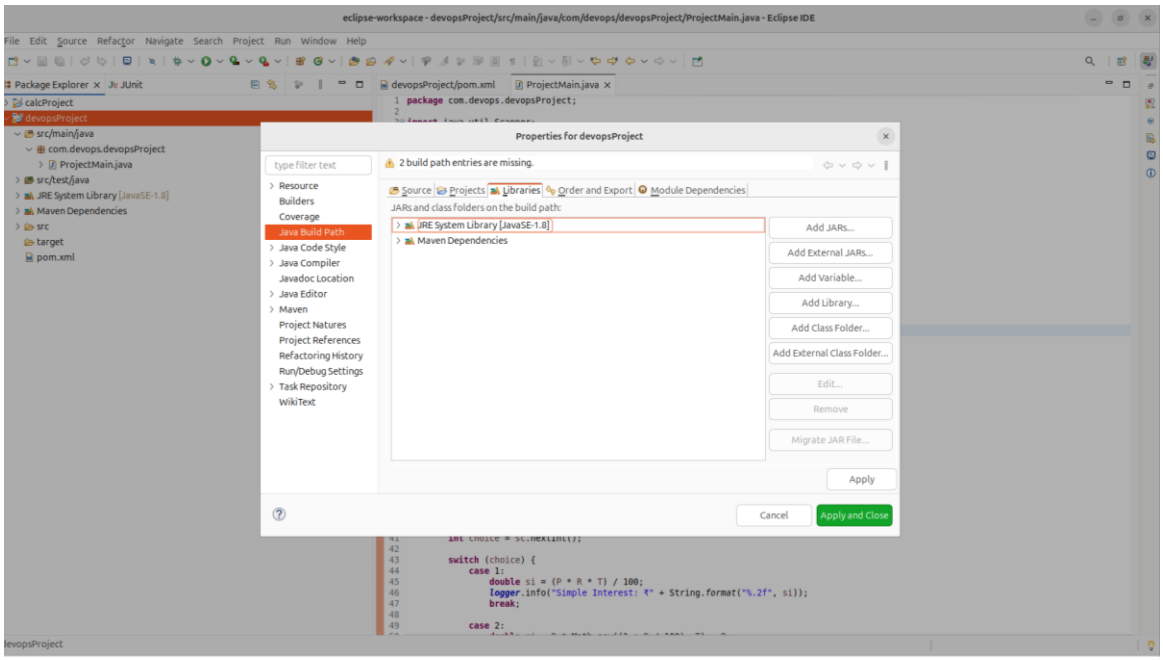
- junit:junit:4.12 for testing
- log4j:log4j:1.2.17 for logging
- maven-assembly-plugin for creating a JAR with dependencies

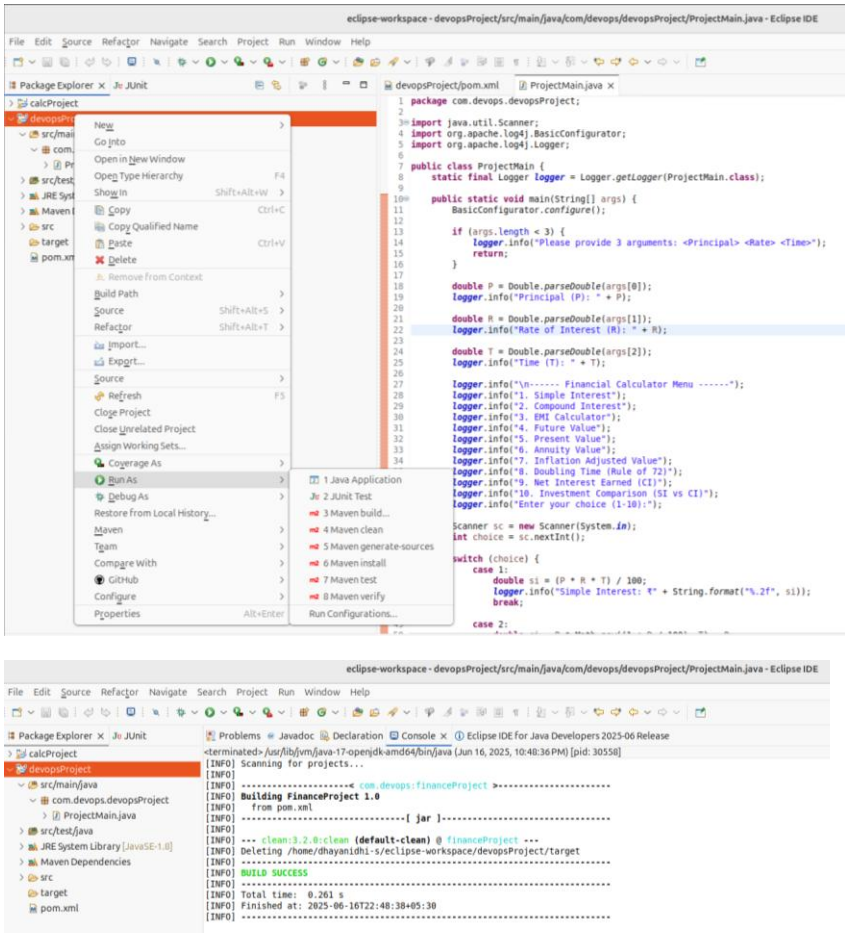


🚀 7. Building and Package the App

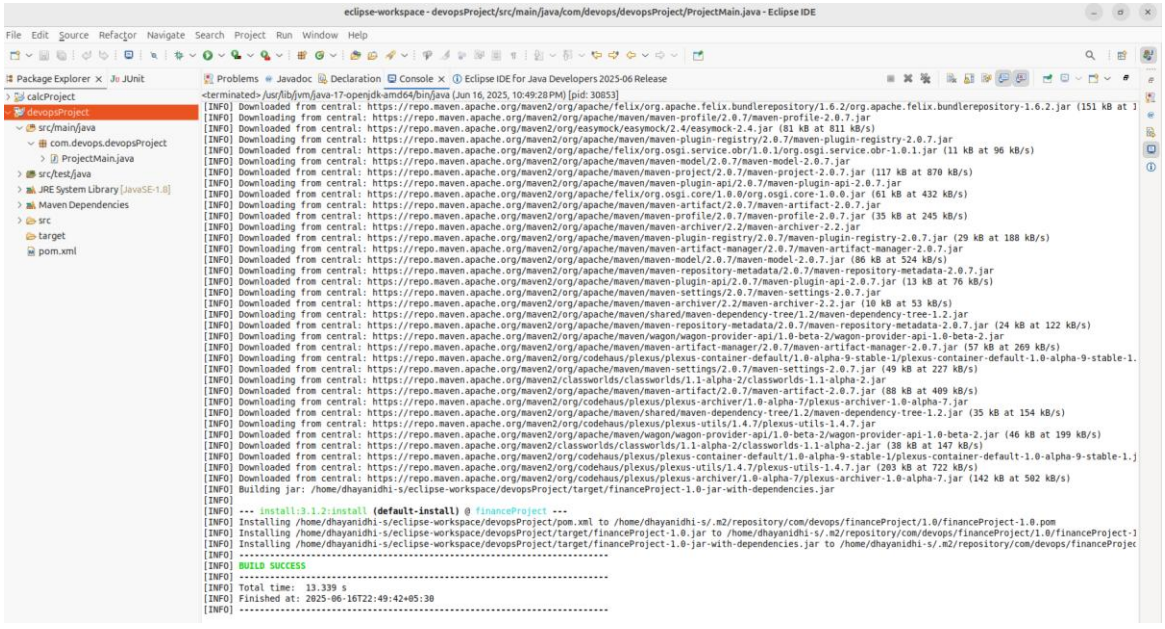
- Configure Build path
- Right-click project → Run As → Maven clean
- Then Run As → Maven install
- This generates a runnable **JAR file** in the target/ directory.



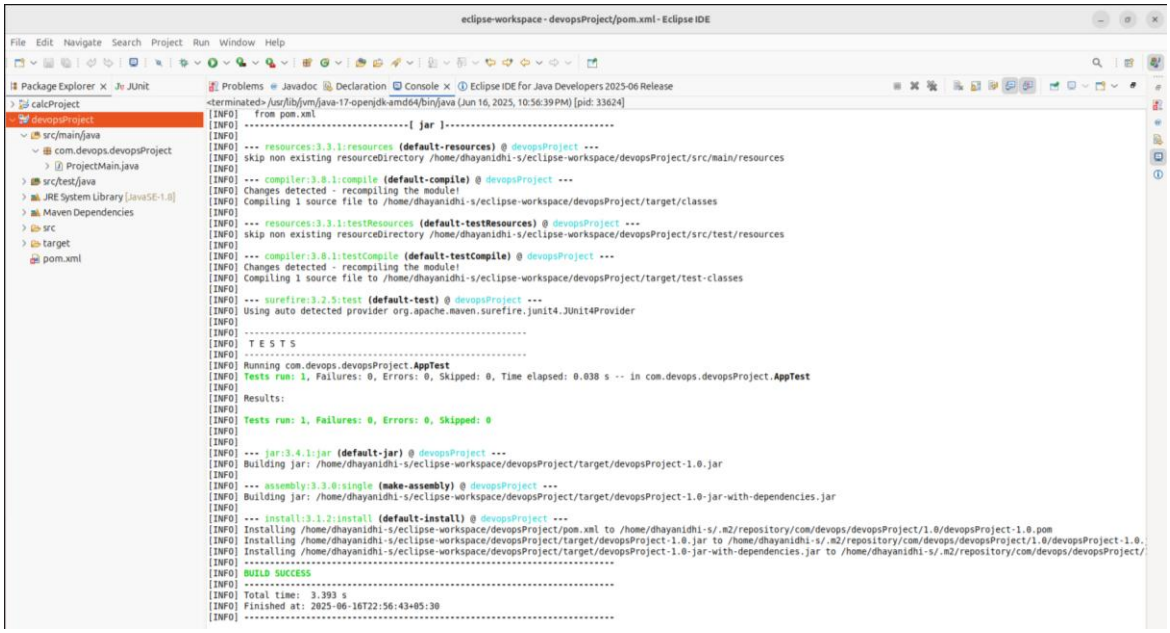


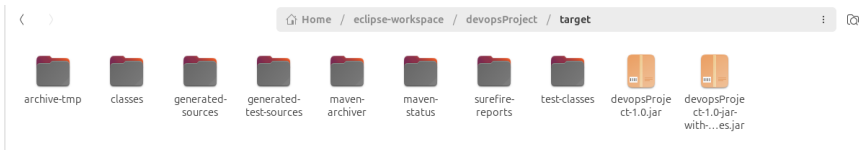


Maven clean completed successfully



Maven install completed successfully





Runnable **JAR file is successfully created on the target directory .**

🚀 8. Run the Executable JAR

Test the app by running the executable JAR :

```
java -jar target/finance-calc-1.0-jar-with-dependencies.jar 10000 7.5 3 1
```

```
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ java -jar /home/dhayanidhi-s/eclipse-workspace/devopsProjec
t/target/devopsProject-1.0-jar-with-dependencies.jar "10000" "7.5" "3" "1"
0 [main] INFO com.devops.devopsProject.ProjectMain  - Principal (P): 10000.0
0 [main] INFO com.devops.devopsProject.ProjectMain  - Rate of Interest (R): 7.5
0 [main] INFO com.devops.devopsProject.ProjectMain  - Time (T): 3.0
0 [main] INFO com.devops.devopsProject.ProjectMain  -
------ Financial Calculator Menu ------
0 [main] INFO com.devops.devopsProject.ProjectMain  - 1. Simple Interest
0 [main] INFO com.devops.devopsProject.ProjectMain  - 2. Compound Interest
1 [main] INFO com.devops.devopsProject.ProjectMain  - 3. EMI Calculator
1 [main] INFO com.devops.devopsProject.ProjectMain  - 4. Future Value
1 [main] INFO com.devops.devopsProject.ProjectMain  - 5. Present Value
1 [main] INFO com.devops.devopsProject.ProjectMain  - 6. Annuity Value
1 [main] INFO com.devops.devopsProject.ProjectMain  - 7. Inflation Adjusted Value
1 [main] INFO com.devops.devopsProject.ProjectMain  - 8. Doubling Time (Rule of 72)
1 [main] INFO com.devops.devopsProject.ProjectMain  - 9. Net Interest Earned (CI)
1 [main] INFO com.devops.devopsProject.ProjectMain  - 10. Investment Comparison (SI vs CI)
1 [main] INFO com.devops.devopsProject.ProjectMain  - Enter your choice (1-10):
38 [main] INFO com.devops.devopsProject.ProjectMain  - Simple Interest: ₹2250.00
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ sudo docker build -t devops_project:1.0 .
[+] Building 0.2s (1/1) FINISHED                                        docker:default
```

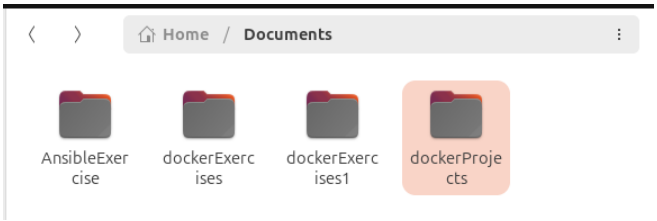
(Where 10000 7.5 3 2 are input arguments for operations)

🐳 Docker Setup & Dockerization of Java Financial Calculator

🔧 1. Install Docker on Ubuntu

👤 2. Add User to Docker Group

🗂️ 3. Create the dockerProjects Directory

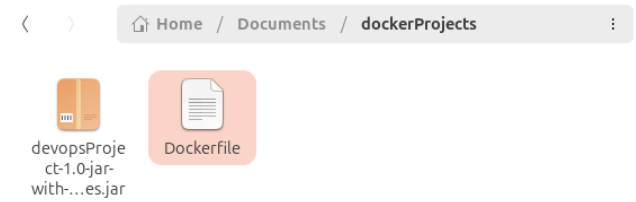


Created a dedicated folder for all Docker-related files, keeping things organized.

📦 4. Copy the JAR File into the Directory

Copied the compiled executable JAR (from the Maven project) into the Docker workspace for image creation.

🛠️ 5. Create the Dockerfile



Defines the Java environment, copies the app, and sets default execution parameters for your calculator.

👀 6. Confirm Dockerfile Content

Use this to double-check the contents of your Dockerfile before building to avoid syntax or path errors.

```
cat dockerProjects/Dockerfile
```

```
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ cd Documents/dockerProjects
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/Documents/dockerProjects$ cat Dockerfile
# Dockerfile for running java application
FROM anapsix/alpine-java
MAINTAINER devopsuser@gmail.com
COPY devopsProject-1.0-jar-with-dependencies.jar /home/devopsProject-1.0-jar-with-dependencies.jar
CMD ["java", "-jar", "/home/devopsProject-1.0-jar-with-dependencies.jar", "10000", "7.5", "3","2"]
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/Documents/dockerProjects$
```

🏗️ 7. Build the Docker Image

```
cd dockerProjects
docker build -t devops_project:1.0 .
```

Builds a Docker image from the Dockerfile. -t tags it for easy identification as devops_calc version 1.0.

🚀 8. Run the Docker Container

```
docker run devops_calc:1.0
```

Launches a container from the image and runs the financial calculator app with sample arguments (like Principal, Rate, Operation Type, and Time).

```
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ cd Documents/dockerProjects
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/Documents/dockerProjects$ sudo docker build -t devops_projec
t:1.0 .
[+] Building 3.2s (7/7) FINISHED                                                docker:default
 => [internal] load build definition from Dockerfile                                      0.0s
 => => transferring dockerfile: 336B                                                      0.0s
 => WARN: MaintainerDeprecated: Maintainer instruction is deprecated in favor of using label  0.0s
 => [internal] load metadata for docker.io/anapsix/alpine-java:latest                     2.6s
 => [internal] load .dockerignore                                                         0.0s
 => => transferring context: 2B                                                           0.0s
 => [internal] load build context                                                         0.0s
 => => transferring context: 487.68kB                                                     0.0s
 => CACHED [1/2] FROM docker.io/anapsix/alpine-java:latest@sha256:1d24bc352e07b84c073acfff8b  0.0s
 => [2/2] COPY devopsProject-1.0-jar-with-dependencies.jar /home/devopsProject-1.0-jar-with-  0.2s
 => exporting to image                                                                    0.1s
 => => exporting layers                                                                   0.1s
 => => writing image sha256:1479186483db6cff96dfc119b033c0ddbcda1c579ace47b8b27b54310d749762  0.0s
 => => naming to docker.io/library/devops_project:1.0                                     0.0s

 1 warning found (use docker --debug to expand):
 - MaintainerDeprecated: Maintainer instruction is deprecated in favor of using label (line 3)
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/Documents/dockerProjects$ sudo docker run devops_project:1.0
0 [main] INFO com.devops.devopsProject.ProjectMain  - Principal (P): 10000.0
1 [main] INFO com.devops.devopsProject.ProjectMain  - Rate of Interest (R): 7.5
1 [main] INFO com.devops.devopsProject.ProjectMain  - Time (T): 3.0
1 [main] INFO com.devops.devopsProject.ProjectMain  -
------ Financial Calculator Menu ------
1 [main] INFO com.devops.devopsProject.ProjectMain  - 1. Simple Interest
1 [main] INFO com.devops.devopsProject.ProjectMain  - 2. Compound Interest
1 [main] INFO com.devops.devopsProject.ProjectMain  - 3. EMI Calculator
1 [main] INFO com.devops.devopsProject.ProjectMain  - 4. Future Value
1 [main] INFO com.devops.devopsProject.ProjectMain  - 5. Present Value
1 [main] INFO com.devops.devopsProject.ProjectMain  - 6. Annuity Value
1 [main] INFO com.devops.devopsProject.ProjectMain  - 7. Inflation Adjusted Value
1 [main] INFO com.devops.devopsProject.ProjectMain  - 8. Doubling Time (Rule of 72)
1 [main] INFO com.devops.devopsProject.ProjectMain  - 9. Net Interest Earned (CI)
1 [main] INFO com.devops.devopsProject.ProjectMain  - 10. Investment Comparison (SI vs CI)
1 [main] INFO com.devops.devopsProject.ProjectMain  - Enter your choice (1-10):
54 [main] INFO com.devops.devopsProject.ProjectMain  - Compound Interest: ₹2422.97
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/Documents/dockerProjects$
```

Runs financial calculator JAR and prints the result for the provided inputs

🧪 JUnit Testing for Java Financial Calculator

🧩 1. Add JUnit Dependency in pom.xml

Open pom.xml file and ensure the following snippet is added inside the <dependencies> section:

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

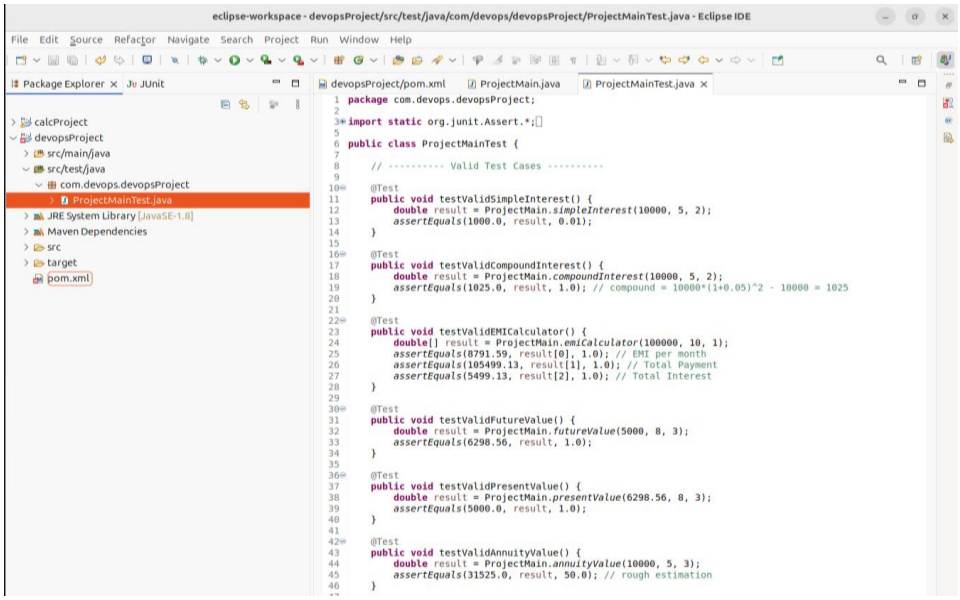
This imports the JUnit 4.12 framework, allowing you to write and run test cases for all calculator methods.

📁 2. Create Test Class CalcTest.java

Inside your Maven project, navigate to:
src/test/java → com.devops.calcProject
Then, create a new Java class named ProjectMainTest.java. This folder is dedicated for unit testing in Maven-based projects.

🧪 3. Write JUnit Test Cases

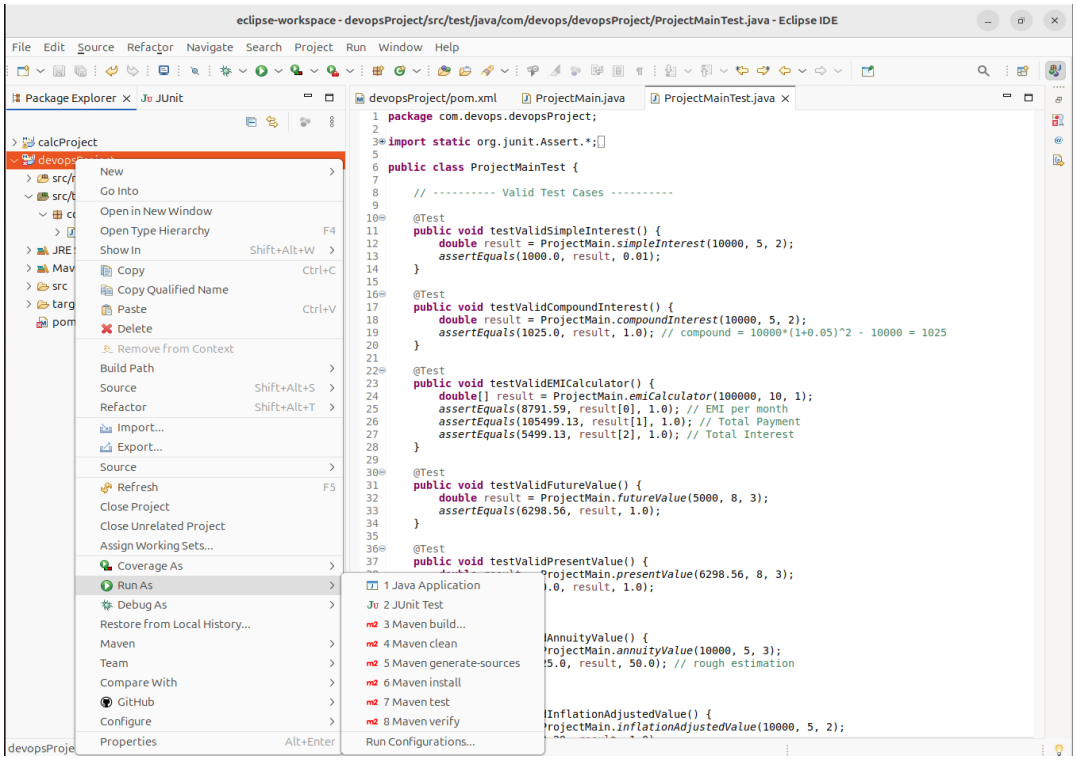
Paste JUnit test code into ProjectMainTest.java.



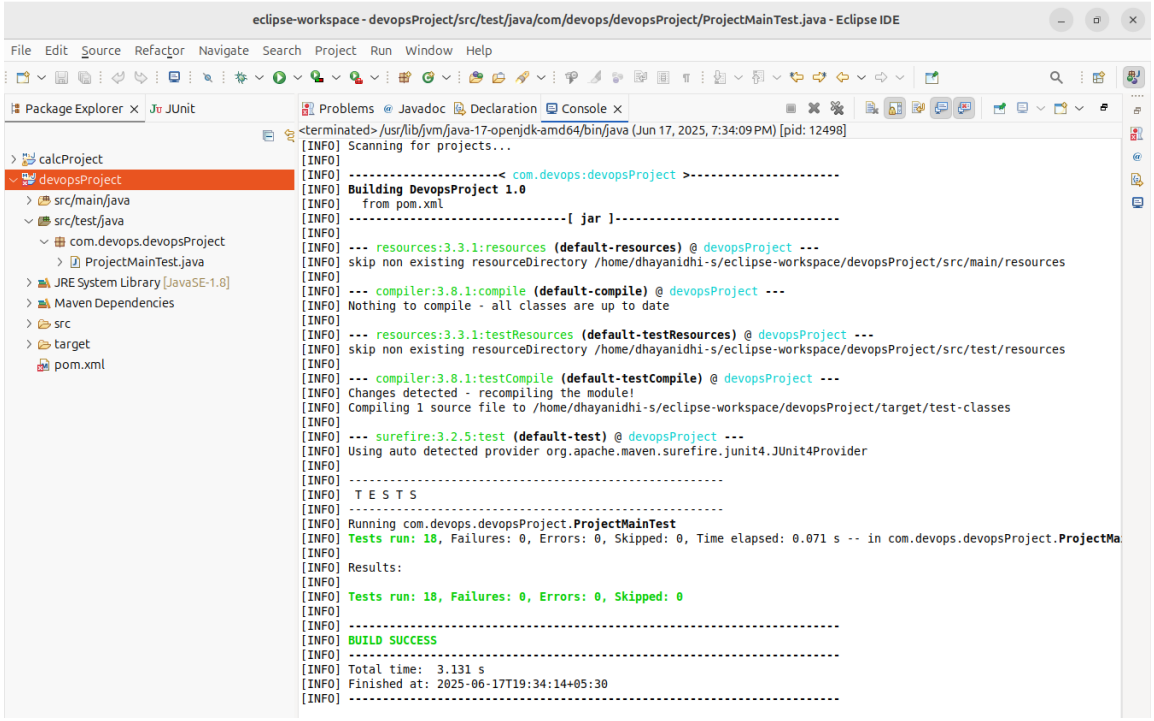
These test both **valid** and **invalid** cases for your calculator methods, ensuring correctness and catching bugs.

🧪 4. Run All Tests via Maven

Right-click on your project → **Run As** → **Maven test**



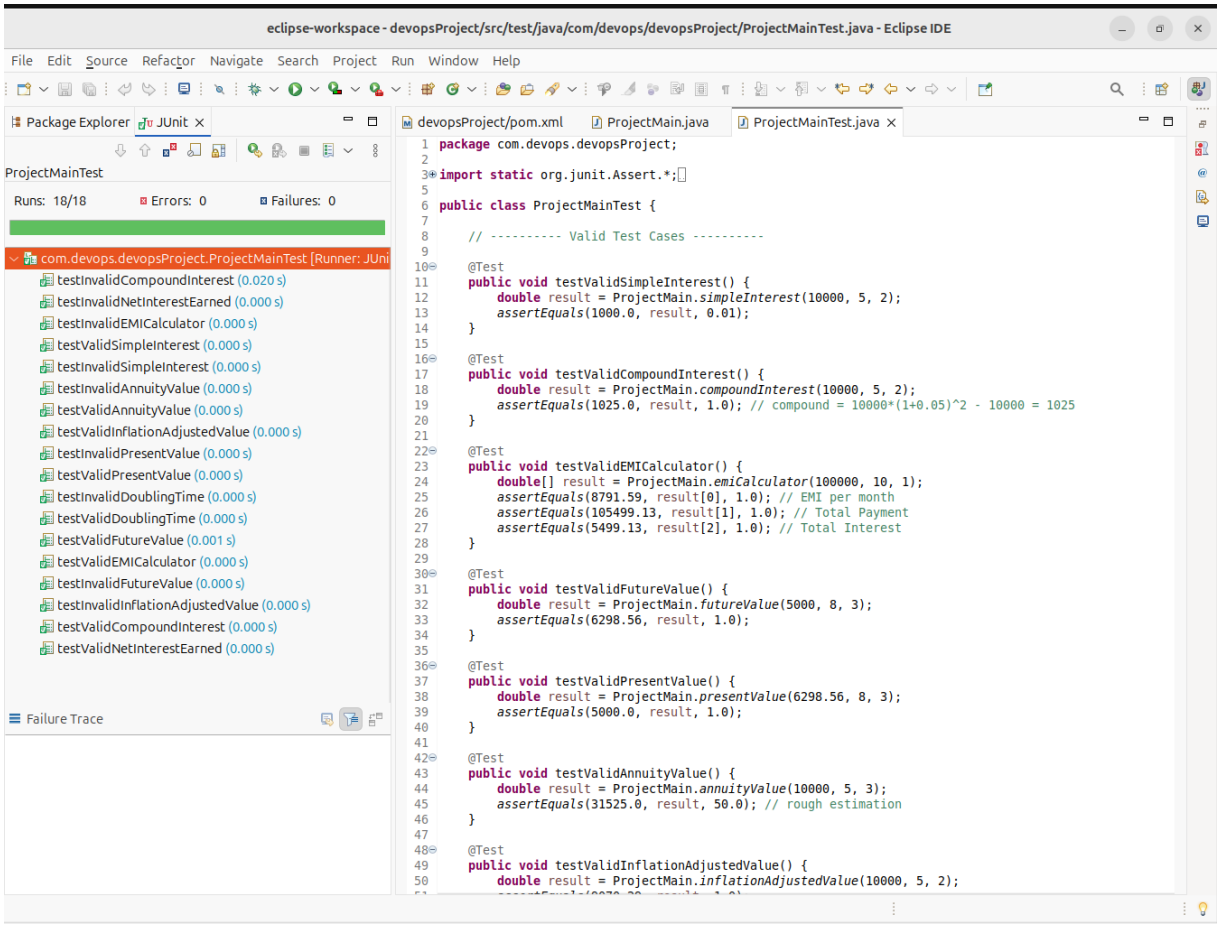
This command automatically finds and runs all test files under src/test/java and shows results in the console.



🔍 5. Run JUnit Test Class Individually (Optional)

To run just ProjectMainTest.java:

- Right-click on the class file
- Click **Run As → JUnit Test**
  This displays a graphical green/red bar interface indicating which tests passed or failed.



All Test Cases were successful .

☸️ Deploying Financial Calculator on Kubernetes with Minikube

---

🔧 1. Install Kubernetes (Minikube)

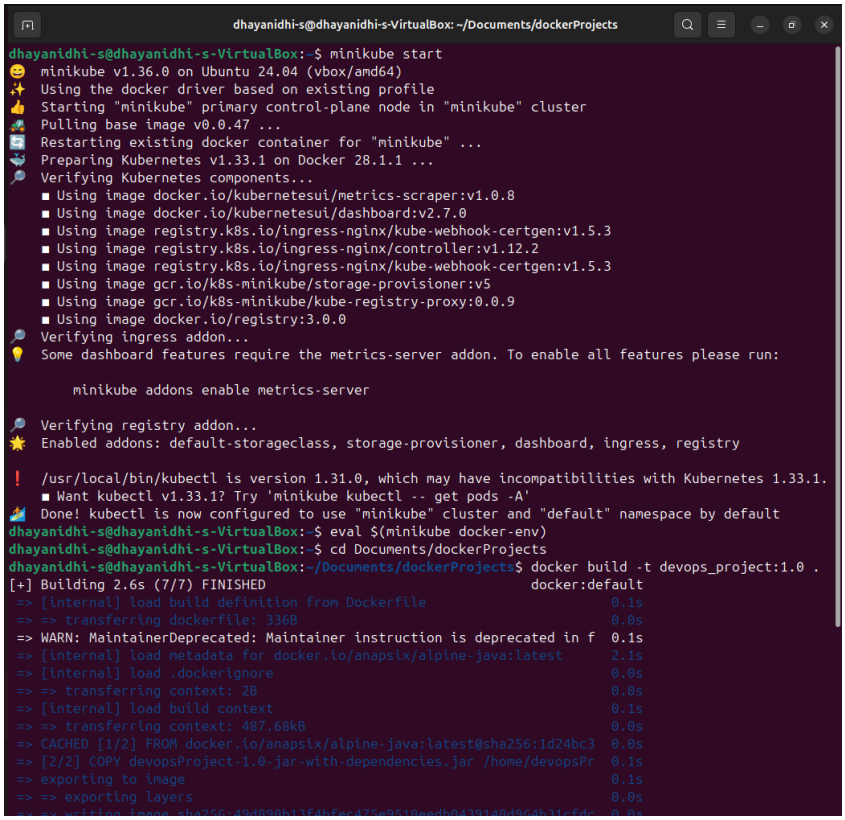
Minikube helps you create a lightweight Kubernetes cluster locally.
Set up Minikube on your Ubuntu system.

🚀 2. Start Minikube and Connect Docker Daemon

To work with Minikube's internal Docker environment:

```
minikube start
eval $(minikube docker-env)
```

This ensures Docker images built will be available to your Kubernetes pods.



🛠️ 3. Build the Docker Image Inside Minikube

Navigate to Dockerfile directory and build the image:

```
docker build -t devops_project:1.0 .
```

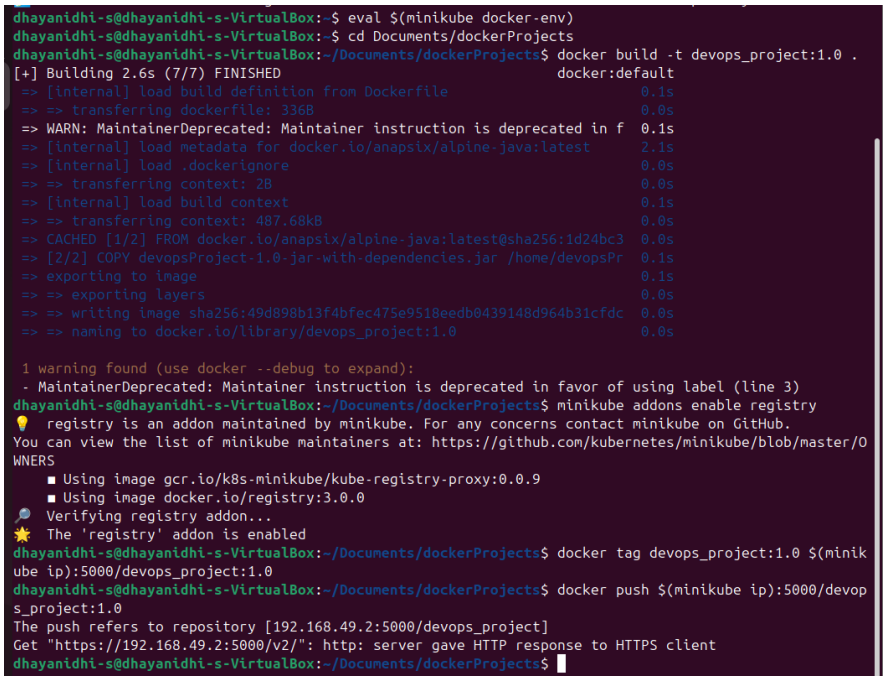
This creates the devops_project:1.0 image directly inside Minikube’s Docker engine.

📦 4. Enable Minikube Registry & Push Image

If you prefer pushing to Minikube's internal registry:

```
minikube addons enable registry
docker tag devops_project:1.0 $(minikube ip):5000/devops_project:1.0
docker push $(minikube ip):5000/devops_project:1.0
```

⚠️ Skip if image was already built with eval $(minikube docker-env).

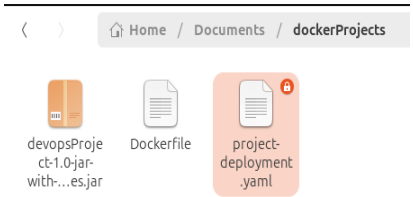


📝 5. Create Kubernetes Deployment YAML File

Create the deployment file:

```
sudo nano calculator-deployment.yaml
```

```
  GNU nano 7.2                                project-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: project-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: moneymetrics
  template:
    metadata:
      labels:
        app: moneymetrics
    spec:
      containers:
        - name: moneymetrics    # ✅ all lowercase
          image: devops_project:1.0
          imagePullPolicy: Never
          ports:
            - containerPort: 8080
```



This configures a pod to run the Dockerized calculator app.

🚀 6. Apply Deployment and Expose Service

Run the following commands to deploy and expose the app:

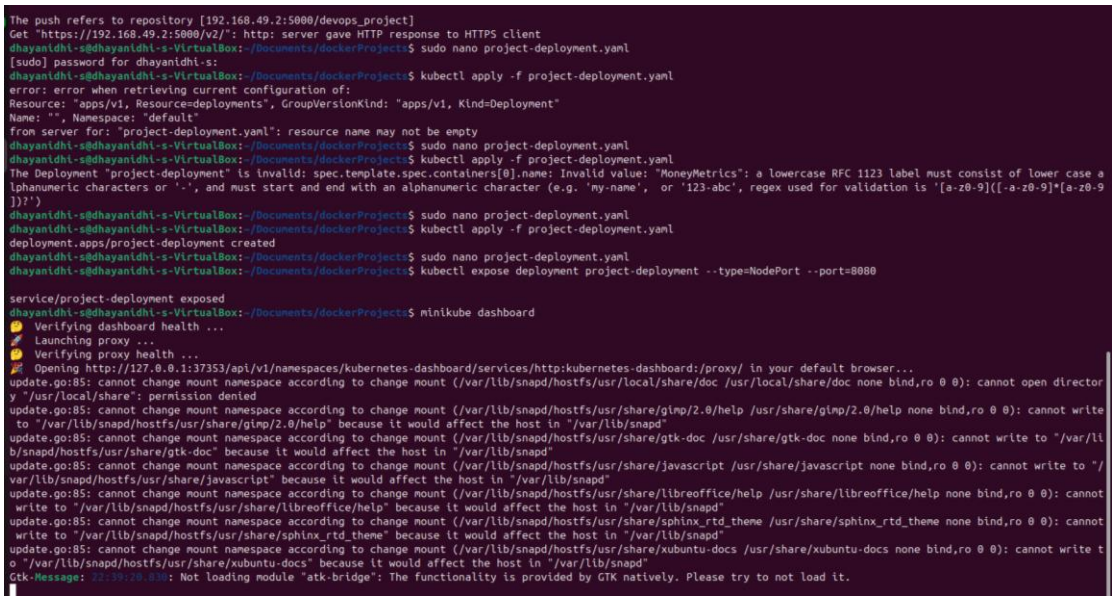
```
kubectl apply -f calculator-deployment.yaml
kubectl expose deployment calculator-deployment --type=NodePort --port=8080
```

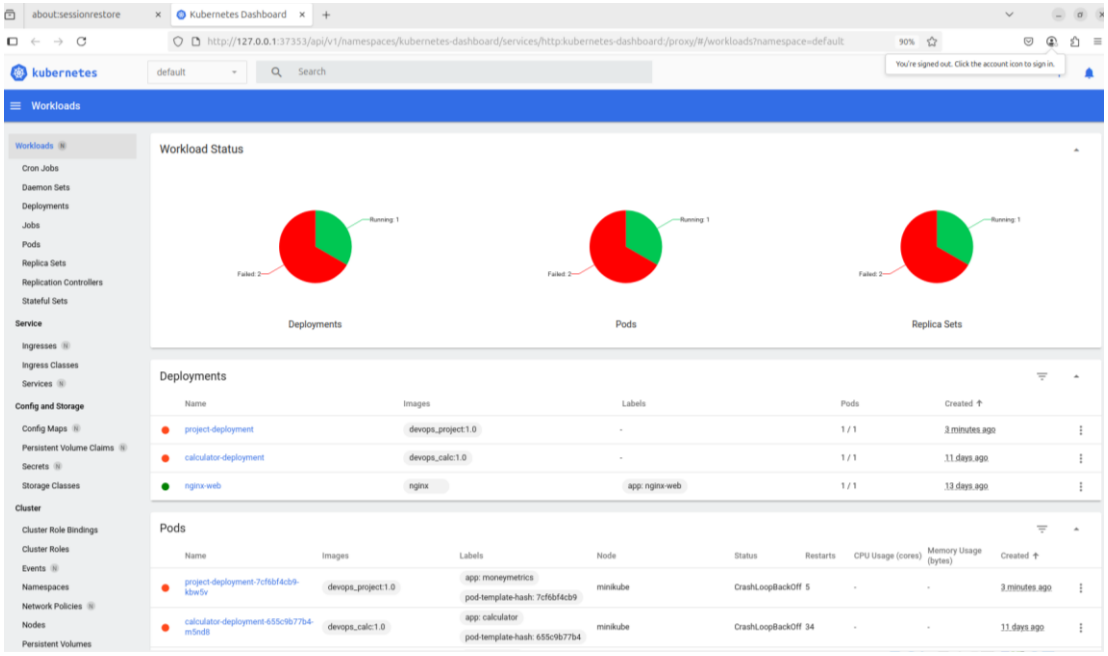
This makes the app available over a NodePort in your Minikube cluster.

📊 7. Launch Dashboard and Monitor Logs

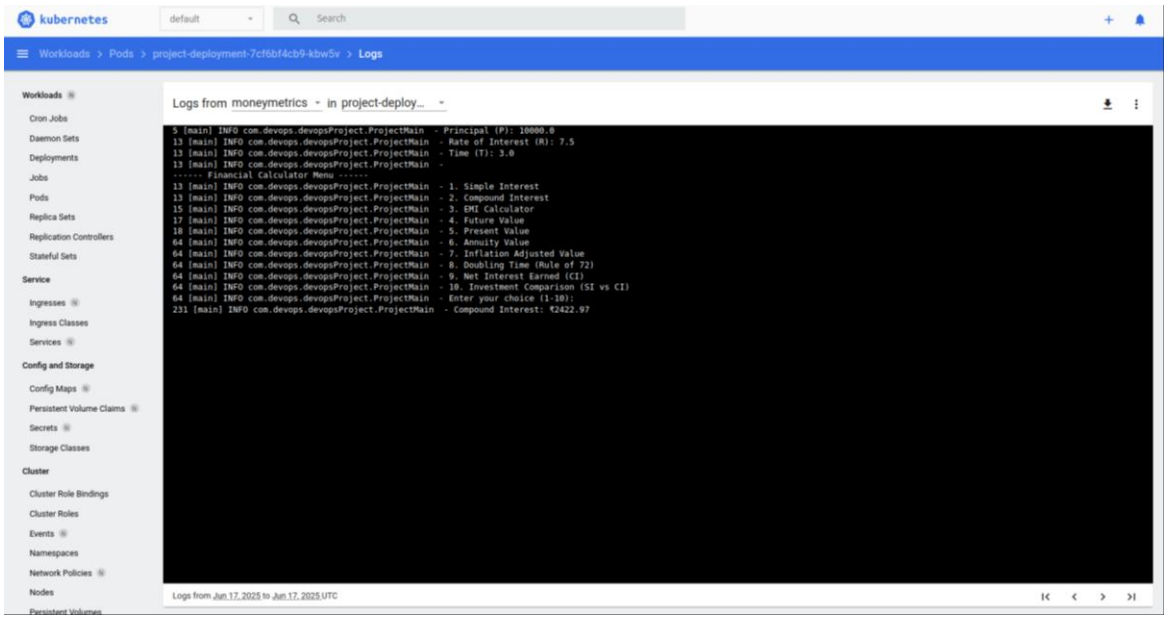
Start the Minikube dashboard to visually monitor pods and logs:

```
minikube dashboard
```





Go to Pods → calculator-deployment → Logs to verify the container output.



📊 Monitoring using Grafana and Graphite

This stage integrates **Graphite** (for storing system metrics) and **Grafana** (for beautiful visual dashboards) to monitor your **Java-based Financial Calculator application**.

🐬 1. Install Docker and Docker Compose (and Nagios)

```
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ sudo curl -L "https://github.com/docker/compose/releases/download
/v2.37.0/docker-compose-$(uname -s)-$(uname -m)" \
  -o /usr/local/bin/docker-compose && sudo chmod +x /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 71.3M  100 71.3M    0     0  2091k      0  0:00:34  0:00:34 --:--:-- 2631k
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ sudo chmod +x /usr/local/bin/docker-compose
```

📁 2. Configure Docker Compose for Grafana and Graphite

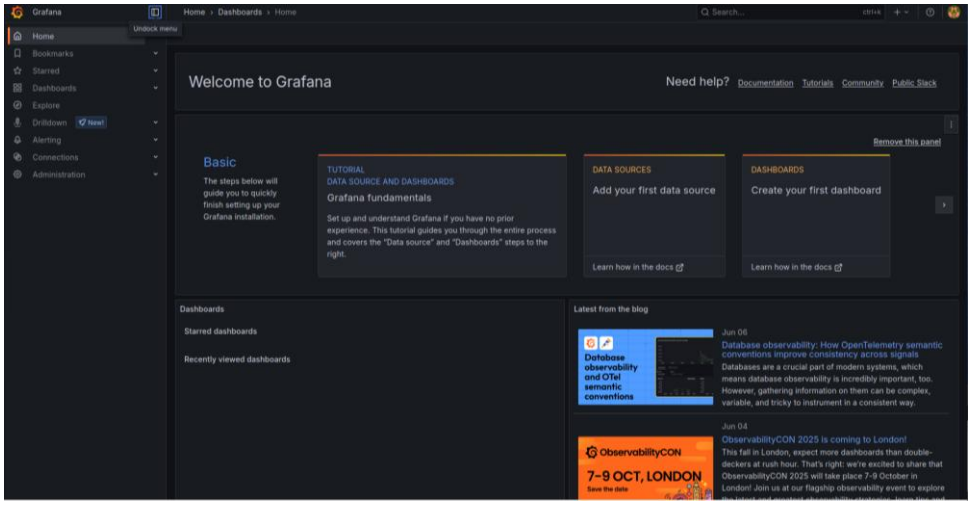
```
dhayanidhi-s@dhayanidhi-s-VirtualBox:~$ mkdir ~/grafana && cd ~/grafana
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/grafana$ nano docker-compose.yml
```

```
 GNU nano 7.2                                  docker-compose.yml
version: "3"
services:
  grafana:
    image: grafana/grafana
    container_name: grafana
    restart: always
    ports:
      - 3000:3000
    networks:
      - grafana-net
    volumes:
      - grafana-volume:/var/lib/grafana

  graphite:
    image: graphiteapp/graphite-statsd
    container_name: graphite
    restart: always
    networks:
      - grafana-net

networks:
  grafana-net:

volumes:
  grafana-volume:
    external: true
```

```
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/grafana$ sudo docker volume create --name=grafana-volume
grafana-volume
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/grafana$ sudo docker-compose up -d
WARN[0000] /home/dhayanidhi-s/grafana/docker-compose.yml: the attribute `version` is obsolete, it will be
 ignored, please remove it to avoid potential confusion
[+] Running 17/17
 ✔ graphite Pulled                                                                          188.5s
   ✔ 0cdfa0c98ed7 Pull complete                                                              15.5s
   ✔ 723656d3099c Pull complete                                                             174.0s
   ✔ 620093e6e6f0 Pull complete                                                             174.2s
   ✔ fcf6a0ec82d9 Pull complete                                                             174.2s
   ✔ f09364cff1fd Pull complete                                                             183.2s
 ✔ grafana Pulled                                                                           128.7s
   ✔ f18232174bc9 Pull complete                                                               4.2s
   ✔ 65babbe3dfe5 Pull complete                                                               4.3s
   ✔ 651b0ba49b07 Pull complete                                                               4.8s
   ✔ d953cde4314b Pull complete                                                               7.1s
   ✔ aecd4cb03450 Pull complete                                                               7.2s
   ✔ 13fa68ca8757 Pull complete                                                               7.3s
   ✔ f836d47fdc4d Pull complete                                                             116.1s
   ✔ 8b5292c940e1 Pull complete                                                             123.3s
   ✔ 454a4350d439 Pull complete                                                             123.3s
   ✔ 9a8c18aee5ea Pull complete                                                             123.4s
[+] Running 3/3
 ✔ Network grafana_grafana-net  Created                                                       0.2s
 ✔ Container graphite           Started                                                       1.4s
 ✔ Container grafana            Started                                                       1.4s
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/grafana$
```

🌐 3. Access Grafana Dashboard

🔗 Open Grafana in browser and sign in with credentials . If new both username and password are admin then set new password.

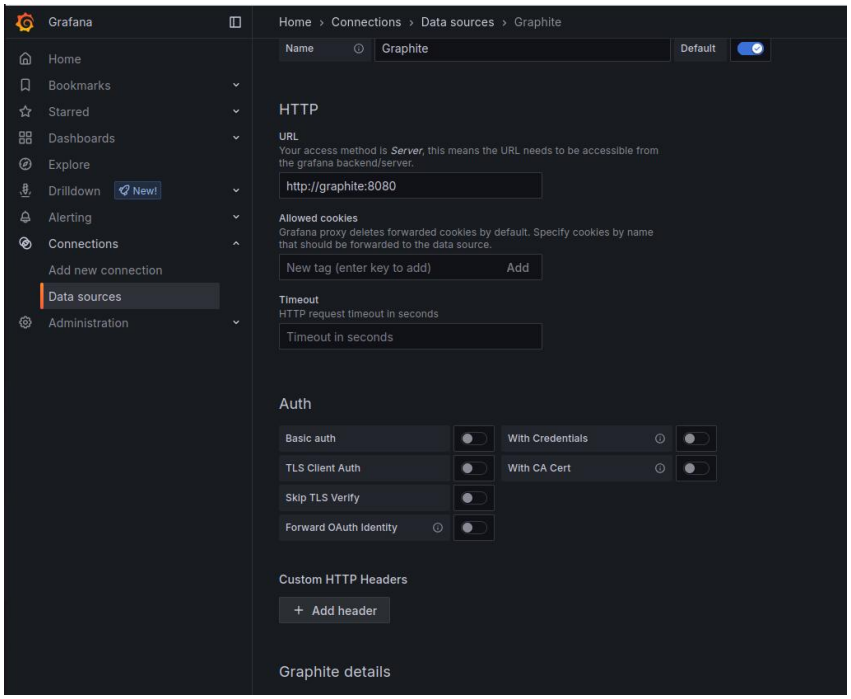


➕ 4. Add Graphite as Data Source in Grafana

🧩 Navigate:

Settings → Data Sources → Add Data Source → Choose "Graphite"

🛠️ After proper Configuration Click **Save & Test**



📈 5. Import Dashboard Template

📥 Import community dashboard:

- Click **Dashboard → Import**
- Enter **Dashboard ID: 55**
- Select **Graphite** as data source and **import**

🖥️ Now see a predefined system metrics dashboard.

🔭 6. Install and Configure Collectd

📦 Install Collectd:

```
dhayanidhi-s@dhayanidhi-s-VirtualBox:~/grafana$ sudo apt-get install collectd collectd-
utils
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  collectd-core default-jre-headless intel-cmt-cat libbson-1.0-0t64
```

🔌 Enable essential plugins: Important

🔁 7. Restart Collectd

♻️ Restart the service:

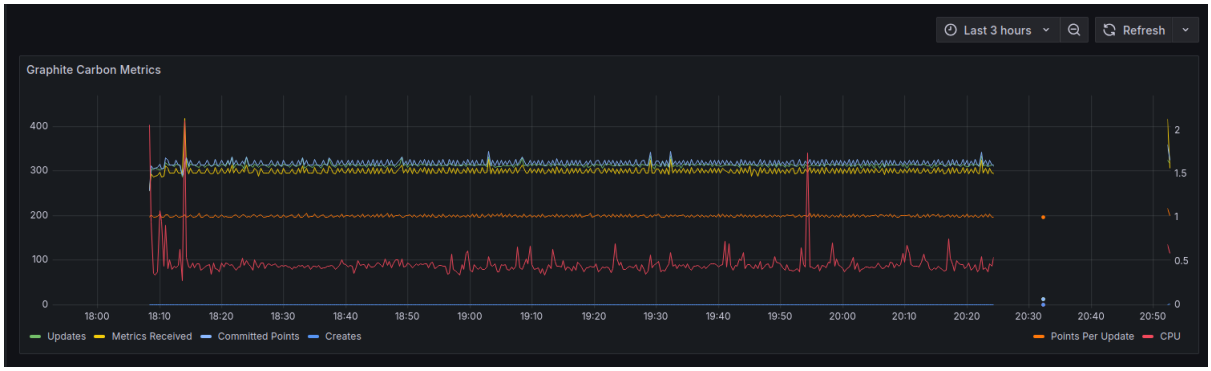
```
sudo systemctl restart collectd
sudo systemctl status collectd
```

📊 8. Visualize Metrics in Grafana

📊 Metrics like:

- CPU usage
- RAM usage
- Disk utilization
- Network bandwidth

will now stream live from **Collectd → Graphite → Grafana**, helping to monitor the application/server health.



🛠️ Ansible Automation Setup for devops_Project

🔧 1. Install Ansible

Install Ansible using apt:

```
sudo apt update
sudo apt install ansible -y
ansible –version
```

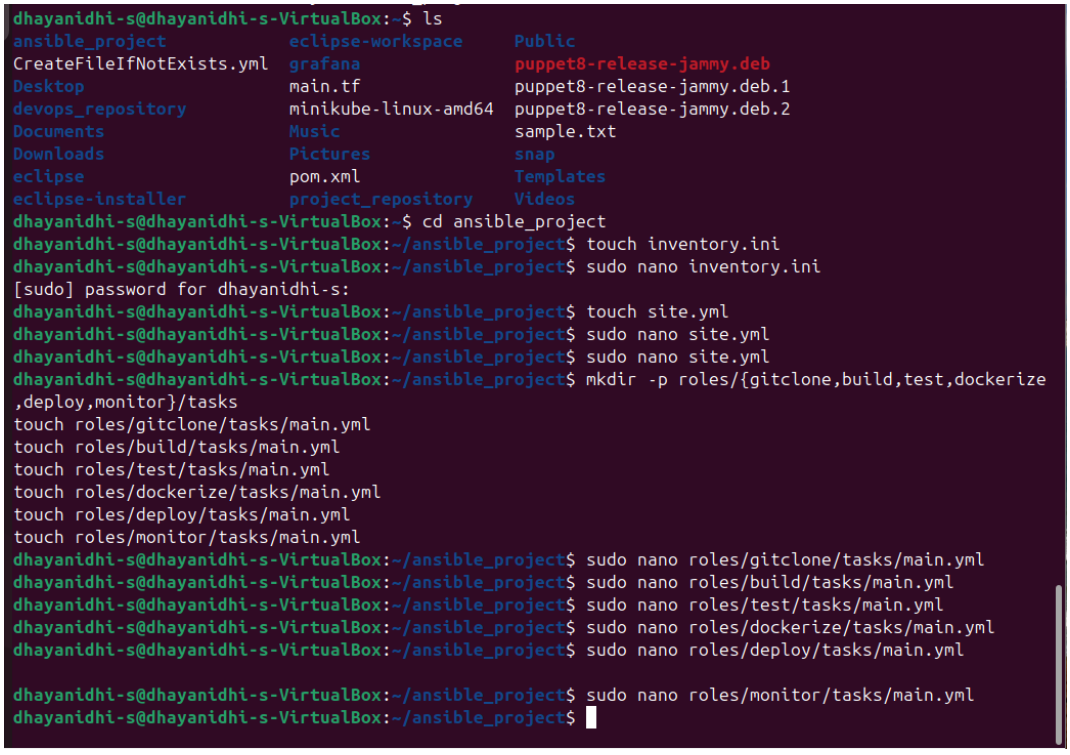
Installs Ansible, a configuration management tool used to automate the entire CI/CD and monitoring pipeline.

📁 2. Create Ansible Project Directory

Navigate to Git project repository and create a folder for Ansible:
Keeps the Ansible configuration separate and organized within the project repository.

🧱 3. Create Ansible Roles

Inside the ansible_project, create the required Ansible roles and supporting files:



Each role automates a distinct step in the pipeline (e.g., Java installation, Docker build, Kubernetes deploy, etc.)

📜 4. Create site.yml (Master Playbook)

Create a central playbook file to execute all roles in sequence:

Open ▾ site.yml ~/project_repository/ansible_project

```yaml
---
- name: Full DevOps Pipeline
  hosts: local
  become: true

  vars:
    build_number: "{{ build_number }}"
    build_path: "{{ build_path }}"

  roles:
    - { role: build, tags: ['build'] }
    - { role: test, tags: ['test'] }
    - { role: dockerize, tags: ['dockerize'] }
    - { role: deploy, tags: ['deploy'] }
    - { role: monitor, tags: ['monitor'] }
    - { role: rollback, tags: ['rollback'] }
```

This playbook controls the execution order of all roles on your local host.

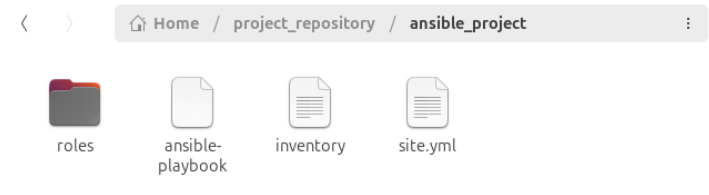
📁 5. Create inventory.ini File

Create inventory file to define the target machine (here: localhost):

Open ▾ inventory ~/project_repository/ansible_project

```
[local]
localhost ansible_connection=local
```

Specifies where the Ansible roles should run — in this case, on your own system.



📁 6. Add Tasks in Each Role (main.yml)

Navigate into each role's tasks folder and add commands specific to the pipeline.

GNU nano 7.2 /home/dhayanidhi-s/ansible_project/roles/gitclone/tasks/main.yml *

```yaml
- name: Clone local Git project repository
  git:
    repo: /home/dhayanidhi-s/project_repository/
    dest: /home/dhayanidhi-s/ci_cd_workspace/
    version: master
    force: yes
```

```
GNU nano 7.2                    roles/build/tasks/main.yml *
- name: Build the Java project using Maven
  command: mvn clean package
  args:
    chdir: /home/dhayanidhi-s/ci_cd_workspace
```

```
GNU nano 7.2                    roles/test/tasks/main.yml *
- name: Run JUnit tests using Maven
  command: mvn test
  args:
    chdir: /home/dhayanidhi-s/ci_cd_workspace
```

```
GNU nano 7.2                    roles/deploy/tasks/main.yml *
- name: Run Docker container from built image
  docker_container:
    name: financial-calc
    image: financial-calc:v1
    state: started
    restart_policy: always
    ports:
      - "8081:8080"
```

```
GNU nano 7.2                              roles/dockerize/tasks/main.yml
- name: Copy Dockerfile to workspace
  copy:
    src: /home/dhayanidhi-s/Documents/dockerProjects/Dockerfile
    dest: /home/dhayanidhi-s/ci_cd_workspace/Dockerfile

- name: Build Docker image from project
  command: docker build -t financial-calc:v1 .
  args:
    chdir: /home/dhayanidhi-s/ci_cd_workspace
```

```
GNU nano 7.2                    roles/monitor/tasks/main.yml *
- name: Copy Docker Compose file to monitoring folder
  copy:
    src: docker-compose.yml
    dest: /home/dhayanidhi-s/ci_cd_workspace/monitoring/docker-compose.yml

- name: Start monitoring stack with Docker Compose
  command: docker-compose up -d
  args:
    chdir: /home/dhayanidhi-s/ci_cd_workspace/monitoring
```

```
GNU nano 7.2              /roles/monitor/files/docker-compose.yml *
version: '3'

services:
  graphite:
    image: graphiteapp/graphite-statsd
    container_name: graphite
    ports:
      - "80:80"
      - "2003-2004:2003-2004"
      - "2023-2024:2023-2024"
      - "8125:8125/udp"
      - "8126:8126"
    restart: always

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    restart: always
```

Each role automates one major DevOps phase of the project.

🚀 7. Run the Ansible Playbook

Now execute the entire pipeline automation:

```
bash
Copy code
ansible-playbook -i inventory.ini site.yml
```

This will run all roles sequentially: Java setup → Maven build → Docker build → Test → Deploy → Enable Monitoring.

CI/CD Pipeline Setup for Java Financial Calculator using Jenkins

---

🧩 1. Install Jenkins and set up

🧩 2. Install Required Jenkins Plugins

Go to http://localhost:8080 → **Manage Jenkins** → **Manage Plugins**

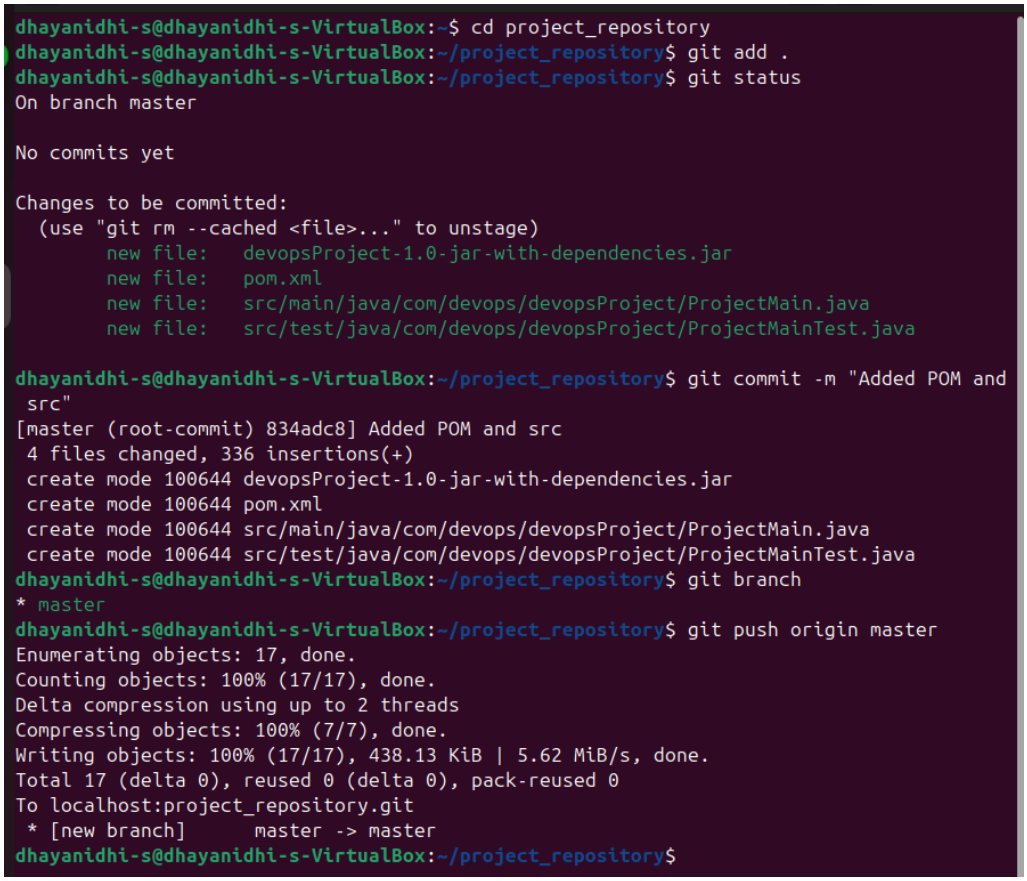
- Under "Available" tab, install:
    - *Hudson Post Build Task Plugin*
    - *PostBuildScript Plugin*

These plugins allow Jenkins to run post-build tasks like executing the generated JAR file.

📩 3. Push Your Maven Project to Git Server

Move your pom.xml and src/ folder into the Git repository, then:

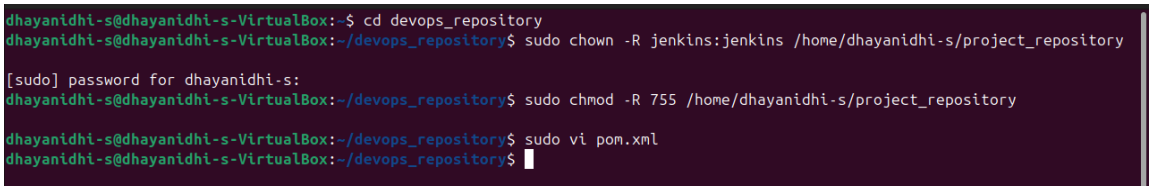
```
git add .
git status
git commit -m "Added POM and src"
```



🔐 4. Grant Jenkins Ownership of the Repository

Ensure Jenkins can read/write to the Git repo:

```
sudo chown -R jenkins:jenkins /home/devops/devops_repository
sudo chmod -R 755 /home/devops/devops_repository
```



📝 5. Add Required Maven Plugin to pom.xml

Edit the pom.xml file to ensure it compiles with Java 17

```
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.1</version>
<configuration>
<source>17</source>
<target>17</target>
</configuration>
</plugin>
</plugins>
```

🧪 6. Validate Jenkins Git Plugin

Go to: **Manage Jenkins → Script Console** and run:

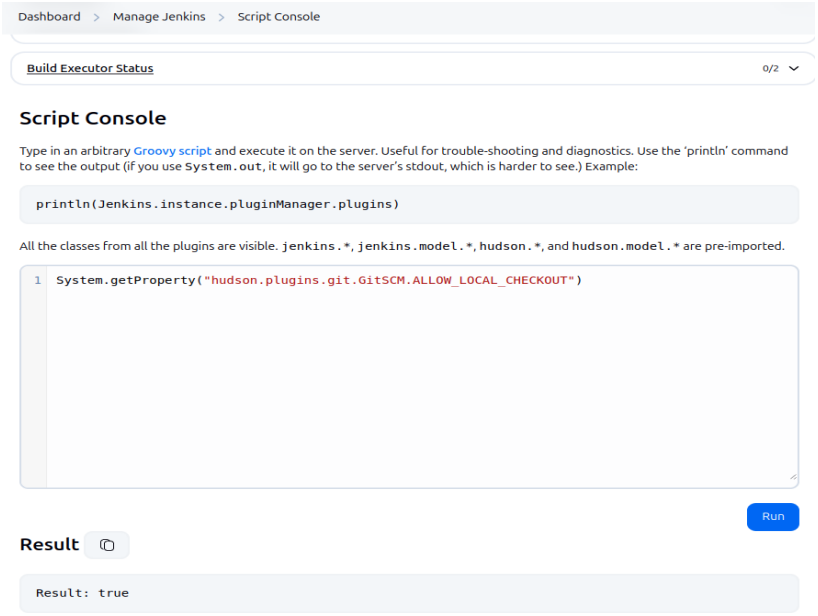
```
System.properties['hudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT']
```

If result ≠ true, edit:

```
sudo nano /etc/default/jenkins
```

Add:

```
JAVA_ARGS="-Dhudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true"
```

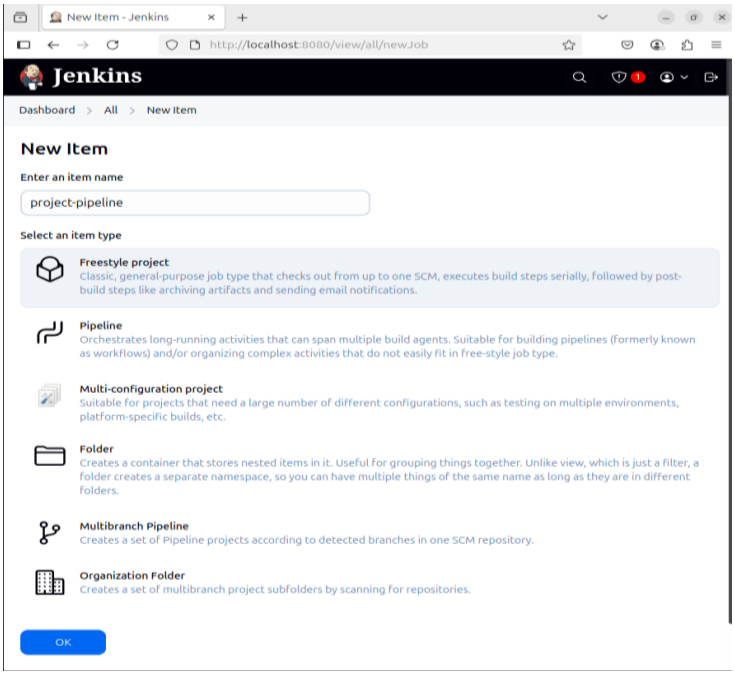


👉 **In this project, we will explore both the *Freestyle Project* and the *Pipeline* methods in Jenkins to automate the build, test, and deployment processes of our Java-based Financial Calculator.**

For freestyle project continue from step 7 .

🛠️ 7. Create a Freestyle Jenkins Project

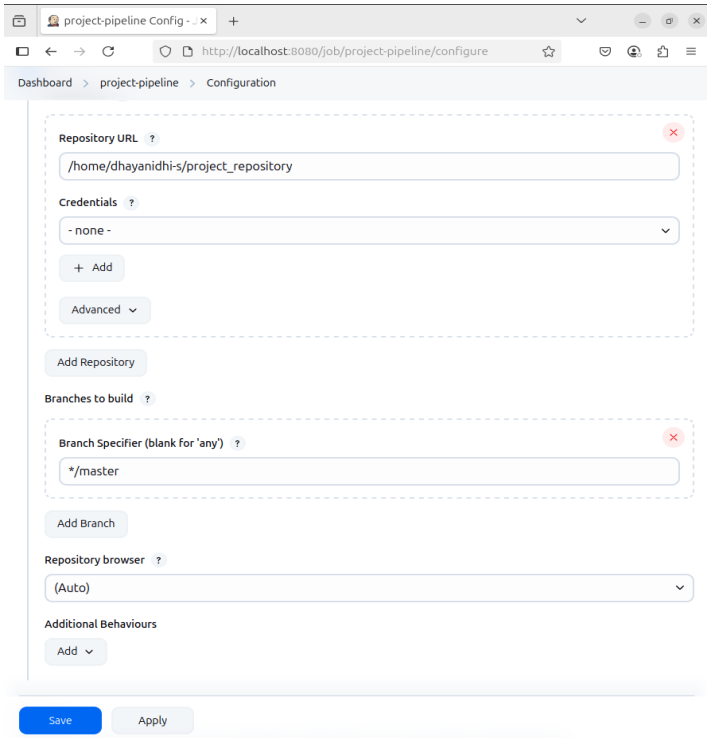
From Jenkins homepage:
→ Click **New Item**
→ Select **Freestyle Project**
→ Name it project-pipeline



This is our CI/CD job configuration.

🔗 8. Connect Git Repository to Jenkins

Inside your new Jenkins project:
→ Go to **Source Code Management**
→ Select **Git**
→ Enter local repo path or URL



Jenkins will now pull code from the Git repo.

🔨 9. Configure Build Command

→ Click on **Add Build Step → Execute Shell**

Paste:

```
mvn clean package -Dmaven.compiler.source=17 -Dmaven.compiler.target=17
```

This builds the JAR using Maven inside Jenkins workspace.
To automate all the processes

🔑 10. Grant Jenkins Sudo Access Without Password

Edit the sudoers file:

```
sudo visudo
```

Add this line:

```
jenkins ALL=(ALL) NOPASSWD: ALL
```

```
# User privilege specification
root    ALL=(ALL:ALL) ALL
devops  ALL=(ALL:ALL) ALL
Jenkins ALL=(ALL) NOPASSWD: ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d
```

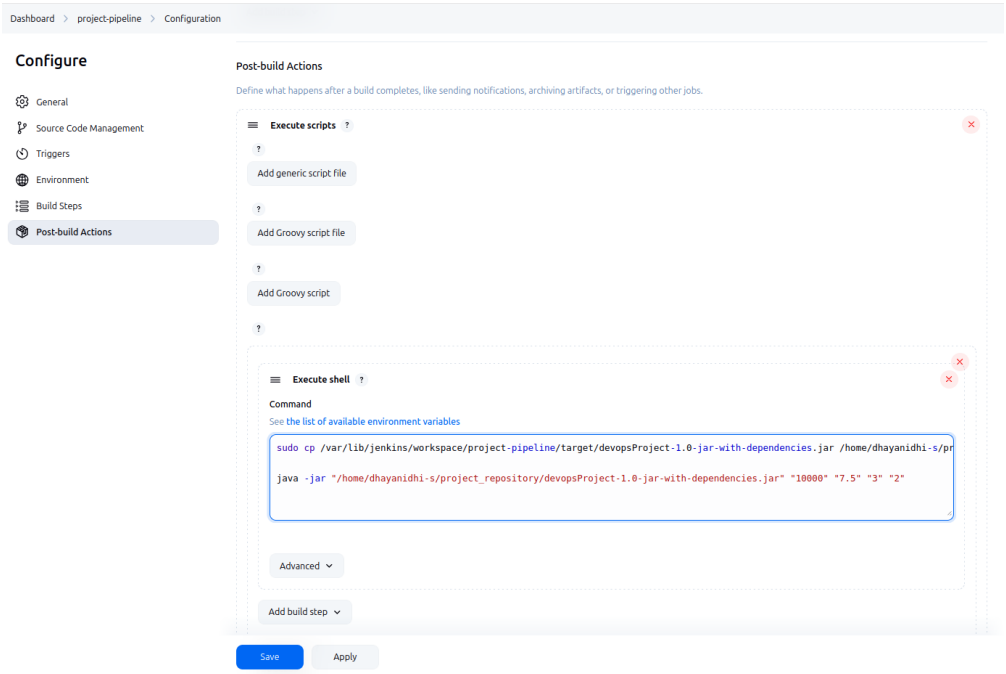
Jenkins can now execute post-build scripts with root access.

📜 11. Post-Build Script to Copy & Run JAR

In **Post-build Actions** → Add:

- **Add post-build action → Execute Scripts**
- Paste:

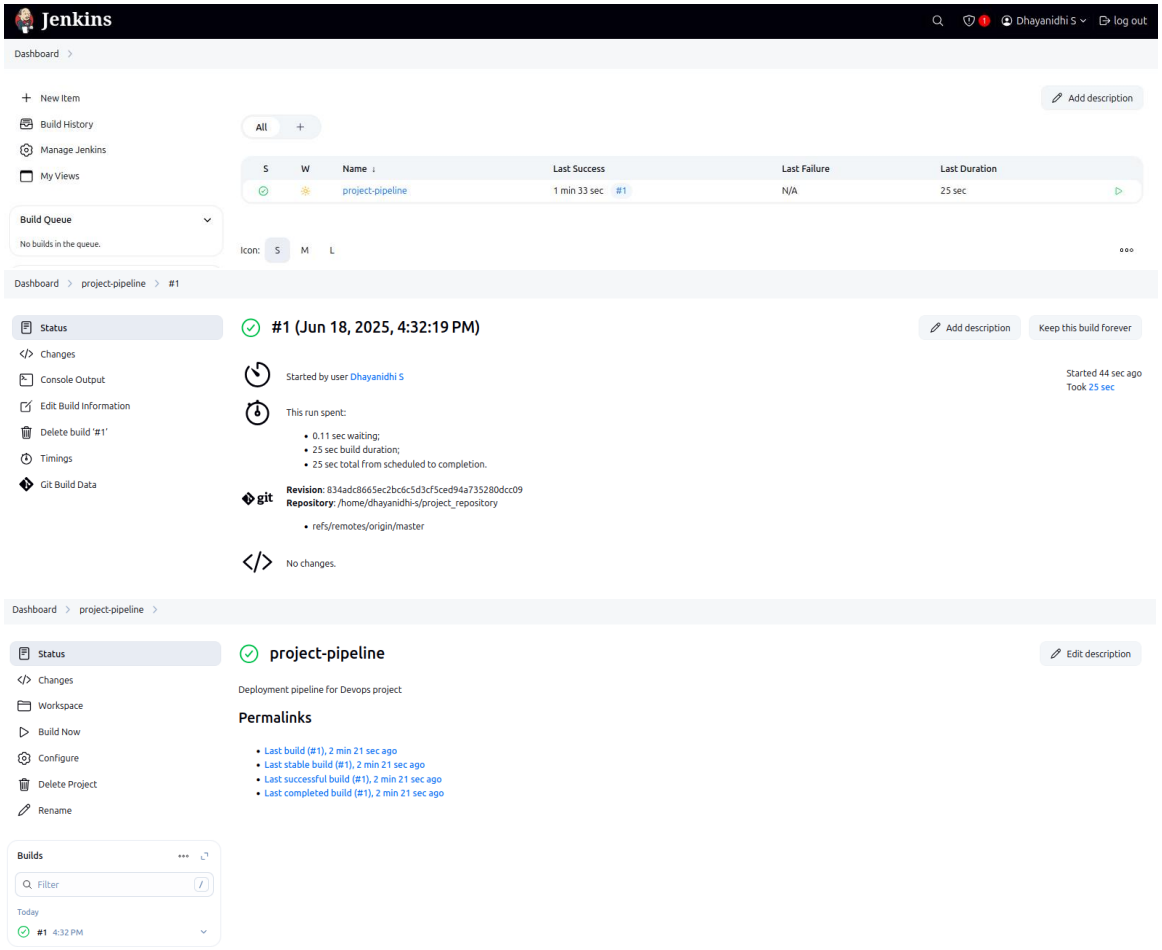
```
sudo cp /var/lib/jenkins/workspace/project-pipeline/target/devopsProject-1.0-jar-with-dependencies.jar /home/devops/devops_repository

java -jar "/home/devops/devops_repository/devopsProject-1.0-jar-with-dependencies.jar" "10000" "7.5" "3" "2"
```



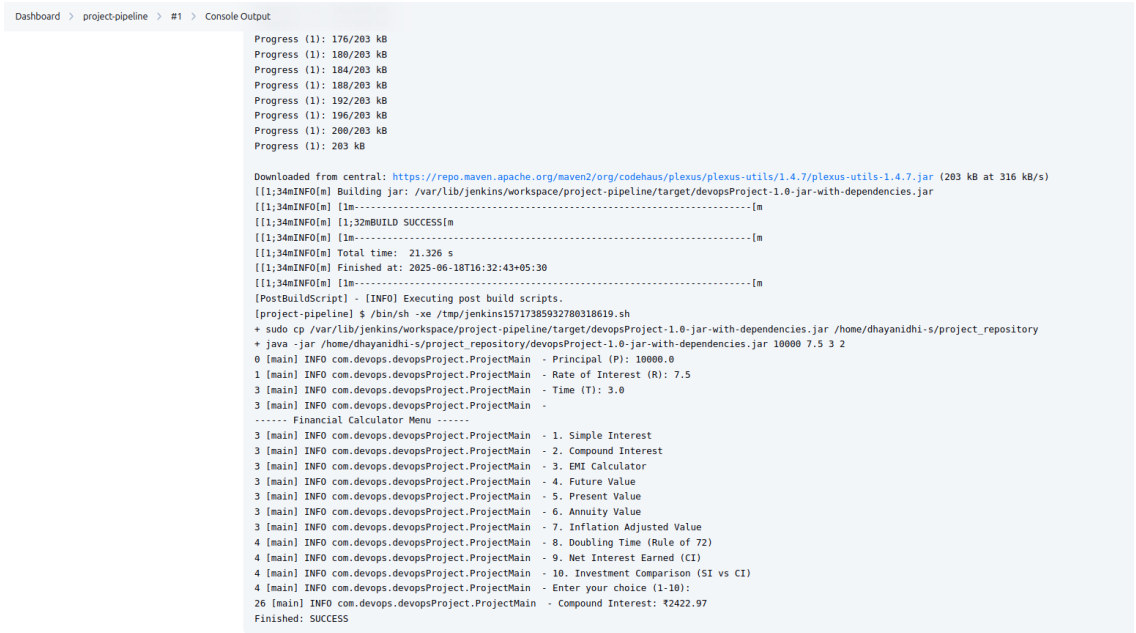
This runs the compiled JAR file after successful build.

🧪 12. Test the Jenkins Pipeline

From Jenkins homepage:
→ Click **Build Now** on project



View real-time logs from **Build History → Console Output**.



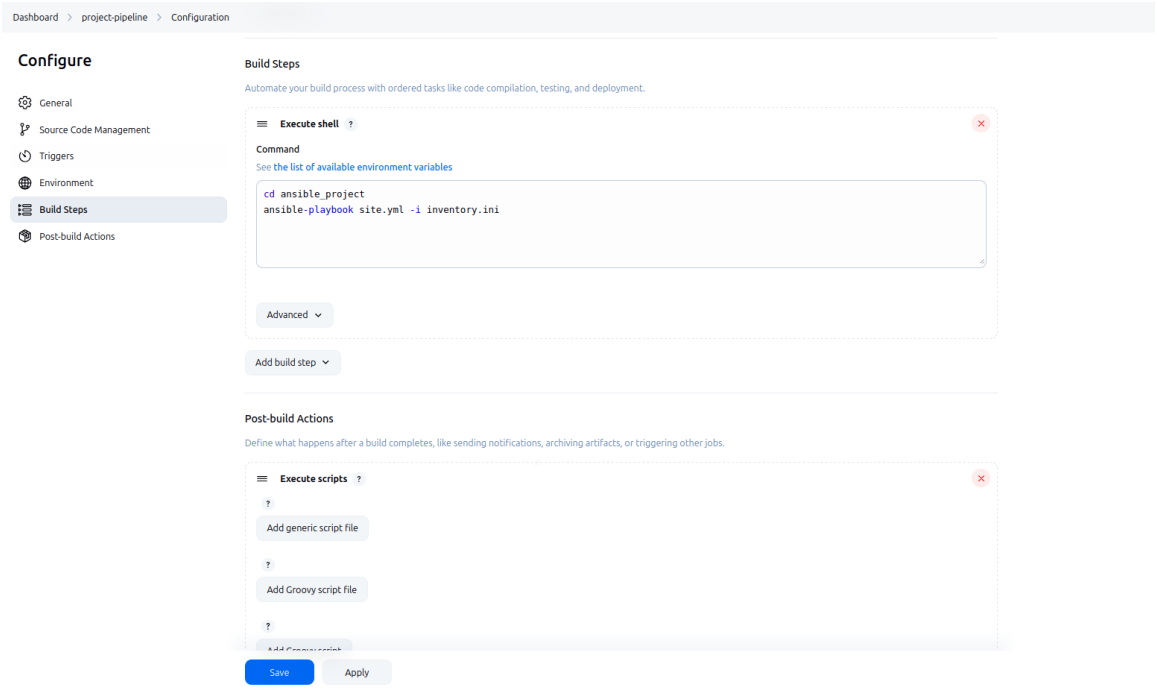
⚙️ Automating All Roles By Running Ansible Playbook In Jenkins

In the project so far, Jenkins automates key tasks like performing a maven clean, building the project, and running the generated JAR file during every build. This ensures continuous integration is in place. However, to **streamline the entire DevOps process even further**, we are now taking it a step ahead — by **triggering the Ansible playbook from Jenkins**. This playbook includes all roles such as Maven clean and build, Docker contanerization, JUnit testing, Jenkins setup, Minikube deployment, and monitoring configuration using Grafana and Graphite.
By doing this, Jenkins not only handles the build but also **automates the full environment setup** with a single trigger, making the pipeline **faster, consistent, and completely hands-free** for end-to-end deployment and monitoring.

▶️ Run the entire pipeline using the Ansible playbook

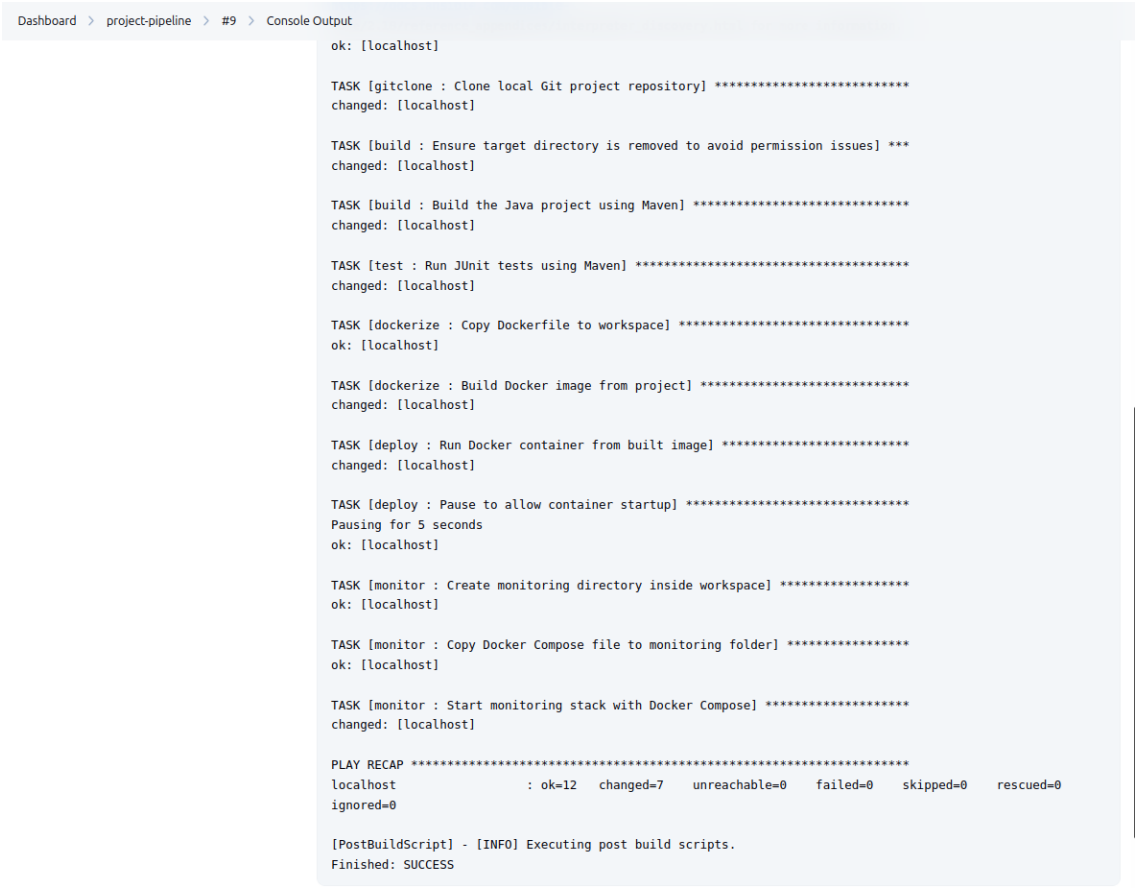
👉 Once all roles and their tasks are defined, execute the entire pipeline:

```
ansible-playbook site.yml -i inventory.ini
```



📦 This will **automatically install, configure, test, deploy, monitor**, and integrate everything defined across all roles in a **single command**.

📋 **Note**: Checking the **Console Output** in Jenkins after a build helps in **identifying and resolving errors** during execution. If a failure occurs while running the Ansible playbook, it's recommended to **review and modify the main.yml file** under each role's tasks/ directory. Ensuring the tasks, paths, modules, and configurations in these files are correct is key to troubleshooting and fixing build or deployment issues efficiently. Make sure you commit and push the changes .



🔁 **Now, we transition from using a Freestyle Project to a Jenkins Pipeline** for our project. The pipeline approach provides **better automation, flexibility, and control**, allowing us to define the entire CI/CD workflow in a script (Jenkinsfile). This ensures a more scalable and manageable way to build, test, and deploy our financial calculator application.

🚀 Creating Jenkins Pipeline with Ansible Integration

🧱 1. Create a Pipeline Project in Jenkins

- 🔹 Go to **Jenkins Dashboard -> New Item**
- 🔹 Enter a name, eg . **pipeline_for_project.**
- 🔹 Select **“Pipeline”** as the project type.
- 🔹 Click **OK** to create the pipeline project.

🎯 *A Pipeline job allows to define the complete build process—build, test, deploy, monitor, and rollback—as code in a Jenkinsfile.*

🛠️ 2. Configure the Pipeline Job

After the project is created:

- 🔹 Scroll down to the **Build Triggers** section and Check the box ✅ **Poll SCM**.

🕒 What is Poll SCM?

This tells Jenkins to **poll the Git repository** periodically to check for changes (commits, file updates). If changes are found, the pipeline build is triggered automatically.

- In the **Schedule** textbox, type:

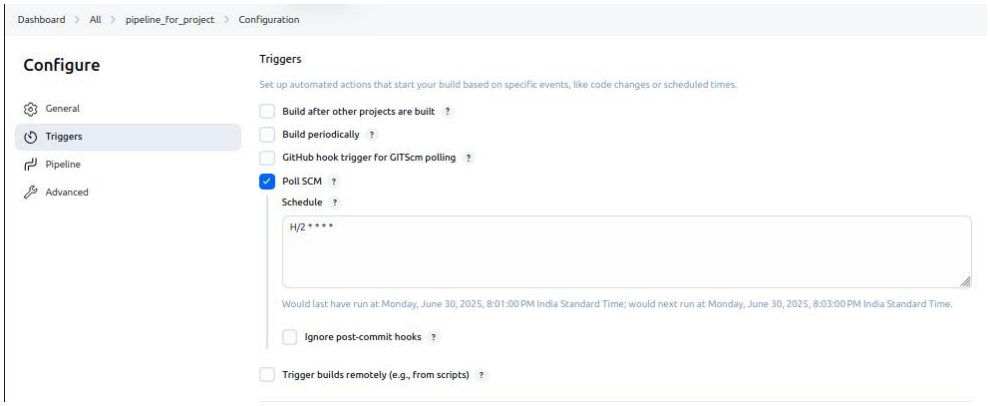
H/2 * * * *

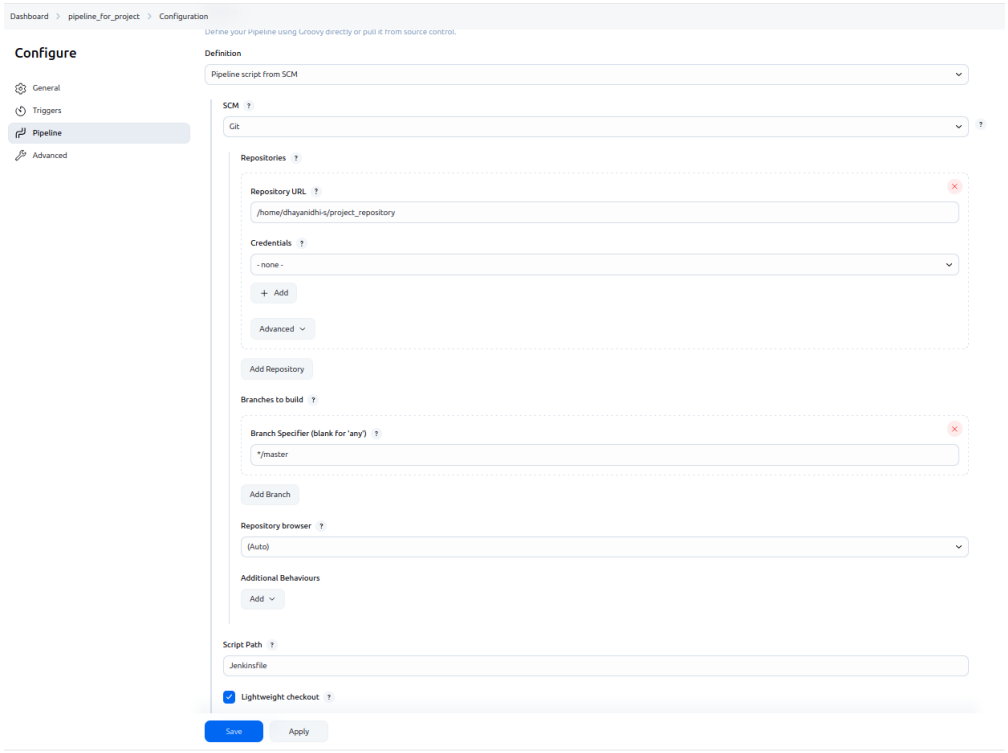
⏱️ This means: “Check for changes every 2 minutes, randomized per job to avoid overload.”

H/2 → Distributes the load evenly by using a hash of the job name.
* → Every minute/hour/day/week/month.

📁 Configure the Pipeline Source:

- 🔹 In the **Pipeline** section below:
  - Select **“Pipeline script from SCM”**.
  - Choose **Git** under SCM.
  - In the **Repository URL**, enter the path to your local Git repository (e.g., /home/administrator/devops_repository/).
  - Leave credentials empty (for local access).
  - Set **Script Path** as: Jenkinsfile (Assuming Jenkinsfile is located at the root of the repo.)
- 🔹 Click **Save**.





🧠 Jenkins will now run the Jenkinsfile from Git repository every 2 minutes (if changes are found) and follow the defined stages.

📝 3. Create the Jenkinsfile

Inside your Git project directory (e.g., devops_repository), create a file named Jenkinsfile and add the following:

Open

Jenkinsfile
~/project_repository

```
pipeline {
    agent any

    environment {
        BUILD_PATH = "${env.WORKSPACE}"
        BUILD_NUMBER = "${env.BUILD_NUMBER}"
    }

    stages {
        stage('📦 Build') {
            steps {
                dir('ansible_project') {
                    sh 'ansible-playbook -i inventory site.yml --tags build --extra-vars "build_path=$BUILD_PATH build_number=$BUILD_NUMBER"'
                }
            }
        }

        stage('🧪 Test') {
            steps {
                dir('ansible_project') {
                    sh 'ansible-playbook -i inventory site.yml --tags test --extra-vars "build_path=$BUILD_PATH build_number=$BUILD_NUMBER"'
                }
            }
        }

        stage('🐳 Dockerize') {
            steps {
                dir('ansible_project') {
                    sh 'ansible-playbook -i inventory site.yml --tags dockerize --extra-vars "build_path=$BUILD_PATH build_number=$BUILD_NUMBER"'
                }
            }
        }

        stage('🚀 Deploy') {
            steps {
                dir('ansible_project') {
                    sh 'ansible-playbook -i inventory site.yml --tags deploy --extra-vars "build_path=$BUILD_PATH build_number=$BUILD_NUMBER"'
                }
            }
        }

        stage('📊 Monitor') {
            steps {
                dir('ansible_project') {
                    sh 'ansible-playbook -i inventory site.yml --tags monitor --extra-vars "build_path=$BUILD_PATH build_number=$BUILD_NUMBER"'
                }
            }
        }

        stage('🔁 Rollback (if needed)') {
            steps {
                dir('ansible_project') {
                    sh 'ansible-playbook -i inventory site.yml --tags rollback --extra-vars "build_path=$BUILD_PATH build_number=$BUILD_NUMBER"'
                }
            }
        }
```

🧩 Each stage here corresponds to a role or task in the **Ansible automation**. The playbook (site.yml) is executed with the respective --tags like build, test, etc., and passes required variables (build_path, build_number) to make the process dynamic and traceable.

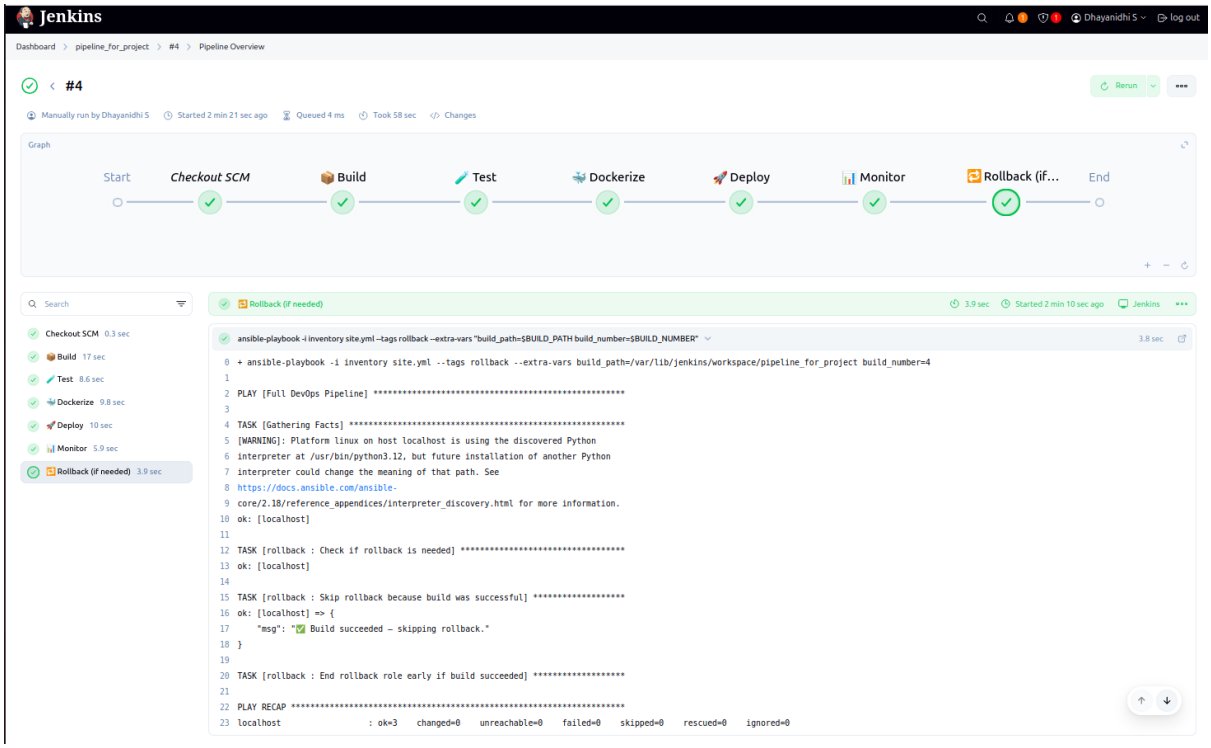
⚙️ 4. Build the Pipeline

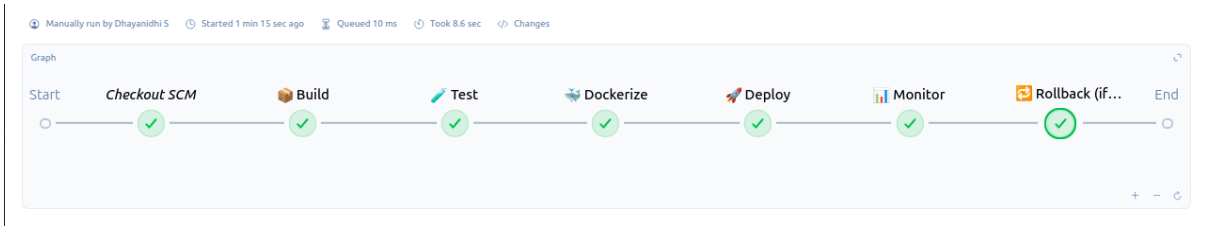
To trigger the pipeline No need to manually click **"Build Now"** every time.

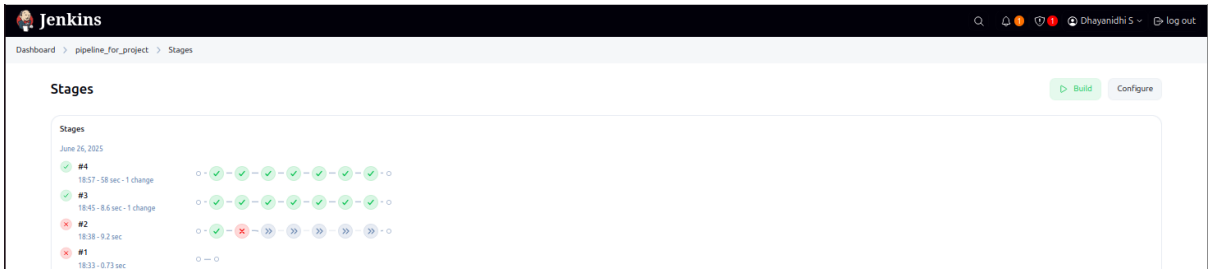
- Every 2 minutes, Jenkins checks the Git repository (your local repo path in the config).
- If it detects **any new commit**, Jenkins will **automatically trigger the build** using the latest version of the Jenkinsfile.

So the workflow becomes:

1. You make changes in code or configuration.
2. You commit and push the changes to Git repository.
3. Jenkins polls the repo, sees the new commit, and triggers the entire pipeline automatically.



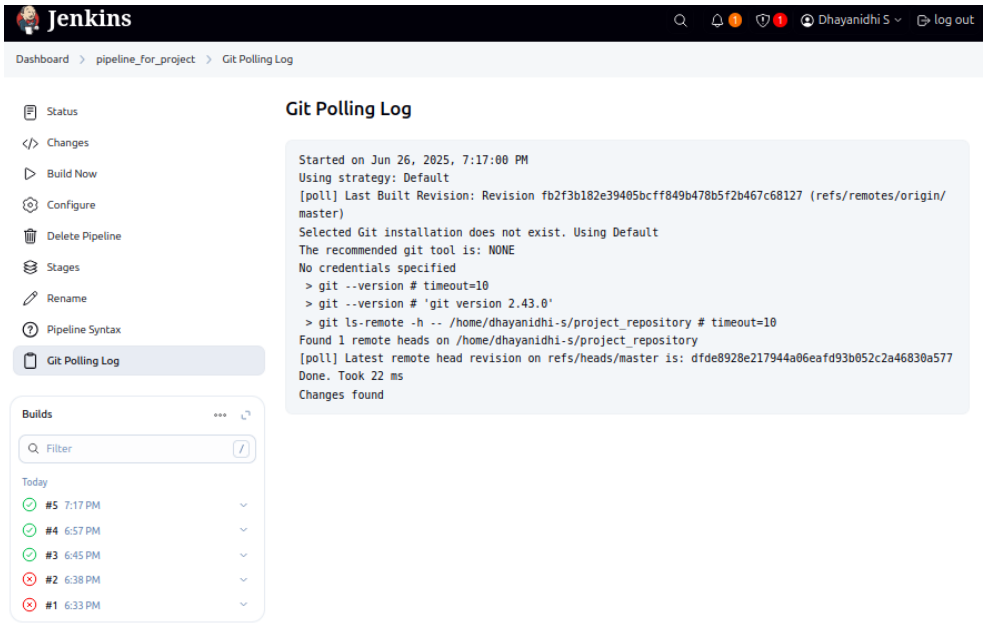




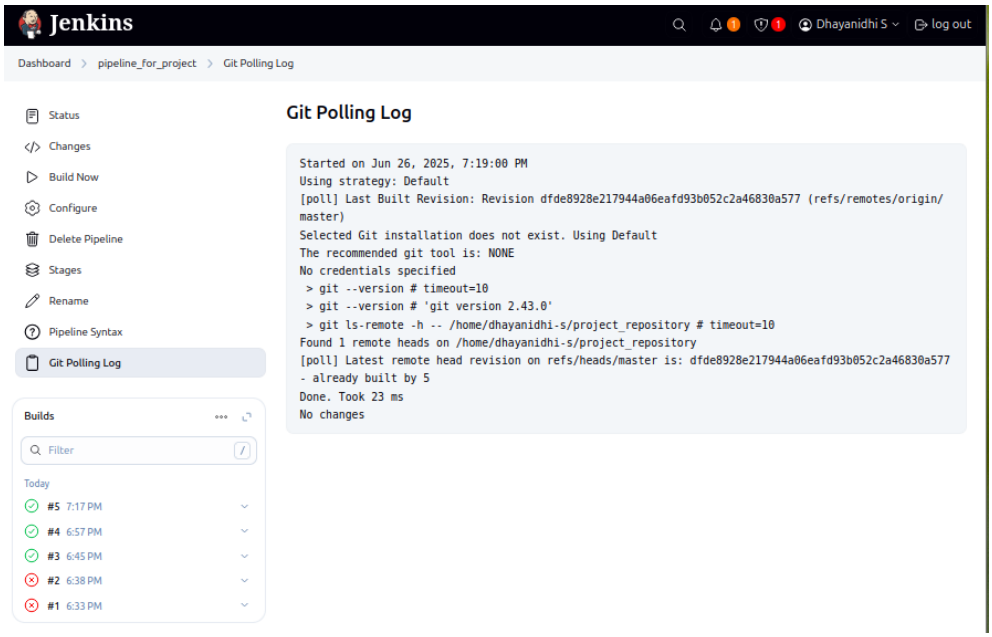
Pipeline working successfully 🛠️

Git Polling and Build Trigger Demonstration

Jenkins is configured to poll the Git repository every 2 minutes for changes.
If any new commits are detected, it automatically triggers the pipeline build process.
This helps ensure that the latest updates are always tested and deployed without manual intervention.



If there are no changes, the pipeline remains idle, avoiding unnecessary builds.



Updated Playbook and Roles for Smooth Pipeline Execution and Achieving Proper Rollback

Open main.yml ~/project_repository/ansible_project/roles/build/tasks

```yaml
- name: Ensure target directory is removed to avoid permission issues
  file:
    path: "{{ build_path }}/target"
    state: absent

- name: Build the Java project using Maven
  command: mvn clean package
  args:
    chdir: "{{ build_path }}"
  register: build_result
  ignore_errors: yes

- name: Create rollback trigger file if build failed
  file:
    path: /tmp/build_failed
    state: touch
  when: build_result.rc != 0

- name: Remove rollback trigger file if build succeeded
  file:
    path: /tmp/build_failed
    state: absent
  when: build_result.rc == 0
```

Open main.yml ~/project_repository/ansible_project/roles/deploy/tasks

```yaml
---
- name: Remove old financial-calc container if it exists
  docker_container:
    name: financial-calc
    state: absent
    force_kill: true
    purge_networks: true
  ignore_errors: yes  # In case container doesn't exist

- name: Run Docker container from built image
  docker_container:
    name: financial-calc
    image: financial-calc:build-{{ build_number }}  # We use build-specific tag
    state: started
    restart_policy: always
    ports:
      - "8182:8080"
    detach: true

- name: Pause to allow container startup
  pause:
    seconds: 5
```

Open main.yml ~/project_repository/ansible_project/roles/dockerize/tasks

```yaml
- name: Copy Dockerfile to workspace
  copy:
    src: /home/dhayanidhi-s/Documents/dockerProjects/Dockerfile
    dest: "{{ build_path }}/Dockerfile"

- name: Build Docker image with unique version tag and 'latest'
  command: >
    docker build
    -t financial-calc:build-{{ build_number }}
    -t financial-calc:latest
    .
  args:
    chdir: "{{ build_path }}"

- name: Save successful build number to file
  copy:
    content: "{{ build_number }}"
    dest: "{{ build_path }}/last_success_build.txt"

- name: Tag current successful image as 'previous'
  shell: docker tag financial-calc:build-{{ build_number }} financial-calc:previous
```

Open main.yml ~/project_repository/ansible_project/roles/monitor/tasks

main.yml | main.yml

```yaml
- name: Create monitoring directory inside workspace
  file:
    path: "{{ build_path }}/monitoring"
    state: directory
    mode: '0755'

- name: Copy Docker Compose file to monitoring folder
  copy:
    src: docker-compose.yml      # This looks inside roles/monitor/files/
    dest: "{{ build_path }}/monitoring/docker-compose.yml"

- name: Start monitoring stack with Docker Compose
  command: docker-compose up -d
  args:
    chdir: "{{ build_path }}/monitoring"
```

Open main.yml ~/project_repository/ansible_project/roles/rollback/tasks

```yaml
---
- name: Check if rollback is needed
  stat:
    path: /tmp/build_failed
  register: rollback_flag

- name: Skip rollback because build was successful
  debug:
    msg: "✅ Build succeeded — skipping rollback."
  when: not rollback_flag.stat.exists

- name: End rollback role early if build succeeded
  meta: end_play
  when: not rollback_flag.stat.exists

- name: Check if previous Docker image exists
  shell: docker images -q financial-calc:previous
  register: previous_image
  changed_when: false

- name: Remove broken container (if any)
  docker_container:
    name: financial-calc
    state: absent
    force_kill: true
  ignore_errors: yes

- name: Rollback to previous Docker image (if it exists)
  docker_container:
    name: financial-calc
    image: financial-calc:previous
    state: started
    restart_policy: always
    ports:
      - "8182:8080"
  when: previous_image.stdout != ""
```

Open main.yml ~/project_repository/ansible_project/roles/test/tasks

```yaml
- name: Run JUnit tests using Maven
  command: mvn test
  args:
    chdir: "{{ build_path }}"
  register: test_result
  ignore_errors: yes  # Allow Ansible to continue even if tests fail

- name: Create rollback trigger file if tests failed
  file:
    path: /tmp/build_failed
    state: touch
  when: test_result.rc != 0
```

🚒 Rollback Implementation – Auto-Recovery

🔁 1. Install Version Number Plugin

Go to **Manage Jenkins → Plugin Manager → Available**
→ Install **Version Number Plugin**
Helps manage versions like 0.2.$BUILD_NUMBER for rollback.

💣 2. Intentionally Break a Test Case

→ In Git repo, modify ProjectMainTest.java to fail a test.

Correct Test Case (with testValidCompoundInterest result as 1025)

```
    @Test
    public void testValidCompoundInterest() {
        double result = ProjectMain.compoundInterest(10000, 5, 2);
        assertEquals(1025.0, result, 1.0); // compound = 10000*(1+0.05)^2 - 1000
0 = 1025
    }
```

Intentionally making the Test case to Fail by changing the value as 125

```
    @Test
    public void testValidCompoundInterest() {
        double result = ProjectMain.compoundInterest(10000, 5, 2);
        assertEquals(125.0, result, 1.0); // compound = 10000*(1+0.05)^2 - 10000
 = 1025
    }
```

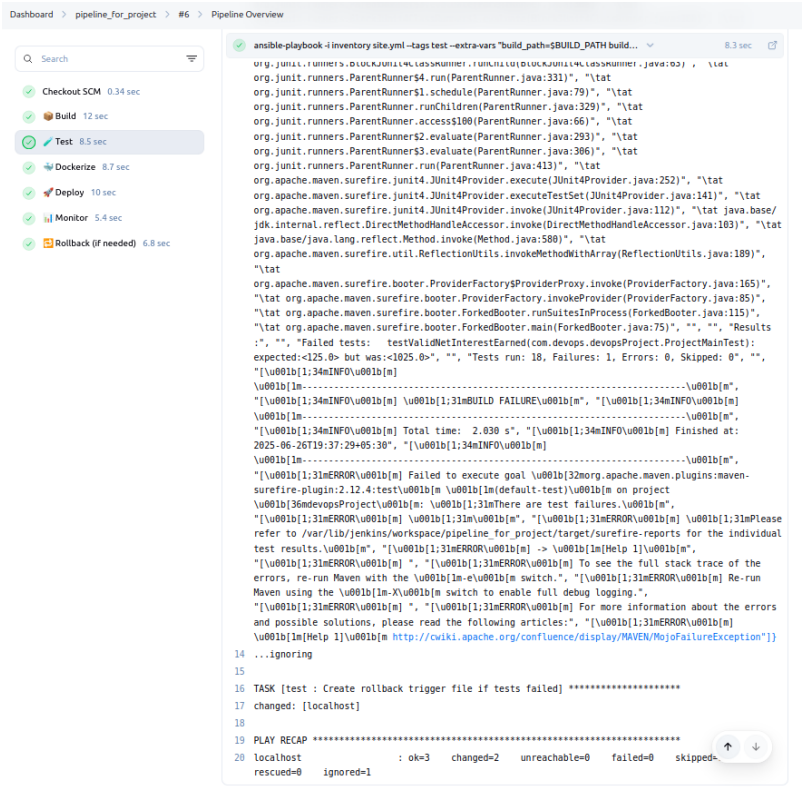
🔧 3. Update Build Commands or Jenkinsfile for Versioning

📈 4. Trigger Jenkins Build and Observe

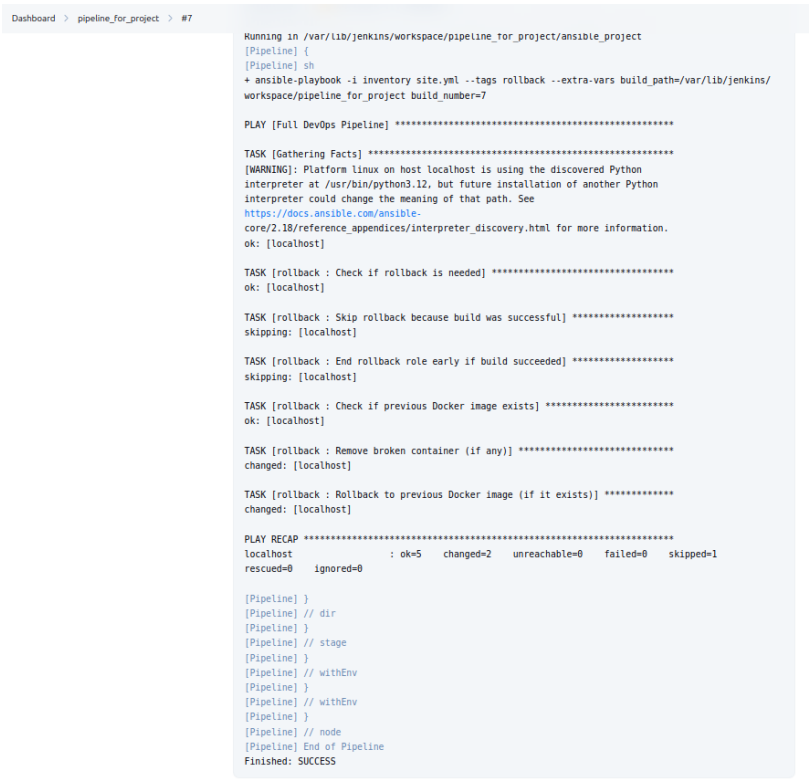
→ Commit the change to trigger a build failure:
→ Watch for:

❌ Build Failure

✅ Rollback Script Execution
✅ Jenkins has now auto-recovered from the failed build using rollback.



Failure of Test case



Rolling back to Previous Build

📌 Migration from Local Git Repository to GitHub

In the initial stages of the project setup, we created a **local Git server and bare repository** (git@localhost:project_repository.git) to manage version control and integrate with Jenkins. This worked effectively in a closed environment, allowing us to test Git-based automation like **SCM polling** and **CI/CD triggers**.

However, to make the project more **scalable, collaborative, and accessible remotely**, **migrate the Git repository to GitHub**.

🔄 What We Changed

1. **Jenkins Configuration:**
    - Replace or use Github repository URL in place of Local repository path . Update under
        - Freestyle Project: Git → Repository URL
        - Pipeline Project: Script from SCM → Repository URL

        Jenkins is now connected to the **GitHub repository** instead of the local Git server.
2. **SCM Trigger:**
    - No change needed in trigger logic (Poll SCM still works), but it now checks GitHub instead of local Git.
    - Optionally, **GitHub Webhooks** can be configured for real-time triggering instead of polling.
3. **Push Workflow:**
    - Code is now committed and pushed to **GitHub** .

⚙️ How Jenkins Works After Migration

- It continues to:
    - **Poll GitHub** every 2 minutes for changes (based on H/2 * * * * schedule).
    - **Trigger builds** automatically if new commits are detected.
    - **Run Ansible playbooks** for build, test, dockerization, deployment, monitoring, and rollback.
- No disruption to your existing Ansible roles or pipeline logic — only the **Git source URL changed**.

💡 Why We Switched to GitHub

Reason	Benefit
🌍 Remote Access	Access code from anywhere, not just the local machine
🤝 Collaboration	Easily share and collaborate with team members
🔐 Better Security	GitHub offers role-based access, SSH key integration
🧩 CI/CD Integration	Jenkins works seamlessly with GitHub for SCM polling or webhook triggers
☁️ Cloud Backup	Prevents data loss by storing your code in the cloud

---

✅ Conclusion

This project presents a comprehensive end-to-end DevOps pipeline designed to automate the entire lifecycle of a Java-based CLI application. It starts with setting up a Maven project for building and managing dependencies, followed by the integration of **JUnit** for automated testing to ensure code quality. The application is then **containerized using Docker**, making it portable and consistent across environments. This Dockerized application is deployed to a **Kubernetes (Minikube)** cluster, offering scalability and efficient resource management.

To enable seamless automation, **Ansible** is used for orchestrating various stages like build, test, dockerization, deployment, monitoring, and rollback. A centralized site.yml playbook and well-structured role-based automation scripts ensure that each task is repeatable, maintainable, and modular. These Ansible playbooks are executed through **Jenkins**, which acts as the heart of the CI/CD pipeline. The Jenkins pipeline is configured with **SCM polling**, so every new code push triggers the entire process automatically. This minimizes manual intervention and ensures rapid and reliable delivery of updates.

Furthermore, robust **monitoring using Collectd, Graphite, and Grafana** is integrated to visualize system metrics in real time. This allows for proactive identification of issues in deployed environments. In case of a failure during build or deployment, **automatic rollback** mechanisms ensure stability by reverting to the last successful version of the application. Altogether, this project demonstrates not only the technical integration of industry-standard DevOps tools but also emphasizes the importance of automation, resilience, and observability in modern software delivery workflows.

🔁 **DevOps as a practice** bridges the gap between development and operations teams by promoting collaboration, continuous integration, and delivery. It helps organizations accelerate product releases, improve deployment frequency, and ensure system reliability. By embracing DevOps principles, this project reflects how automation and toolchain integration can drastically reduce deployment risks, enhance feedback loops, and maintain high-quality software standards at scale

---

🔗 GITHUB LINK

https://github.com/Dhaya09/DevOpsProject.git