

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.

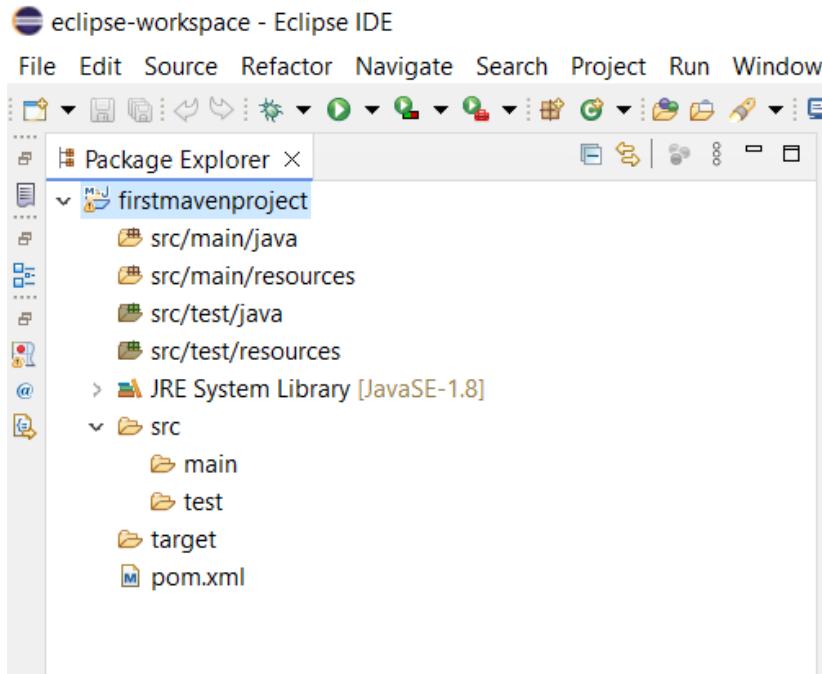
Tools and Technologies Used

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

Steps

1. Set Up Maven Project in Eclipse:

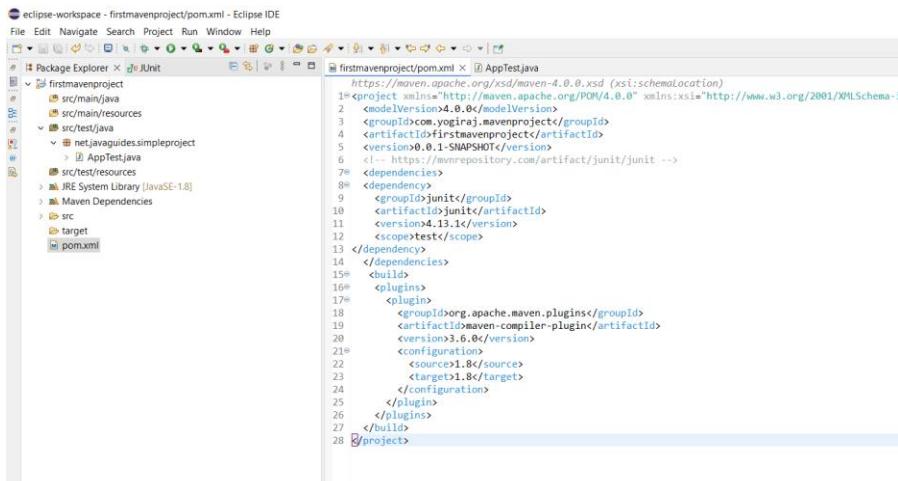
- Install Eclipse IDE from [Eclipse Downloads](#).
- Open Eclipse and create a new Maven project (File → New → Maven Project).
- Select options for a simple project and use the default workspace location.
- Enter GroupId and ArtifactId.



2. Configure pom.xml:

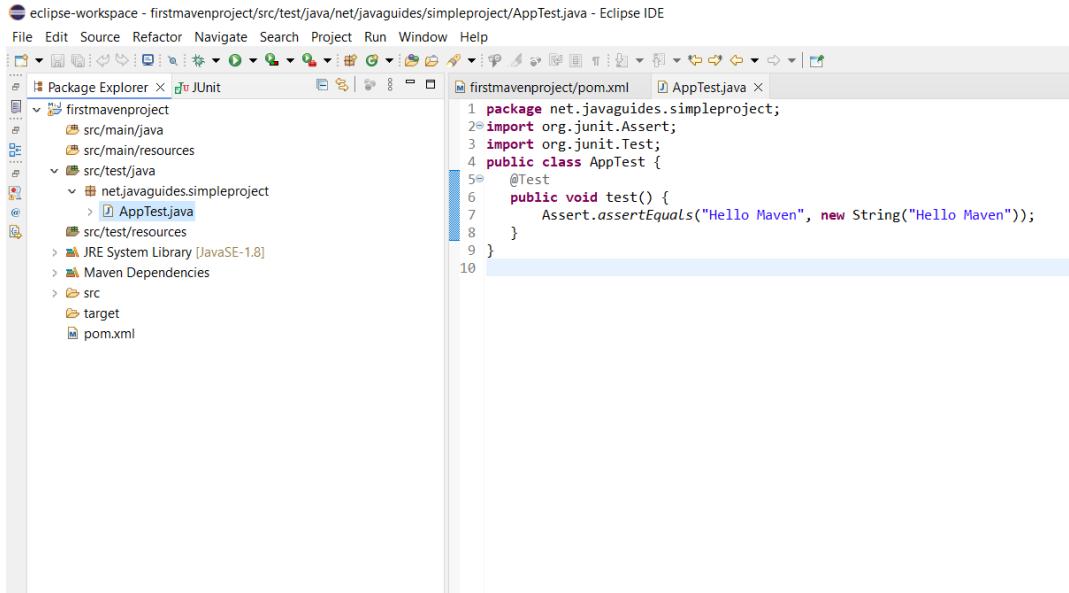
- Add the necessary dependencies (e.g., JUnit) and configure the Maven compiler plugin:

```
<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:



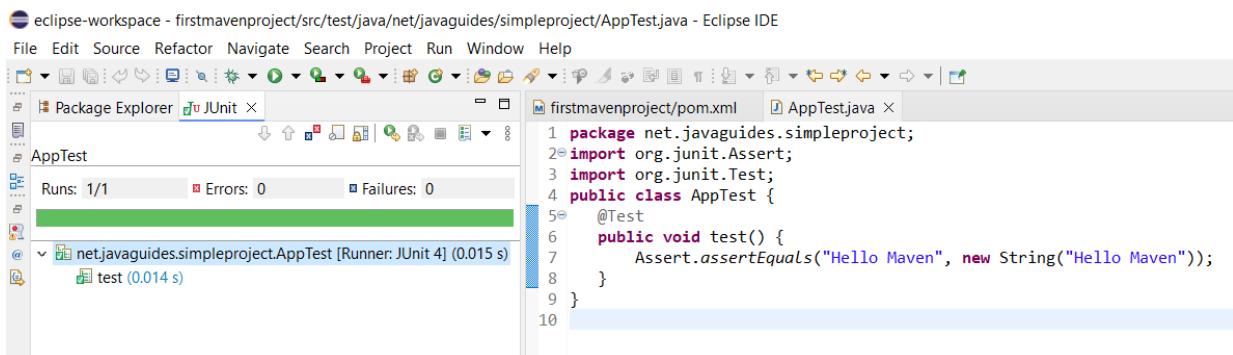
The screenshot shows the Eclipse IDE interface with the following details:

- File Menu:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard toolbar items like New, Open, Save, Cut, Copy, Paste, Find, etc.
- Package Explorer View:** Shows the project structure:
 - firstmavenproject
 - src/main/java
 - src/main/resources
 - src/test/java
 - net.javaguides.simpleproject
 - AppTest.java
 - src/test/resources
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - src
 - target
 - pom.xml
- Editor View:** Shows the file `AppTest.java` with the following code:

```
1 package net.javaguides.simpleproject;
2 import org.junit.Assert;
3 import org.junit.Test;
4 public class AppTest {
5     @Test
6     public void test() {
7         Assert.assertEquals("Hello Maven", new String("Hello Maven"));
8     }
9 }
```

4. Run the Maven Project:

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



The screenshot shows the Eclipse IDE interface after running the JUnit test, with the following details:

- File Menu:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard toolbar items like New, Open, Save, Cut, Copy, Paste, Find, etc.
- Package Explorer View:** Shows the project structure and the test results:
 - AppTest
 - Runs: 1/1 Errors: 0 Failures: 0
- Editor View:** Shows the file `AppTest.java` with the same code as before, and the test results are displayed in the status bar: "test (0.014 s)".

5. Push the Maven Project to GitHub:

- Commit and push the project to your GitHub repository.

The screenshot shows a GitHub repository page for 'yogirajbshinde21/first-maven-project'. The repository is public and has 0 stars, 0 forks, 1 watching, 1 branch, and 0 tags. The commit history shows an initial commit by 'yogirajmbshinde21' with a timestamp of 6:49d34 - 4 minutes ago. The terminal log on the right shows the command sequence for committing changes and pushing them to the GitHub origin master branch, including a failed attempt due to authentication issues.

```
YUMINODESSKTOP-RU88E96 MINGW64 ~/eclipse-workspace/Firstmavenproject (master)
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file: .gitignore
    new file: .gitignore.swp
    new file: pom.xml
    new file: src/test/java/net/javaguides/simpleproject/AppTest.java

YUMINODESSKTOP-RU88E96 MINGW64 ~/eclipse-workspace/Firstmavenproject (master)
$ git commit -m "Added initial commit"
[master (root-commit) e494d3d] Added initial commit
 4 files changed, 41 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .gitignore.swp
 create mode 100644 pom.xml
 create mode 100644 src/test/java/net/javaguides/simpleproject/AppTest.java

YUMINODESSKTOP-RU88E96 MINGW64 ~/eclipse-workspace/Firstmavenproject (master)
$ git status
On branch master
nothing to commit, working tree clean

YUMINODESSKTOP-RU88E96 MINGW64 ~/eclipse-workspace/Firstmavenproject (master)
$ git remote add origin https://github.com/yogirajbshinde21/first-maven-project.git
$ git branch -M master
$ git push -u origin master
remote: Invalid username or password.
fatal: Authentication failed for 'https://github.com/yogirajbshinde21/first-maven-project.git'
YUMINODESSKTOP-RU88E96 MINGW64 ~/eclipse-workspace/Firstmavenproject (master)
$ git push -u origin master
Enumerating objects: 12, done.
Counting objects: 100%, 12/12, done.
Delta compression using up to 4 threads
Compressing objects: 100% (12/12), 1.38 KiB / 174.00 KiB, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/yogirajbshinde21/first-maven-project.git
 * [new branch] master -> master
Branch 'master' set up to track 'origin/master'.

YUMINODESSKTOP-RU88E96 MINGW64 ~/eclipse-workspace/Firstmavenproject (master)
```

6. Integrate with Jenkins:

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

The screenshot shows the Jenkins 'Enter an item name' dialog with 'maven-github-project' entered. Below it, a list of project types is shown: 'Freestyle project', 'Maven project', 'Pipeline', and 'Multi-configuration project'. The 'Maven project' option is highlighted. At the bottom is an 'OK' button.

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

The screenshot shows the Jenkins configuration interface for a GitHub project. The left sidebar lists various configuration sections: General, Source Code Management (selected), Build Triggers, Build Environment, Pre Steps, Build, Post Steps, Build Settings, and Post-build Actions. The main panel is titled 'Git' under 'Source Code Management'. It contains fields for 'Repository URL' (set to <https://github.com/yogirajbshinde21/first-maven-project.git>) and 'Branches to build' (set to `*/master`). A note at the bottom of the panel says: 'Jenkins needs to know where your Maven is installed. Please do so from the tool configuration.' At the bottom right of the panel are 'Save' and 'Apply' buttons.

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

The screenshot shows the Jenkins configuration interface for the build section. The left sidebar lists: General, Source Code Management, Build Triggers (selected), Build Environment, Pre Steps (selected), Build, Post Steps, Build Settings, and Post-build Actions. The main panel is titled 'Build'. It includes fields for 'Maven Version' (with a note: 'Jenkins needs to know where your Maven is installed. Please do so from the tool configuration.'), 'Root POM' (set to `pom.xml`), and 'Goals and options' (set to `package`). An 'Advanced' dropdown is also present. At the bottom right of the panel are 'Save' and 'Apply' buttons.

- Build the project.

The screenshot shows the Jenkins build history for a project named 'maven-github-project'. The top bar indicates the build number is '#3 (28 Sept 2024, 19:25:24)'. The left sidebar provides links to Status, Changes, Console Output, Edit Build Information, Delete build '#3', Timings, Git Build Data, Test Result, Redeploy Artifacts, and See Fingerprints. The main content area displays the build status: 'No changes.' (indicated by a green checkmark icon). It shows the build was started by user 'Yogiraj Balkrishna Shinde'. A summary of the run time is provided: 'This run spent:' with items: '10 ms waiting', '30 sec build duration', and '30 sec total from scheduled to completion'. The 'git' section shows the revision and repository information: 'Revision: e494d344a19c2b6be38c42f07b9b9d7edc3b665' and 'Repository: <https://github.com/yogirajbshinde21/first-maven-project.git>'. Below this, a 'Test Result' section states '(no failures)' and a 'Module Builds' section shows a single build for 'firstmavenproject' which took 17 seconds. On the right side, there are buttons for 'Keep this build forever', 'Add description', and build statistics: 'Started 1 min 0 sec ago' and 'Took 30 sec'.

> firstmavenproject > #3 > Test Results > net.javaguides.simpleproject > AppTest

Test Result : AppTest

0 failures

1 tests

Took 10 ms.

 Add description

All Tests

Test name	Duration	Status
test	10 ms	Passed

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:

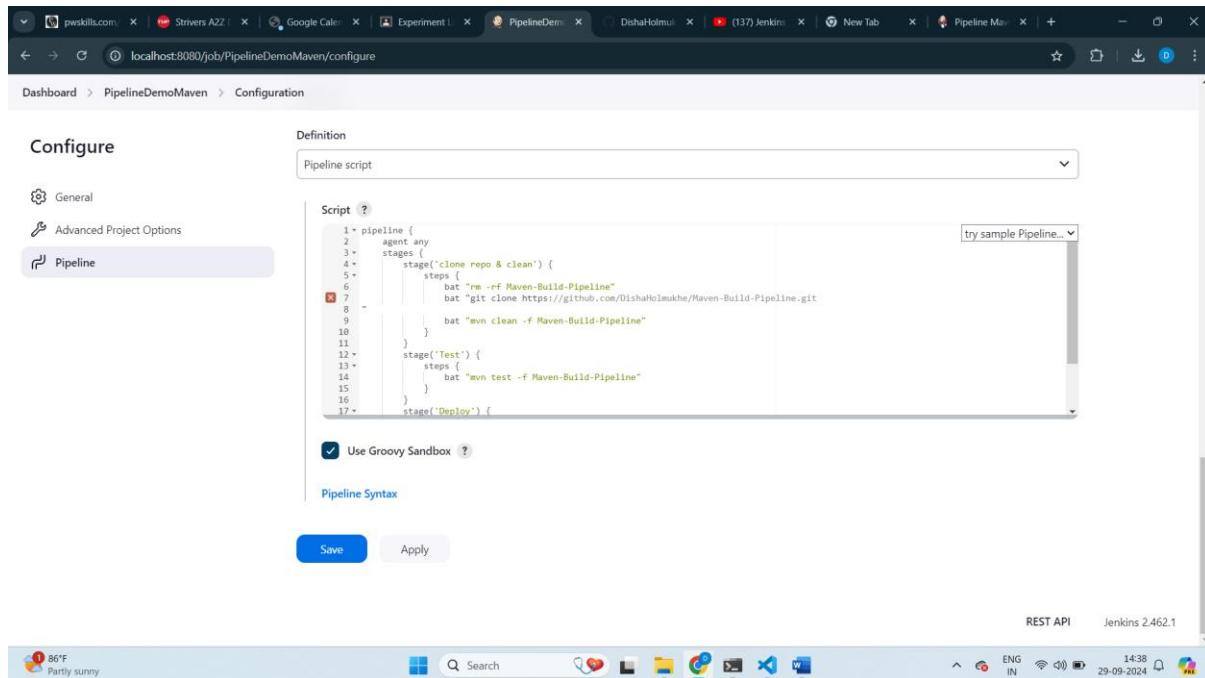
Step1: Create a new item/job with item type as pipeline.

The screenshot shows the Jenkins interface for creating a new item. The title bar says 'localhost:8080/view/all/newJob'. The main area is titled 'New Item' with a sub-section 'Enter an item name' containing 'PipelineDemoMaven'. Below it, 'Select an item type' is set to 'Pipeline'. Other options shown are 'Freestyle project', 'Maven project', and 'Multi-configuration project'. At the bottom is an 'OK' button.

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

The screenshot shows the Jenkins configuration page for 'PipelineDemoMaven'. The 'General' tab is selected. Under 'GitHub project', the 'Project url' field contains 'https://github.com/DishaHolmukhe/Maven-Build-Pipeline'. Other tabs like 'Advanced' and 'Pipeline' are also visible. At the bottom are 'Save' and 'Apply' buttons.

Step3: In Pipeline choose Pipeline Script

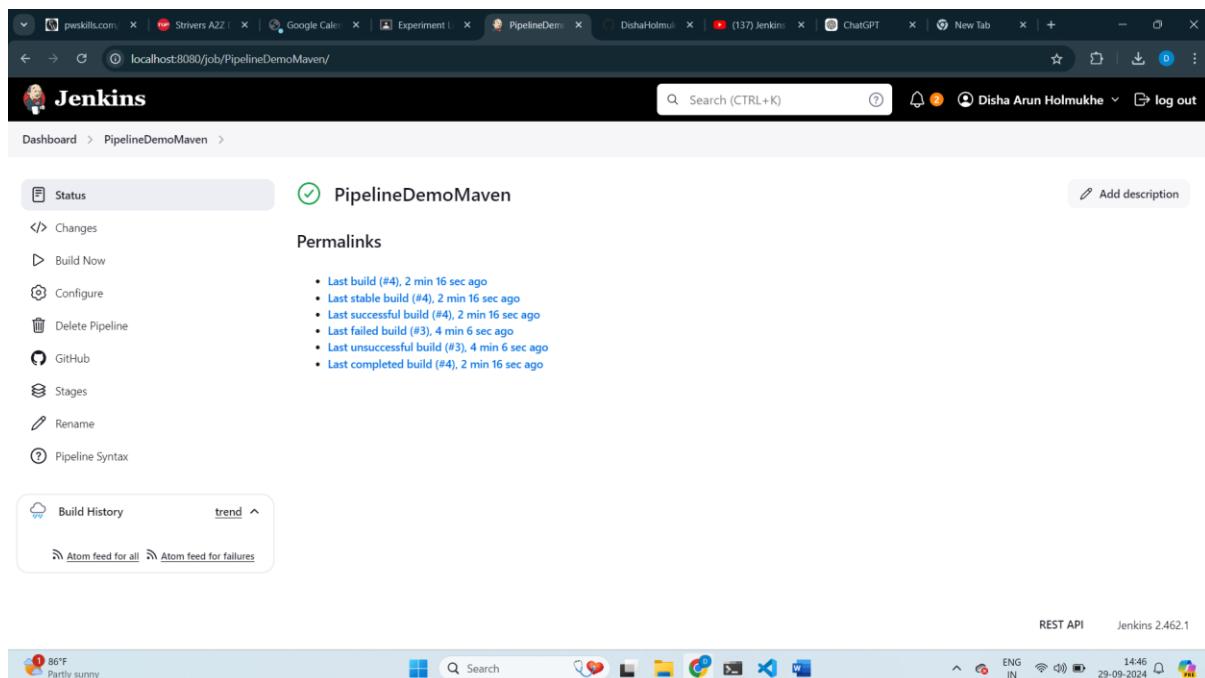


The screenshot shows the Jenkins Pipeline configuration page for a job named "PipelineDemoMaven". The "Pipeline" tab is selected. The "Definition" dropdown is set to "Pipeline script". The main area contains the following Groovy pipeline script:

```
1 > pipeline {  
2 >     agent any  
3 >     stages {  
4 >         stage('clone repo & clean') {  
5 >             steps {  
6 >                 bat "rm -rf Maven-Build-Pipeline"  
7 >                 bat "git clone https://github.com/DishaHolmukhe/Maven-Build-Pipeline.git"  
8 >                 bat "mvn clean -f Maven-Build-Pipeline"  
9 >             }  
10 >         }  
11 >         stage('Test') {  
12 >             steps {  
13 >                 bat "mvn test -f Maven-Build-Pipeline"  
14 >             }  
15 >         }  
16 >         stage('Deploy') {  
17 >     }
```

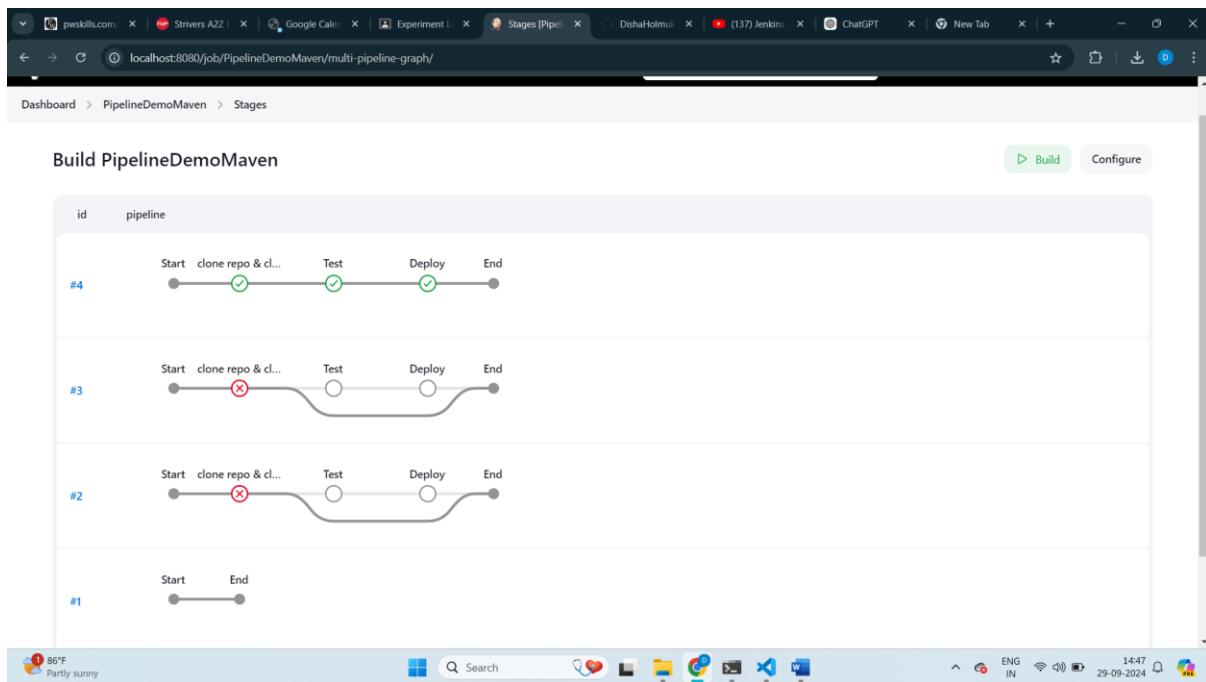
Below the script, there is a checkbox labeled "Use Groovy Sandbox". At the bottom, there are "Save" and "Apply" buttons.

Step4: Click save and Build now.



The screenshot shows the Jenkins Pipeline status page for the "PipelineDemoMaven" job. The status is green with a checkmark icon. The pipeline name "PipelineDemoMaven" is displayed. On the left, a sidebar lists options: Status, Changes, Build Now, Configure, Delete Pipeline, GitHub, Stages, Rename, and Pipeline Syntax. Below the sidebar, a "Build History" section shows the last seven builds, all of which are successful. The top navigation bar includes the Jenkins logo, user information (Disha Arun Holmukhe), and a log out link. The bottom status bar shows weather information (86°F, Partly sunny) and system details (14:38, 29-09-2024).

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.

The screenshot shows the Jenkins Pipeline Demo Maven build #4 console output. The log starts with the build being started by user Disha Arun Holmukhe. It then shows the pipeline starting, running on Jenkins, and cloning the repository. The Maven build command is run, followed by the clean command. The log concludes with the message "Finished: SUCCESS".

```

Started by user Disha Arun Holmukhe
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\PipelineDemoMaven
[Pipeline] {
[Pipeline] stage
[Pipeline] {
  [Pipeline] {
    [Pipeline] bat
C:\ProgramData\Jenkins\.jenkins\workspace\PipelineDemoMaven>if exist "Maven-Build-Pipeline" rmdir /S /Q "Maven-Build-Pipeline"
[Pipeline] bat
C:\ProgramData\Jenkins\.jenkins\workspace\PipelineDemoMaven>git clone https://github.com/DishaHolmukhe/Maven-Build-Pipeline.git
Cloning into 'Maven-Build-Pipeline'...
[Pipeline] bat
C:\ProgramData\Jenkins\.jenkins\workspace\PipelineDemoMaven>mvn clean -f Maven-Build-Pipeline
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] < com.mycompany.app:my-app >-----
[INFO] Building my-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----
[INFO] -----
[INFO] Finished: SUCCESS

```

```

Progress (1): 6.6/6.8 MB
Progress (1): 6.7/6.8 MB
Progress (1): 6.8/6.8 MB
Progress (1): 6.8/6.8 MB
Progress (1): 6.8 MB

Downloaded from central: https://repo.maven.apache.org/maven2/com/github/luben/zstd-jni/1.5.5-11/zstd-jni-1.5.5-11.jar (6.8 MB at 1.5 MB/s)
[INFO] Building jar: C:\ProgramData\Jenkins\jenkins\workspace\PipelineDemoMaven\Haven-Build-Pipeline\target\my-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 16.683 s
[INFO] Finished at: 2024-09-29T14:44:53+05:30
[INFO] -----
[Pipeline] )
[Pipeline] // stage
[Pipeline] )
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

REST API Jenkins 2.462.1

86°F Partly sunny 14:47 ENG IN 29-09-2024

Building Maven pipeline using Pipeline script SCM

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

New Item

Enter an item name
PipelineDemoMavenSCM

Select an item type

- Freestyle project**
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
- Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

OK

27 min delay NES / Eastern Ex... 14:52 29-09-2024

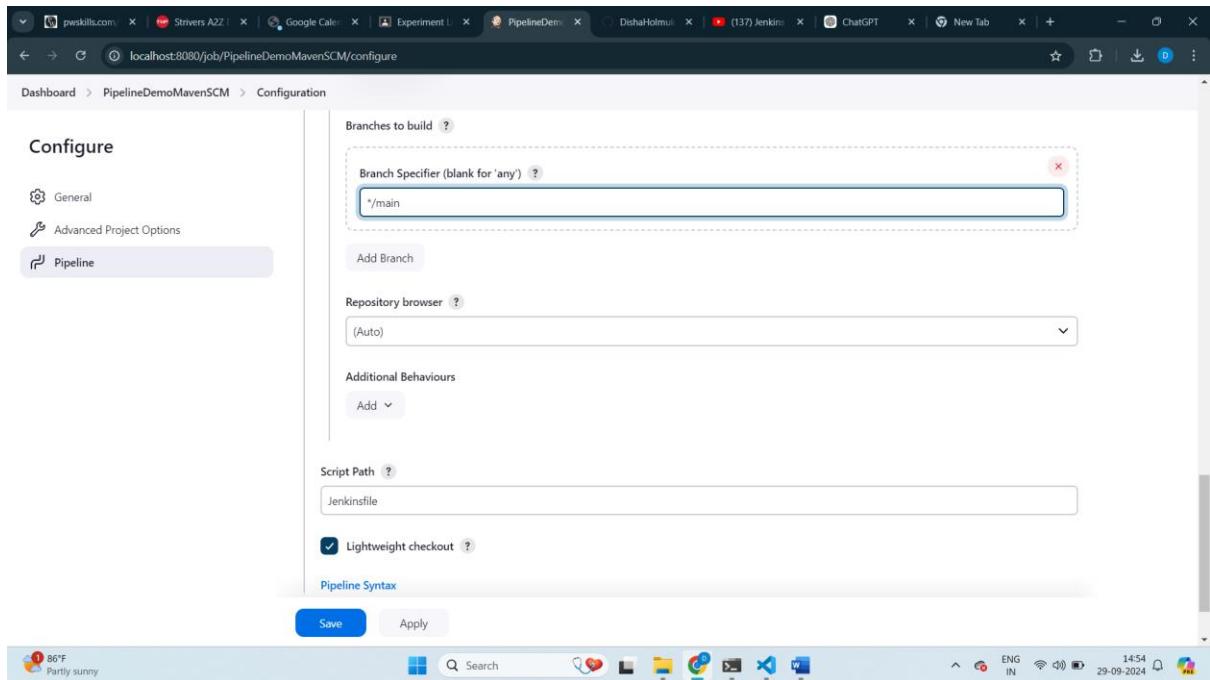
Step2: In General set github project with the github url to the maven project.

The screenshot shows the Jenkins Pipeline configuration page for a project named 'PipelineDemoMaven'. The 'General' section is selected. Under 'GitHub project', the 'Project url' field contains the URL 'https://github.com/DishaHolmukhe/Maven-Build-Pipeline'. Other options like 'Discard old builds', 'Do not allow concurrent builds', and 'Do not allow the pipeline to resume if the controller restarts' are available but not selected. The 'Pipeline' section is also visible. At the bottom are 'Save' and 'Apply' buttons.

Step3: In pipeline select Pipeline Script from SCM.

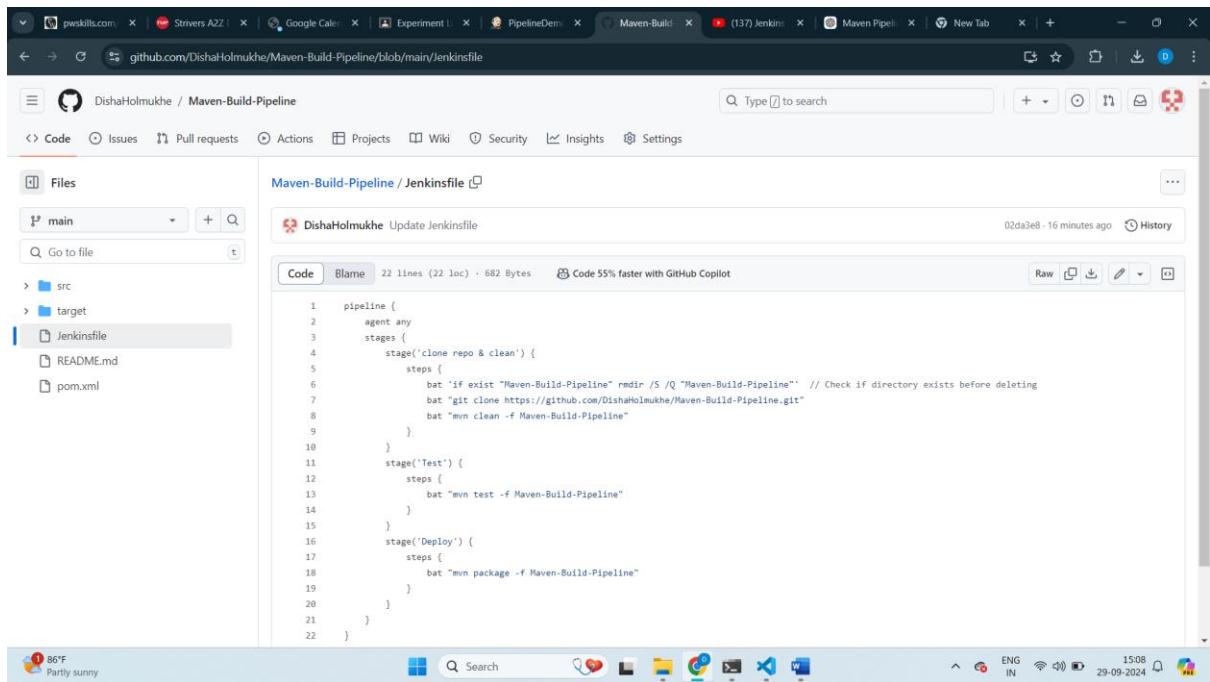
The screenshot shows the Jenkins Pipeline configuration page for a project named 'PipelineDemoMavenSCM'. The 'Pipeline' section is selected. Under 'Definition', 'Pipeline script from SCM' is chosen. The 'SCM' dropdown is set to 'Git'. The 'Repositories' section shows a single repository with the 'Repository URL' set to 'https://github.com/DishaHolmukhe/Maven-Build-Pipeline.git'. The 'Credentials' dropdown is currently empty. At the bottom are 'Save' and 'Apply' buttons.

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.

The screenshot shows the Jenkins interface for a pipeline named 'PipelineDemoMavenSCM'. At the top, there's a navigation bar with tabs like 'Dashboard', 'PipelineDemoMavenSCM', and 'Stages'. Below the navigation is a search bar and a user profile. The main content area displays a 'Build PipelineDemoMavenSCM' section. It includes a table with columns 'id' and 'pipeline', showing a single row '#1' with a pipeline diagram. The diagram consists of six nodes: 'Start', 'Checkout SCM', 'clone repo & cl...', 'Test', 'Deploy', and 'End', connected by arrows. All nodes are marked with green checkmarks, indicating successful execution. There are 'Build' and 'Configure' buttons at the top right of this section.



This screenshot shows the Jenkins pipeline details page for 'PipelineDemoMavenSCM'. The top navigation bar is identical to the previous one. The main content area has a 'Status' tab selected, which is highlighted in grey. To its right is a green checkmark icon and the pipeline name 'PipelineDemoMavenSCM'. Below the status are several actions: 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'GitHub', 'Stages', 'Rename', and 'Pipeline Syntax'. A 'Permalinks' section lists recent builds. At the bottom, there's a 'Build History' section with a 'trend' dropdown, and links for 'Atom feed for all' and 'Atom feed for failures'. The bottom of the screen shows the same Windows taskbar as the previous screenshot.

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

```
Started by user Disha Arun Holmukhe
Obtained Jenkinsfile from git https://github.com/DishaHolmukhe/Maven-Build-Pipeline.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\jenkins\workspace\PipelineDemoMavenSCM
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/DishaHolmukhe/Maven-Build-Pipeline.git
> git.exe init C:\ProgramData\Jenkins\jenkins\workspace\PipelineDemoMavenSCM # timeout=10
Fetching upstream changes from https://github.com/DishaHolmukhe/Maven-Build-Pipeline.git
> git.exe --version # timeout=10
> git --version # 'git' version 2.42.0.windows.2'
> git.exe fetch --tags --force --progress -- https://github.com/DishaHolmukhe/Maven-Build-Pipeline.git +refs/heads/*:refs/remotes/origin/*
timeout=10
> git.exe config remote.origin.url https://github.com/DishaHolmukhe/Maven-Build-Pipeline.git # timeout=10
> git.exe config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git.exe rev-parse "refs/remotes/origin/main"(commit) # timeout=10
```

```
[INFO] -----
[INFO] Running com.mycompany.app.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.092 s -- in com.mycompany.app.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.4.2:jar (default-jar) @ my-app ---
[INFO] Building jar: C:\ProgramData\Jenkins\jenkins\workspace\PipelineDemoMavenSCM\Maven-Build-Pipeline\target\my-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.459 s
[INFO] Finished at: 2024-09-29T14:56:03+05:30
[INFO] -----
[Pipeline] )
[Pipeline] // stage
[Pipeline] )
[Pipeline] // withEnv
[Pipeline] )
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Conclusion: Successfully completed the demonstration to Create Maven Pipeline.

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

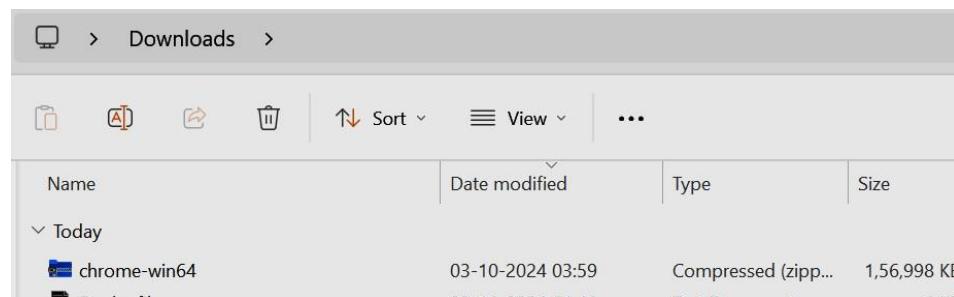
First, ensure you have Python installed on your machine. Then, install Selenium using pip:

`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 selenium) (2.2.8)
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) (4.18.0)
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>=0.19 (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>=0.19->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
```

The screenshot shows a Google search results page. The search bar at the top contains the query "Selenium WebDriver". Below the search bar, there is a "Sign in to Google" dialog box with options for "Stay signed out" and "Sign in". The main search results area displays several links related to Selenium WebDriver, including one from the official Selenium documentation website.

Below the search results, there is a "People also ask" section with two questions: "What is a Selenium WebDriver?" and "How many types of webdrivers are in Selenium?".

```
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.

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.

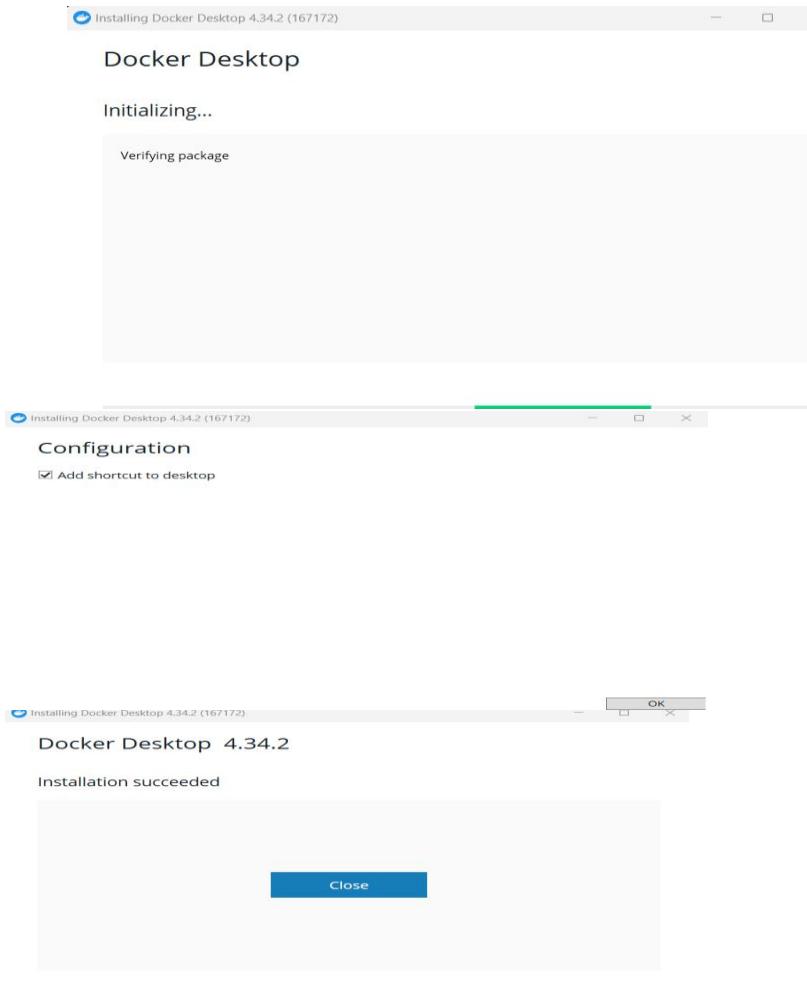
► Installation of Docker

- 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.

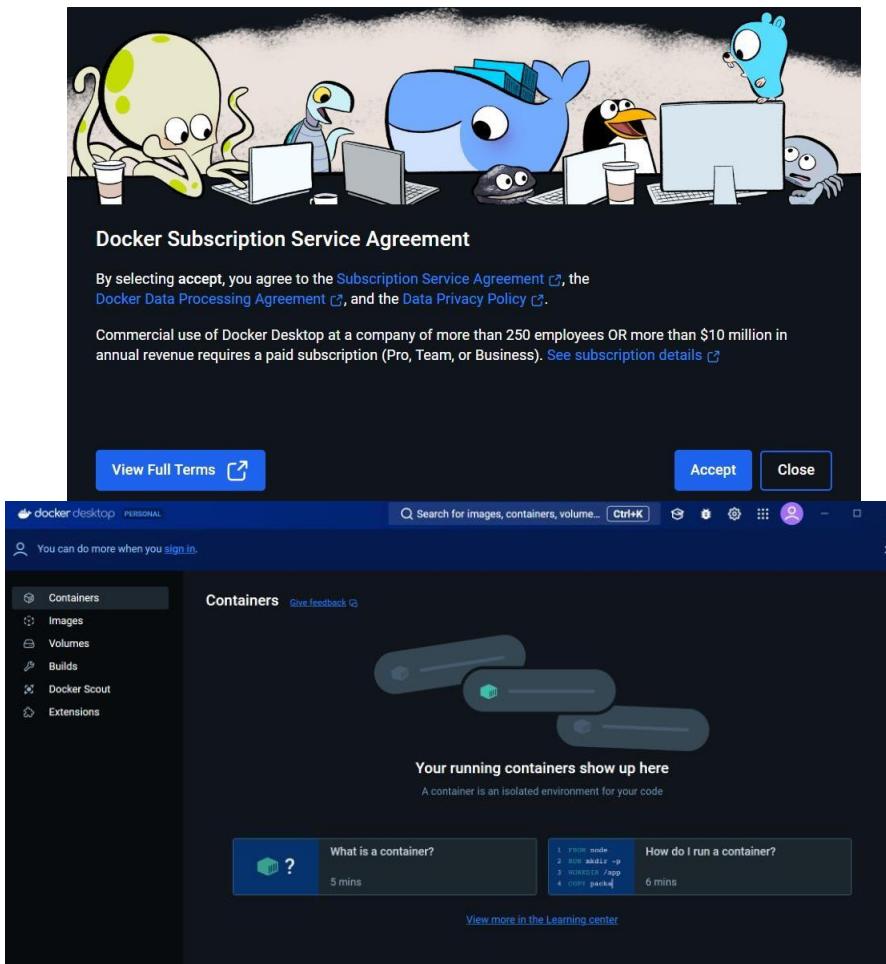


The screenshot shows the Docker Docs website with the URL [https://docs.docker.com/manuals/install/windows/](#). The page title is "Install Docker Desktop on Windows". The left sidebar has a "Manuals" section with "Docker Build Cloud", "Docker Compose", "Docker Desktop", "Install" (which is expanded), "Mac", "Understand permission requirements...", and "Windows" (which is selected). The main content area includes "Docker Desktop terms" and a note about commercial use requiring a paid subscription. It also contains links for "Docker Desktop for Windows - x86_64" and "Docker Desktop for Windows - Arm (Beta)". The right sidebar has links for "Edit this page", "Request changes", "Table of contents", "System requirements", and "Where to go next".

- 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.

The top screenshot shows the Docker sign-in page with fields for 'Username or email address*' and 'Continue' button, and options to 'Continue with Google' or 'Continue with GitHub'. The bottom screenshot shows the Docker Home page with a 'Welcome to Docker Home, Kamal' message, 'Get started' and 'Docker concepts' links, and sections for 'Explore areas of Docker' (Docker Desktop, Build Cloud, Scout, Hub), 'Give feedback', and a 'Build Cloud' section.

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

```
c:\ Select Command Prompt
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Users\gyand>docker --version
Docker version 27.2.0, build 3ab4256

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

Conclusion:

Hence, we successfully installed the Docker.

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



```
1 #STEP1 sepcify the base image
2 FROM alpine
3
4 #STEP2 download and install dependencies
5 RUN apk add --update redis
6
7 #STEP3 setup the strtup command
8 CMD ["redis-server"]
```

2. Build the Docker Image

```
PS C:\Users\Sayali Nimbalkar\Desktop\Docke 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 Dockerfile
=> => transferring dockerfile: 211B 0.0s
=> [internal] load metadata for docker.io/1 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\Docke File> docker run a4e47e3c2b51
1:C 02 Oct 2024 12:00:09.836 * 000000000000 Redis is starting 000000000000
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\Docke 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:beefbdb8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d
=> => resolve docker.io/library/alpine:latest@sha256:beefbdb8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d
=> CACHED [2/2] RUN apk add --update redis
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:111f83d7a91f0b1bf7b7063368e9db52a9ae8e65b1ae22326c2ca71a9af8b7e9
=> => exporting config sha256:68f0da0c4ea6b47fc9c4d0a87c9924021dde87d654438d1608a67b539466420
=> => 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.

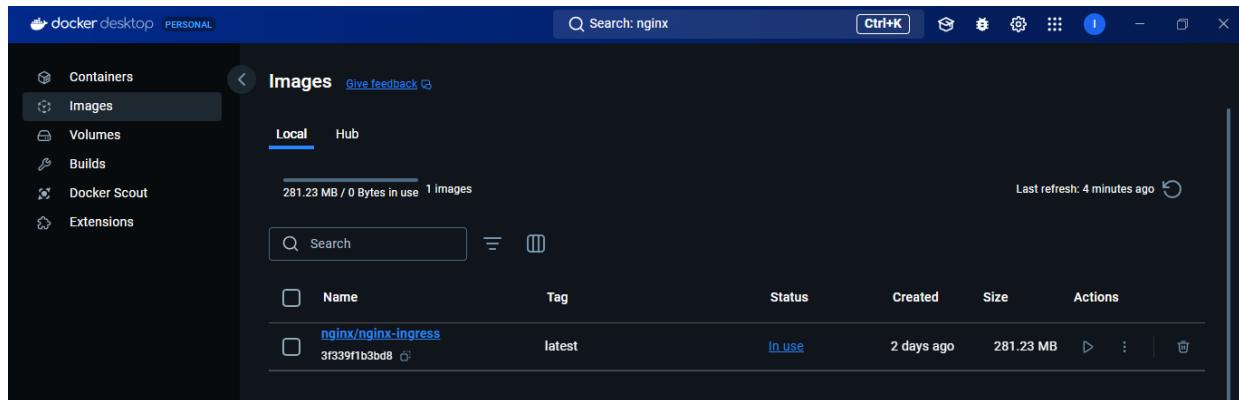
```
PS C:\Users\Sayali Nimbalkar\Desktop\Docke File> docker run sayali/redis
1:C 02 Oct 2024 12:10:20.241 * 000000000000 Redis is starting 000000000000
1:C 02 Oct 2024 12:10:20.241 * Redis version=7.2.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 02 Oct 2024 12:10:20.241 # Warning: 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 12:10:20.244 * monotonic clock: POSIX clock_gettime
1:M 02 Oct 2024 12:10:20.247 * Running mode=standalone, port=6379.
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\Docke 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

- Ansible Installed:** Ensure Ansible is installed on your control machine (the machine from which you will run Ansible).
- SSH Access:** Ensure that you have SSH access to the target machines and that they are listed in the Ansible inventory file.
- 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 ansible_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

Assignment – 1

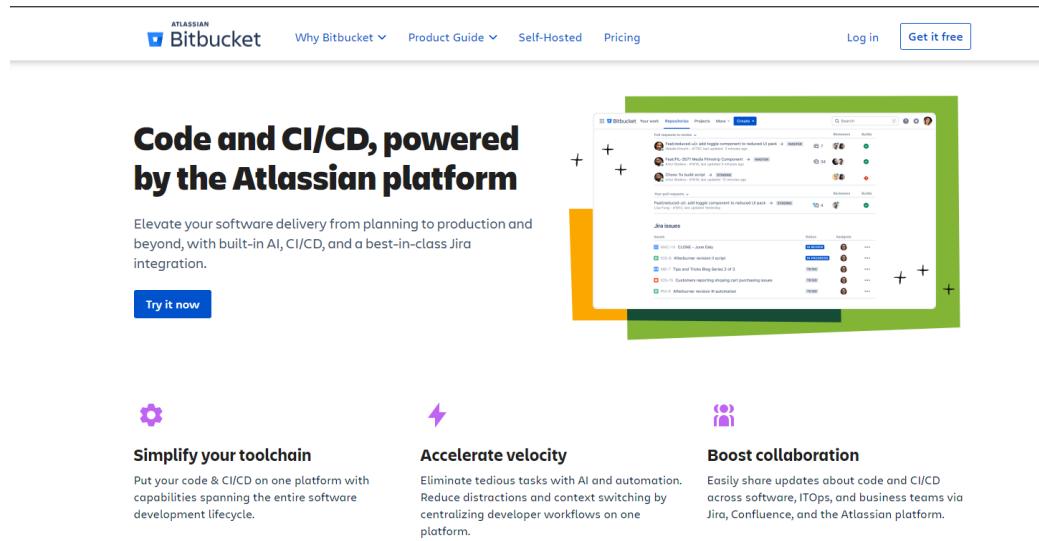
Aim: Write Case Study on Bitbucket version control System

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.



The screenshot shows the Bitbucket homepage. At the top, there's a navigation bar with links for 'Why Bitbucket', 'Product Guide', 'Self-Hosted', 'Pricing', 'Log in', and a 'Get it free' button. Below the navigation, a large banner highlights 'Code and CI/CD, powered by the Atlassian platform'. It includes a callout for 'Elevate your software delivery from planning to production and beyond, with built-in AI, CI/CD, and a best-in-class Jira integration.' A 'Try it now' button is visible. To the right of the banner is a composite image showing a Bitbucket dashboard with multiple tabs (Issues, Pull Requests, Pipelines) and a Jira interface below it, all connected by a network of plus signs. Below this composite image are three sections: 'Simplify your toolchain', 'Accelerate velocity', and 'Boost collaboration', each with a brief description and a small icon.

- Simplify your toolchain**: Put your code & CI/CD on one platform with capabilities spanning the entire software development lifecycle.
- Accelerate velocity**: Eliminate tedious tasks with AI and automation. Reduce distractions and context switching by centralizing developer workflows on one platform.
- Boost collaboration**: Easily share updates about code and CI/CD across software, ITOps, and business teams via Jira, Confluence, and the Atlassian platform.

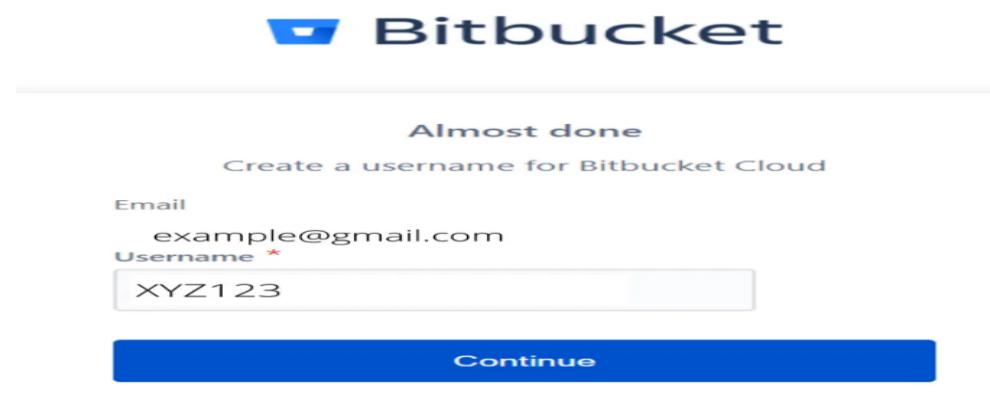
Moreover, Bitbucket seamlessly integrates with other Atlassian products such as Jira and Confluence, enabling a unified workflow for project management and documentation. This integration helps teams align their development efforts with project goals and user feedback, ensuring that the software meets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

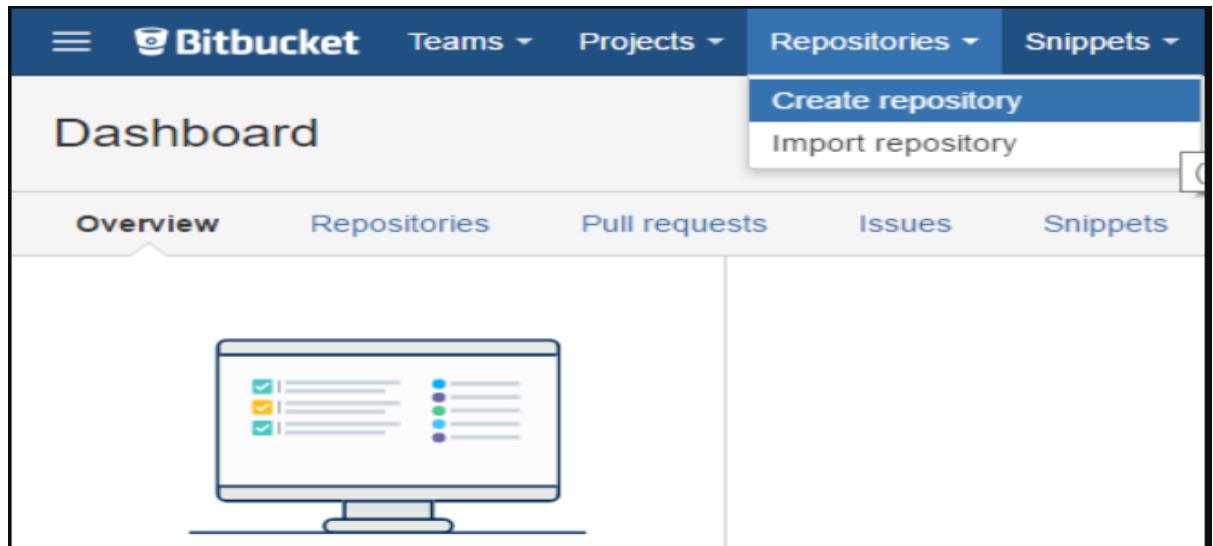
ii) Implementation Details

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>
```

```
Dell@Projectlab-PC-5 MINGW64 ~/mywebpage (master)
$ git add pqr.txt
```

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

```
git commit -m "Commit message"
```

```
Dell@Projectlab-PC-5 MINGW64 ~/mywebpage (master)
$ git commit
hint: Waiting for your editor to close the file...
```

```
Open ▾  *COMMIT_EDITMSG
Local Disk (C:) \Users\Del\mywebpage\.git

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       new file:    trex.txt
#
# Untracked files:
#       python/
This is my commit |
```

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

```
git push origin <branch>
```

```
Dell@Projectlab-PC-5 MINGW64 ~/mywebpage (master)
$ git push origin main
```

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

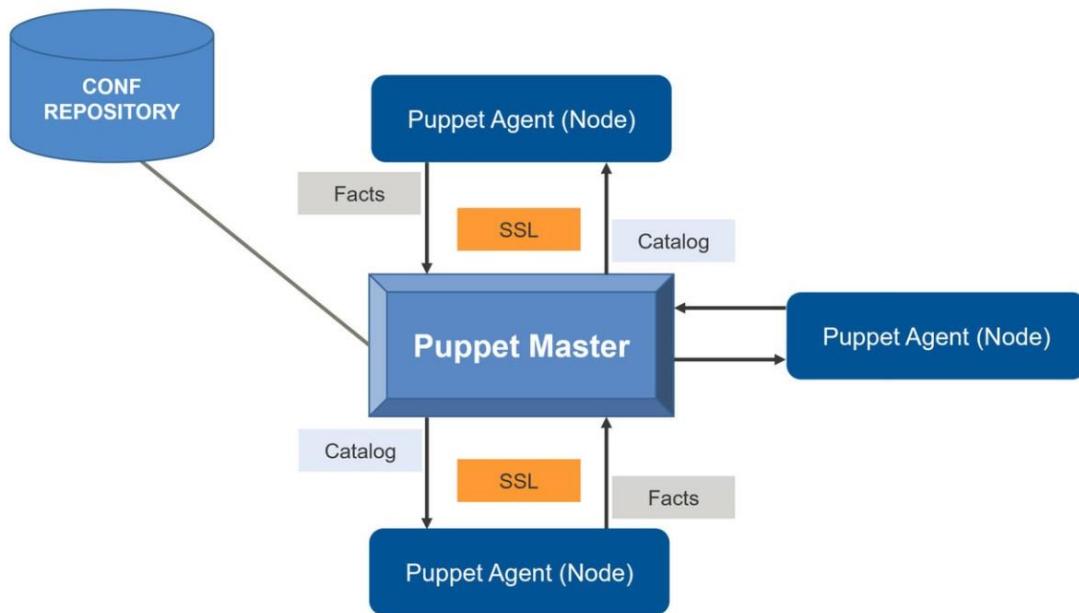
Bitbucket is a powerful version control system that enhances collaboration and productivity for software development teams. Its user-friendly interface, robust features, and integration capabilities make it a valuable tool for managing code repositories. By adopting Bitbucket, teams can effectively manage 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.

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 processes.