Name: Kamal Agrahari
ID: VU4F2223028

Lab: DevOps Lab
TE IT A | Batch B

**Experiment No. 7**

**Aim** - : Execute a simple Maven project and integrate it with Jenkins using GitHub.

**Theory** –

Maven is a build automation tool primarily used for Java projects, hosted by the Apache Software Foundation. It simplifies the build process and provides a uniform system for managing project dependencies and configurations through the Project Object Model (POM), defined in the `pom.xml` file.

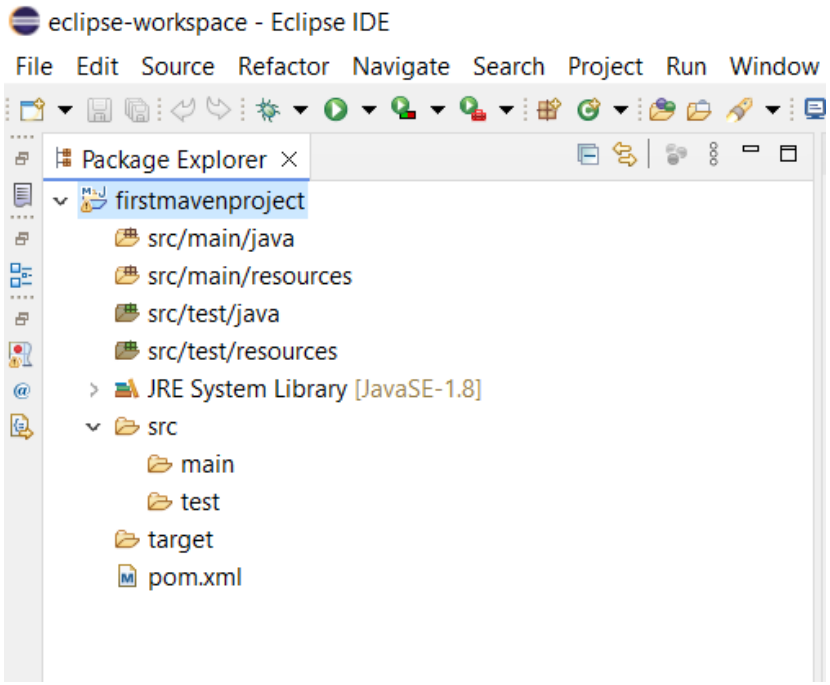
## Tools and Technologies Used

- **Eclipse Neon**
- **Maven** - 3.5.3
- **JDK** - 1.8

**Steps**

### 1. Set Up Maven Project in Eclipse:

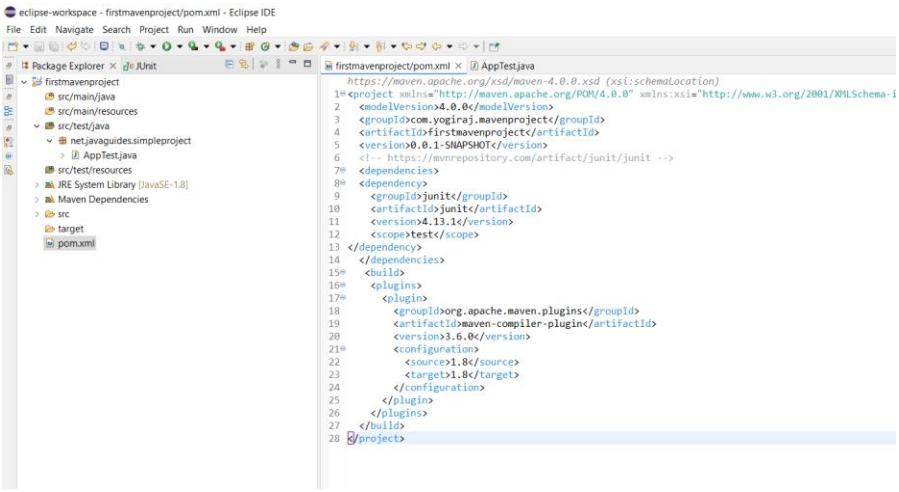
- Install Eclipse IDE from Eclipse Downloads.
- Open Eclipse and create a new Maven project (`File -> New -> Maven Project`).
- Select options for a simple project and use the default workspace location.
- Enter `GroupId` and `ArtifactId`.



## 2. Configure `pom.xml`:

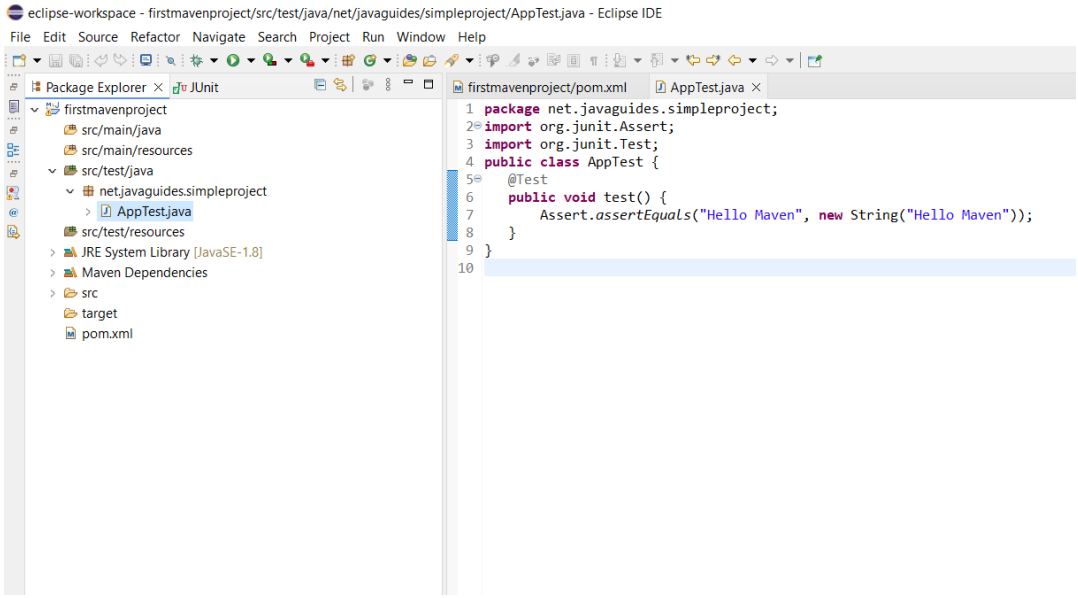
- Add the necessary dependencies (e.g., JUnit) and configure the Maven compiler plugin:

```
<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>
  <groupId>com.yogiraj.mavenproject</groupId>
  <artifactId>firstmavenproject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <!-- https://mvnrepository.com/artifact/junit/junit -->
  <dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.1</version>
    <scope>test</scope>
</dependency>
  </dependencies>
   <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.6.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
 </build>
</project>
```



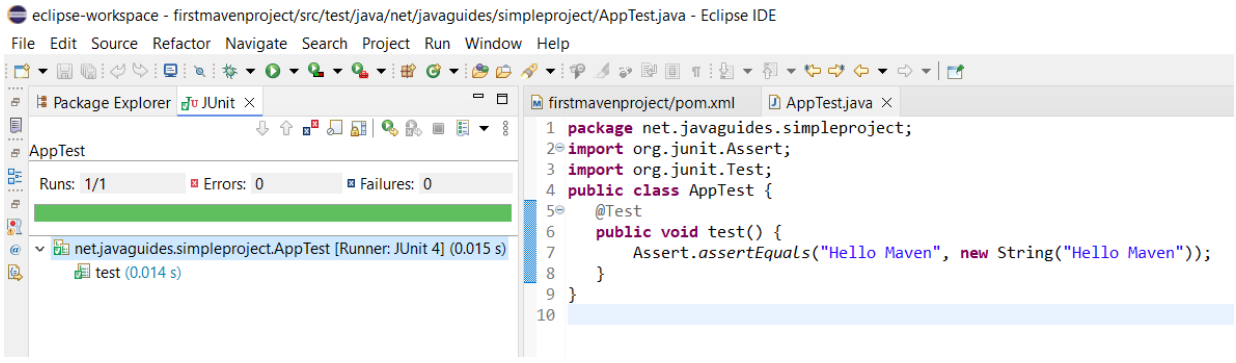
### 3. Create a JUnit Test:

- Create a package `net.javaguides.simpleproject` under `src/test/java`.
- Add `AppTest.java` and write a simple test:



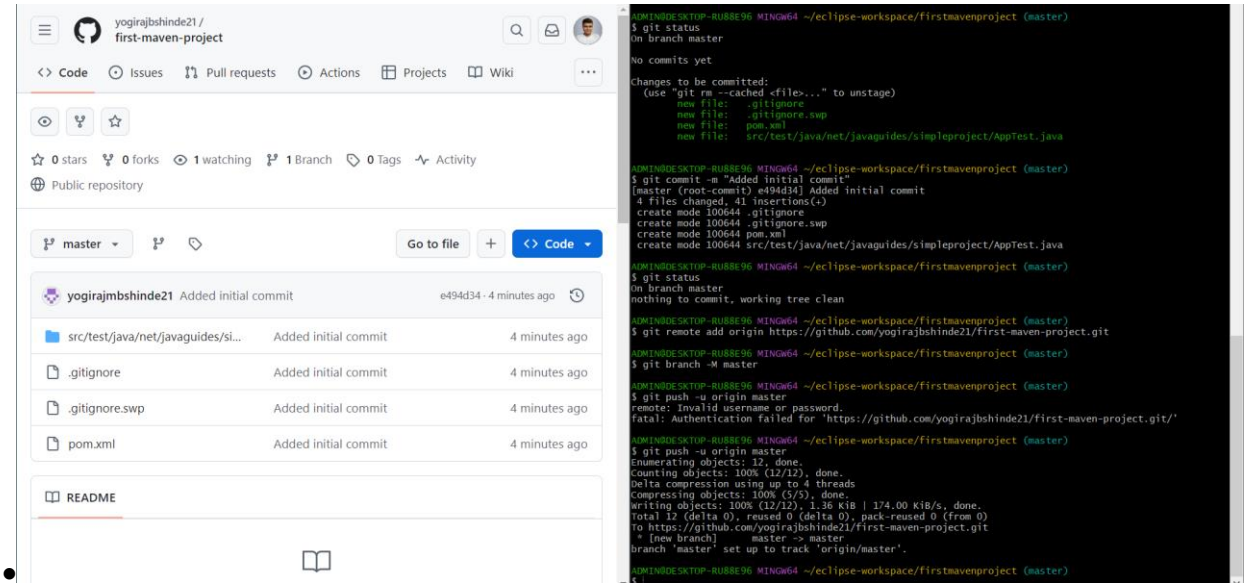

### 4. Run the Maven Project:

- Right-click on `AppTest.java` and select `Run as -> JUnit Test`.



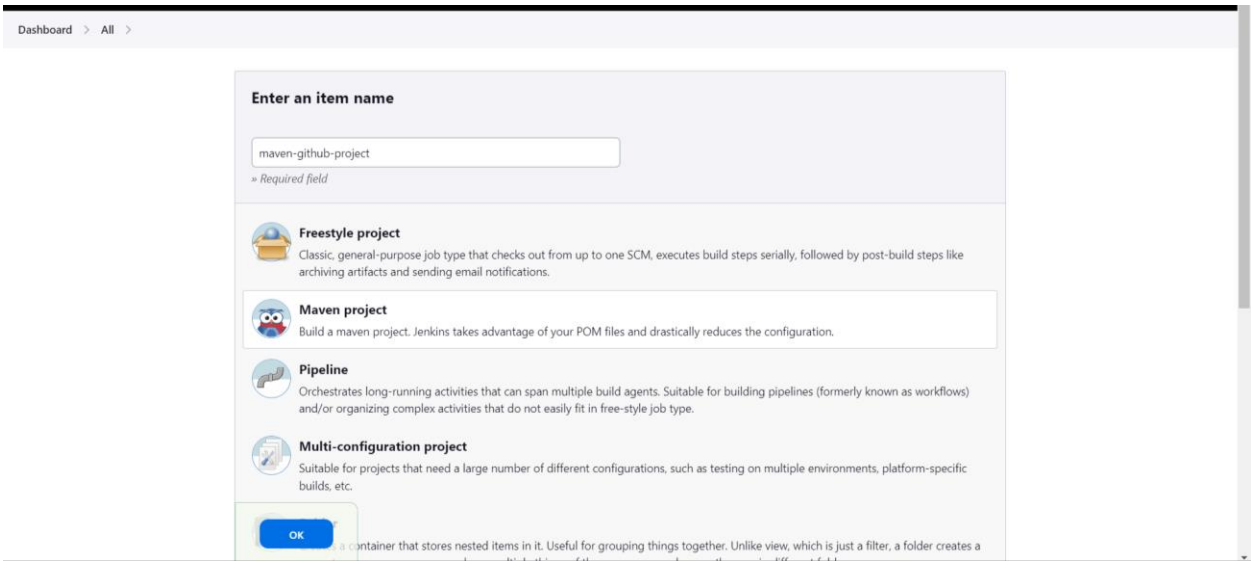
## 5. Push the Maven Project to GitHub:

- Commit and push the project to your GitHub repository.

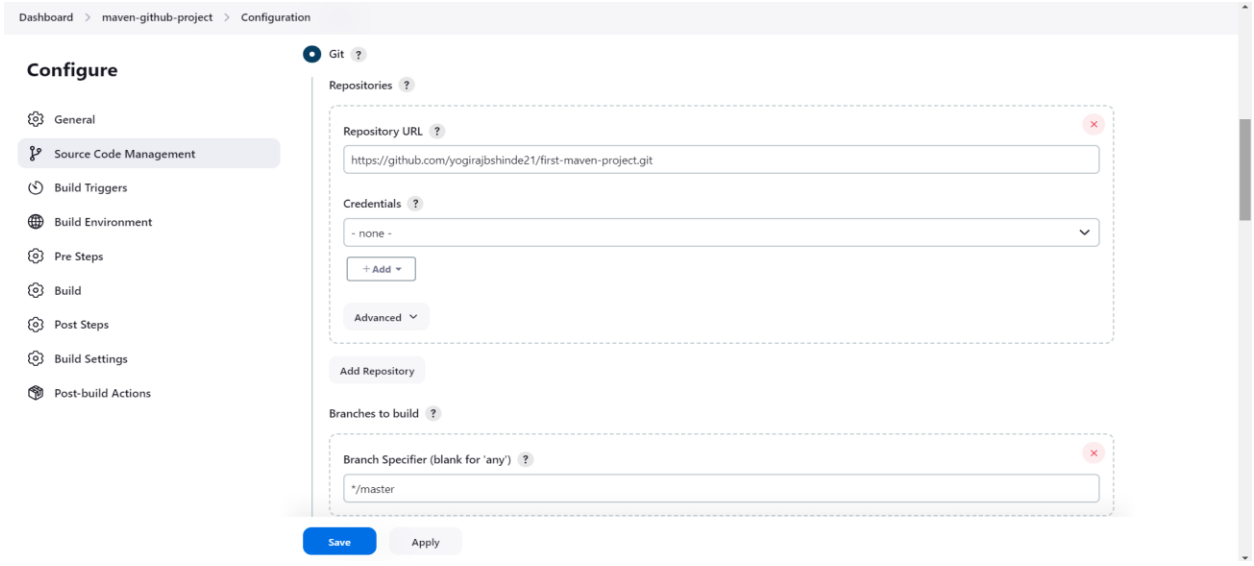



## 6. Integrate with Jenkins:

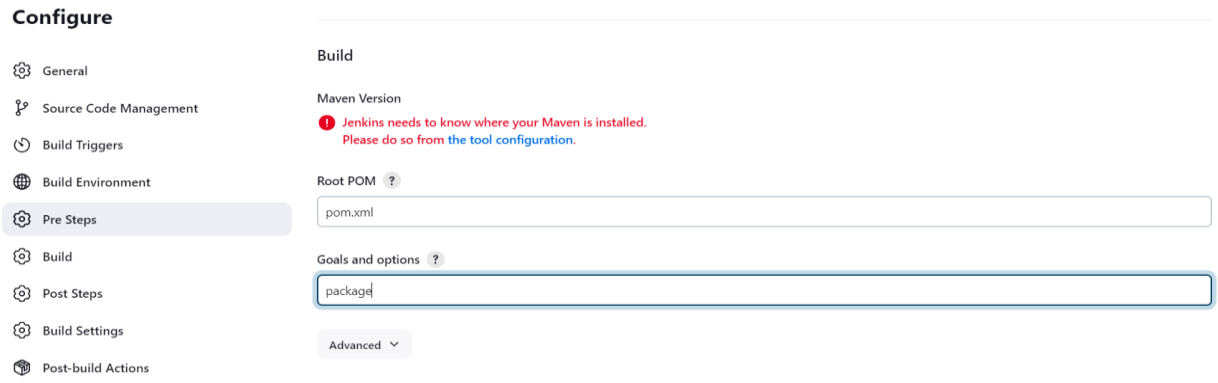
- Create a new job in Jenkins and select **Maven Project**.



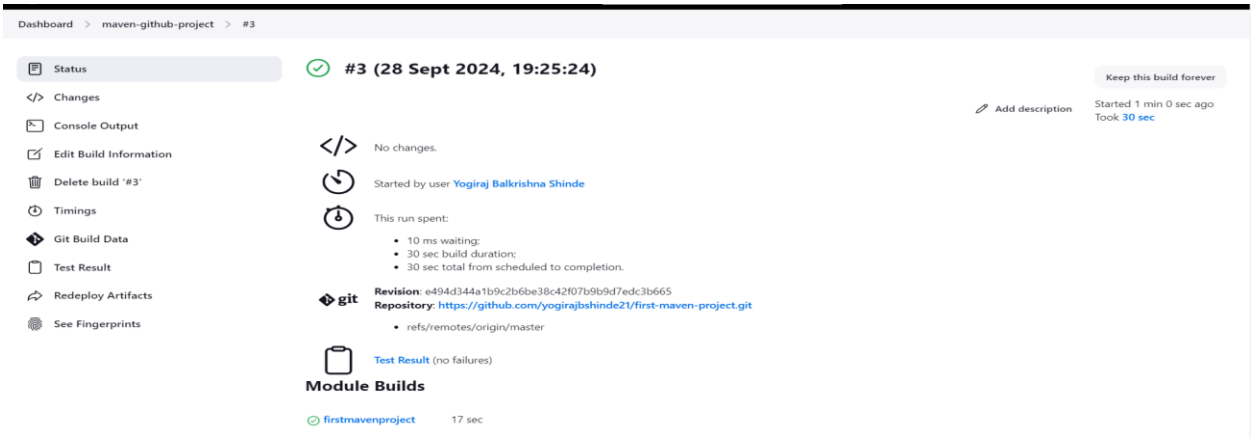
- Configure GitHub repository details in the Source Code Management section.

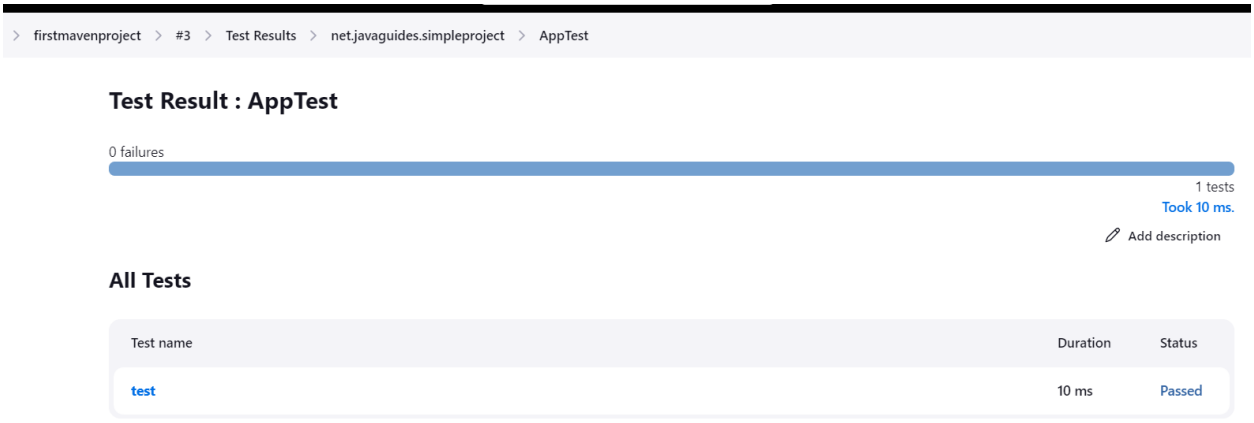



- Set build triggers and goals in the build section (e.g., `clean install`).




- Build the project.






**Conclusion:**

Successfully executed a simple Maven project in Eclipse, pushed it to GitHub, and integrated it with Jenkins for automated builds.

**Name: Kamal Agrahari** **Lab: DevOps**

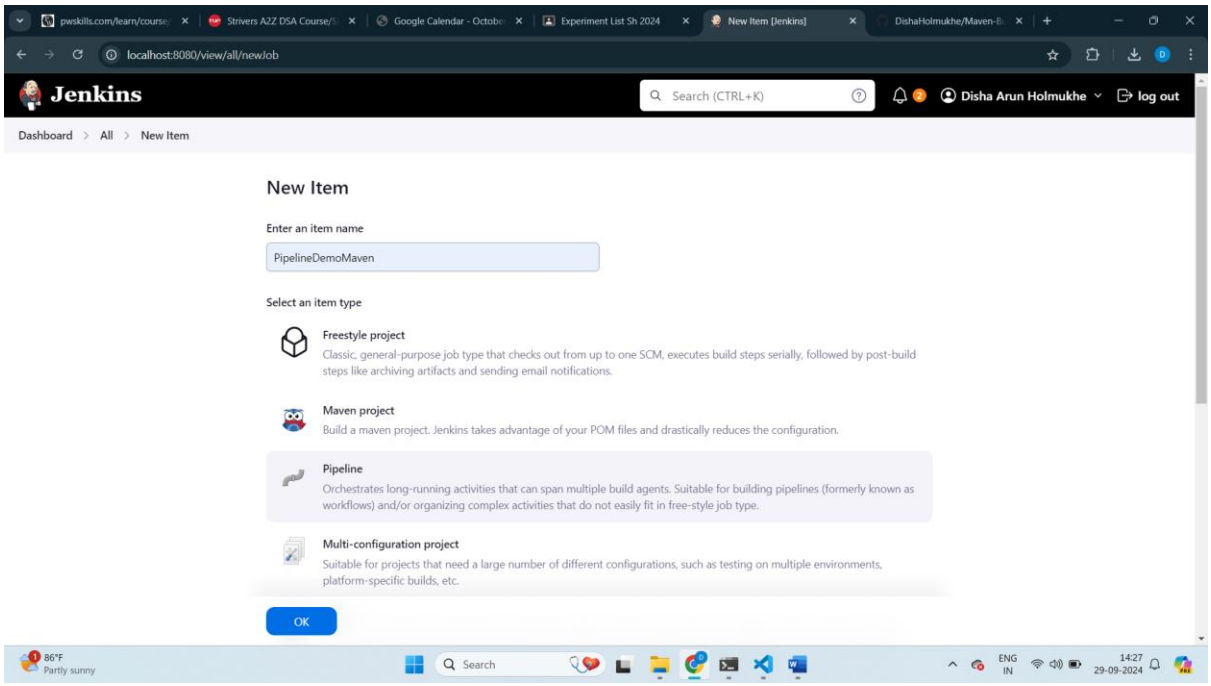
**ID: VU4F2223028** **TE | IT | A**

**Experiment:08**

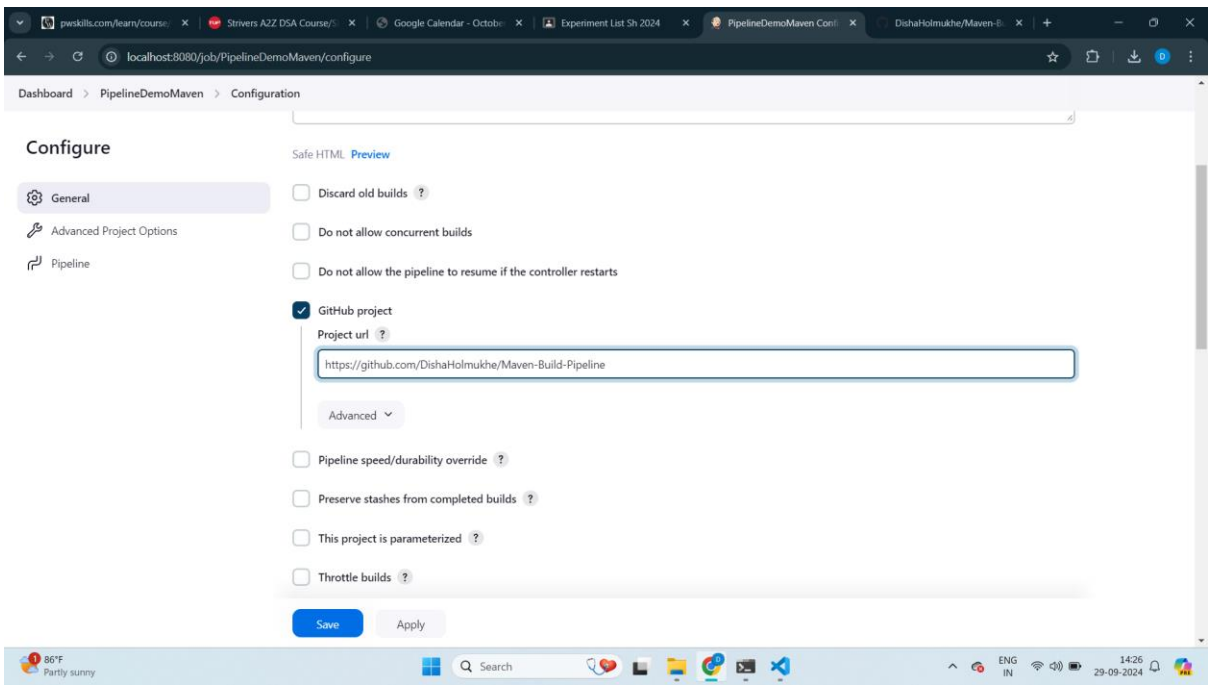
**Aim:** Demonstration to Create Maven Pipeline.

**Theory:**

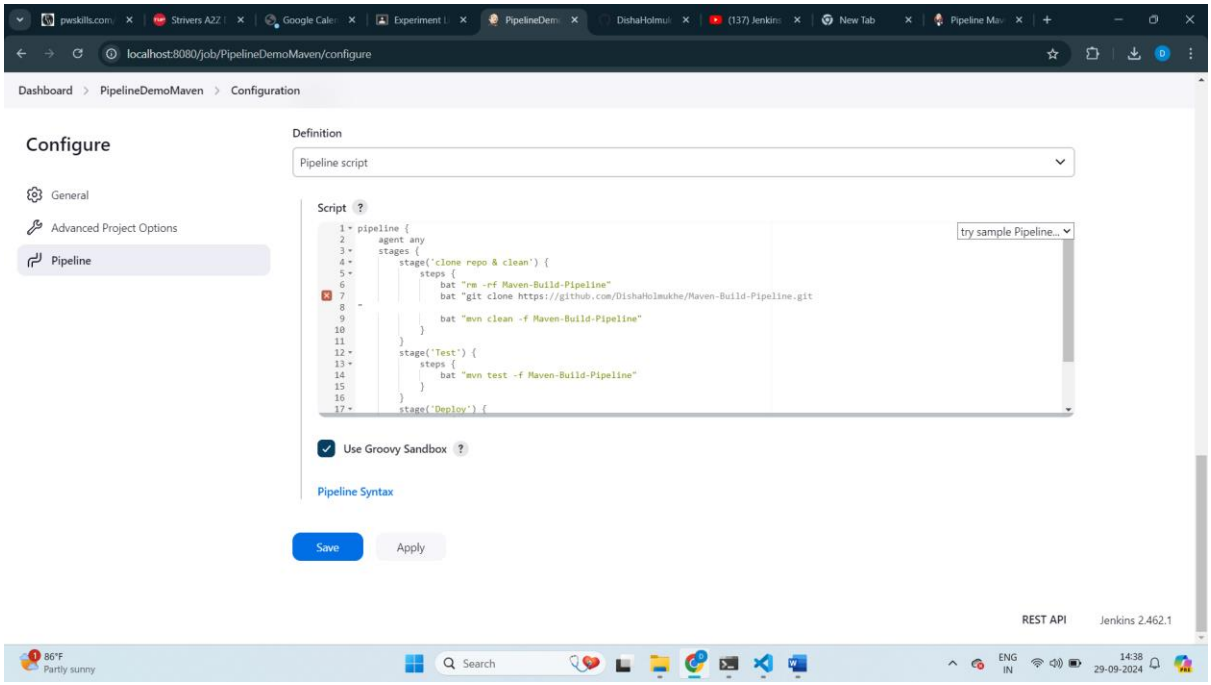
Step1: Create a new item/job with itme type as pipeline.



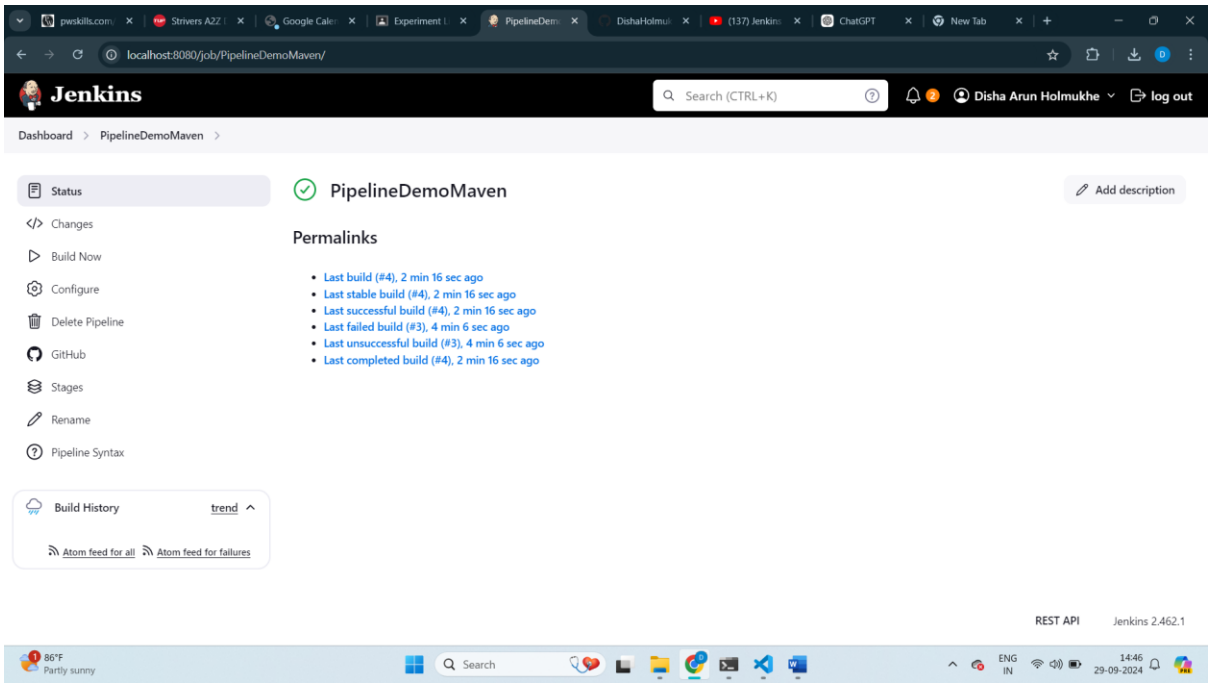

Step2: In General set github project with the github url to the maven project.



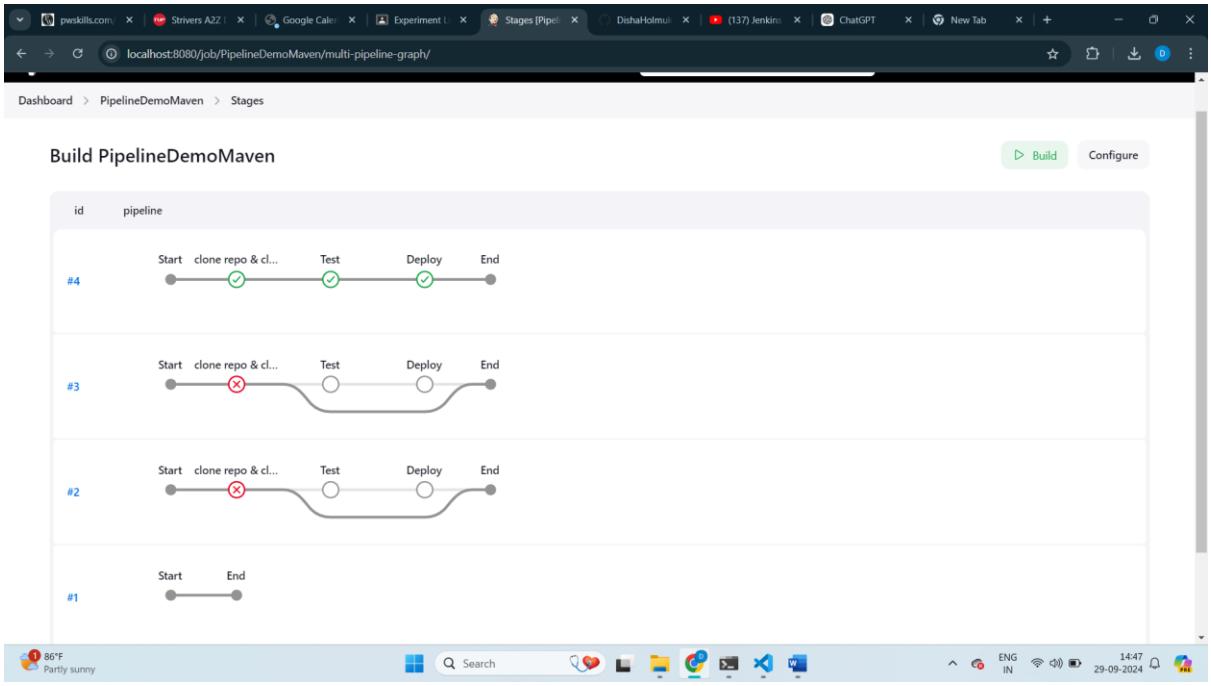
Step3: In Pipeline choose Pipeline Script



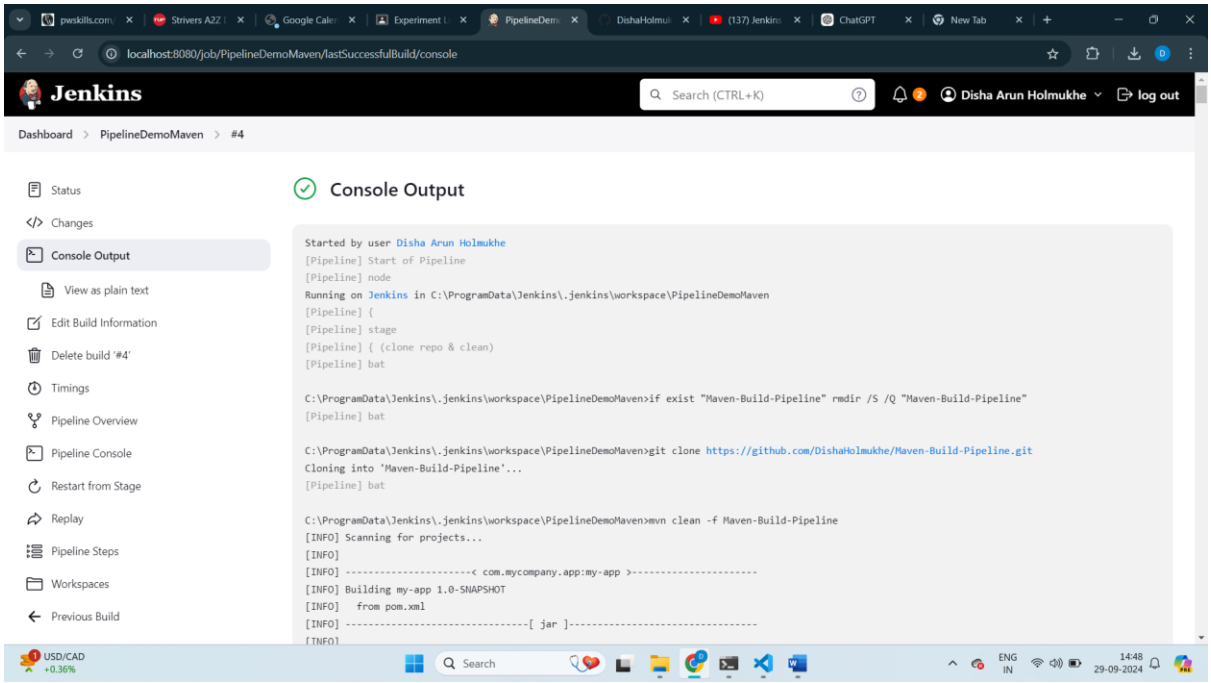

Step4: Click save and Build now.

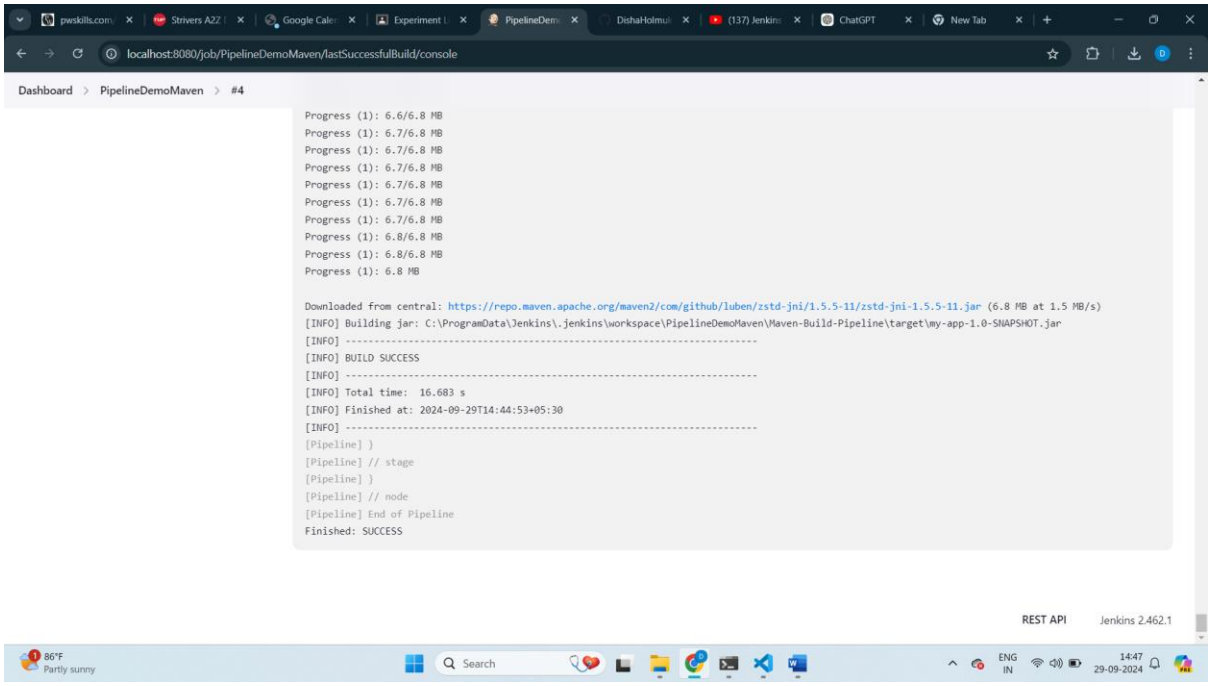



Step5: Click on stages to see the execution of pipeline in each stage.



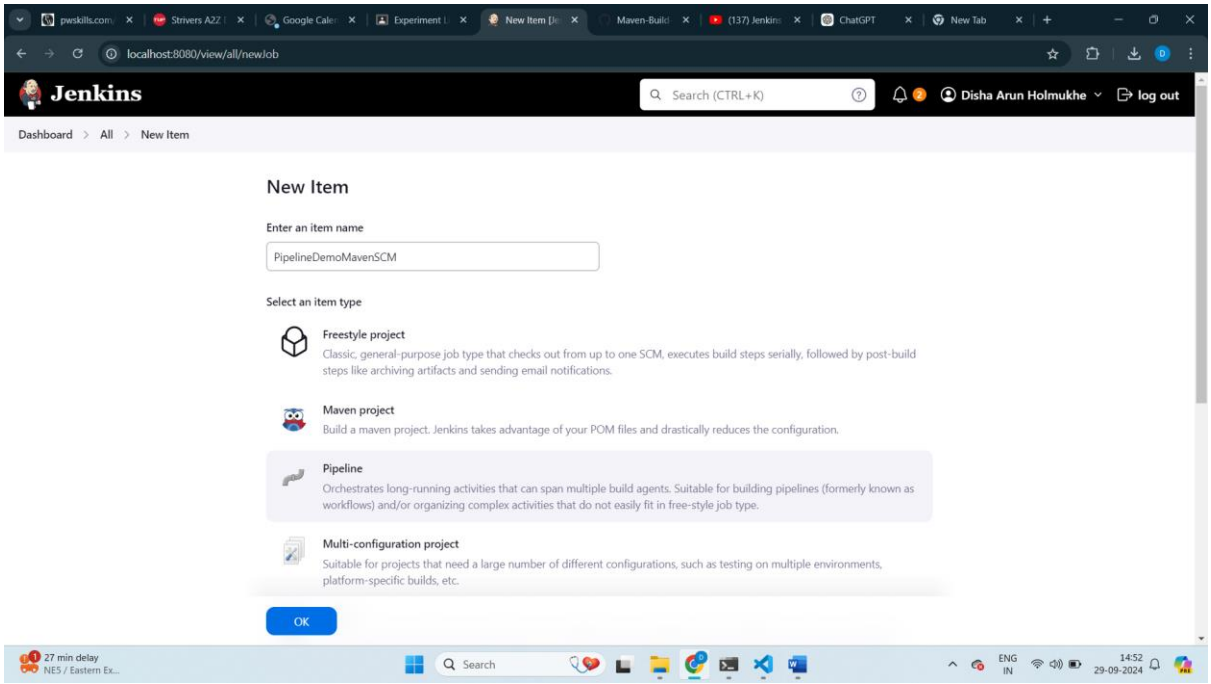

Step6: The console output displays “Finished:SUCCESS”. The pipeline is executed successfully.



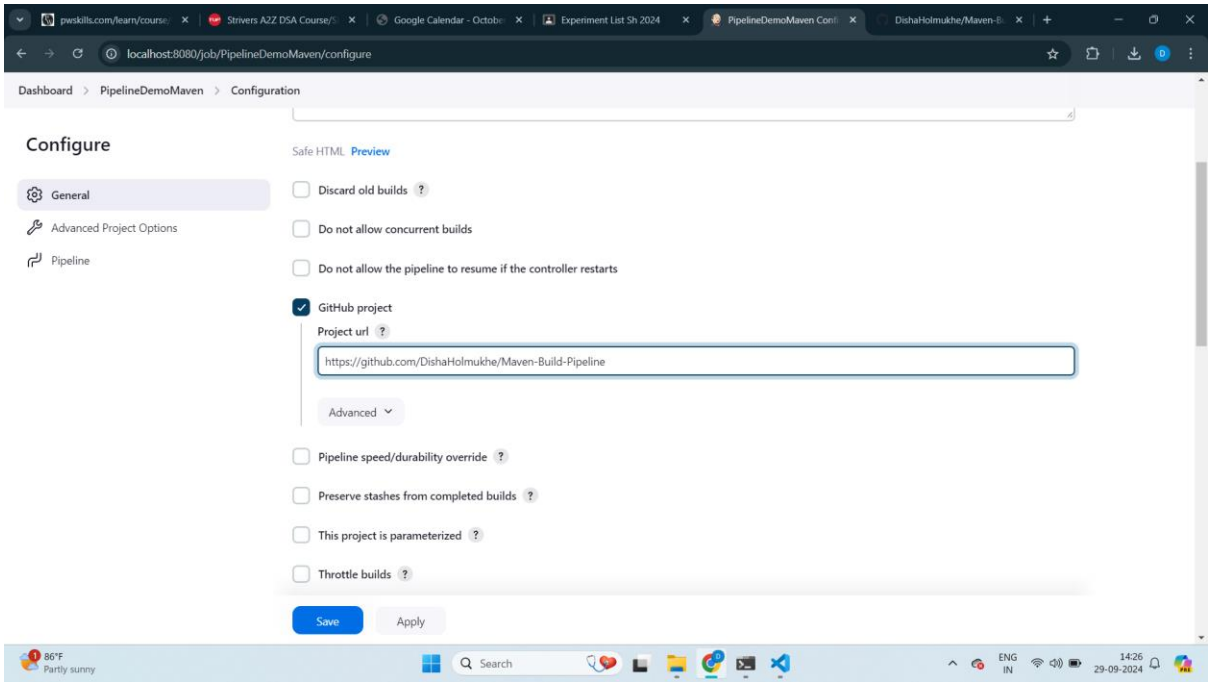



## Building Maven pipeline using Pipeline script SCM

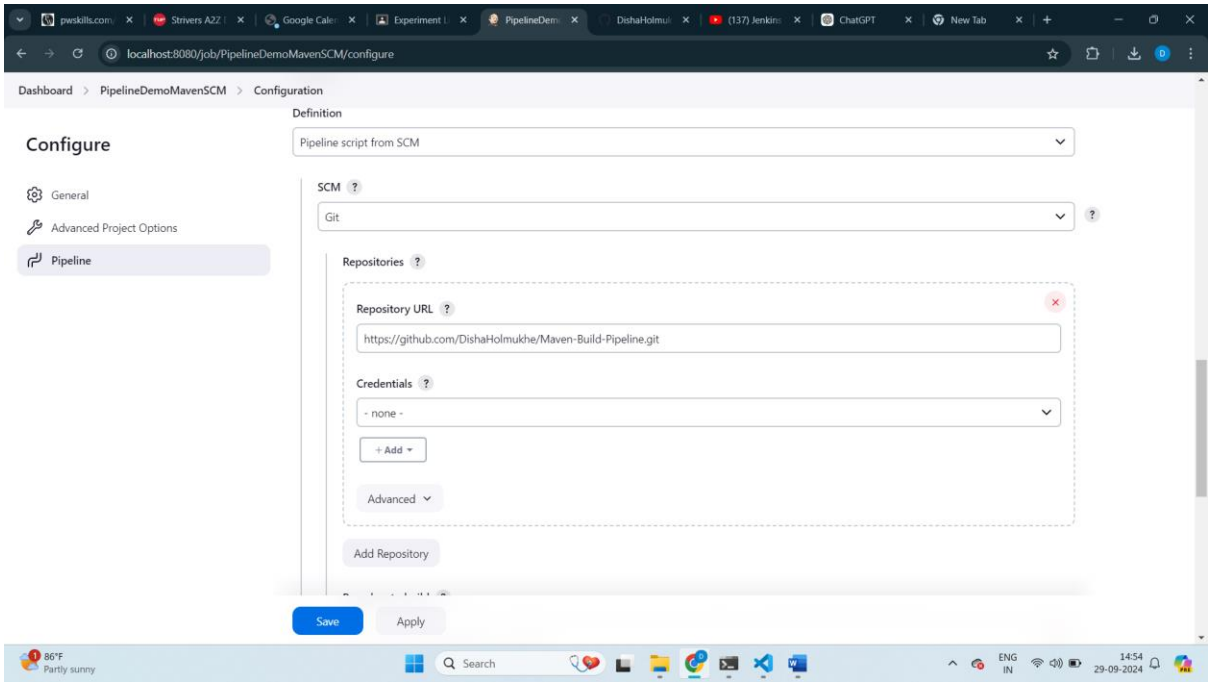
Step1: Create a new item/job with item type as pipeline.



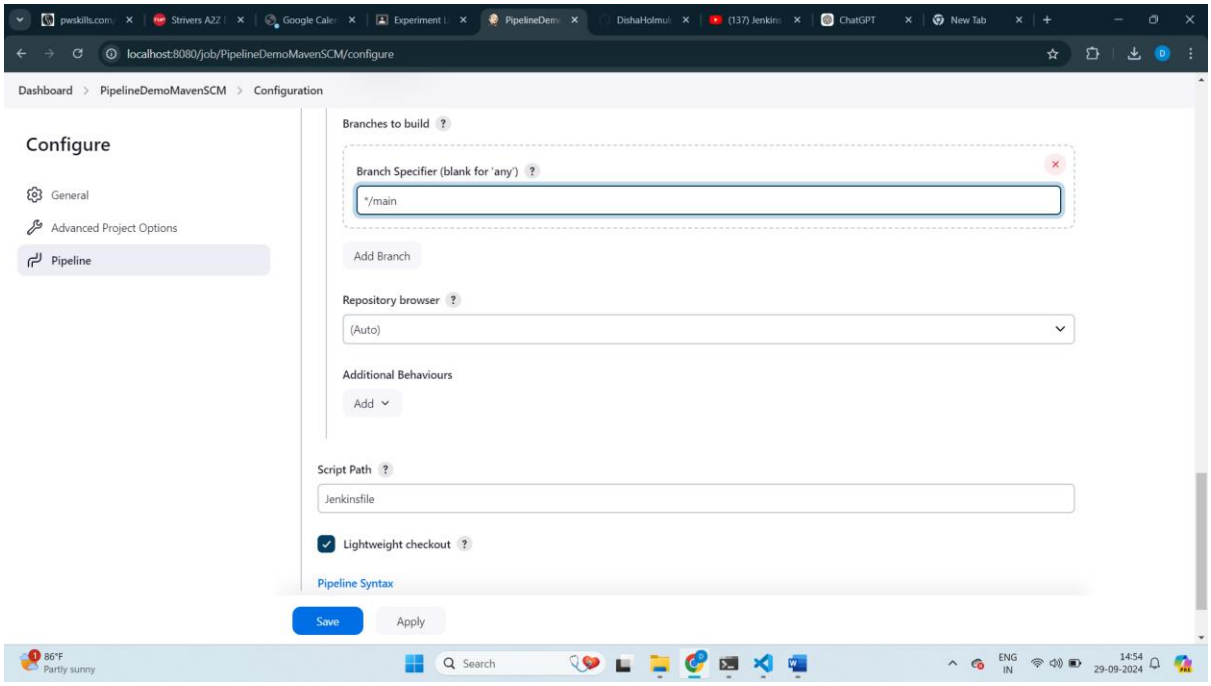

Step2: In General set github project with the github url to the maven project.




Step3: In pipeline select Pipeline Script from SCM.

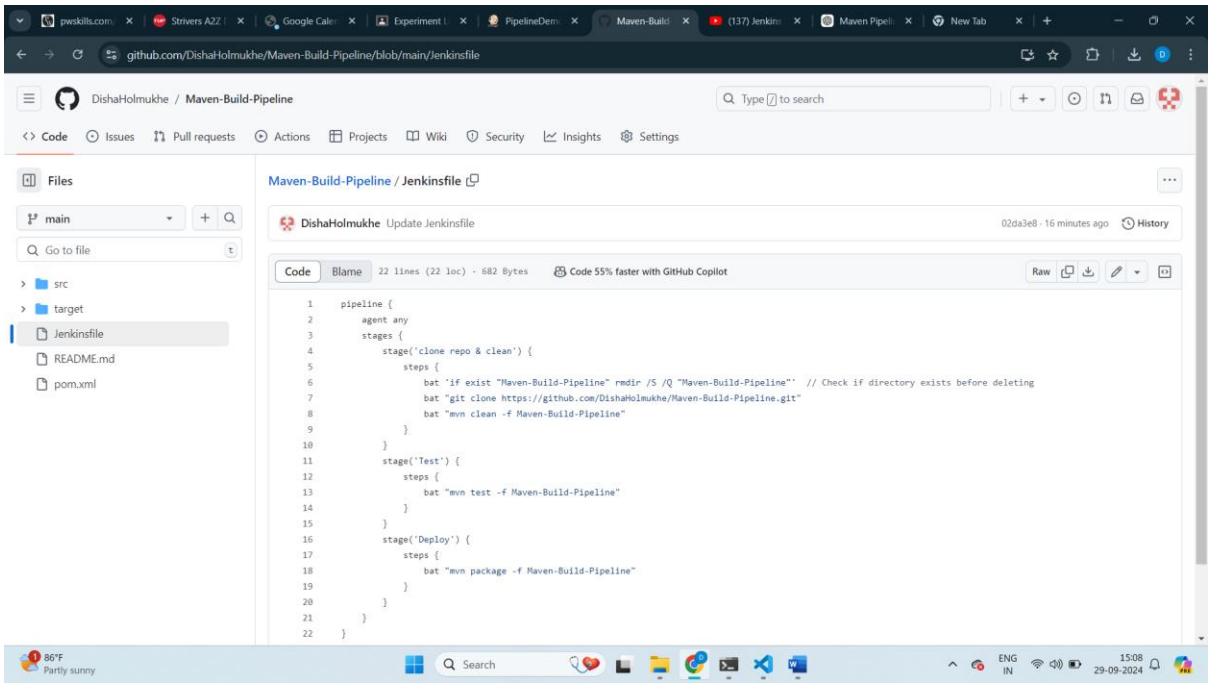



Step4: Set branch to main and Script path to the Jenkins file of your github repository.

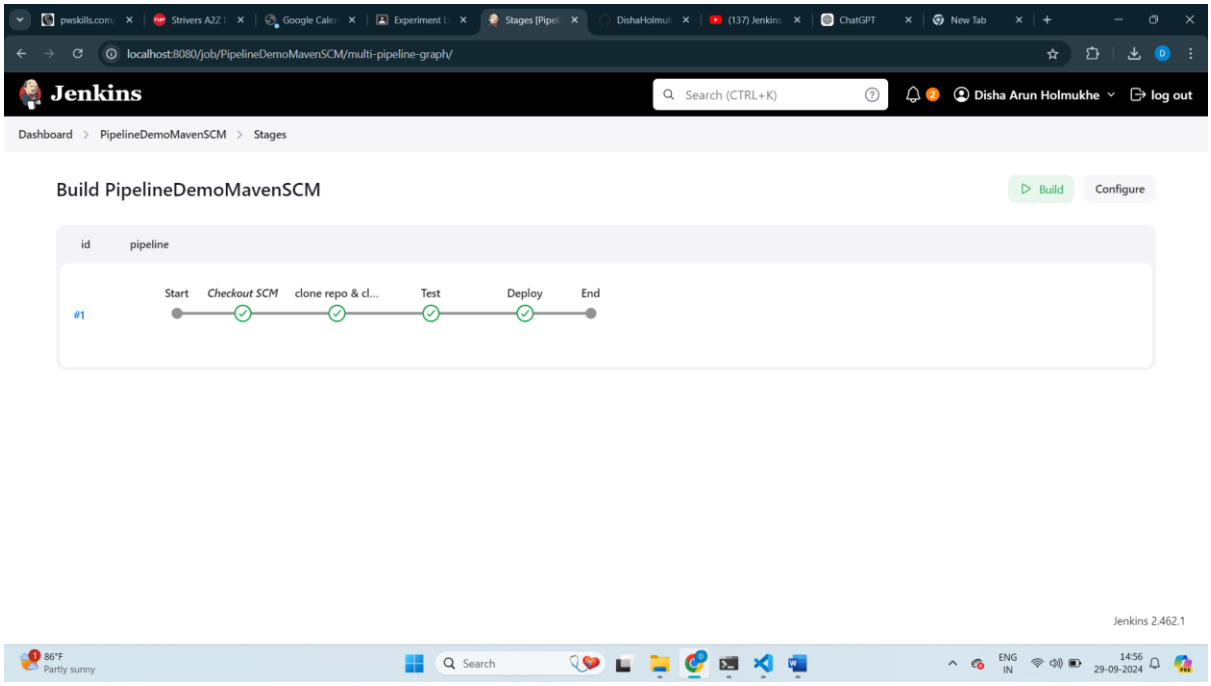


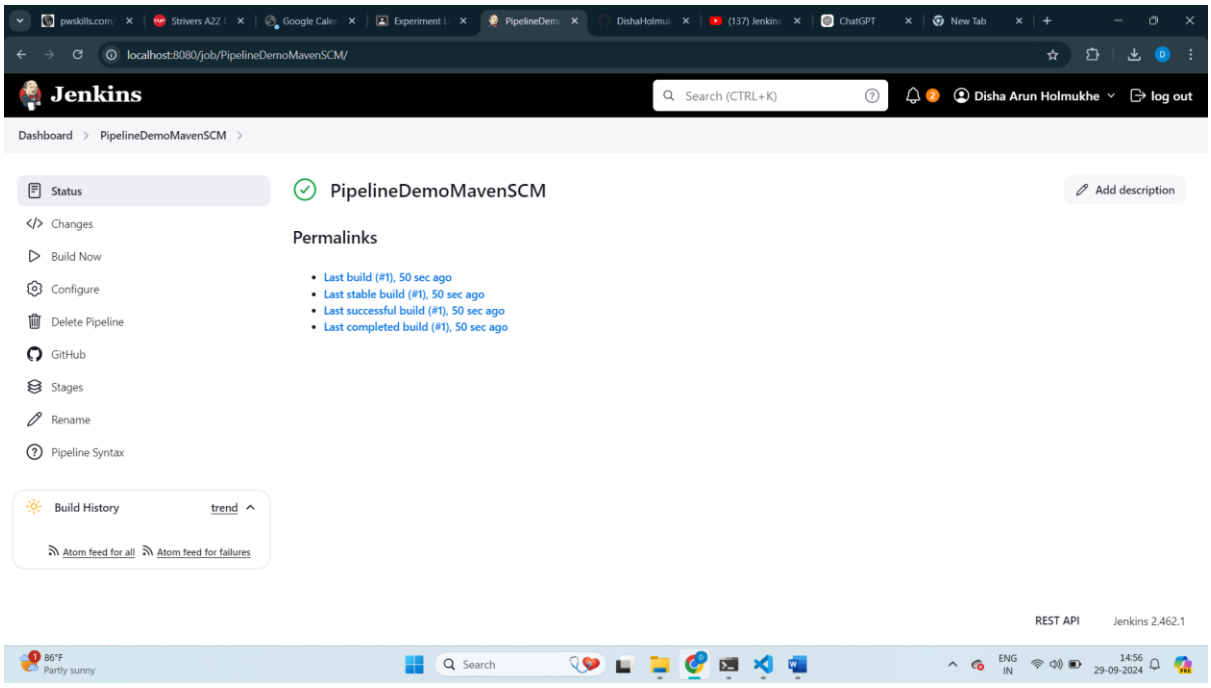

Step5: Click save and build now.

Step6: The Jenkinsfile should be in the Root of your repository.

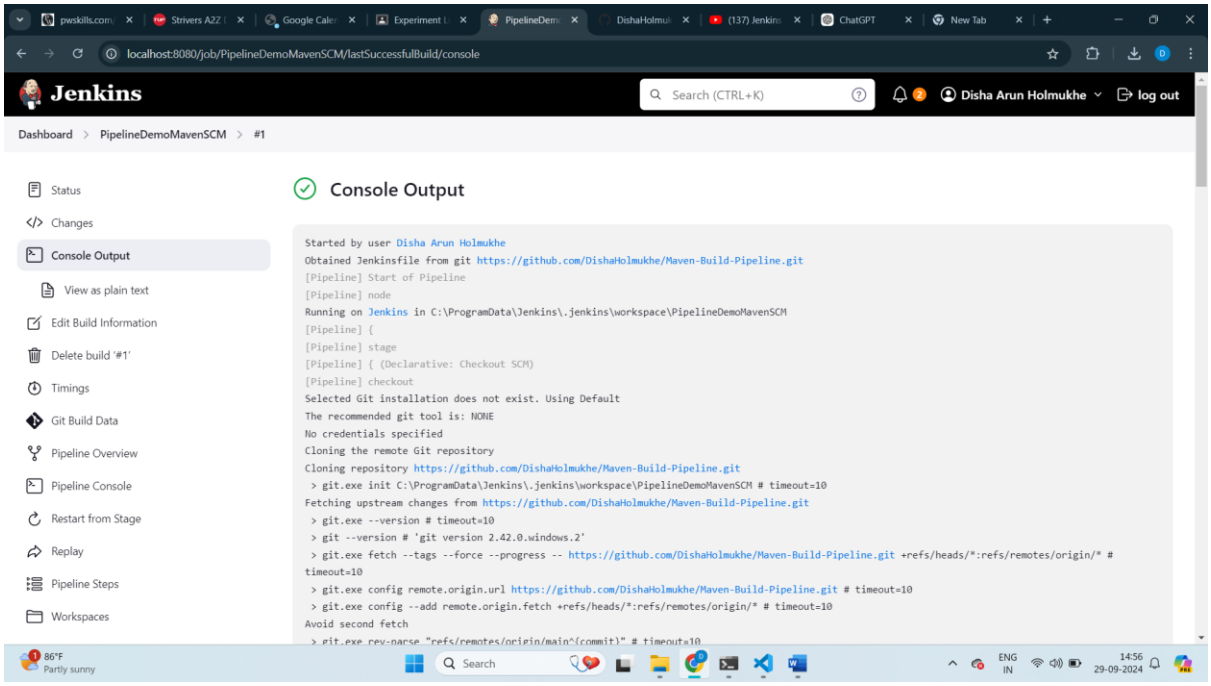


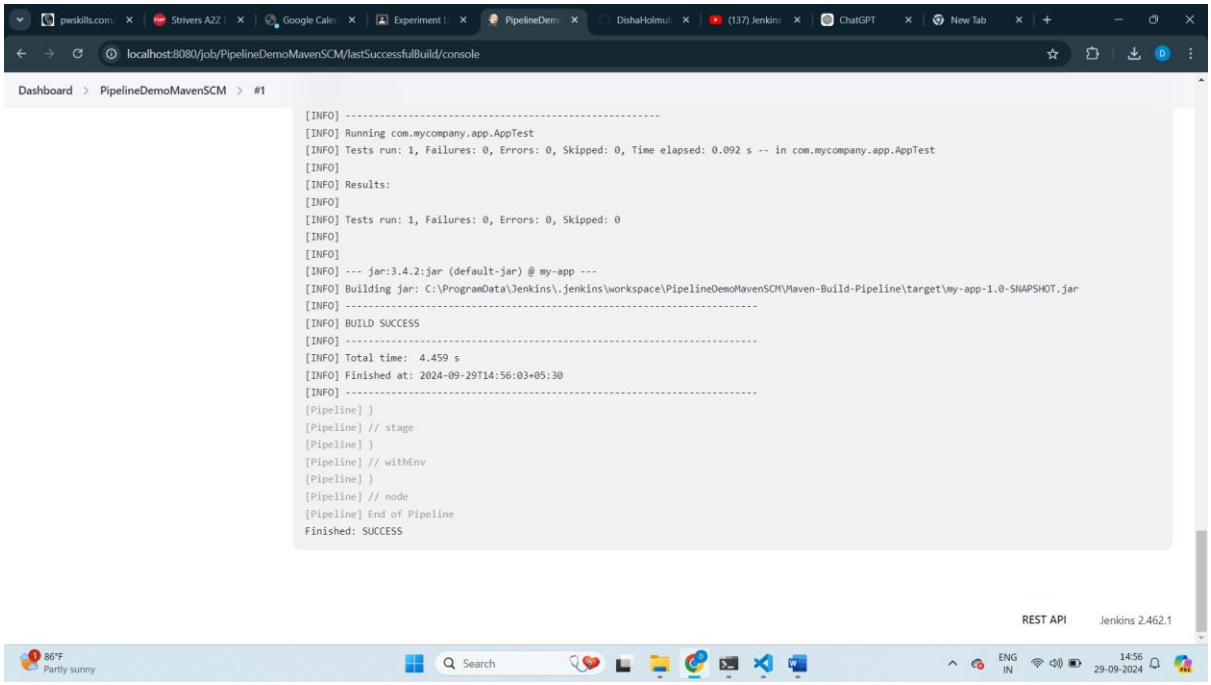

Step5: Click on stages to see the execution of pipeline in each stage.




Step6: The console output displays “Finished : SUCCESS”. The pipeline is executed successfully.




**Conclusion:** Successfully completed the demonstration to Create Maven Pipeline.

Name: Kamal Agrahari
ID: VU4F2223028

Lab: DevOps
Class: TE/A

**Experiment no. 9**

**Aim** : Running selenium test cases to test web-based UI components.

**Theory** :

Running Selenium test cases to test web-based UI components in Python involves several steps. Below, I'll outline a basic setup and provide an example to help you get started.

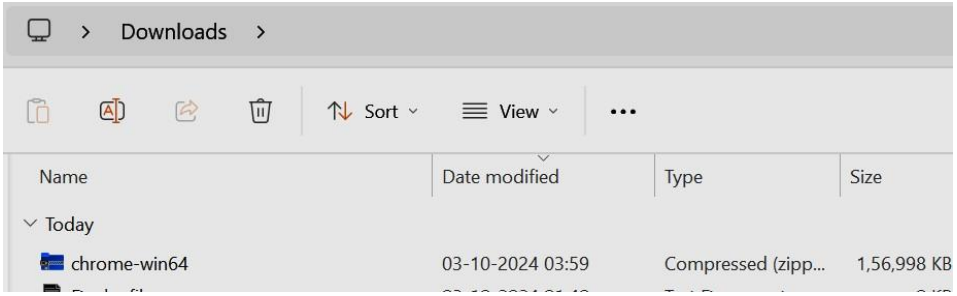
**1. Install Required Packages**

First, ensure you have Python installed on your machine. Then, install Selenium using pip:
pip install selenium

```
C:\Users\ADMIN>pip install selenium
Collecting selenium
  Downloading selenium-4.25.0-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: urllib3<3,>=1.26 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from urllib3[socks]<3,>=1.26->s
elenium) (2.2.0)
Collecting trio~=0.17 (from selenium)
  Downloading trio-0.26.2-py3-none-any.whl.metadata (8.6 kB)
Collecting trio-websocket~=0.9 (from selenium)
  Downloading trio_websocket-0.11.1-py3-none-any.whl.metadata (4.7 kB)
Requirement already satisfied: certifi>=2021.10.8 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from selenium) (2024.2.2)
Requirement already satisfied: typing_extensions~=4.9 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from selenium) (4.10.0)
Collecting websocket-client~=1.8 (from selenium)
  Downloading websocket_client-1.8.0-py3-none-any.whl.metadata (8.0 kB)
Collecting attrs>=23.2.0 (from trio~=0.17->selenium)
  Downloading attrs-24.2.0-py3-none-any.whl.metadata (11 kB)
Collecting sortedcontainers (from trio~=0.17->selenium)
  Downloading sortedcontainers-2.4.0-py2.py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: idna in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from trio~=0.17->selenium) (3.6)
Collecting outcome (from trio~=0.17->selenium)
  Downloading outcome-1.3.0.post0-py2.py3-none-any.whl.metadata (2.6 kB)
Collecting sniffio>=1.3.0 (from trio~=0.17->selenium)
  Downloading sniffio-1.3.1-py3-none-any.whl.metadata (3.9 kB)
Requirement already satisfied: cffi>=1.14 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from trio~=0.17->selenium) (1.16.0)
Collecting wsproto>=0.14 (from trio-websocket~=0.9->selenium)
  Downloading wsproto-1.2.0-py3-none-any.whl.metadata (5.6 kB)
Collecting pysocks!=1.5.7,<2.0,>=1.5.6 (from urllib3[socks]<3,>=1.26->selenium)
  Downloading PySocks-1.7.1-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: pycparser in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from cffi>=1.14->trio~=0.17->selenium)
 (2.21)
Collecting h11<1,>=0.9.0 (from wsproto>=0.14->trio-websocket~=0.9->selenium)
  Downloading h11-0.14.0-py3-none-any.whl.metadata (8.2 kB)
Downloading selenium-4.25.0-py3-none-any.whl (9.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.3/9.7 MB 685.6 kB/s eta 0:00:04
```

**2. Set Up a WebDriver**

Selenium requires a WebDriver to interface with the browser. For example, if you're using Chrome, you'll need ChromeDriver. Make sure you download the driver that matches your installed Chrome version from here.



**3. Example Test Case**

Here's a basic example of a Selenium test case that opens a web page, interacts with elements, and verifies the page title.

Example Code-

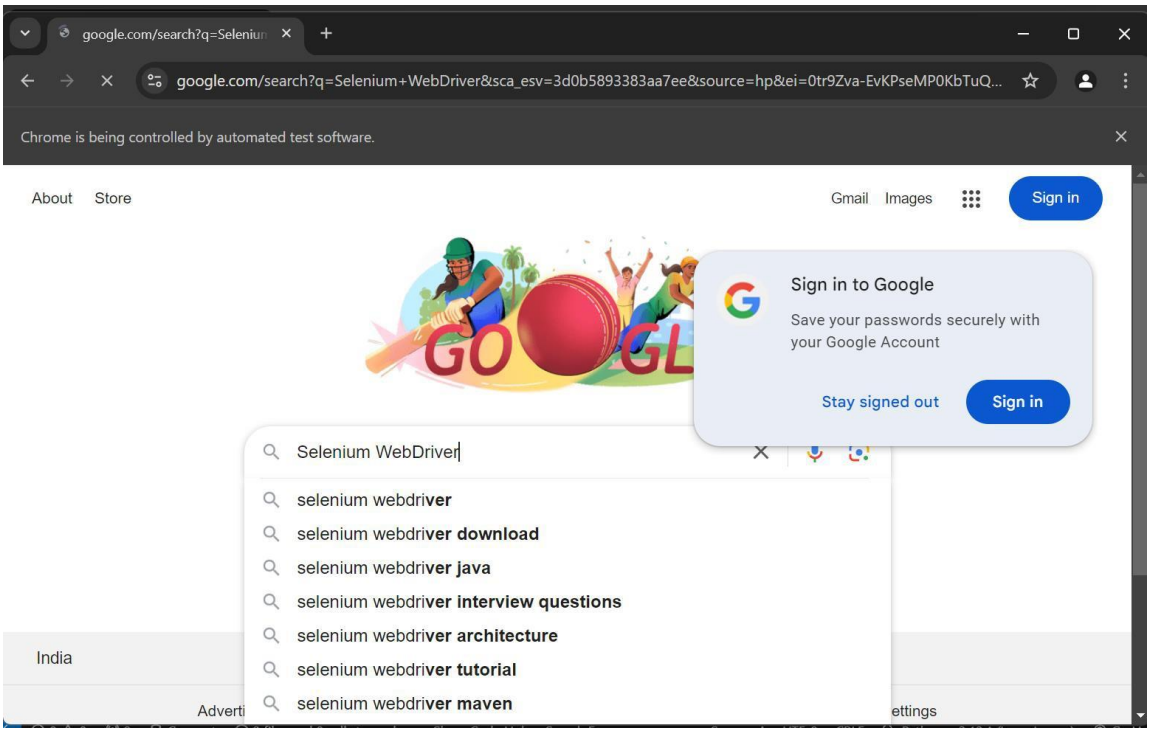
***Test.py***

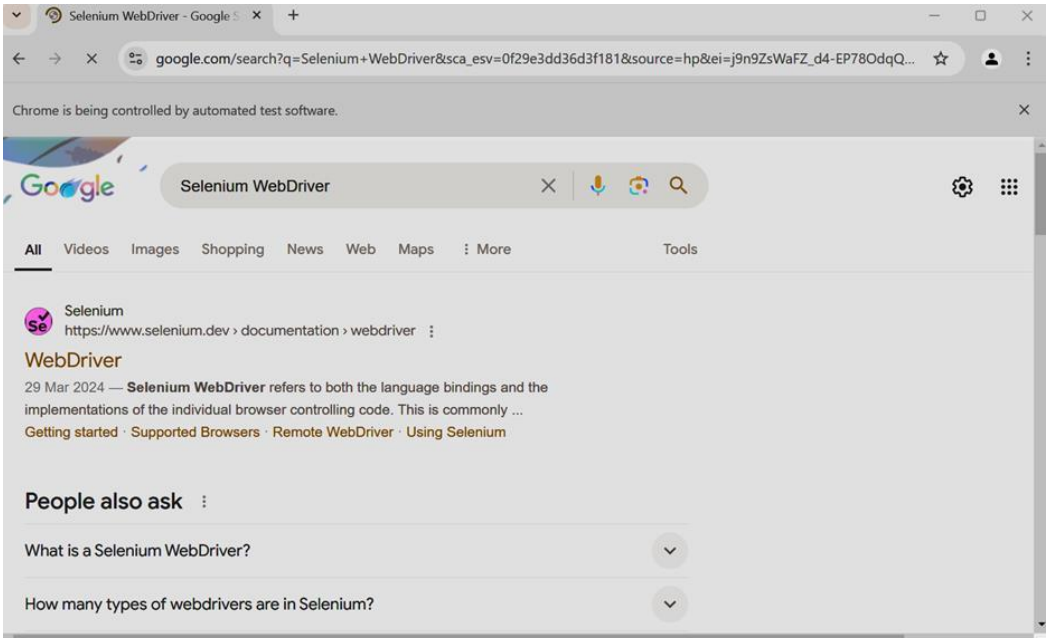
```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys from
selenium.webdriver.chrome.service import Service import
time

# Set the path to your WebDriver (e.g., ChromeDriver)
driver_path='C:\\Users\\ADMIN\\chromedriver-win64\\chromedriver.exe'
#yourChromedriverPath service
= Service(driver_path)
driver = webdriver.Chrome(service=service)
try:
  # Open a web page
  driver.get('http://google.com')  # Replace with your target URL

  # Wait for the page to load
  time.sleep(2)  # Implicit wait; you can also use WebDriverWait for better practices

  # Interact with elements
  search_box = driver.find_element(By.NAME, 'q')  # Example element
  search_box.send_keys('Selenium WebDriver' + Keys.RETURN)

  # Wait for results to load
  time.sleep(2)

  # Verify the title
  assert 'Selenium WebDriver' in driver.title

  print("Test Passed: Page title is correct.")
except Exception as e:
  print(f"Test Failed: {e}")
finally:
  # Close the browser
  driver.quit()
```

**4. Run Your Test Case**

1. Save the script in a .py file (e.g., test_script.py).
2. Run it using Python:

$ python test_script.py




```
C:\Users\ADMIN>python test.py

DevTools listening on ws://127.0.0.1:49242/devtools/browser/5e479cb9-4873-
49b2-8426-051f2a8c8b81
Test Passed: Page title is correct.
```

Conclusion: Hence we have performed selenium test on web-based UI components.

Name: Kamal Agrahari
ID: VU4F2223028

Lab: DevOps
Class: TE / A

**Experiment No. 10**

Aim: Installation of Docker.

Theory:

- Docker

- Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization.
- Containers are lightweight, portable, and self-sufficient environments that package an application and all its dependencies, ensuring consistent operation across different environments.

- Key Concepts:

  - Container: A lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools.
  - Image: A read-only template used to create containers. Images can be thought of as the blueprint for a container.
  - Dockerfile: A text file containing a series of instructions on how to build a Docker image.
  - Docker Hub: A cloud-based repository for sharing Docker images, allowing you to download existing images or upload your own.
  - Docker Engine: The core component of Docker, responsible for running containers and managing container lifecycle.

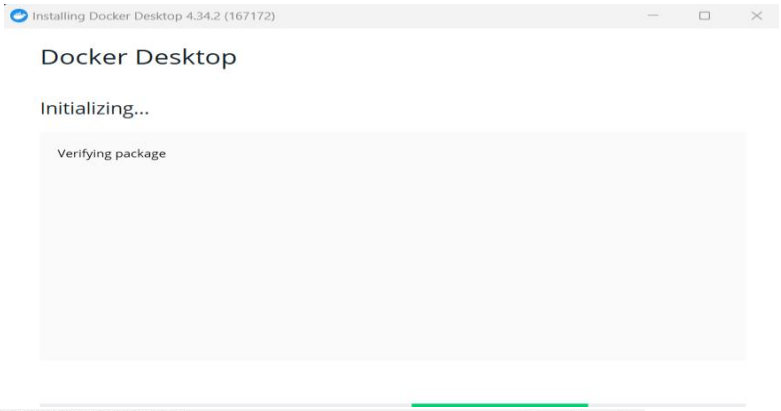
- Installation of Docker

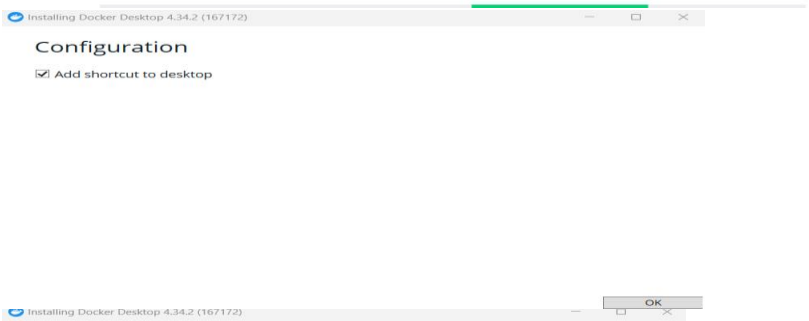
- Requirements: Windows 10 64-bit: Pro, Enterprise, or Education (Build 15063 or later).

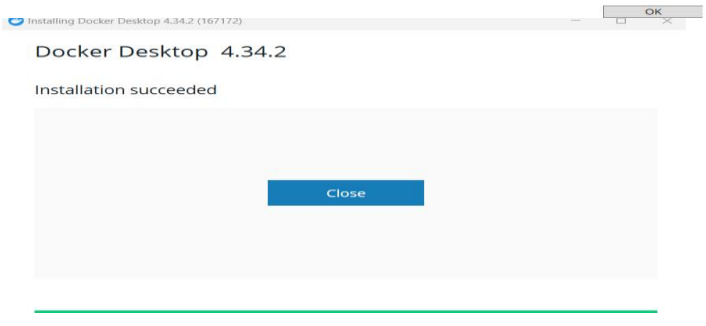
- Steps:

1. Download Docker Desktop: Go to the Docker Desktop for Windows page and download the installer.

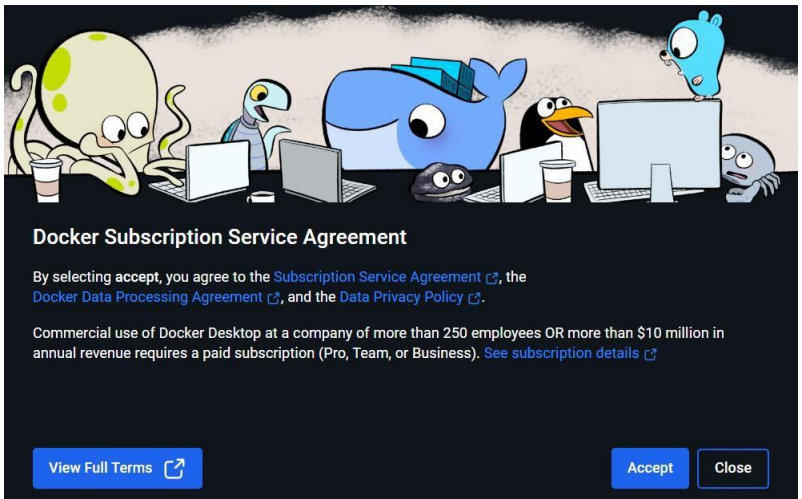

2. Run the Installer: Double-click the downloaded file and follow the installation instructions. Ensure the “Install required Windows components for WSL 2” option is checked if prompted.

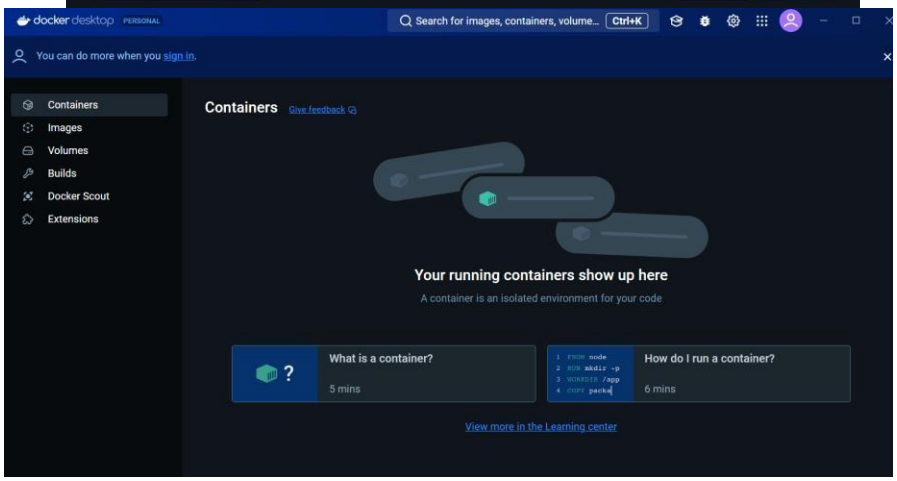


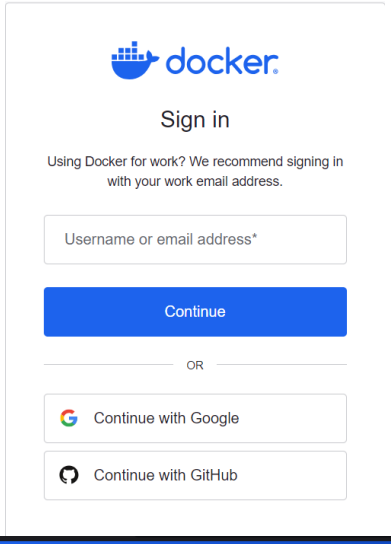



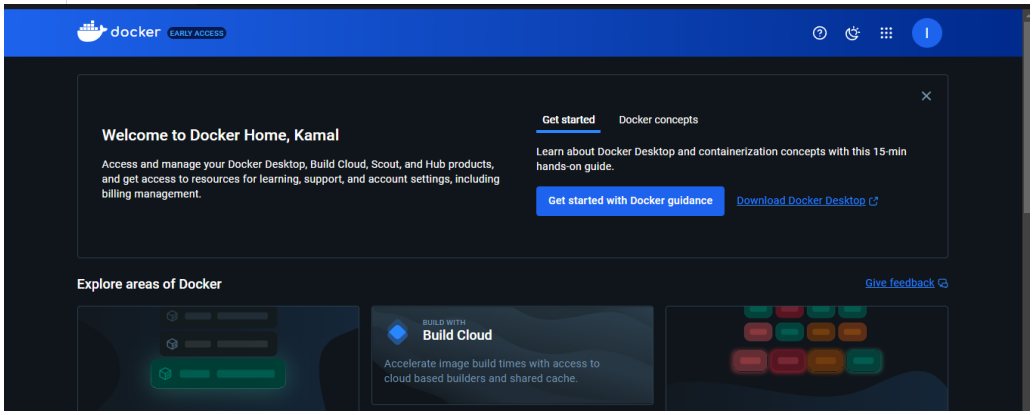
3. Start Docker Desktop: After installation, launch Docker Desktop from the Start menu.



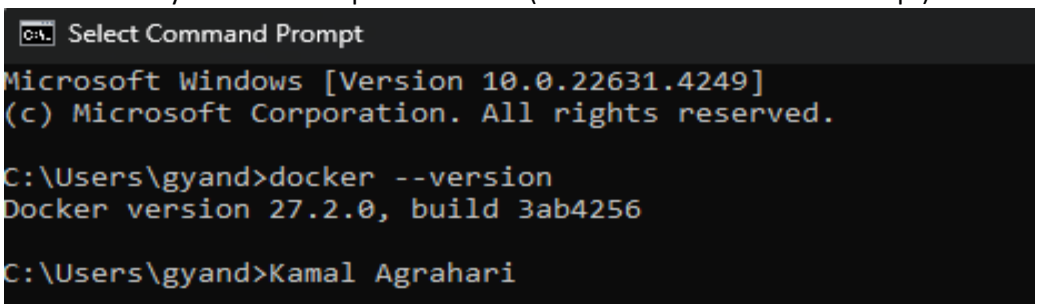



4. Configuration: Follow the onboarding instructions to complete the setup, including creating a Docker account if necessary.




5. Verify Installation: Open a terminal (PowerShell or Command Prompt) and run:




Conclusion:

Hence, we successfully installed the Docker.

Name: Kamal Agrahari
ID: VU4F2223028

Lab: DevOps
Class: TE / A

**Experiment No. 11**

Aim: Building Docker Image.

Theory:

 What is a Docker Image?

- A Docker image is a lightweight, stand-alone, and executable package that includes everything needed to run an application, including the code, runtime, libraries, environment variables, and configuration files.
- Images are immutable, meaning once they are created, they cannot be changed. Instead, new images are built on top of existing images.

 Key Characteristics of Docker Images:

- Layered File System: Docker images are composed of layers. Each layer represents a set of file changes or operations, such as adding files, modifying files, or installing software. This layering system allows for efficient storage and sharing since multiple images can share common layers.
- Read-Only: Once an image is created, it becomes read-only. Any changes made to a running container (an instance of the image) are stored in a new writable layer on top of the image.
- Versioning: Images can be tagged and versioned, allowing you to manage different versions of your application easily.

 Building a Docker Image

1. Create a Dockerfile

Dockerfile

```
#STEP1 sepcify the base image
FROM alpine

#STEP2 download and install dependencies
RUN apk add --update redis

#STEP3 setup the strtup command
CMD ["redis-server"]
```

2. Build the Docker Image

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Sayali Nimbalkar\Desktop\Docker File> docker build .
[+] Building 10.5s (6/6) FINISHED docker:desktop-linux => sha256:43c4264eed91be 3.62MB / 3.62MB  1.3s
 => [internal] load build definition from Do  0.0s
 => => transferring dockerfile: 211B          0.0s
 => [internal] load metadata for docker.io/l  5.0s
 => [internal] load .dockerignore             0.0s
 => => transferring context: 2B               0.0s
 => [1/2] FROM docker.io/library/alpine:late  1.7s
 => => resolve docker.io/library/alpine:late  0.1s
 => => sha256:43c4264eed91be 3.62MB / 3.62MB  1.3s
 => => extracting sha256:43c4264eed91be63b20  0.3s
 => [2/2] RUN apk add --update redis          2.9s
 => => resolve docker.io/library/alpine:late  0.1s
 => => sha256:43c4264eed91be 3.62MB / 3.62MB  1.3s
 => => extracting sha256:43c4264eed91be63b20  0.3s
 => => resolve docker.io/library/alpine:late  0.1s
 => => resolve docker.io/library/alpine:late  0.1s
 => => sha256:43c4264eed91be 3.62MB / 3.62MB  1.3s
 => => extracting sha256:43c4264eed91be63b20  0.3s
 => => resolve docker.io/library/alpine:late  0.1s
 => => sha256:43c4264eed91be 3.62MB / 3.62MB  1.3s
 => => resolve docker.io/library/alpine:late  0.1s
 => => sha256:43c4264eed91be 3.62MB / 3.62MB  1.3s
 => => exporting layers                       0.5s
 => => exporting manifest sha256:111f83d7a91  0.0s
 => => exporting layers                       0.5s
```

3. Verify the Built Image

```
C:\Users\gyand>docker images
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
nginx/nginx-ingress     latest    3f339f1b3bd8   47 hours ago    281MB

C:\Users\gyand>Kamal Agrahari_
```

4. Run the image by the “image ID” to check if it’s working correctly.

```
PS C:\Users\Sayali Nimbalkar\Desktop\Docker File> docker run a4e47e3c2b51
1:C 02 Oct 2024 12:00:09.836 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 02 Oct 2024 12:00:09.836 * Redis version=7.2.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 02 Oct 2024 12:00:09.836 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
1:M 02 Oct 2024 12:00:09.837 * monotonic clock: POSIX clock_gettime
1:M 02 Oct 2024 12:00:09.838 * Running mode=standalone, port=6379.
1:M 02 Oct 2024 12:00:09.839 * Server initialized
1:M 02 Oct 2024 12:00:09.840 * Ready to accept connections tcp
1:signal-handler (1727870691) Received SIGINT scheduling shutdown...
1:M 02 Oct 2024 12:04:51.412 * User requested shutdown...
1:M 02 Oct 2024 12:04:51.413 * Saving the final RDB snapshot before exiting.
1:M 02 Oct 2024 12:04:51.423 * DB saved on disk
1:M 02 Oct 2024 12:04:51.424 # Redis is now ready to exit, bye bye...
```

5. Tagging a name to the image.

```
PS C:\Users\Sayali Nimbalkar\Desktop\Docker File> docker build -t sayali/redis:latest .
[+] Building 2.5s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 211B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d
 => => resolve docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d
 => CACHED [2/2] RUN apk add --update redis
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:111f83d7a91f0b1bf7b7063368e9db52a9ae8e65b1ae22326c2ca71a9af8b7e9
 => => exporting config sha256:68f0da0c44ea6b47fc9c4d0a87c9924021dde87d654438d1608a67b539466420
 => => exporting attestation manifest sha256:8b664ca3422f4fd4b168c21ff2244db317356d2a8b326d40b2af69be27971748
 => => exporting manifest list sha256:0d53756c965006149bdfdf69017a91cd368ce1dd16a4295bb6d30a49eff076c9
 => => naming to docker.io/sayali/redis:latest
 => => unpacking to docker.io/sayali/redis:latest
```

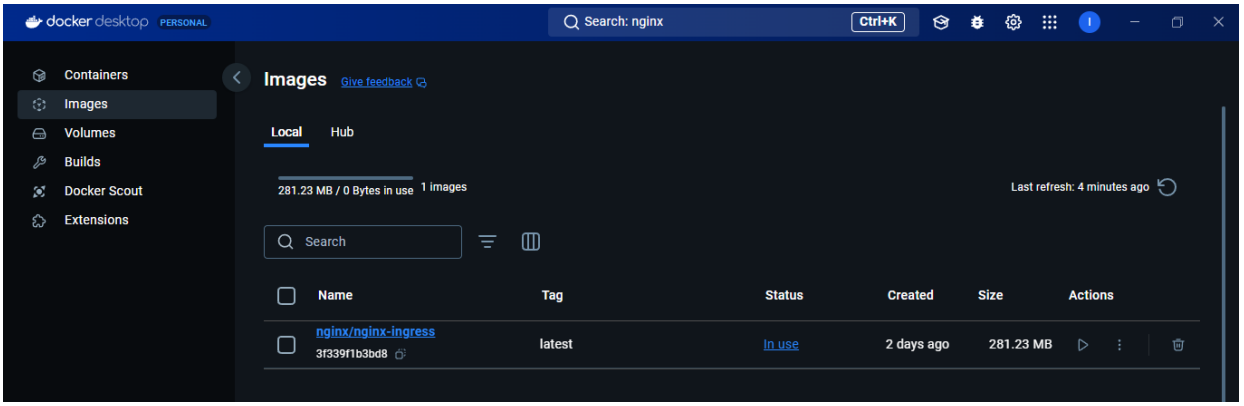
6. Running the image by given name.

```
PS C:\Users\Sayali Nimbalkar\Desktop\Docker File> docker run sayali/redis
1:C 02 Oct 2024 12:10:20.241 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 02 Oct 2024 12:10:20.241 * Redis version=7.2.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 02 Oct 2024 12:10:20.241 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
1:M 02 Oct 2024 12:10:20.241 * monotonic clock: POSIX clock_gettime
1:M 02 Oct 2024 12:10:20.244 * Running mode=standalone, port=6379.
1:M 02 Oct 2024 12:10:20.247 * Server initialized
1:M 02 Oct 2024 12:10:20.251 * Ready to accept connections tcp
1:signal-handler (1727871074) Received SIGINT scheduling shutdown...
1:M 02 Oct 2024 12:11:14.978 * User requested shutdown...
1:M 02 Oct 2024 12:11:14.978 * Saving the final RDB snapshot before exiting.
1:M 02 Oct 2024 12:11:14.988 * DB saved on disk
1:M 02 Oct 2024 12:11:14.988 # Redis is now ready to exit, bye bye...
PS C:\Users\Sayali Nimbalkar\Desktop\Docker File>
```

7. Verifying it.

```
C:\Users\gyand>docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
nginx/nginx-ingress     latest    3f339f1b3bd8   47 hours ago     281MB

C:\Users\gyand>Kamal Agrahari_
```

8. You can see the image in your docker desktop.




Conclusion:

Hence, we successfully build the Docker image.

Name: Kamal Agrahari Lab: DevOps

ID: VU4F2223028 TE | IT | A

## Experiment No. 12

**Title:** Demonstration of ansible automation.

**Theory:**

**Ansible** is an open-source automation tool that simplifies the process of configuration management, application deployment, and task automation across a network of machines. It uses a simple and human-readable language (YAML) for writing automation scripts called **playbooks**. This demonstration outlines the steps to automate a basic task using Ansible.

**Objective**

In this demonstration, we will automate the installation of **Nginx** on multiple remote servers using Ansible.

**Prerequisites**

1. **Ansible Installed**: Ensure Ansible is installed on your control machine (the machine from which you will run Ansible).
2. **SSH Access**: Ensure that you have SSH access to the target machines and that they are listed in the Ansible inventory file.
3. **Target Machines**: Have at least two target machines (remote servers) ready for Nginx installation.

**Steps to follow:**

Step 1: Create an inventory file (hosts.ini) to define the target machines. This file specifies the IP addresses or hostnames of the servers where you want to deploy Nginx.

```
[webservers]
server1 ansible_host=192.168.1.10
server2 ansible_host=192.168.1.11
```

**Step 2:** Create a playbook file (install_nginx.yml) to describe the tasks needed to install Nginx. This playbook will include tasks for installing Nginx and starting the service.

```
---
- name: Install Nginx on web servers
hosts: webservers
become: yes  # Use sudo to install packages
tasks:
- name: Update package manager cache
apt:
update_cache: yes

- name: Install Nginx
apt:
name: nginx
state: present

- name: Start Nginx service
service:
name:  nginx
state: started
enabled: yes
```

**Step 3:** To execute the playbook and install Nginx on the target servers, run the following command in your terminal:

```
ansible-playbook -i hosts.ini install_nginx.yml
```

**Step 4:** After running the playbook, you can verify that Nginx is installed and running on the target servers. SSH into one of the servers and run:

```
systemctl status nginx
```

Ansible to automate the installation of Nginx on multiple servers. By defining the target servers in an inventory file and outlining the installation steps in a playbook, we simplified

the deployment process, ensuring consistency and reducing manual effort. Ansible's ability to manage multiple systems simultaneously makes it an invaluable tool for system administrators and DevOps engineers.

```
devesh@LAPTOP-AKG6B6TB:~$
devesh@LAPTOP-AKG6B6TB:~$ sudo apt install ansible
[sudo] password for devesh:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ansible is already the newest version (9.2.0+dfsg-0ubuntu5).
0 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.
devesh@LAPTOP-AKG6B6TB:~$
```

```
devesh@LAPTOP-AKG6B6TB:~$ cat inventory.txt
target1 ansible_ssh_user=ubuntu ansible_host=54.224.151.191 ansible_ssh_private_key_file=~/learning.pem ansibl
e_python_interpreter=/usr/bin/python3
devesh@LAPTOP-AKG6B6TB:~$
devesh@LAPTOP-AKG6B6TB:~$
```

```
devesh@LAPTOP-AKG6B6TB:~$
devesh@LAPTOP-AKG6B6TB:~$ ansible -m ping all -i inventory.txt
The authenticity of host '54.224.151.191 (54.224.151.191)' can't be established.
ED25519 key fingerprint is SHA256:WYrT2uVrngX0d0LTQiyzh8kEXllAKB0jbdpiZG3PQok.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:1: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
target1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
devesh@LAPTOP-AKG6B6TB:~$
```

**Conclusion:**

Hence, we have successfully demonstrate an ansible automation

**Name: Kamal Agrahari** **Lab: DevOps**
**ID: VU4F2223028** **TE | IT |A**

## Assignment – 1

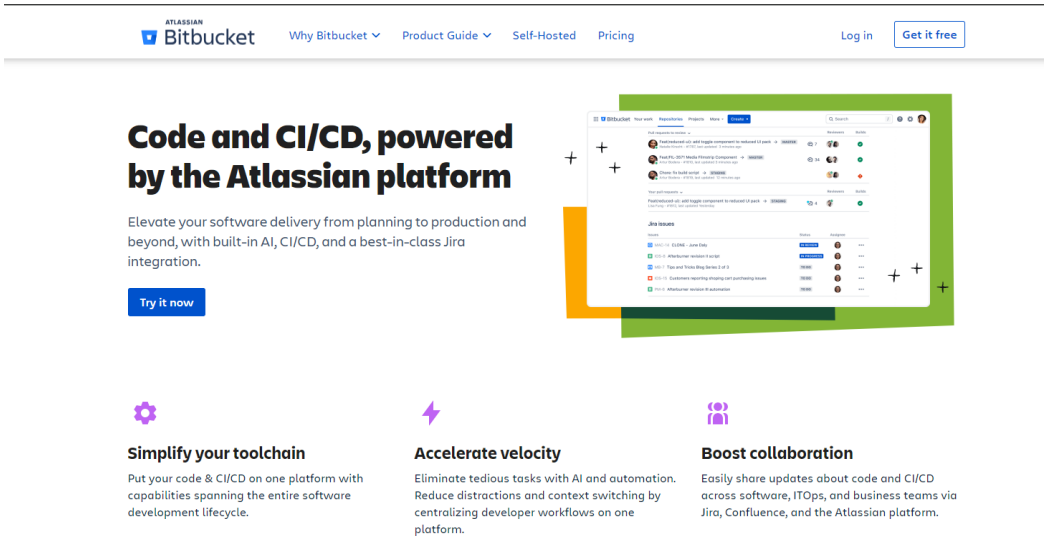
**Aim: Write Case Study on Bitbucket version control System**

### Introduction

Bitbucket is a web-based version control repository hosting service that supports both Git and Mercurial systems. Founded in 2008 and acquired by Atlassian in 2010, Bitbucket has become essential for software development teams globally. Its robust features facilitate collaborative coding and project management, making it a popular platform among developers.

Beyond being a code repository, Bitbucket provides a comprehensive solution for managing software projects. It helps teams maintain code integrity and boost productivity through features like pull requests, issue tracking, and continuous integration and deployment (CI/CD). These capabilities are especially valuable in today's fast-paced development environments.

Security and access control are also key focuses of Bitbucket, offering fine-grained permissions and branch restrictions to ensure only authorized personnel can make significant changes. This is particularly crucial for larger organizations or open-source projects, where maintaining code integrity is vital.



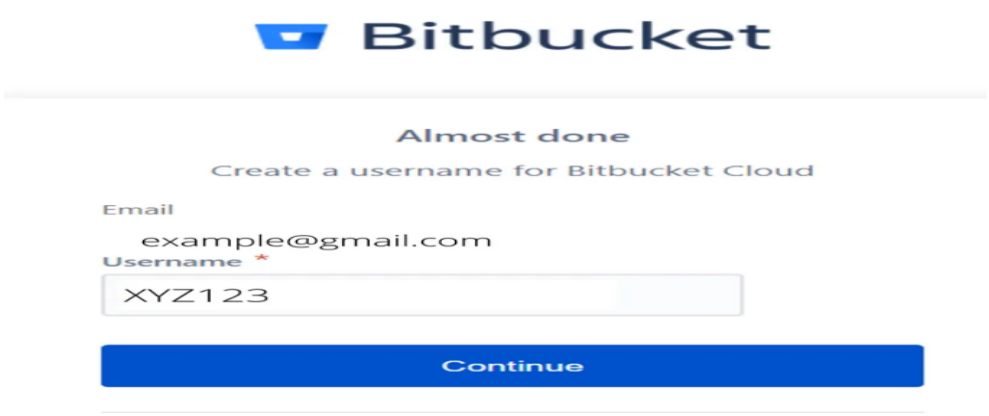

Moreover, Bitbucket seamlessly integrates with other Atlassian products such as Jira and Confluence, enabling a unified workflow for project management and documentation. This integration helps teams align their development efforts with project goals and user feedback, ensuring that the software meets the intended requirements and quality standards.

In this case study, we will explore the implementation details of Bitbucket, its important commands, benefits, and conclude with a reflection on its significance in the software development lifecycle. By understanding Bitbucket's features and capabilities, teams can better leverage this powerful tool to enhance their collaborative efforts and streamline their development processes.

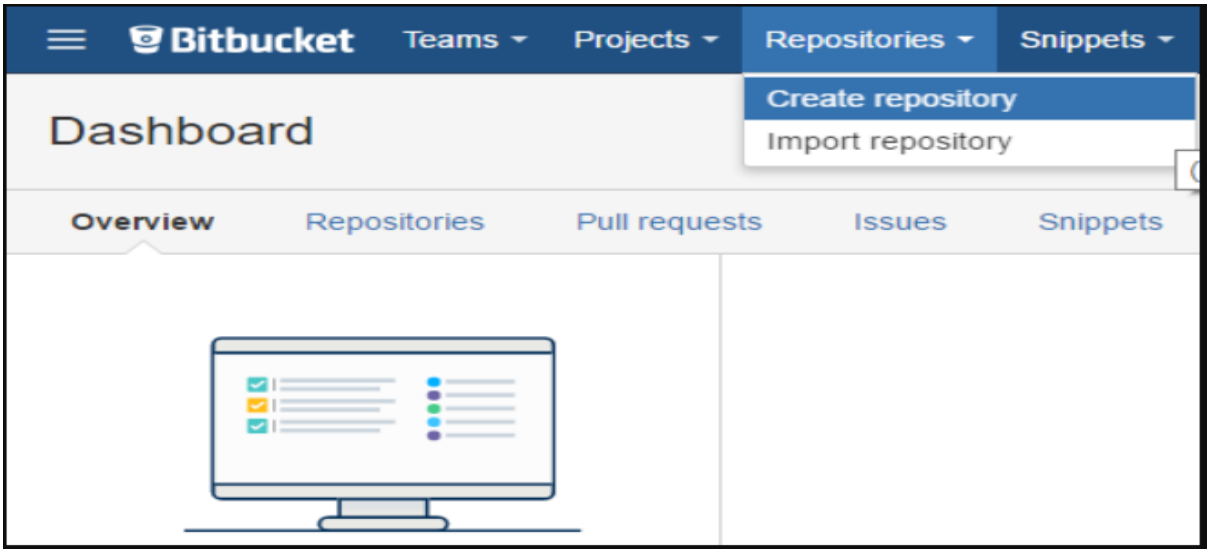
**ii) Implementation Detailss**

**Setting Up Bitbucket**

1. **Creating an Account**: Users need to sign up for a Bitbucket account, which can be free for small teams.




2. **Creating a Repository**: After logging in, users can create a new repository. This can be done via the Bitbucket interface by clicking on "Create Repository" and filling in the necessary details (repository name, access level, etc.).



3. **Cloning the Repository**: Users can clone the repository locally using the command:

git clone https://<username>@bitbucket.org/<team>/<repository>.git

**Basic Workflow**

1. **Making Changes**: Users can make changes to their local files.

2. **Staging Changes**: Staged changes are added using:

git add <file>

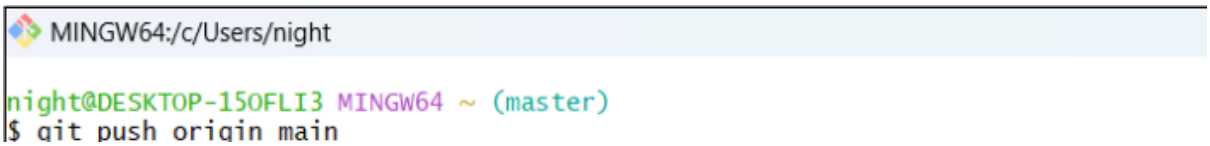

3. **Committing Changes**: Users commit their changes with a descriptive message:

git commit -m "Commit message"



4. **Pushing Changes**: Finally, users push their changes to Bitbucket:

git push origin <branch>



**Branching Strategy**

Bitbucket encourages a branching strategy to keep the codebase organized. Users can create branches for features, bug fixes, or experiments. This can be done with:

git checkout -b <branch-name>

**iii) Important Commands**

- **Cloning a Repository**:

  git clone <repository-url>

- **Checking Status**:

  git status

- **Adding Changes**:

  git add <file>

- **Committing Changes**:

  git commit -m "Your commit message"

- **Pushing Changes**:

  git push origin <branch-name>

- **Pulling Updates**:

  git pull origin <branch-name>

- **Creating a New Branch**:

  git checkout -b <branch-name>

- **Merging Branches**:

  git merge <branch-name>

**iv) Benefits**

1. **Collaboration**: Bitbucket allows multiple users to work on the same project simultaneously, facilitating collaboration through pull requests and code reviews.

2. **Version Control**: Users can track changes made to the code over time, enabling easy rollbacks and history tracking.

3. **Integration**: Bitbucket integrates seamlessly with other Atlassian products like Jira and Trello, enhancing project management and tracking.

4. **Branch Permissions**: Administrators can set branch permissions to control who can push changes to specific branches, ensuring code stability.

5. **Continuous Integration and Deployment**: Bitbucket provides built-in CI/CD capabilities, allowing teams to automate testing and deployment processes.

6. **Code Quality**: With features like pull requests and code reviews, teams can maintain high code quality and catch bugs early in the development process.

**v) Conclusion**

Bitbucket is a powerful version control system that enhances collaboration and productivity for software development teams. Its user-friendly interface, robust features, and integration capabilities make it a valuable tool for managing code repositories. By adopting Bitbucket, teams can effectively manage their projects, improve code quality, and streamline their development workflows. As software development continues to evolve, tools like Bitbucket play a crucial role in enabling teams to work efficiently and deliver high-quality software products.

**Name: Kamal Agrahari**
**ID: VU4F2223028**

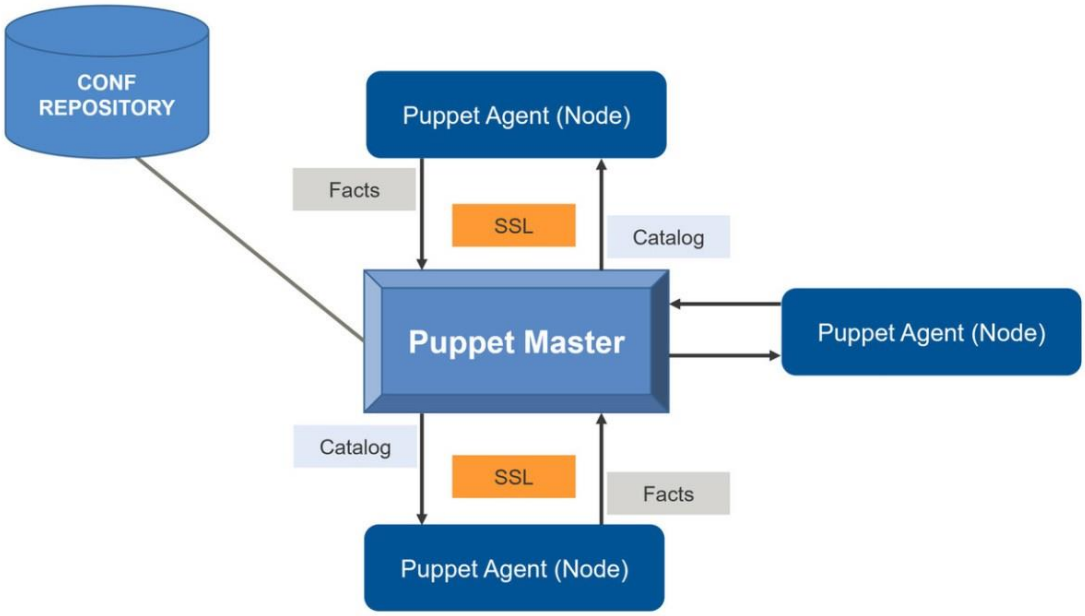
**Lab: DevOps**
**TE | IT |A**

# Case Study: Puppet Configuration Management

## Introduction

Puppet is an open-source configuration management tool used for automating the deployment, configuration, and management of infrastructure. It enables system administrators to automate repetitive tasks, ensuring that systems maintain a desired state. Puppet is based on a client-server architecture where the Puppet Master manages the configurations of nodes (clients or agents) through a defined language. This case study explores the architecture of Puppet, its master-slave communication, core blocks, installation process, and a concluding evaluation.

## i) Puppet Architecture




Puppet operates on a declarative, model-driven approach to automate infrastructure management. Its architecture consists of the following key components:

- Puppet Master: The Puppet Master is the central server responsible for compiling configurations into catalogs and sending them to the agents. It holds the manifest files and manages various nodes across the network.
- Puppet Agent: The Puppet Agent runs on client machines. It sends the Puppet Master facts about the system, which are pieces of information about the node's state. The agent pulls the catalog from the master and applies configurations accordingly.
- Puppet Manifests: These are files containing the desired system state written in Puppet's domain-specific language (DSL). These manifests define what configurations should be applied on the agents.
- Catalogs: Once the Puppet Master compiles the manifest for a specific agent, it creates a catalog. The catalog is a compiled version of the manifest with all variables resolved, and it is sent to the agent for execution.
- Facts and Facter: Facter is a tool that collects system information, like OS, IP address, and available memory. These 'facts' are then sent to the Puppet Master to help determine what configurations should be applied.
- PuppetDB: A database that stores configuration data, facts, reports, and catalogs, allowing Puppet to make more informed decisions during future runs.

## ii) Puppet Master-Slave Communication

Puppet follows a client-server (master-slave) architecture, and the communication between the Puppet Master and Puppet Agents follows a specific workflow:


1. Node Requests Facts: The Puppet Agent sends its facts (collected by Facter) to the Puppet Master when it initiates a connection.

2. Catalog Compilation: Based on the facts sent by the agent, the Puppet Master compiles a catalog for the node, referencing the manifests, modules, and hiera data to define the desired system state.

3. Catalog Transmission: The compiled catalog is then sent back to the Puppet Agent.

4. Application of Configuration: The Puppet Agent applies the configurations based on the catalog and ensures that the system conforms to the defined state.

5. Reporting: After applying the configurations, the agent sends a report back to the Puppet Master to indicate whether the application was successful or if any errors occurred.

Puppet uses HTTPS with SSL certificates for secure communication between the master and the agents.

## iii) Puppet Blocks

Puppet uses various fundamental building blocks to define configurations, ensuring infrastructure is managed effectively:

- Resource: The core unit in Puppet. Resources represent the system's state and could be packages, files, services, etc.

Example:
```
package { 'nginx':
  ensure => 'installed',
}
```

- Class: A collection of resources that define a particular system's behavior. It allows grouping of configurations and reusability.

Example:
```
class webserver {
  package { 'nginx': ensure => 'installed', }
  service { 'nginx': ensure => 'running', }
}
```

- Node Definition: Nodes allow you to specify different configurations for different hosts.

Example:
```
node 'webserver1.example.com' {
  include webserver
}
```

- Module: A module is a collection of manifests, templates, and files that are structured to support code reuse and organization.

- Templates: These are dynamic files used in configuration management, allowing variables to be inserted into configuration files.
- Hiera: A key/value lookup tool for configuration data, which allows separating code from data. This makes it easier to manage multiple configurations for different environments.

## iv) Installation and Configuring Puppet Master and Agent

### Installing Puppet Master on a Linux Machine (Ubuntu)

1. Install Puppet Master:

```
sudo apt update
sudo apt install puppetmaster
```

2. Configure Puppet Master:

```
sudo nano /etc/puppet/puppet.conf
```

Set up your environment and module path within the puppet.conf file:

```
[master]
environmentpath = $confdir/environments
basemodulepath = $confdir/modules
```

3. Start the Puppet Master service:

```
sudo systemctl start puppetmaster
sudo systemctl enable puppetmaster
```

### Installing Puppet Agent

1. Install Puppet Agent:

```
sudo apt install puppet-agent
```

2. Configure Puppet Agent:

Update the /etc/puppet/puppet.conf file to point to the Puppet Master:

```
[agent]
server = puppetmaster.example.com
```

3. Start the Puppet Agent:

```
sudo systemctl start puppet
sudo systemctl enable puppet
```

### Signing the SSL Certificates

1. On the Puppet Master: Check for incoming certificate requests:

sudo puppet cert list

2. Sign the certificate request:

sudo puppet cert sign <agent-node-name>

3. On the Puppet Agent: After signing, the agent can initiate communication:

sudo puppet agent --test

## v) Conclusion

Puppet is a powerful tool for automating and managing infrastructure at scale. Its architecture, built around the master-agent model, ensures that systems across the network are always configured according to the desired state. The block-based system, involving resources, classes, and modules, allows administrators to manage configurations in an efficient and modular way. By automating repetitive tasks and ensuring consistency across systems, Puppet helps improve system reliability and efficiency in DevOps practices.
This case study illustrates how Puppet simplifies system administration tasks, enabling organizations to adopt more scalable and efficient infrastructure management processses.