Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./~~Miss~~ PRATHAM ASNANI of IT Department, Semester V with Roll No. 63 has completed a course of the necessary experiments in the subject DEVOPS LAB under my supervision in the **Thadomal Shahani Engineering College** Laboratory in the year 2025 - 2026

Teacher In- Charge

Head of the Department

Date 10/10/2025

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1	To understand Devops principle practices and Devops Engineer Role	1-4	18/7/25	
2	To understand version control system GIT Installation and GITHUB acc	5-8	12/8/25	
3	To perform various GIT operations	9-15	1/9/25	
4	To understand continous using Jenkin	16-22	9/9/25	
5	To build Java program using Jenkin	23-29	10/9/25	
6	To build pipeline using Jenkin	30-35	16/9/25	
7	To understand Docker architecture and container, life cycle, install docker deploy container in Docker	36-41	19/9/25	
8	To build an image for sample web applications using Docker	42-46	23/9/25	
9	Installation of Nagios on Ubuntu	47-55	9/10/25	
10	To study puppet tools	56-62	9/10/25	
11	Written Assignment I	63-67		
12	Written Assignment II	68-72	11/9/25	

Name-Pratham Asnani

Rollno-63

Batch-T13

Assignment 1

<u>Aim:</u> To understand principles of DevOps

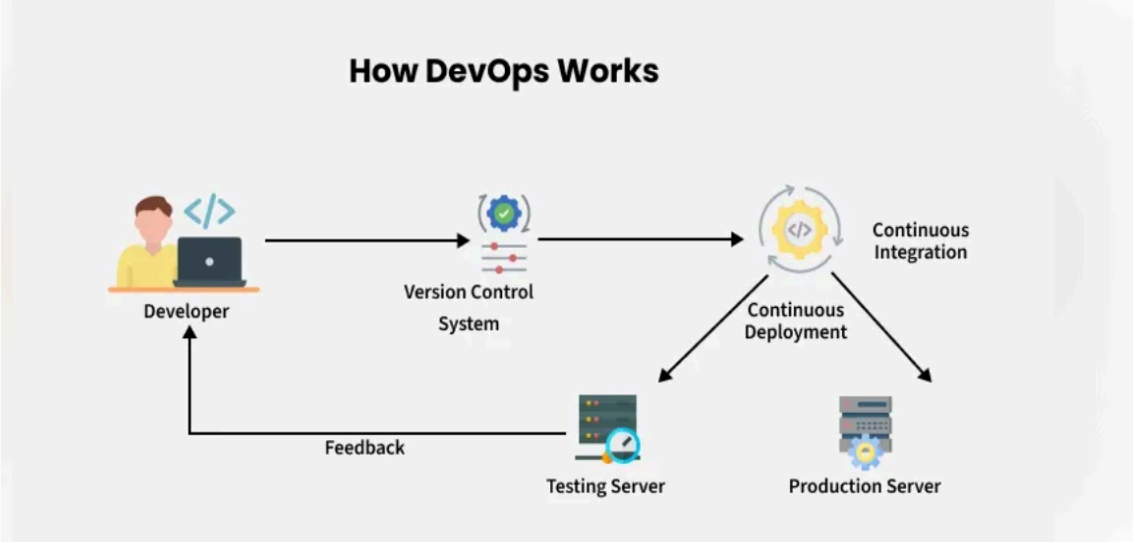
<u>Theory:</u>

Introduction:

DevOps is a combination of two words: **"Development"** and **"Operations."** It's a modern approach where software developers and software operations teams work together throughout the entire software life cycle, from planning and coding to testing, deploying, and monitoring.

The main idea of DevOps is to improve the software delivery process. We achieve this by software release automation, continuous monitoring/logging and testing. The goals of DevOps are:

- Faster and continuous software releases.
- Reduces manual errors through automation.
- Built-in Monitoring**,** detect failures or issues during delivery or in production.
- Automate testing throughout the software delivery process.



<u>Phases of DevOps Lifecycle</u>

The following are the different phases of the DevOps lifecycle.

1. Plan: This phase focuses on understanding the business needs and gathering feedback from end-users. Teams create a plan that aligns the project with business goals and ensures the right results are delivered.

2. Code: In this phase, developers write the actual code for the software. Tools like Git help manage the code, making sure that the code is well-organized and free from security issues or bad coding practices.
3. Build: Once the code is written, it is submitted to a central system using tools like Jenkins. This step ensures the code is compiled, and all components are integrated together smoothly.
4. Test: The software is then tested to ensure it works properly. This includes different types of tests like security, performance, and user acceptance. Tools like JUnit and Selenium are used to automate these tests and verify the software's integrity.
5. Release: After testing, the software is ready to be released to production. The DevOps team ensures that all checks are passed and then sends the latest version to the production environment.
6. Deploy: Using Infrastructure-as-Code (IaC) tools like Terraform, the necessary infrastructure (servers, networks, etc.) is automatically created. Once the infrastructure is set up, the code is deployed to various environments in an automated and repeatable way.
7. Operate: Once deployed, the software is available for users. Tools like Chef help manage the configuration and ongoing deployment of the system to ensure it operates smoothly.
8. Monitor: This phase involves observing how the software is performing in the real world. Data about user behaviour and application performance is collected to identify any issues or bottlenecks. By monitoring the system, the team can quickly spot and fix problems that may affect performance.

Features:

Features of DevOps

1. **Collaboration and Communication**: Breaks silos between development and operations teams for better coordination.
2. **Automation**: Automates tasks like testing, building, deployment, and monitoring to save time and reduce errors.
3. **Continuous Integration & Continuous Delivery (CI/CD)**: Enables frequent, reliable code updates through automated pipelines.
4. **Infrastructure as Code (IaC)**: Manages and provisions infrastructure using code (e.g., with Terraform or Ansible).
5. **Monitoring and Logging**: Continuously tracks application performance and system health using tools like Prometheus and Grafana.
6. **Rapid Feedback Loops**: Gathers and acts on user and system feedback to quickly improve software.
7. **Scalability and Reliability**: Supports scalable deployments and ensures system stability even during frequent changes.

Goals of DevOps

1. **Faster Software Delivery**: Reduce time from development to production.
2. **High-Quality Releases**: Detect and fix bugs early through automated testing.
3. **Improved Collaboration**: Foster a culture of shared responsibility among teams.
4. **Increased Deployment Frequency**: Enable frequent and smooth software updates.
5. **Reduced Failure Rates**: Minimize production issues with continuous monitoring and testing.
6. **Quicker Recovery Times**: Recover rapidly from failures due to automated rollback and incident response.
7. **Customer-Centric Focus**: Continuously deliver features that improve the user experience.

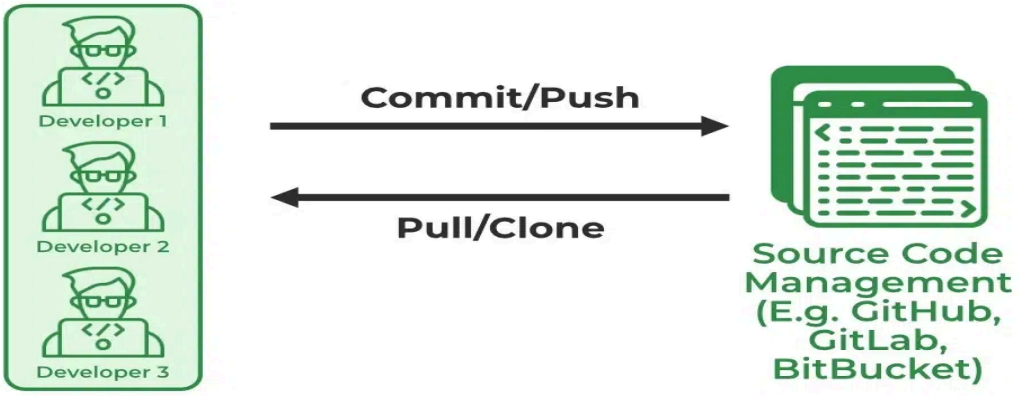
---

7 Cs of DevOps

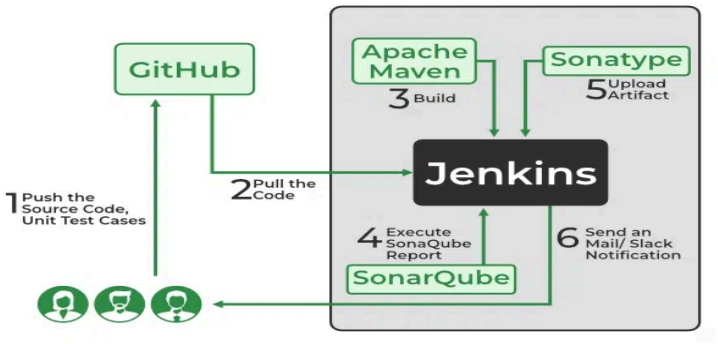
The 7 Cs of DevOps are core principles that help make DevOps successful. They guide how teams work together, build, test, and deliver software faster and more reliably. Each of these Cs contributes to a workflow that enhances the quality, speed, and reliability of delivering software products:

1. Continuous Development
2. Continuous Integration
3. Continuous Testing
4. Continuous Deployment/Continuous Delivery
5. Continuous Monitoring
6. Continuous Feedback
7. Continuous Operations

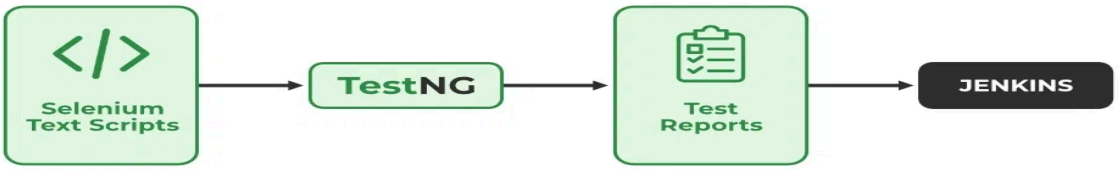
- Continuous Development: Develop features in small increments, enabling rapid testing and deployment (e.g., login and search features built and tested daily).



- Continuous Integration (CI): Automatically build, test, and integrate code changes using tools like GitHub, Jenkins, Maven, SonarQube, and Nexus (e.g., "Track Delivery" feature tested and built automatically).

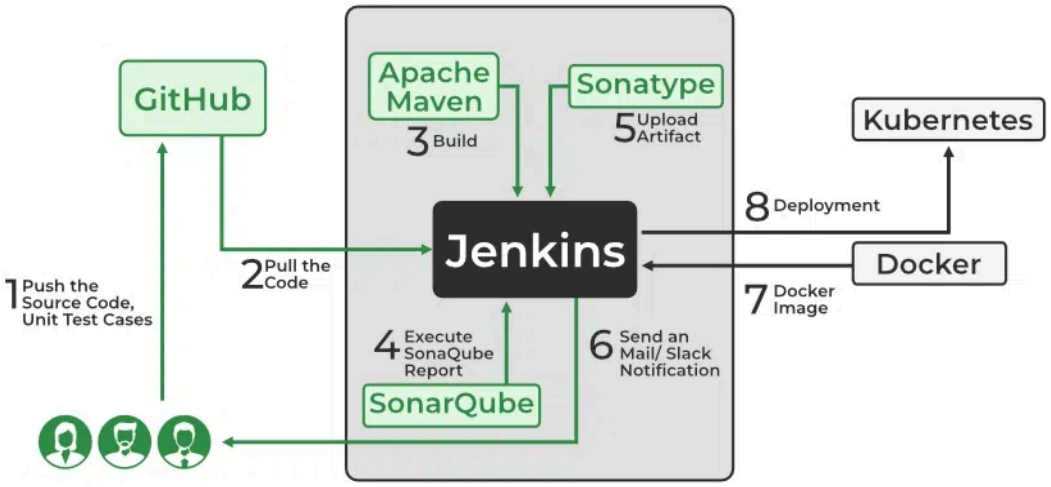


- Continuous Testing: Run automated tests every time code changes using tools like Selenium or Testsigma to catch bugs early (e.g., testing "Apply Coupon" feature instantly after code push).



- Continuous Deployment: Automatically release code to production after passing all tests (e.g., feature goes live without manual intervention).

- Continuous Delivery: Code is tested and ready for release but deployed manually based on business decisions (e.g., "Refer & Earn" feature scheduled for weekend launch).



- Continuous Monitoring: Use tools like Prometheus and Grafana to track app performance in real time and alert teams of issues (e.g., identifying slow "Order History" page during peak hours).
- Continuous Feedback: Collect and analyze user feedback to improve app features and fix bugs (e.g., fixing live tracking based on user complaints).
- Continuous Operations: Ensure app updates and maintenance happen without downtime, keeping services available 24/7 (e.g., updating payment system without interrupting users).

<u>Tools Used:</u>

1) Plan
   a) Tools: Jira, Trello, Asana
   b) Purpose: Task planning, assignment, and progress tracking.1
2) Develop
   a) Tools: Git, GitHub, GitLab, Bitbucket
   b) Purpose: Version control, code collaboration, branching, and merging.
3) Build
   a) Tools: Jenkins, Maven, Gradle
   b) Purpose: Automates code compilation and dependency management.
4) Test
   a) Tools: Selenium, JUnit, TestNG, SonarQube
   b) Purpose: Automated testing for functionality, quality, and security.
5) Release/Deploy
   a) Tools: ArgoCD, GitLab CI/CD, AWS CodeDeploy, Azure DevOps, Spinnaker, Terraform

b) Purpose: Automates release pipelines and deployment to environments.

6) Operate

a) Tools: Terraform, Ansible, Puppet, Chef

b) Purpose: Infrastructure provisioning, configuration management, and system operations.

7) Monitor

a) Tools: Prometheus, Grafana, ELK Stack, Datadog

b) Purpose: Real-time monitoring of performance, logs, metrics, and system health.

<u>Delivery Pipeline</u>

The pipeline represents the different stages that software goes through before it is released to production. These stages might typically include:

1. **Build Stage**

- Developers **write and organize code**, using version control tools like Git to track changes.
- The system **automatically compiles and packages the code** into a deployable format.
- Dependencies (external libraries and tools) are included to ensure smooth operation.
- **Common Tools:** Git, Jenkins, GitLab CI/CD, Gradle, Maven.

2. **Test Stage**

- The software undergoes **thorough testing** to catch bugs and security risks before release.
- Different testing methods include:

- **Unit Testing:** Checks individual pieces of code.
- **Integration Testing:** Ensures different parts of the system work together.
- **Performance Testing:** Measures speed and scalability.
- **Security Testing:** Identifies potential vulnerabilities.

- Automated tests help ensure the software is stable before moving forward.
- **Common Tools:** Selenium, JUnit, TestNG, SonarQube.

3. **Release Stage**

- The software is deployed in a **staging environment** to simulate real-world conditions.
- If everything checks out, the software is **rolled out to production** using deployment strategies like:

- **Blue-Green Deployment:** Two identical environments switch traffic for a seamless update.
- **Canary Deployment:** A small percentage of users get the new version first, ensuring safety.

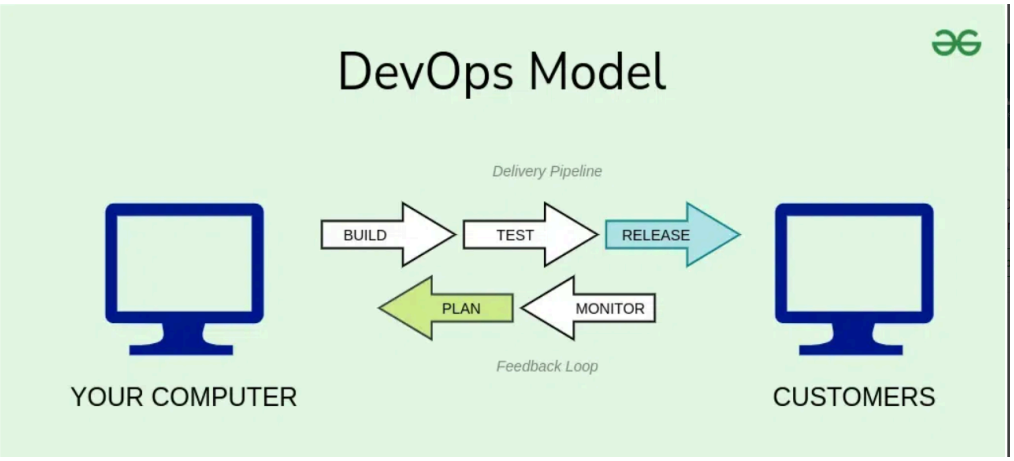
- **Rolling Updates:** The update is gradually pushed out to all users.

- **Common Tools:** Docker, Kubernetes, Ansible, Helm, ArgoCD.

4. **Continuous Feedback Loop**

A key aspect of DevOps is **learning from real-world performance** and using that feedback to improve future releases.

- **Monitoring & Logging:** Track system performance and detect errors.
- **User Feedback:** Gather insights from customers to enhance features.
- **Incident Response:** Alert systems notify teams of failures for quick fixes.
- **Process Improvement:** Teams analyze past releases to optimize automation and workflow.
- **Common Tools:** Prometheus, Grafana, ELK Stack, Datadog, New Relic.



<u>Conclusion:</u>

Conclusion about DevOps

DevOps is a powerful methodology that bridges the gap between development and operations, focusing on collaboration, automation, and continuous delivery. It enhances the speed, quality, and reliability of software by integrating all stages of the software lifecycle—from planning to monitoring. With practices like continuous integration, testing, deployment, and feedback, DevOps ensures faster releases, reduced errors, and improved user experience. Overall, DevOps is not just a set of tools or processes—it's a cultural shift that drives innovation and efficiency in modern software development.

2. Git Installation

Aim: To Understand Version Control System, Git installation & GitHub Account.

Theory:

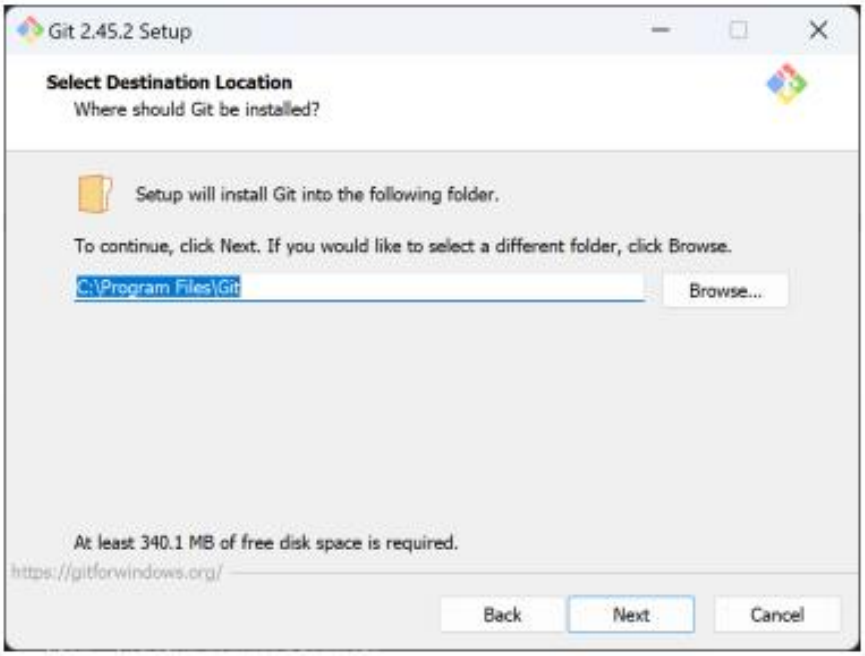
1. Version Control System
    - Version control - also known as source control or revision control - is an important software development practice for tracking and managing changes made to code and other files. It is closely related to source code management.
    - With version control, every change made to the code base is tracked. This allows software developers to see the entire history of who changed what at any given time — and roll back from the current version to an earlier version if they need to. It also creates a single source of truth.
    - Version control (or source control or revision control) serves as a safety net to protect the source code from irreparable harm, giving the development team the freedom to experiment without fear of causing damage or creating code conflicts.
    - If developers code concurrently and create incompatible changes, version control identifies the problem areas so that team members can quickly revert changes to a previous version, compare changes, or identify who committed the problem code through the revision history. With a version control system (VCS), a software team can solve an issue before progressing further into a project. Through code reviews, software teams can analyze earlier versions to understand the changes made to the code over time.
    - Depending on a team's specific needs and development process, a VCS can be local, centralized, or distributed. A local VCS stores source files within a local system, a centralized VCS stores changes in a single server, and a distributed VCS involves cloning a Git repository.

2. Benefits of version control
    • Quality: Teams can review, comment, and improve each other's code and assets.
    • Acceleration: Branch code, make changes, and merge commits faster.
    • Visibility: Understand and spark team collaboration to foster greater release build and release patterns. Better visibility improves everything from project management to code quality.

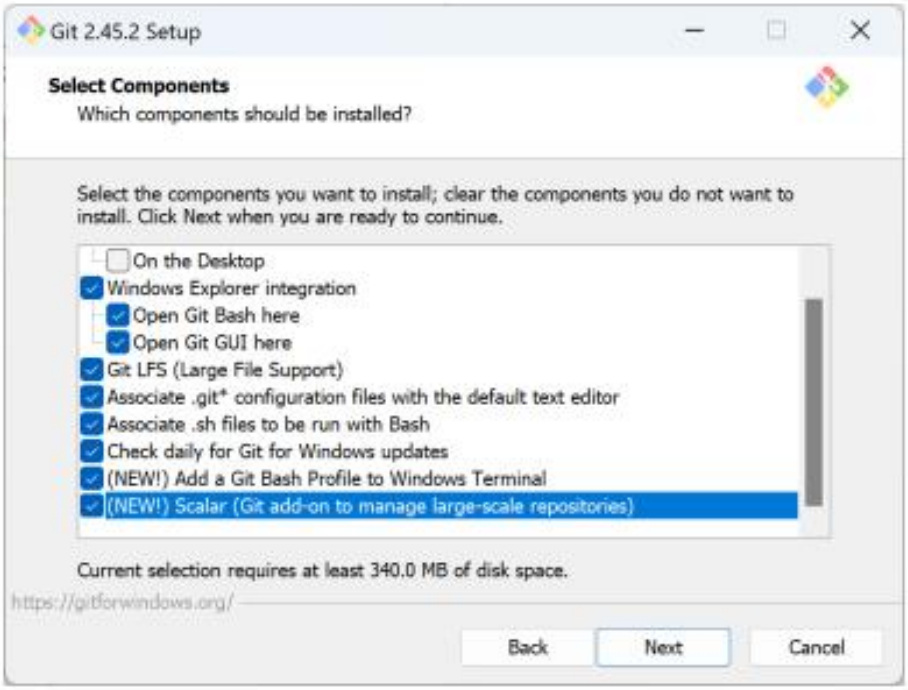
3. Installation Process:

Step – 0 : Click Next.

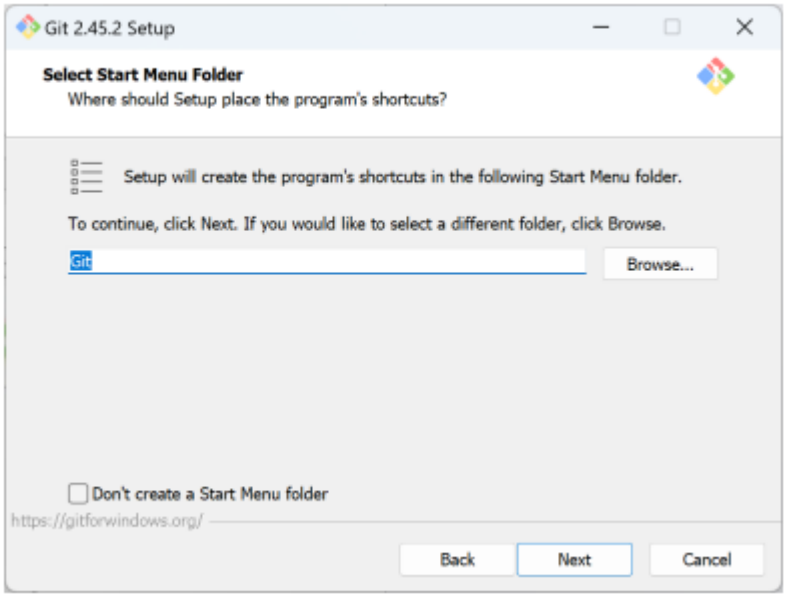
Step – 1: Select the file path for installation.



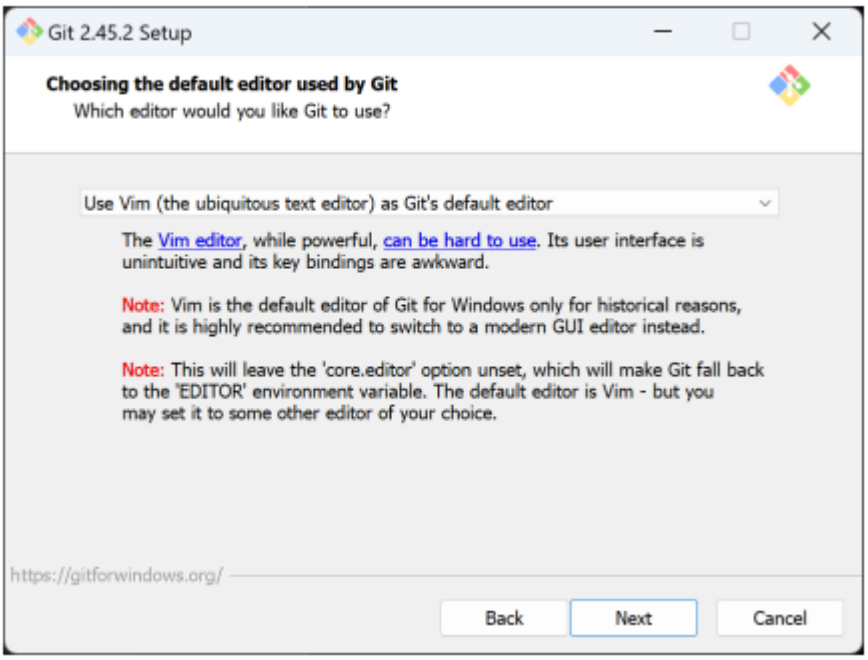
Step – 2: Select the required components for installation.



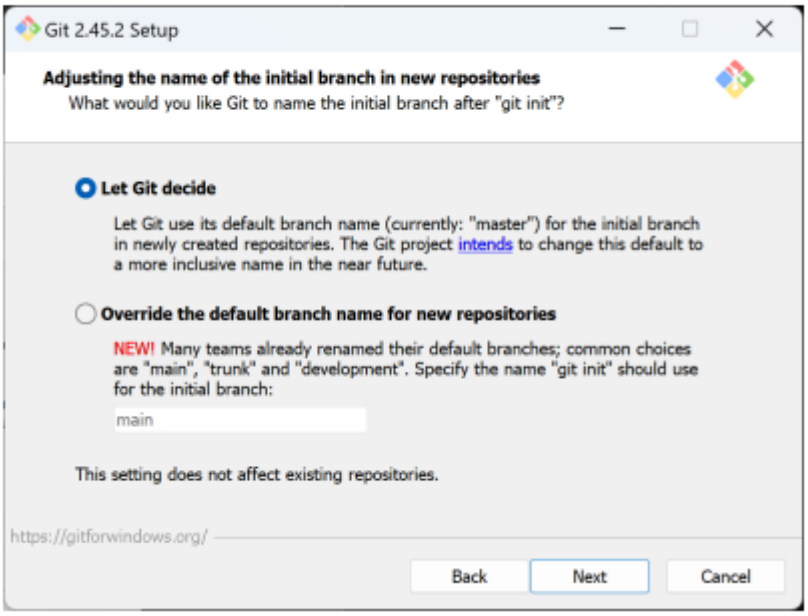
Step – 3: Select the folder name for setup.



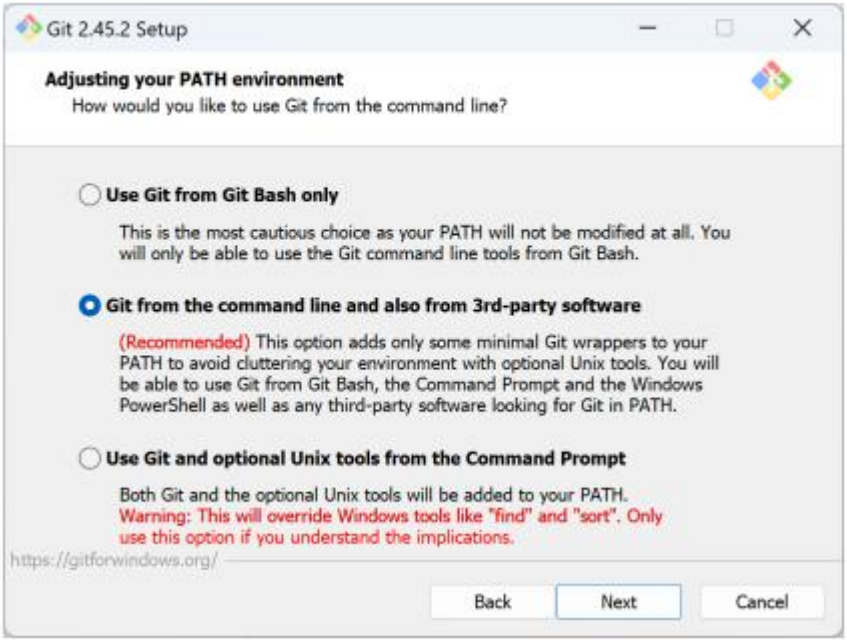
Step – 4: Select the default text editor for using Git.

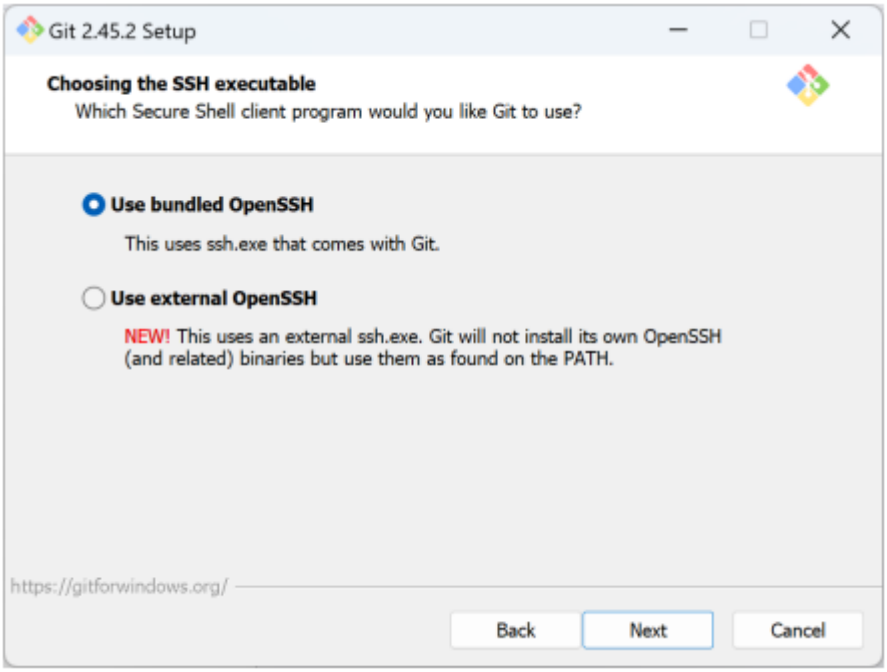


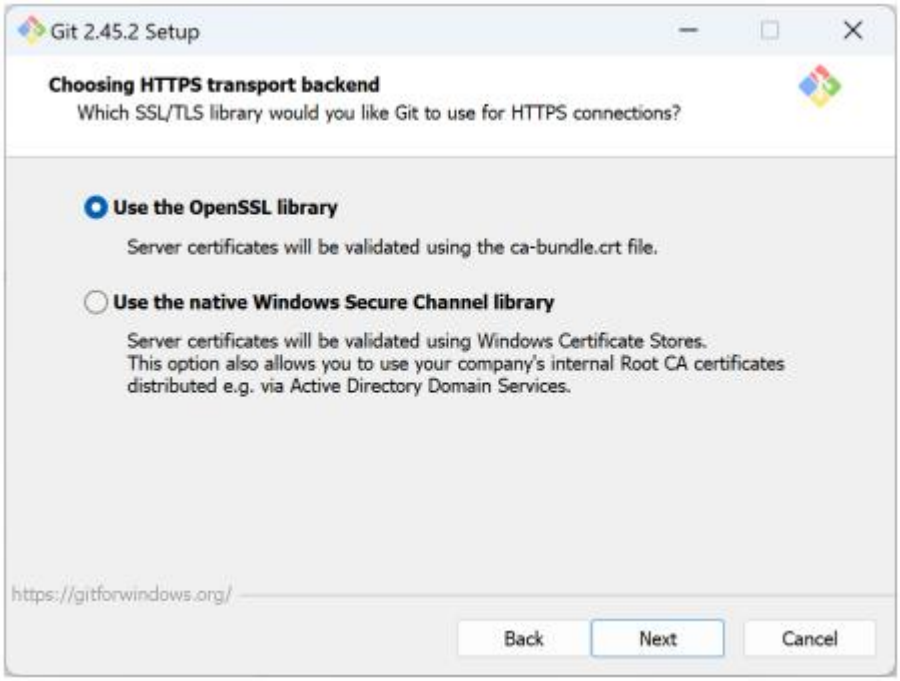
Step – 5: Setup required formats.

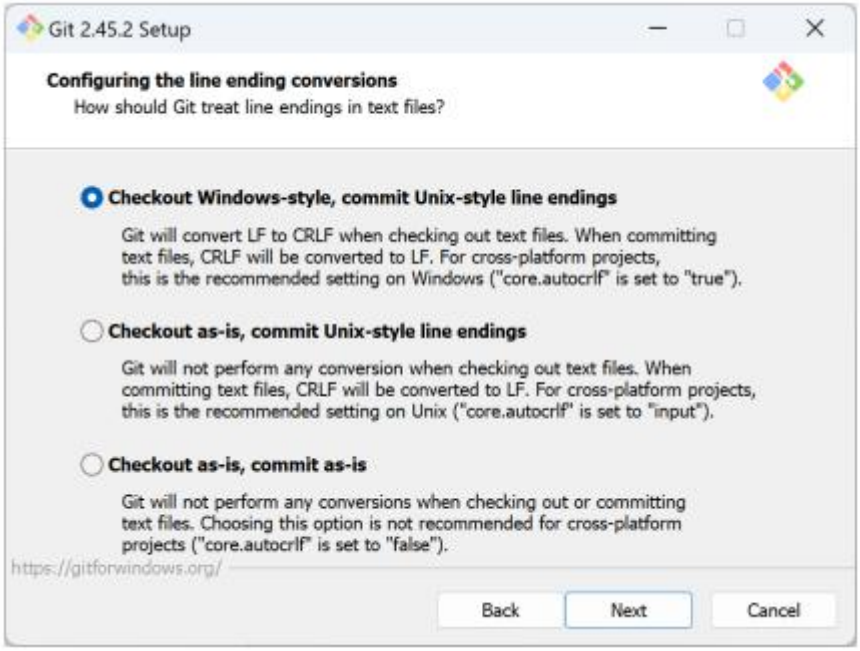


Step – 6: Authorise necessary permissions.

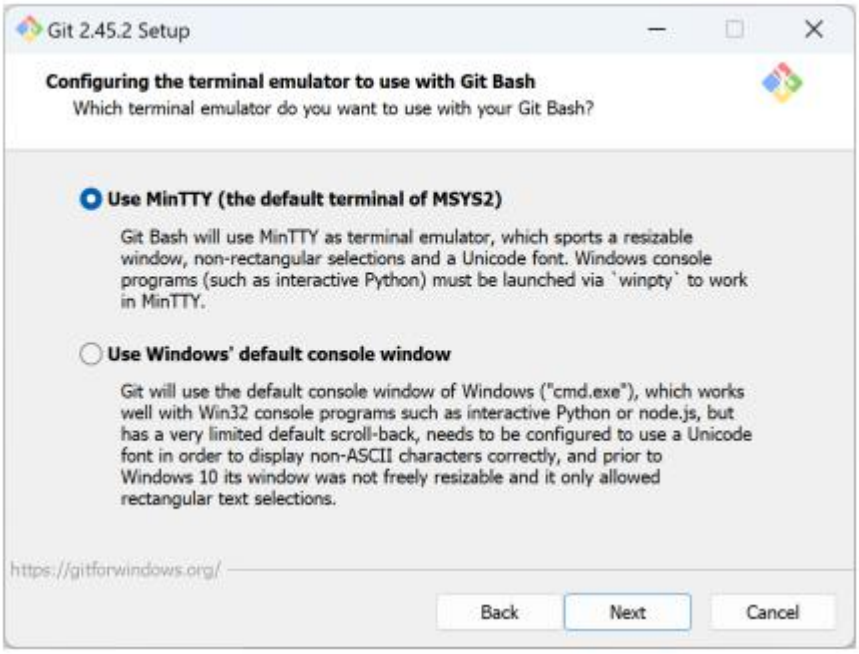


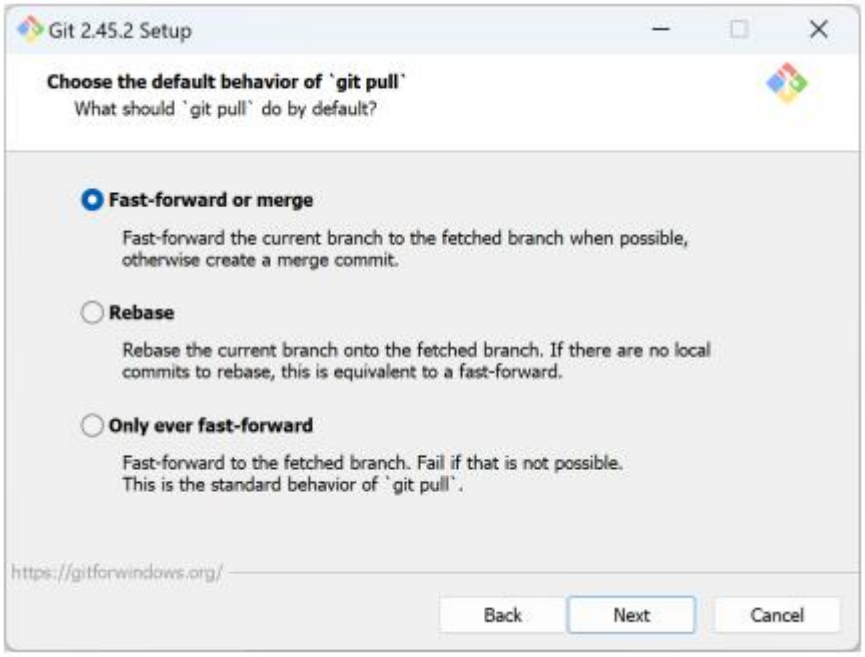


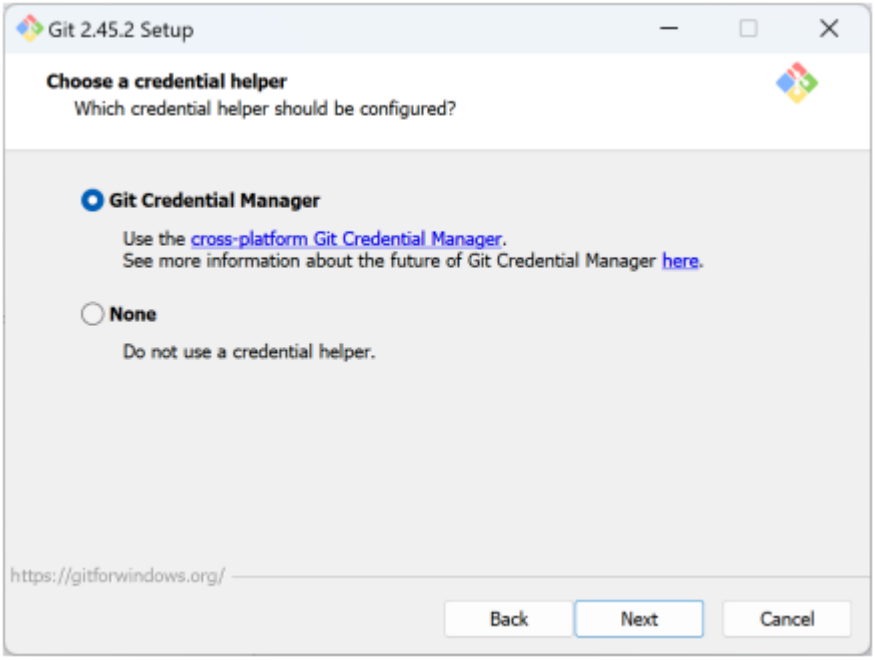

Git 2.45.2 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates
distributed e.g. via Active Directory Domain Services.
https://gitforwindows.org/
Back
Next
Cancel

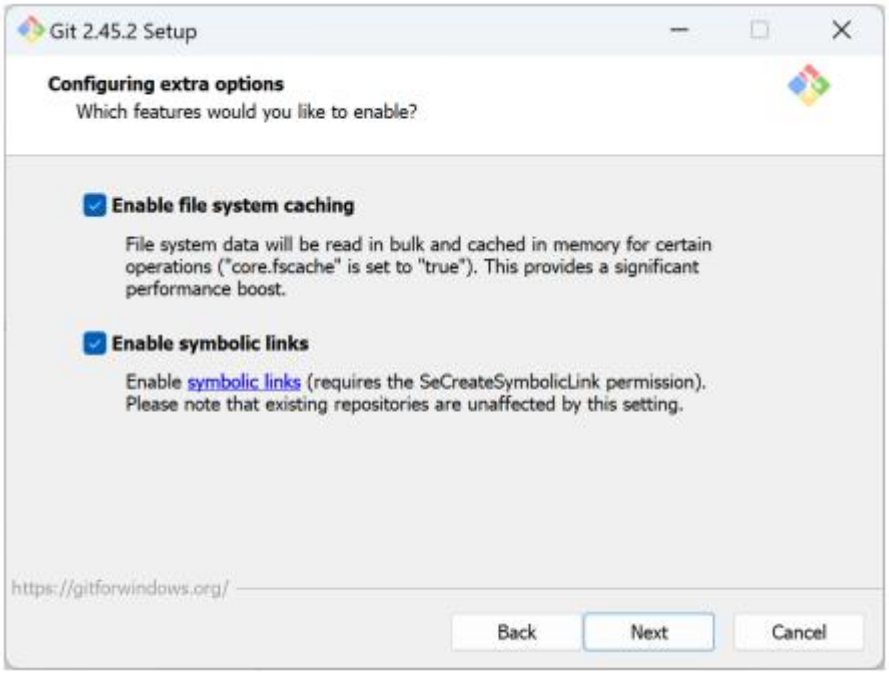

Git 2.45.2 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing
text files, CRLF will be converted to LF. For cross-platform projects,
this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When
committing text files, CRLF will be converted to LF. For cross-platform projects,
this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing
text files. Choosing this option is not recommended for cross-platform
projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
Back
Next
Cancel

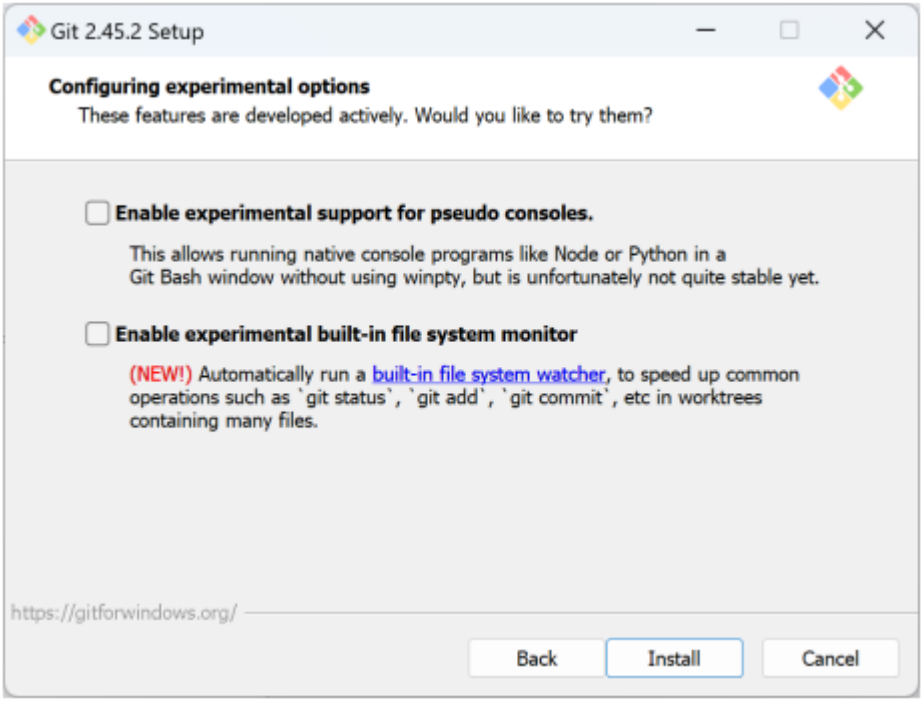
Step – 7: Configure the BASH terminal.



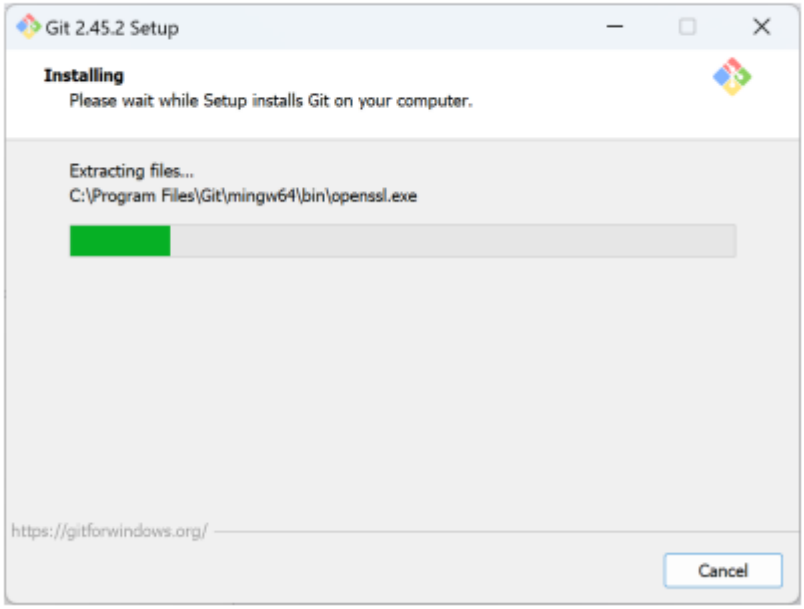



Git 2.45.2 Setup
Choose a credential helper
Which credential helper should be configured?
Git Credential Manager
Use the cross-platform Git Credential Manager.
See more information about the future of Git Credential Manager here.
None
Do not use a credential helper.
https://gitforwindows.org/
Back
Next
Cancel

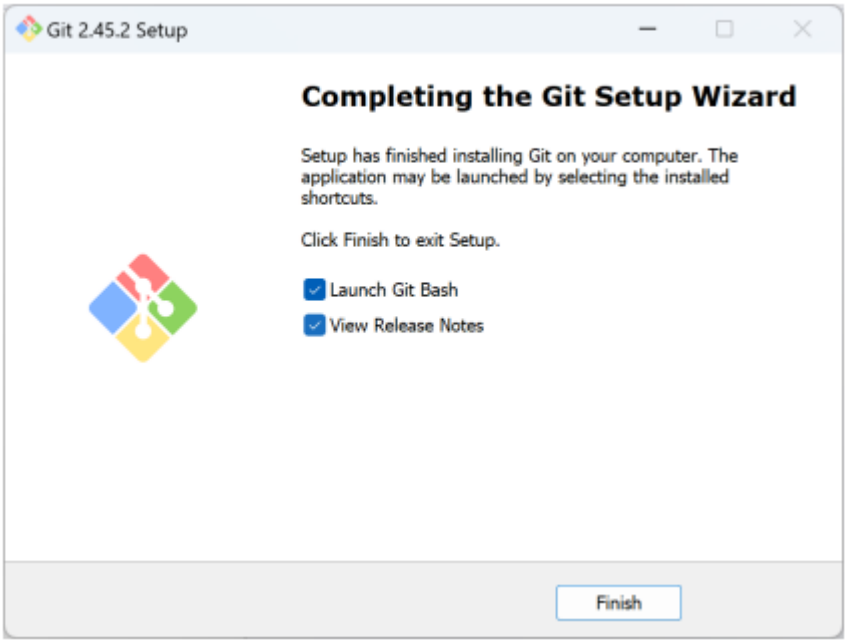

Git 2.45.2 Setup
Configuring extra options
Which features would you like to enable?
Enable file system caching
File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.
Enable symbolic links
Enable symbolic links (requires the SeCreateSymbolicLink permission).
Please note that existing repositories are unaffected by this setting.
https://gitforwindows.org/
Back
Next
Cancel



Step – 8: Click Install.



Step – 9: Setup Complete.



Conclusion: Basic knowledge regarding Git BASH installation and Version Control System to effectively track changes augmented with Git and GitHub.

LO - 2: To obtain complete knowledge of the "Version Control System" to effectively track changes augmented with Git & GitHub, is achieved

3. Assignment 3

Aim: To perform various Git Operations.

Lab Outcome: LO1, LO2 Mapped

Theory:

Git is a distributed version control system that enables collaboration among developers by tracking changes in code. It allows multiple people to work on the same project simultaneously, maintaining different versions of the code through commits and branches. This system facilitates efficient code merging, conflict resolution, and the ability to work on new features independently. Git also provides a remote repository for sharing and backup, making it an essential tool for managing and tracking code changes in software development projects.

Git commands are instructions that you give to the Git version control system to perform various tasks and operations on your source code repository. These commands allow you to interact with your code, track changes, collaborate with others, and manage your project's history effectively. Each Git command corresponds to a specific action, such as initialising a repository, staging changes, committing code, creating branches, merging changes, fetching remote updates, and more. Some of these are given below:

1. git init: Initialises a new Git repository.
   Example: git init

2. git clone: Copies a remote repository to your local machine.
   Example: git clone https://github.com/username/repo.git

3. git add: Stages changes for commit.
   Example: git add file.txt

4. git commit: Records staged changes with a message.
   Example: git commit -m "Added newfeature"

5. git status: Shows the status of your working directory.
   Example: git status

6. git log: Displays commit history.
   Example: git log

7. git branch: Lists, creates, or deletes branches.
   Example: git branch new-feature

8. git checkout: Switches to a different branch.
   Example: git checkout new-feature

9. git merge: Combines changes from different branches.
   Example: git merge feature-branch

11. git pull: Fetches remote changes and merges them into the current branch.
    Example: git pull origin main

12. git push: Uploads local changes to a remote repository.
    Example: git push origin main

13. git remote: Manages remote repositories.
    Example: git remote add origin https://github.com/username/repo.git

15. git fetch: Retrieves remote changes but doesn't merge them.
    Example: git fetch origin

16. git diff: Shows differences between working directory and last commit.
    Example: git diff

17. git reset: Unstaged files or discards changes.
    Example: git reset file.txt

Commands & Output:

```
Lab1002@MU2024 MINGW64 ~
$ git config
usage: git config [<options]

Config file location
--global          use global config file
--system          use system config file
--local           use repository config file
--worktree        use per-worktree config file
-f, --file                use given config file
--blob            read config from given blob object

Action
--get                     get value: name [value-pattern]
--get-all                 get all values: key [value-pattern]
--get-regexp              get values for regexp: name-regex [value-pattern]
--get-urlmatch            get value specific for the URL: section[.var] URL
--replace-all             replace all matching variables: name value [value-pattern]
--add                     add a new variable: name value
--unset remove a variable: name [value-pattern]
--unset-all               remove all matches: name [value-pattern]
--rename-section          rename section: old-name new-name
--remove-section          remove a section: name
-l, --list                list all
--fixed-value             use string equality when comparing values to 'value-pattern'
-e, --edit                open an editor
--get-color               find the color configured: slot [default]
--get-colorbool           find the color setting: slot [stdout-is-tty]

Type
-t, --type        value is given this type
--bool            value is "true" or "false"
--int             value is decimal number
--bool-or-int     value is                  --bool or --int
--bool-or-str     value is                  --bool or string
--path value is a path (file or directory name)
--expiry-date     value is an expiry date

Other
-z, --null                terminate values with NUL byte
--name-only               show variable names only
--includes                respect include directives on lookup
--show-origin             show origin of config (file, standard input, blob, command line)
--show-scope              show scope of config (worktree, local, global, system, command)
--default with --get,     use default value when missing entry
Lab1002@MU2024 MINGW64 ~
$ git config --global user.email aarnabafna@gmail.com


Lab1002@MU2024 MINGW64 ~
$ git status
```

fatal: not a git repository (or any of the parent directories):
.git
Lab1002@MU2024 MINGW64 ~
$ git config --global user.name aarna

Lab1002@MU2024 MINGW64 ~
$ git init
Initialized empty Git repository in C:/Users/Lab1002/.git/

Lab1002@MU2024 MINGW64 ~ (master)
$ git status

On branch master No commits yet
Lab1002@MU2024 MINGW64 ~ (master)
$ git --version
git version 2.41.0.windows.3

Lab1002@MU2024 MINGW64 ~ (master)
$ git remote add origin "https://github.com/AarnaBafna07/DevOps.git"

Lab1002@MU2024 MINGW64 ~ (master)
$ git pull origin main
remote: Enumerating objects: 9, done. remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9 (delta 0), reused 0 (delta 0), pack-reused 0 Unpacking objects: 100% (9/9), 1.83 KiB | 5.00 KiB/s, done.

Conclusions: In conclusion, I've effectively used different Git commands, demonstrating their crucial role in version control and teamwork. This hands-on experience highlights Git's essential role in making coding processes more organised and collaborative

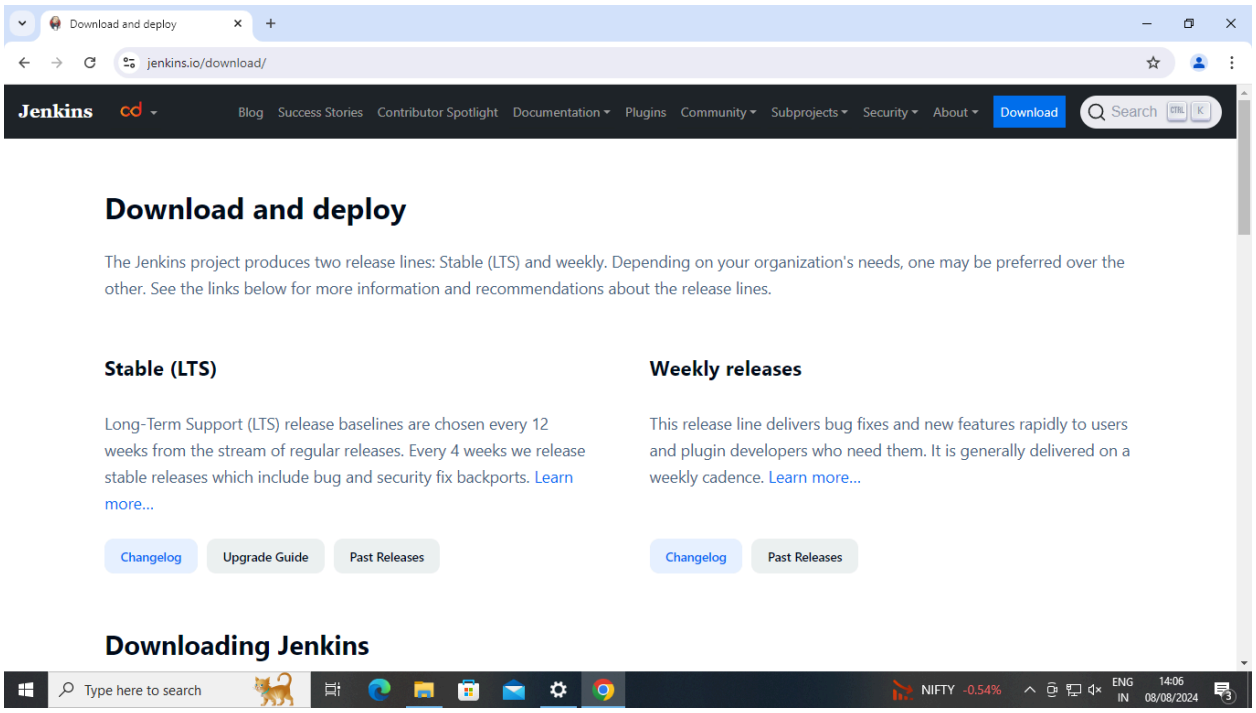
DEVOPS ASSIGNMENT 4

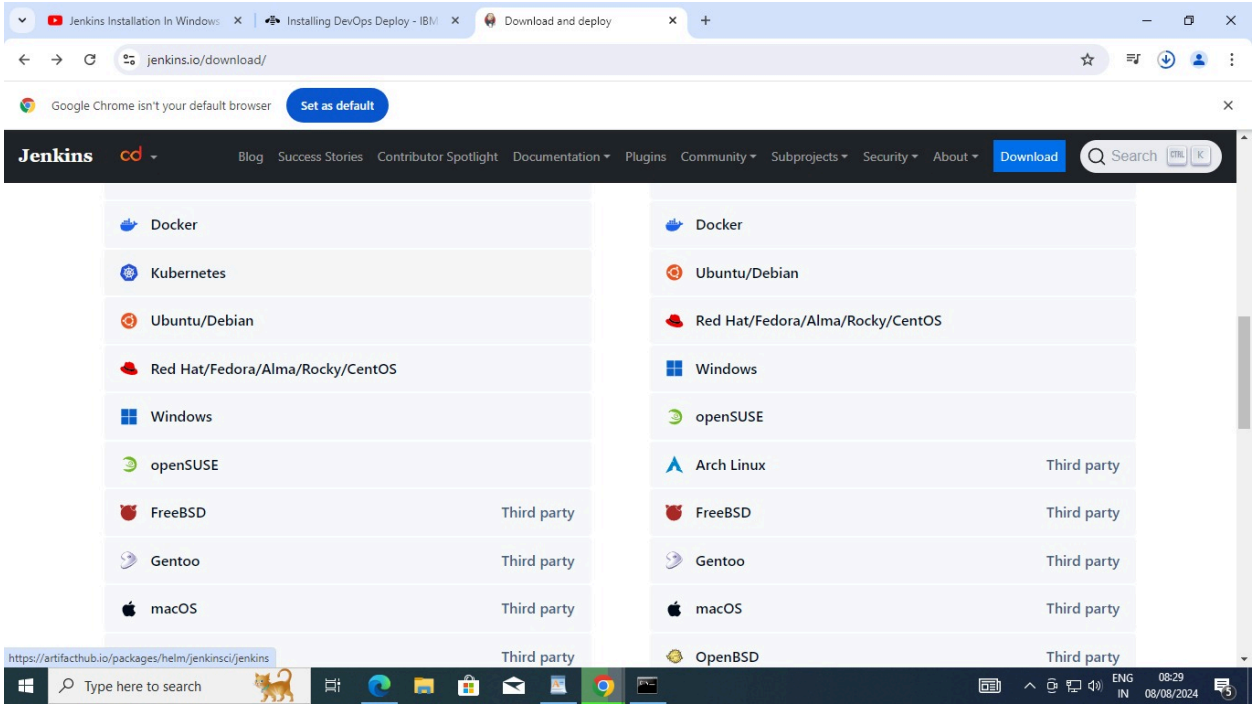
Aim: To install Jenkins on your PC

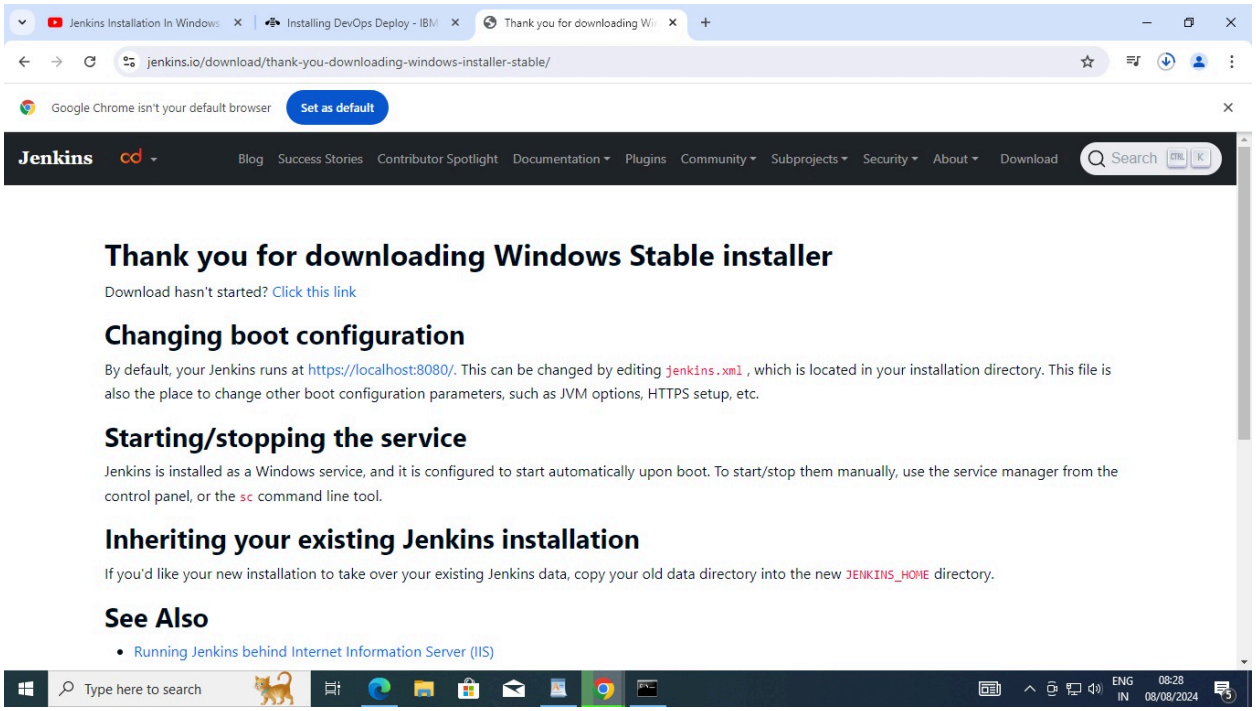
Theory: Jenkins is a tool used for automating parts of the software development process. Specifically, it helps with continuous integration and continuous delivery (CI/CD). This means Jenkins can automatically build, test, and deploy code changes, making it easier for developers to deliver updates quickly and reliably. It helps streamline workflows and catch issues early by integrating various tools and running tasks on code changes.

Steps for Installation of Jenkins:

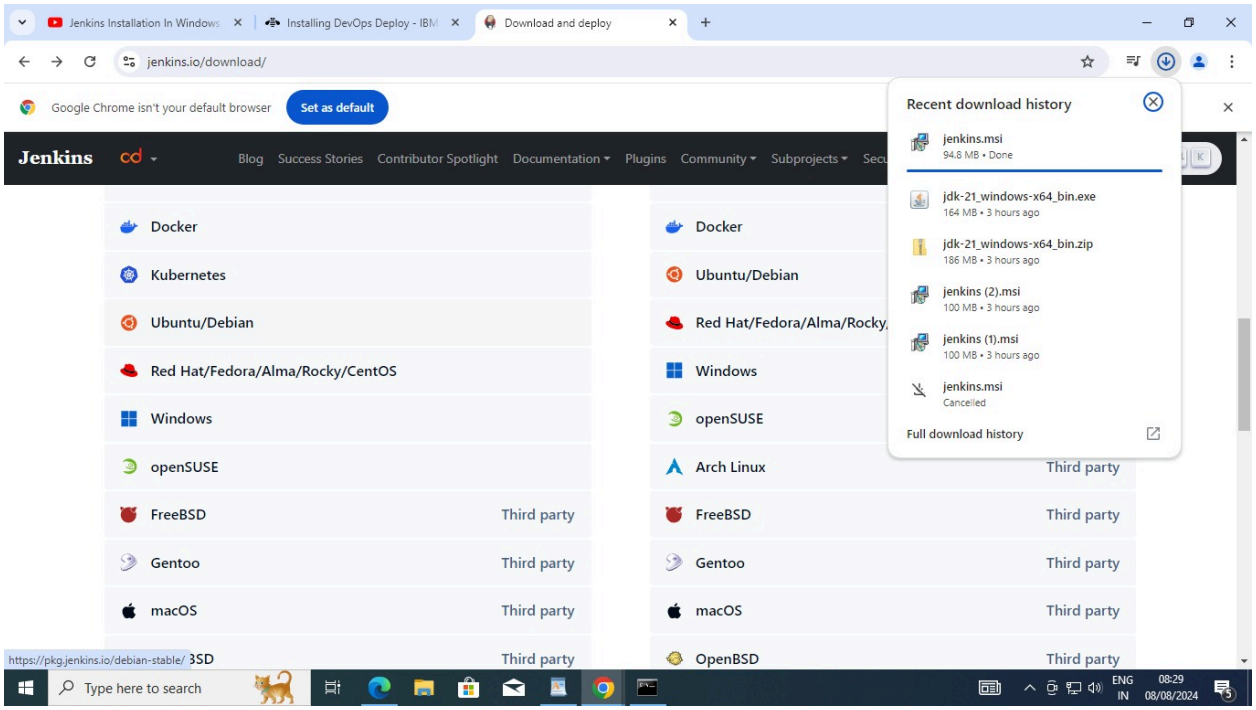
1) **Visit this website and download https://www.jenkins.io/download/ and select windows which will redirect you to the download page and the file will start downloading automatically.**



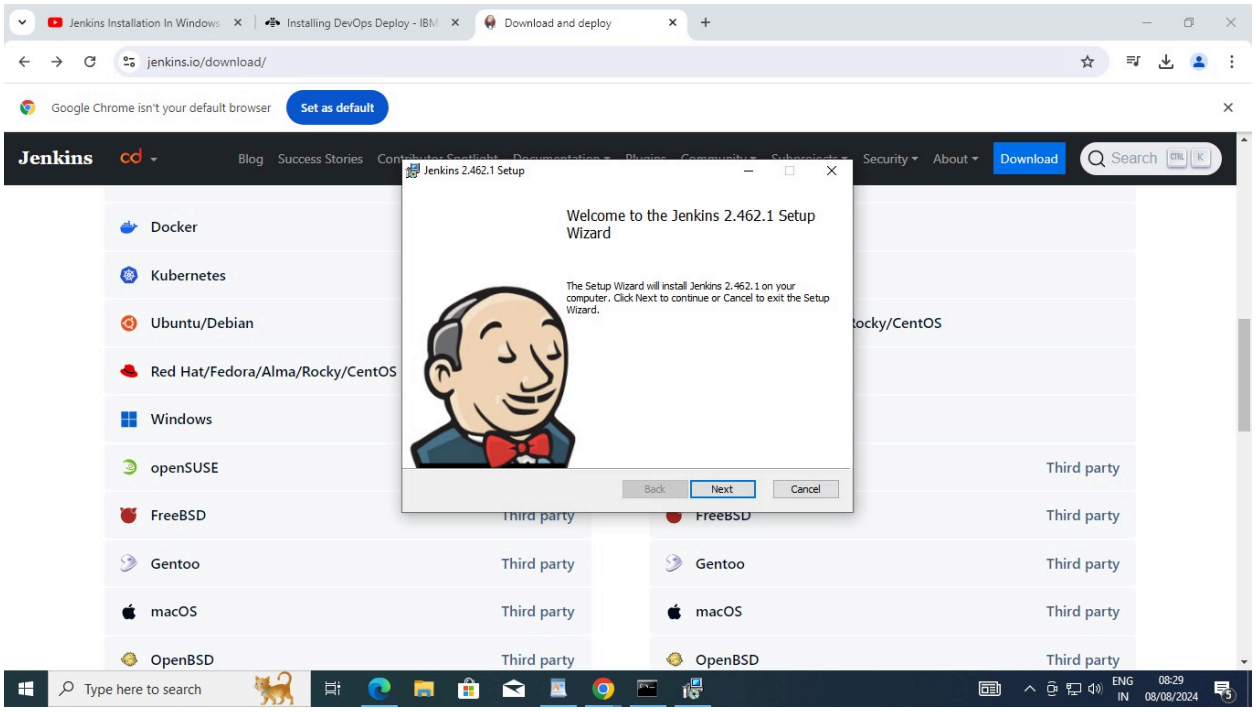




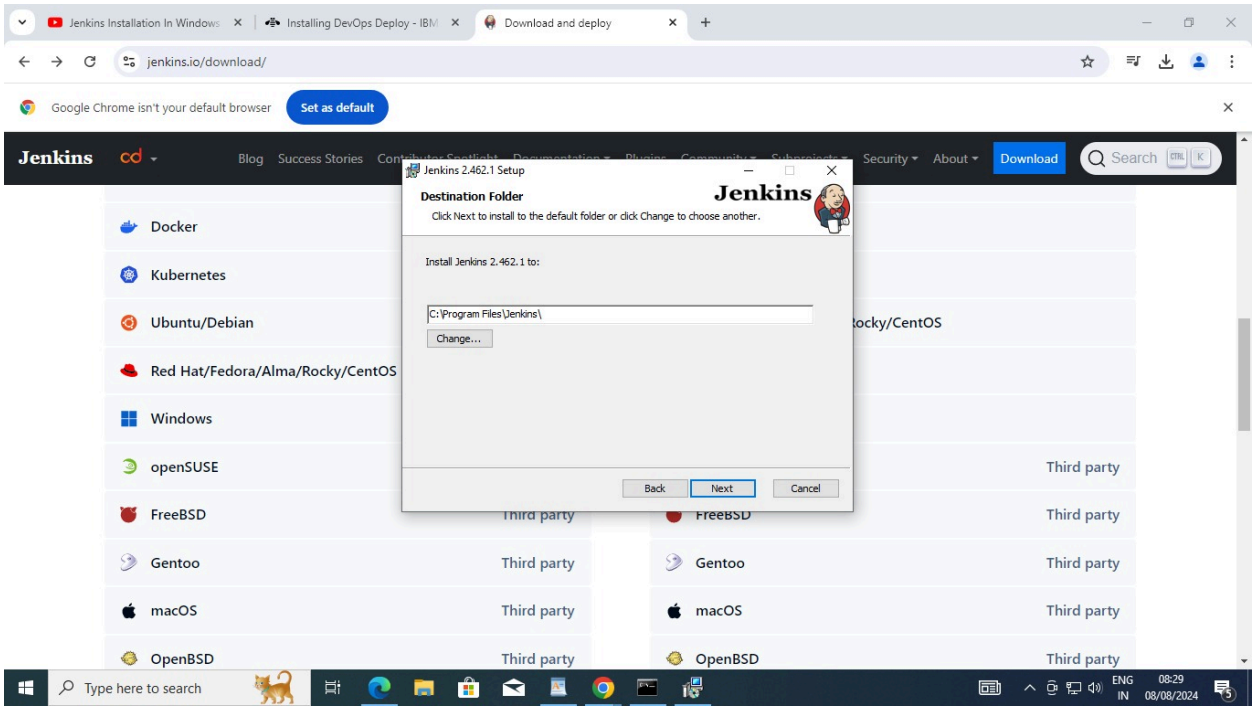
2) Download has been completed as seen in below screenshot



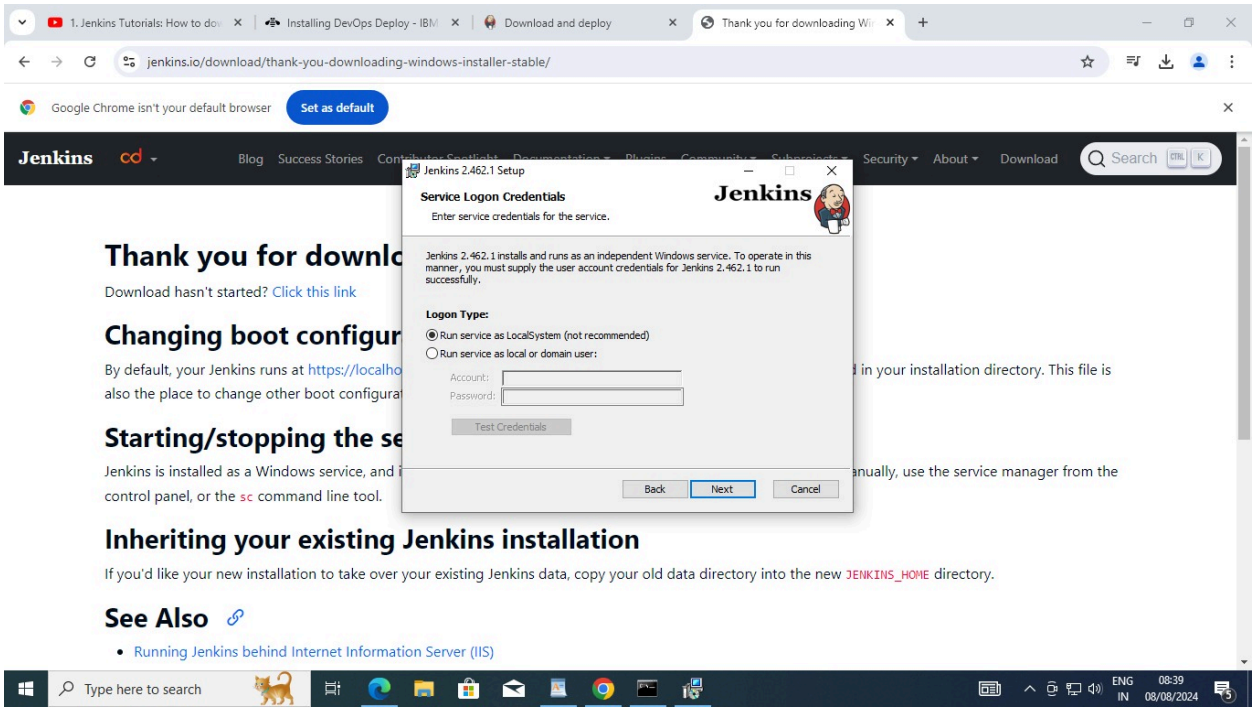
3) **Click on the exe file and start installing by clicking NEXT**



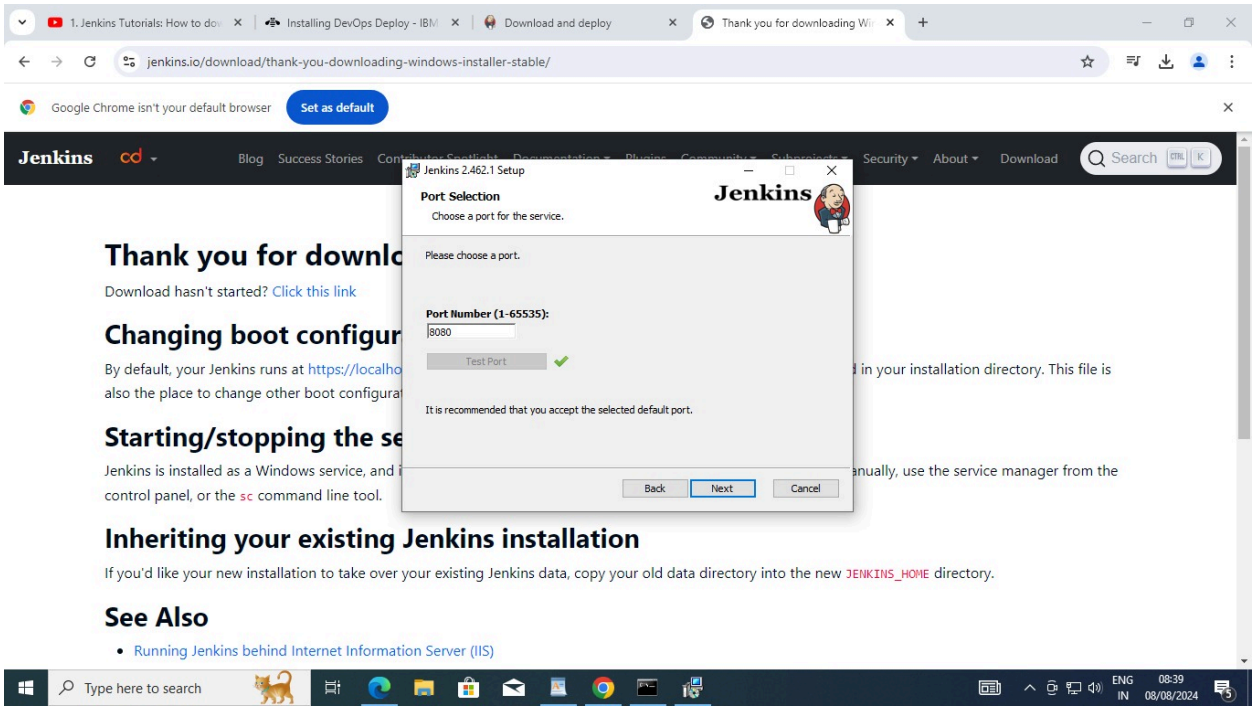
4) **Then change the path to the path where your jdk folder is located. Click on NEXT**



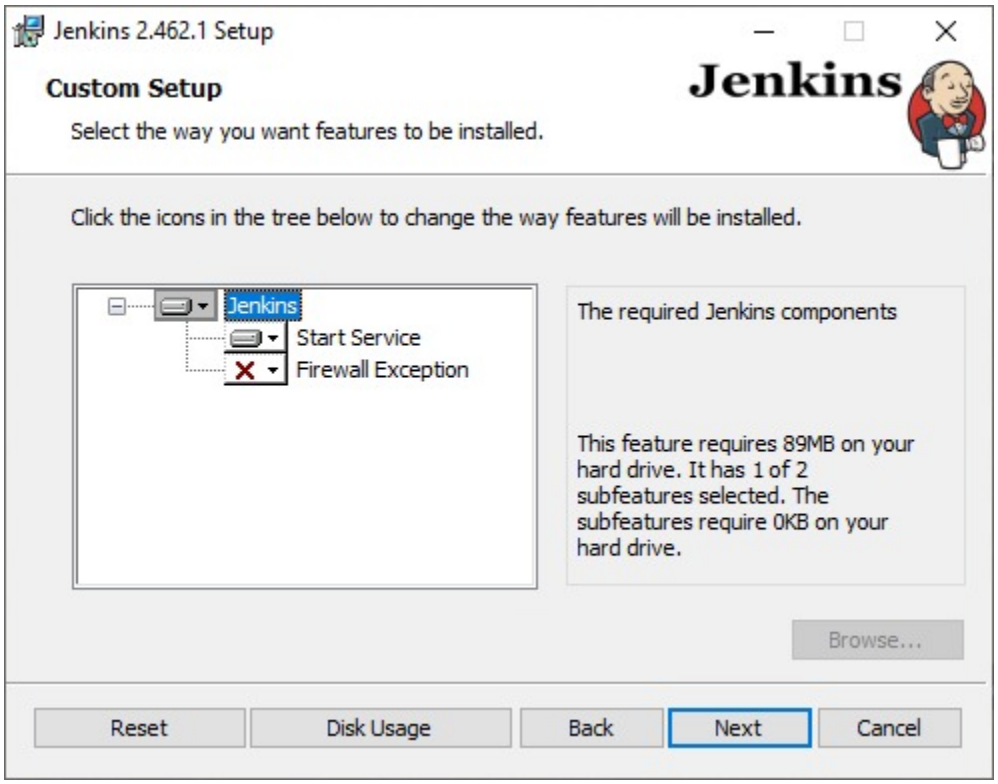
5) **Here select 'Run service as LocalHost' and click next**



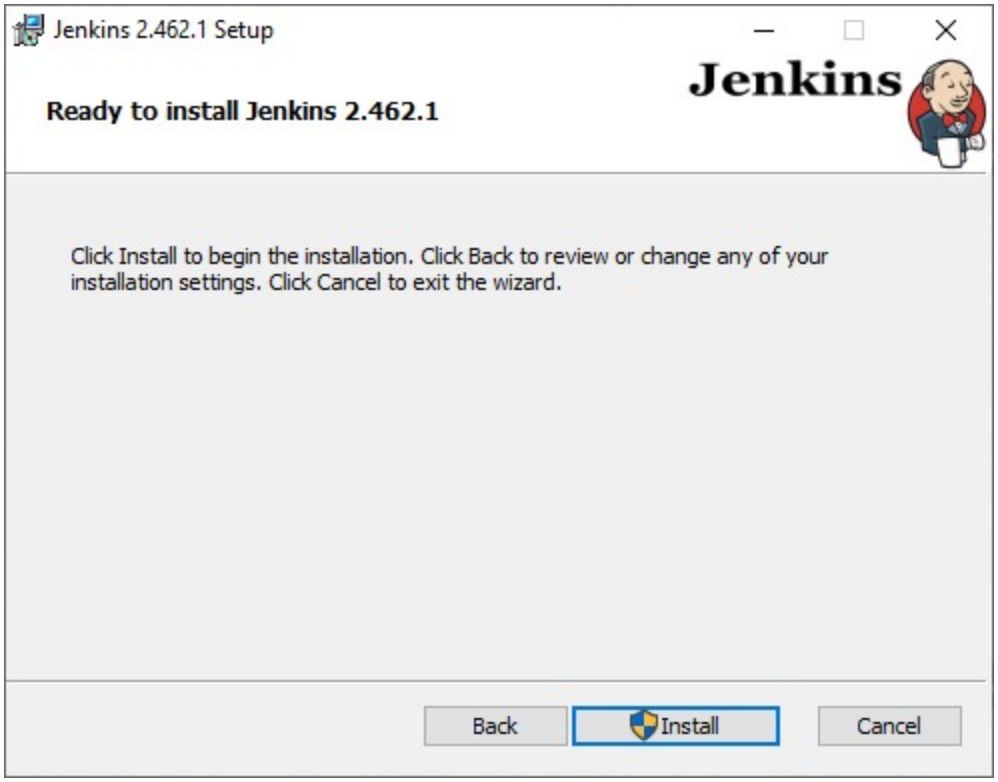
6) **In the next screen click on 'Test Port' and remember your port number. Then click NEXT**

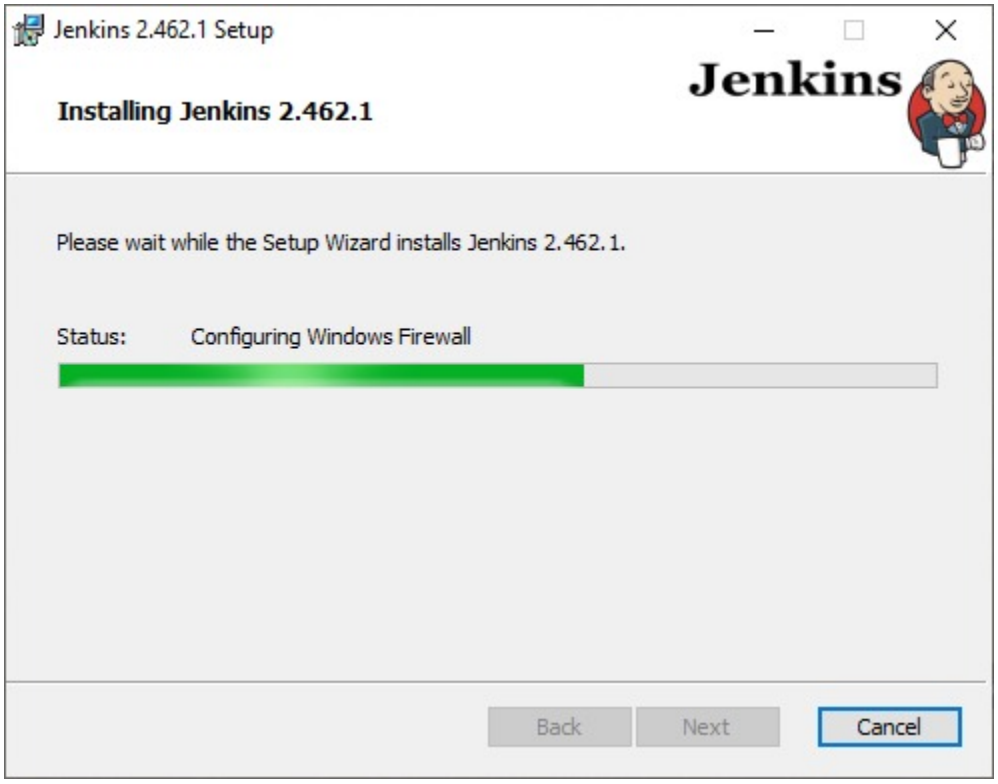


7) **In the next screen click on the firewall drop down and select 'Will be installed on Local Hard Drive'. Then click next**

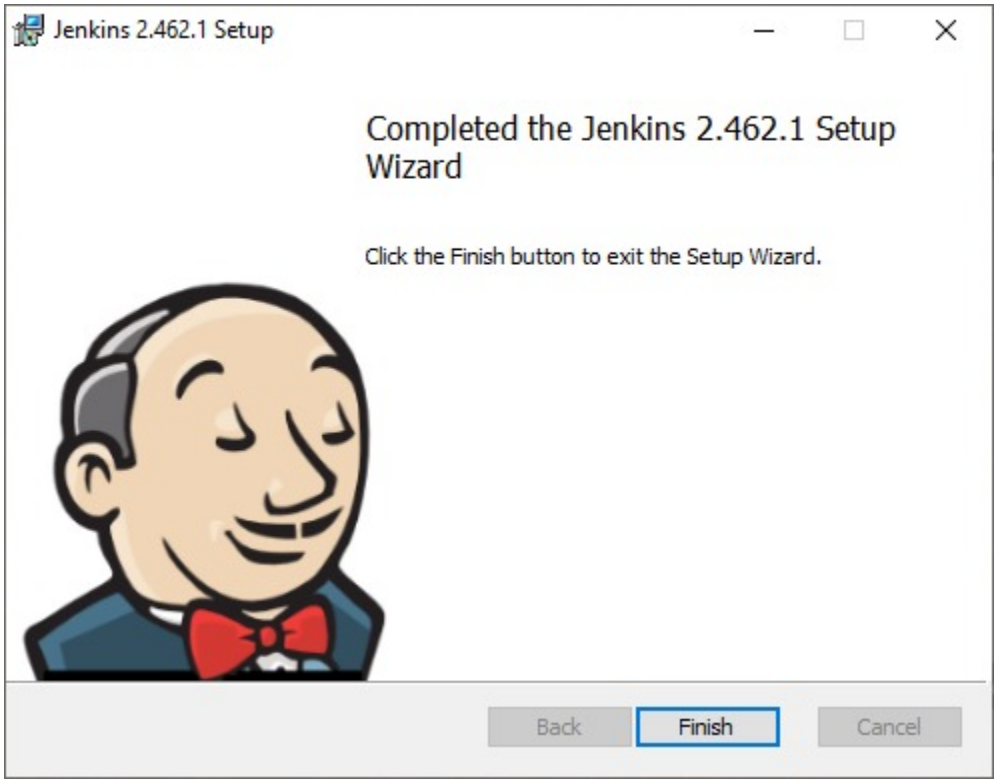


8) **Click on Install now**

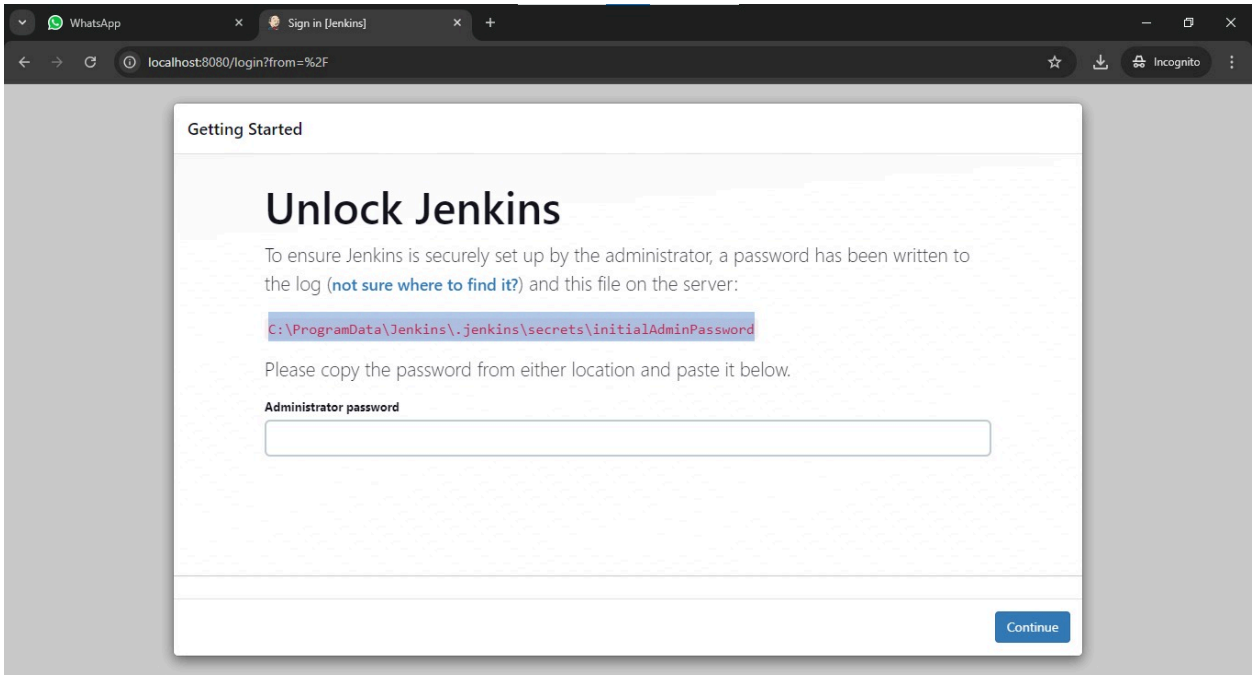

Jenkins 2.462.1 Setup
Jenkins
Ready to install Jenkins 2.462.1
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel


Jenkins 2.462.1 Setup
Jenkins
Installing Jenkins 2.462.1
Please wait while the Setup Wizard installs Jenkins 2.462.1.
Status:
Configuring Windows Firewall
Back
Next
Cancel

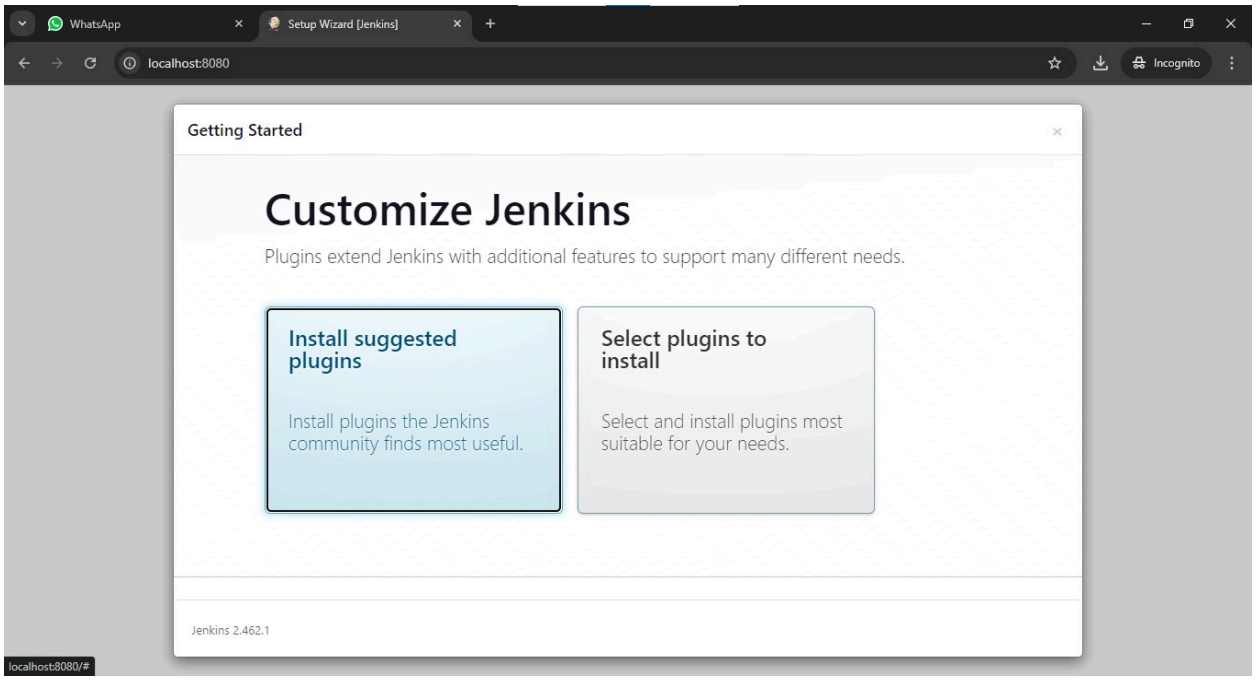
9) **Click on FINISH**



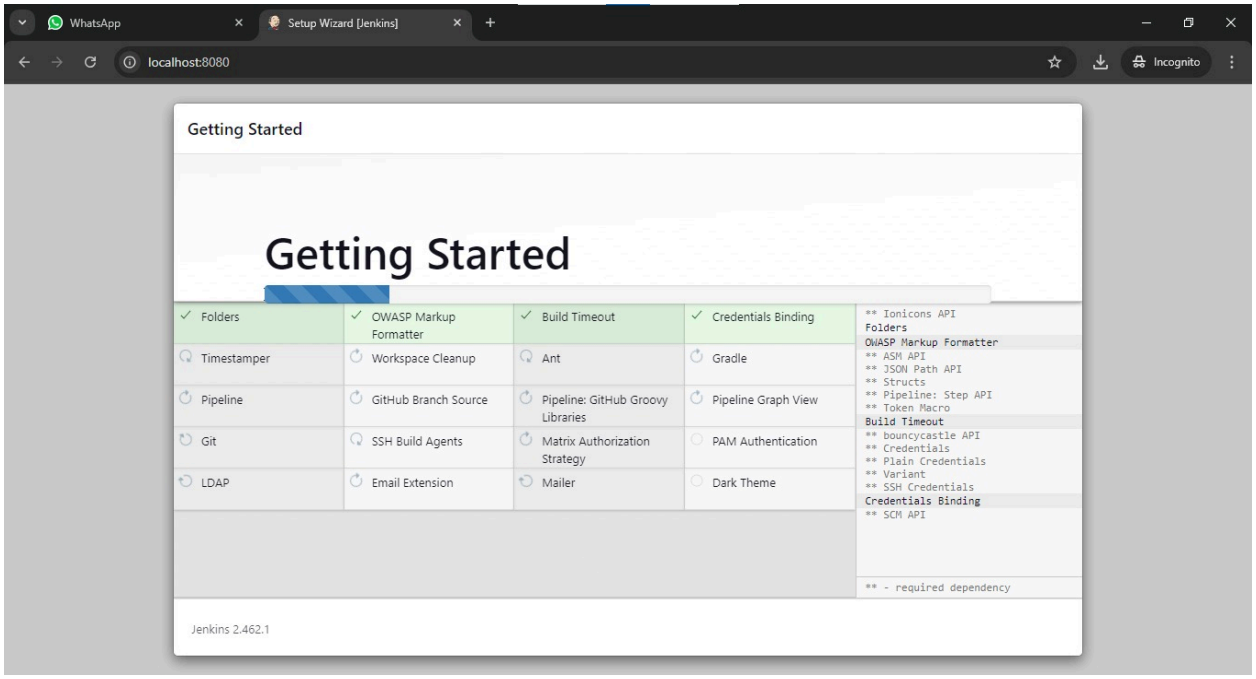
10) **Login in Jenkins**



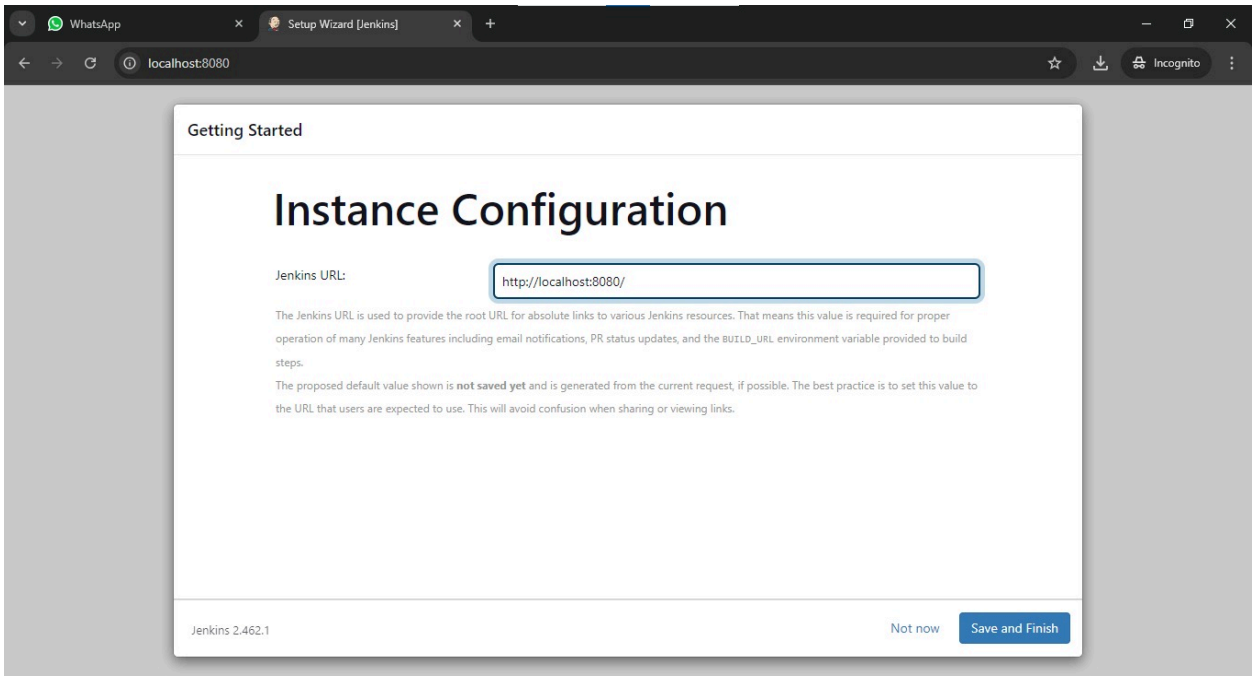
11) **Install plugins**



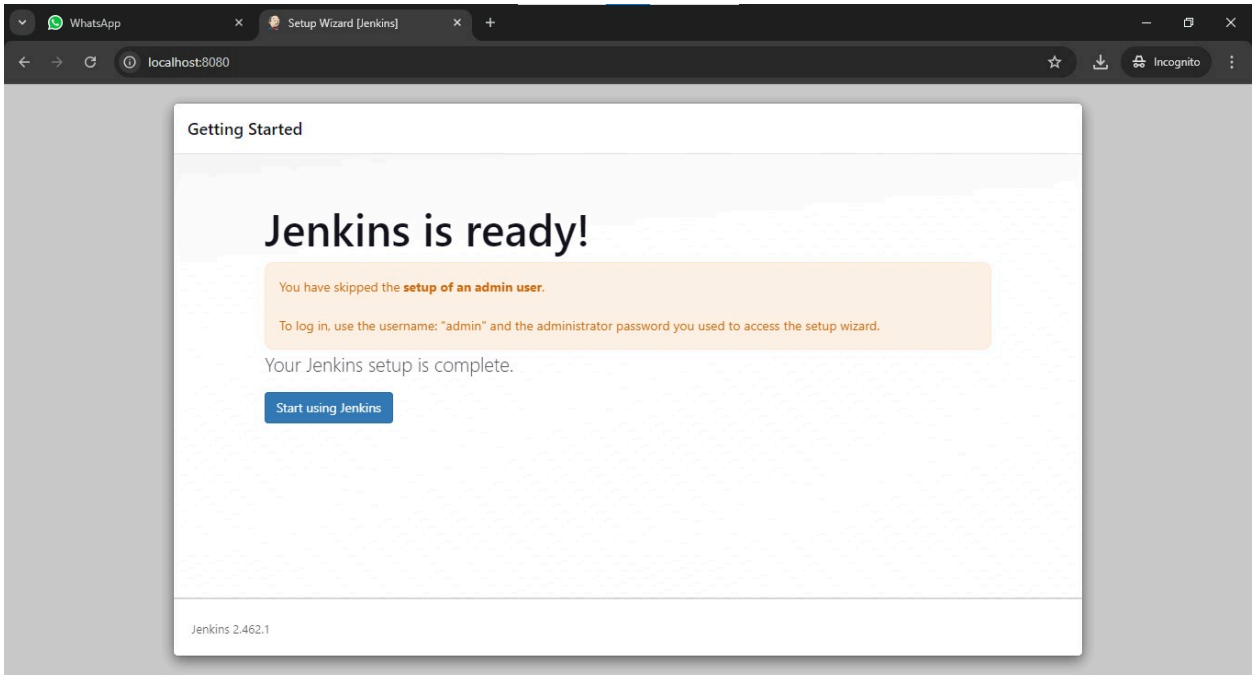
12) Installation

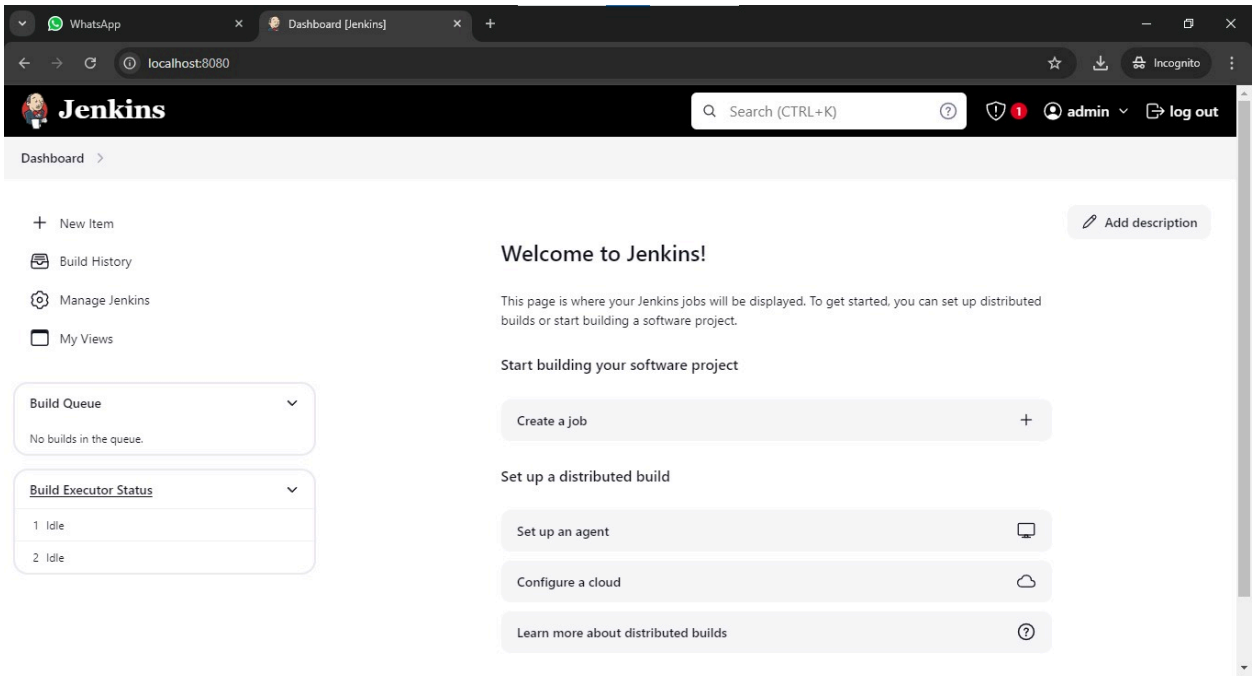


12) Instance configuration



13) Jenkins is ready




WhatsApp
Dashboard [Jenkins]
localhost:8080
Incognito
Jenkins
Search (CTRL+K)
admin
log out
Dashboard
New Item
Build History
Manage Jenkins
My Views
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
Add description
Welcome to Jenkins!
This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.
Start building your software project
Create a job
Set up a distributed build
Set up an agent
Configure a cloud
Learn more about distributed builds

LAB ASSIGNMENT 5

AIM: To build a Java program using Jenkins.

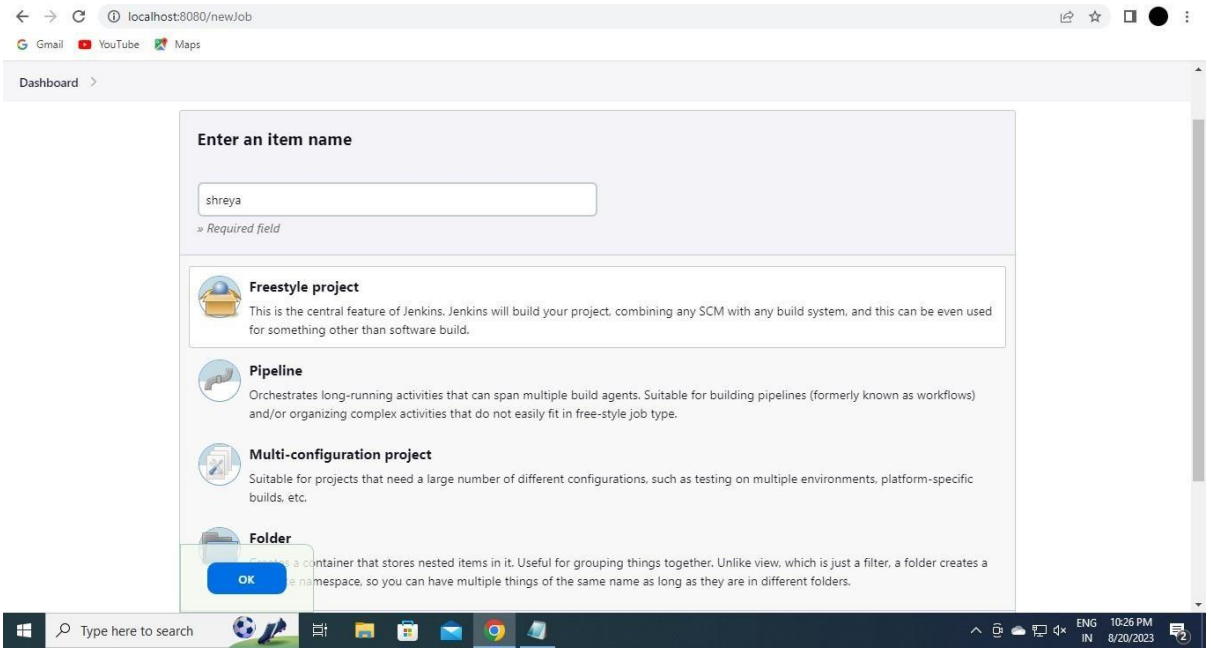
LAB OUTCOME:
LO1, LO3 Mapped.

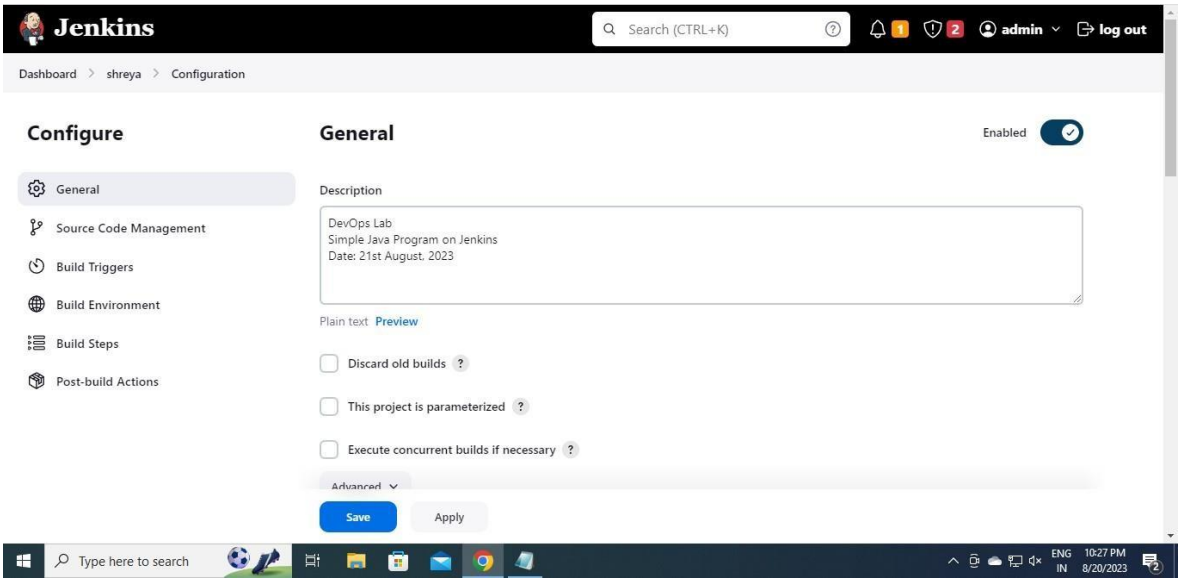
THEORY:
Building programs using Jenkins involves automating the development workflow, particularly for Java applications. Through Jenkins, developers create a structured process that automatically manages tasks such as code compilation, testing, and potential deployment.

When a Jenkins job is established, it constructs a framework that guides the progression of software development activities. Jenkins can be configured to monitor version control repositories, subsequently triggering a build each time code changes are committed. This preemptive identification of errors during automated builds aids in identifying and addressing issues before they propagate further within the development cycle.

Steps to build Java Programs using Jenkins:

1. Install Jenkins: Install Jenkins on the system.
2. Create Job: Make a new job in Jenkins.
3. Setup Code: Configure version control (e.g., Git) for your job.
4. Build Steps: Add steps to compile your Java program, e.g., use "Execute shell" with javac.
5. Post-Build: Optionally, add post-build actions like archiving artefacts.
6. Save and Build: Save the job and trigger a build.
7. Check Results: Review build results, including console output and test reports.
8. Test Steps: If applicable, add test steps using tools like JUnit.
9. Automate: Set up auto-build triggers for continuous integration.


localhost:8080/newJob
Gmail
YouTube
Maps
Dashboard
Enter an item name
shreya
» Required field
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
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.
Folder
OK
Type here to search
ENG
IN
10:26 PM
8/20/2023


Jenkins
Search (CTRL+K)
admin
log out
Dashboard
shreya
Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
General
Enabled
Description
DevOps Lab
Simple Java Program on Jenkins
Date: 21st August, 2023
Plain text
Preview
Discard old builds
This project is parameterized
Execute concurrent builds if necessary
Advanced
Save
Apply
Type here to search
ENG
IN
10:27 PM
8/20/2023

Dashboard > shreya > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

Execute Windows batch command

Command

See the list of available environment variables

```
javac Helloworld.java
java Helloworld
```

Advanced

Add build step

Post-build Actions

Add post-build action

Save Apply

Helloworld.java - Notepad

File Edit Format View Help

```
public class Helloworld
{
        public static void main(String[] args)
        {
                System.out.println("Hello World!");
        }
}
```

Ln 7, Col 2 | 100% | Windows (CRLF) | UTF-8

Jenkins Search (CTRL+K) admin log out

Dashboard > shreya >

- Status
- Changes
- Workspace
- Build Now
- Configure
- Delete Project
- Rename

Build History trend

No builds

shreya

DevOps Lab
Simple Java Program on Jenkins
Date: 21st August, 2023

Edit description

Disable Project

Permalinks

Jenkins

Search (CTRL+K)

admin log out

Dashboard > shreya > #2

- Status
- Changes
- Console Output
- Edit Build Information
- Delete build '#2'
- Previous Build

#2 (Aug 20, 2023, 10:33:34 PM)

Keep this build forever

Add description

Started 6.4 sec ago
Took 1.2 sec

No changes.

Started by user admin

REST API Jenkins 2.417

Type here to search

ENG IN 10:33 PM 8/20/2023

Jenkins

Search (CTRL+K)

admin log out

Dashboard > shreya > #2 > Console Output

- Status
- Changes
- Console Output
- View as plain text
- Edit Build Information
- Delete build '#2'
- Previous Build

Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\shreya
[shreya] $ cmd /c call C:\WINDOWS\TEMP\jenkins7542321336782794327.bat

C:\ProgramData\Jenkins\.jenkins\workspace\shreya>javac Helloworld.java

C:\ProgramData\Jenkins\.jenkins\workspace\shreya>java Helloworld
Hello World!

C:\ProgramData\Jenkins\.jenkins\workspace\shreya>exit 0
Finished: SUCCESS
```

REST API Jenkins 2.417

Type here to search

ENG IN 10:33 PM 8/20/2023

Jenkins

Search (CTRL+K)

admin log out

Dashboard > shreya >

- Status
- Changes
- Workspace
- Build Now
- Configure
- Delete Project
- Rename

Build History trend

Filter builds... /

#2
Aug 20, 2023, 10:33 PM

shreya

DevOps Lab
Simple Java Program on Jenkins
Date: 21st August, 2023

Edit description

Disable Project

Permalinks

- Last build (#2), 44 sec ago
- Last stable build (#2), 44 sec ago
- Last successful build (#2), 44 sec ago
- Last failed build (#1), 2 min 37 sec ago
- Last unsuccessful build (#1), 2 min 37 sec ago
- Last completed build (#2), 44 sec ago

Type here to search

ENG IN 10:34 PM 8/20/2023

Dashboard > shreya > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

Execute Windows batch command

Command

See the list of available environment variables

```
javac Helloworld.java
java Helloworld
javac Forloop.java
java Forloop
```

Advanced

Add build step

Post-build Actions

Add post-build action

Save Apply

Forloop.java - Notepad

File Edit Format View Help

```
public class Forloop{
        public static void main(String[] args){
                for(int i=0; i<5; i++){
                        System.out.println(i);
}
}
}
```

Ln 7, Col 2 | 100% | Windows (CRLF) | UTF-8

Jenkins | Search (CTRL+K) | admin | log out

Dashboard > shreya > #5

- Status
- Changes
- Console Output
- Edit Build Information
- Delete build '#5'
- Previous Build

#5 (Aug 20, 2023, 10:39:00 PM)

Keep this build forever

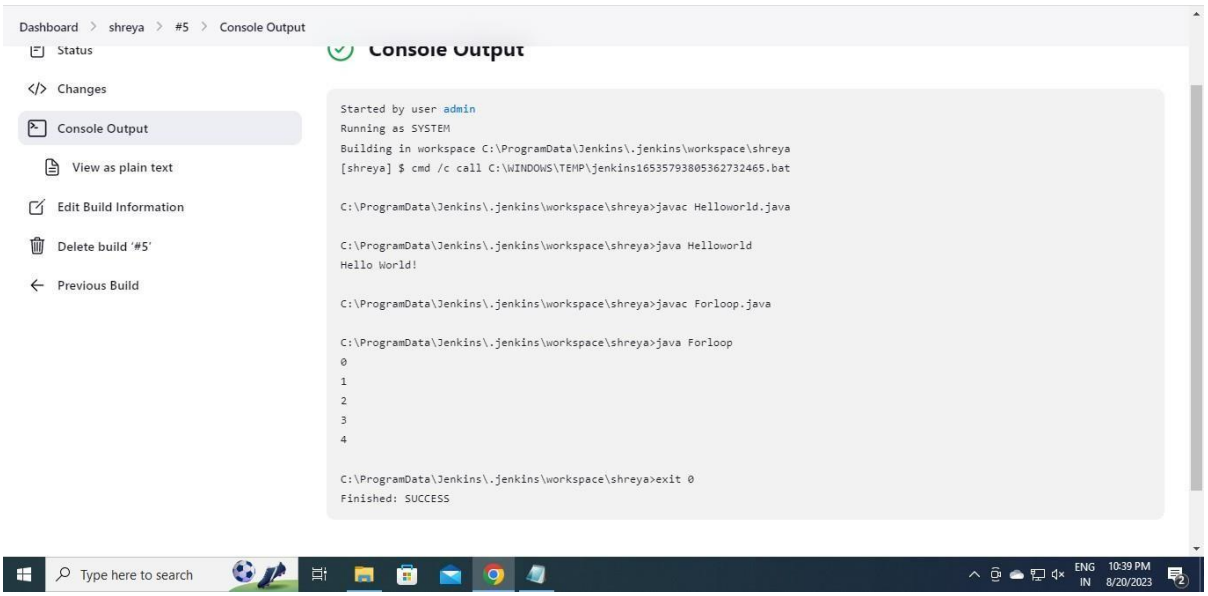
Add description

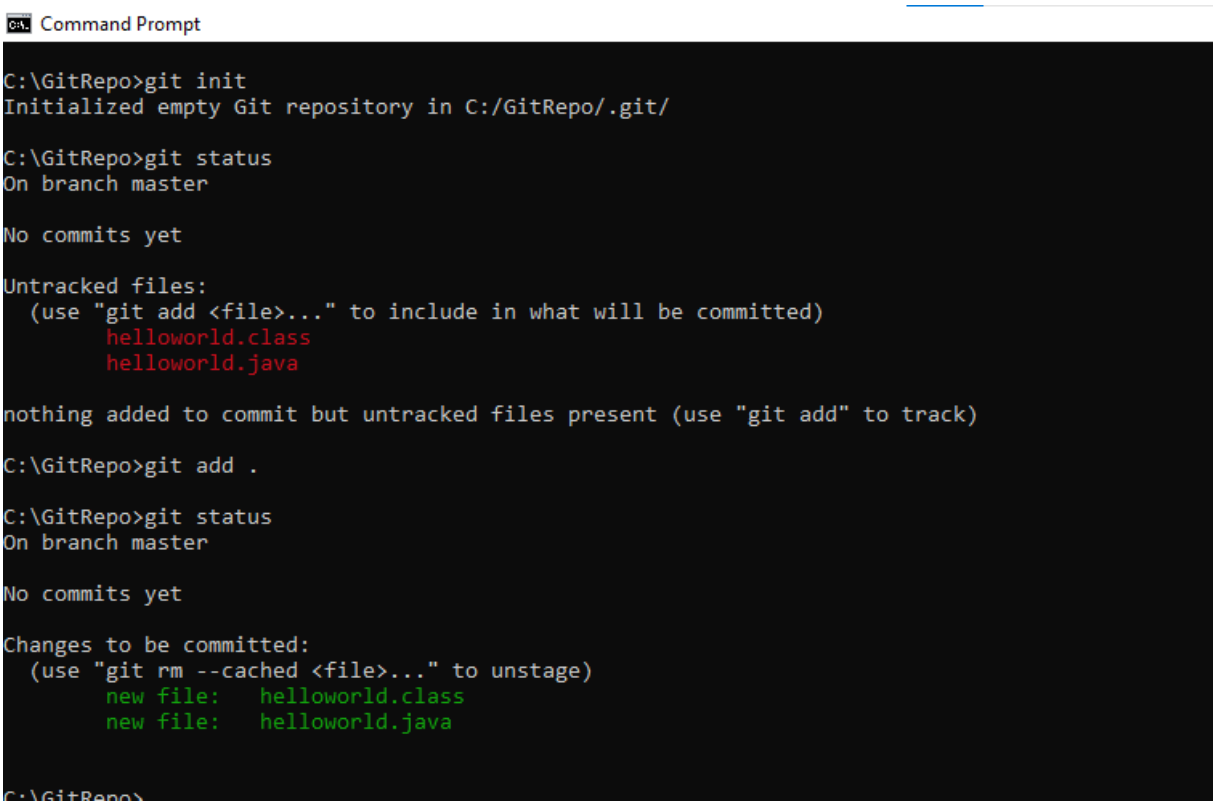
Started 6.4 sec ago
Took 1.3 sec

No changes.

Started by user admin

REST API Jenkins 2.417





CONCLUSION:

Hence, through this assignment, I have successfully built a Hello World and a program with for loops in Java using Jenkins.

6. Pipeline using Jenkins

AIM:
To build pipeline using Jenkins.

THEORY:
What is a Pipeline in Jenkins?
A Jenkins Pipeline is a set of automated processes that help developers and DevOps teams build, test, and deploy code in a continuous integration/continuous delivery (CI/CD) environment.
In simple terms, it's a workflow that defines how software moves from code to production in an automated and repeatable manner.

Key Benefits of Jenkins Pipeline:

- Automates CI/CD processes
- Provides better visibility into the build process
- Offers version control of build logic (Pipeline as Code)
- Supports complex workflows (parallel execution, conditional logic, etc.) Types of

Jenkins Pipelines:

There are two main types of syntax used to define Jenkins pipelines:

1. Declarative Pipeline

Simpler and more structured syntax
Designed for users who want an easy-to-read format
Introduced later to make pipelines more readable and maintainable Key Features:

- Uses pipeline {} block
- Organized into stages and steps
- Easier error handling
- Better validation in Jenkins UI

2. Scripted Pipeline

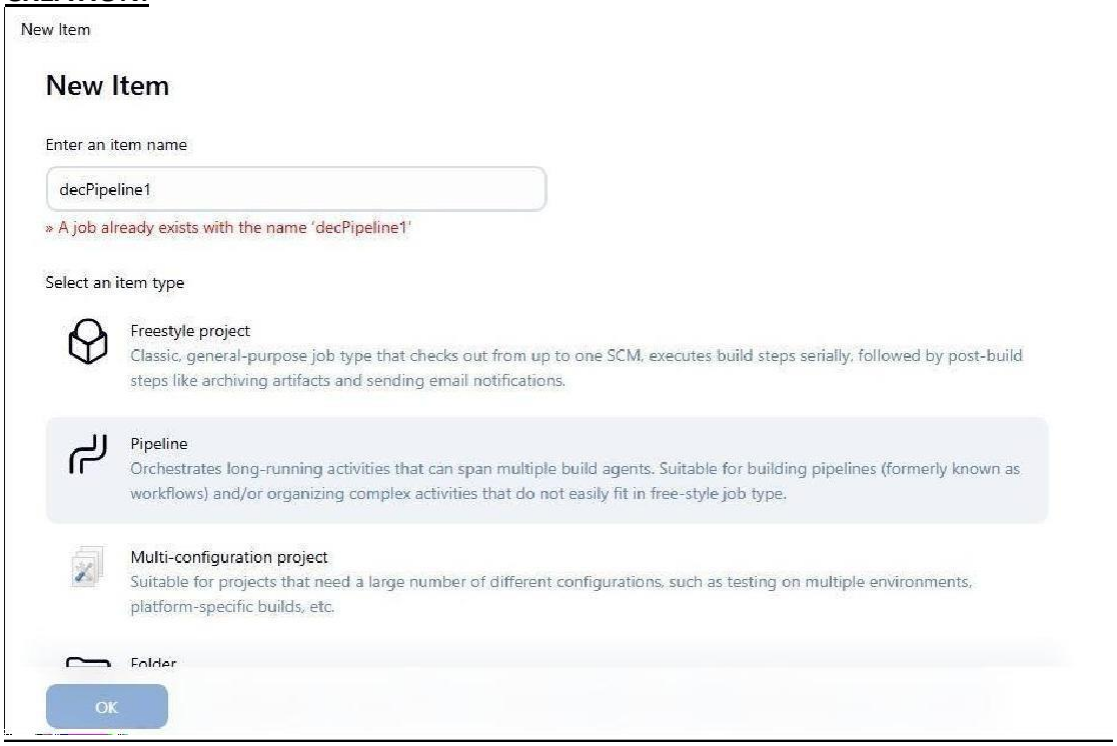
- More flexible and powerful
- Written in Groovy-based syntax
- Gives full control over the build process
- Suitable for advanced users Key

Features:

- Uses node {} block
- Everything is written as code
- Allows complex scripting and logic
- Less user-friendly for beginners

1. <u>Declarative Pipeline</u>

CREATION:



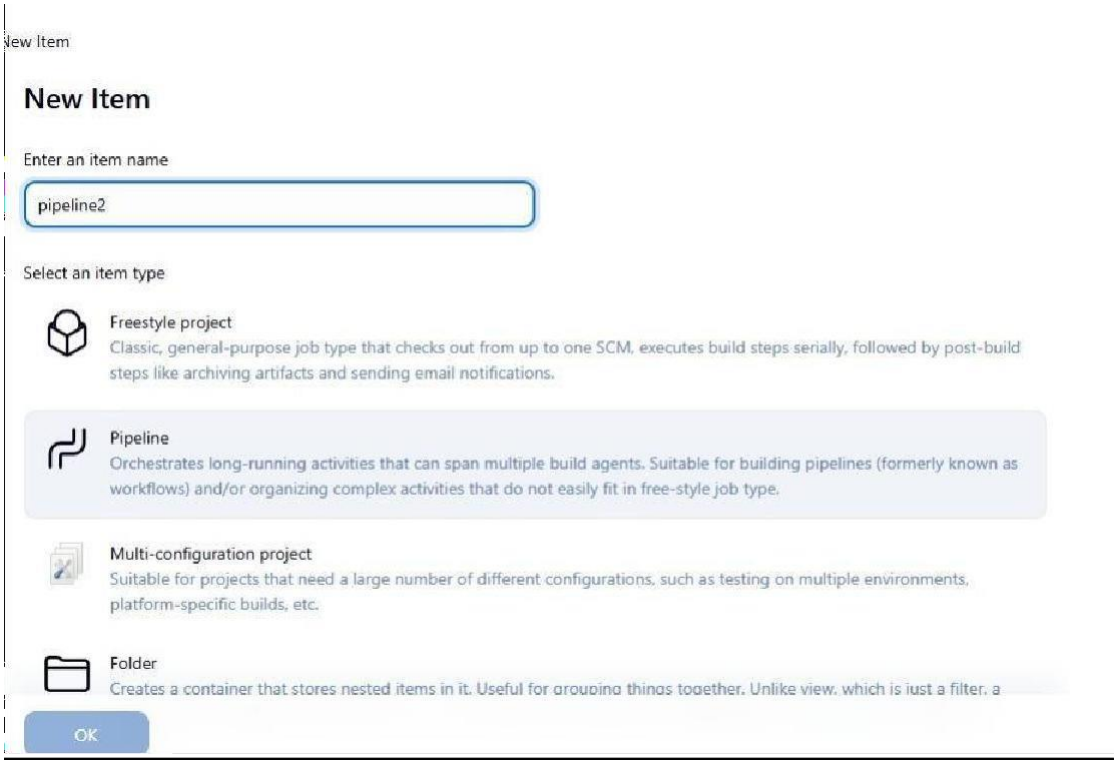
CODE:

```
pipeline {      agent any       stages {
stage('Build') {                steps {
echo 'Building..'
                }       }       stage('Test') {
    steps {                 echo 'Testing...'
            }       }       stage('Deploy') {
 steps {                 echo 'Deploying...'
            }
        }
    }
}
```

OUTPUT:

```
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\decPipeline1
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] echo
Building..
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo
Testing...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deploying...
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

2. <u>SCRIPTED</u> CREATION:



CODE:

```
node {
stage('Build') {
//      }
stage('Test') {
        //      }
 stage('Deploy') {
//
    }
}
```

OUTPUT:

```
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\pipeline2
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] echo
Building...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo
Testing...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deploying...
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

CONCLUSION: LO1, LO3

In this assignment, we learned how to create Jenkins Pipelines to automate the build, test, and deployment process. We explored both Declarative and Scripted pipelines, understanding their structure, benefits, and use cases. By implementing different pipeline examples, we demonstrated how Jenkins enables efficient, repeatable, and reliable CI/CD workflows.

LO MAPPING: LO1,LO3 Mapped.

7. Docker Installation

AIM:

To understand Docker architecture and container life cycle, install docker ,deploy container in Docker

THEORY:

Docker is an open-source platform that automates the deployment, scaling, and management of applications inside lightweight, portable containers. Containers allow developers to package an application with all its dependencies and run it reliably in different computing environments.

2. Docker Architecture

Docker's architecture consists of the following key components:

Docker Client: The command-line interface where users issue Docker commands.

Docker Daemon: A background service that handles building, running, and monitoring containers.

Docker Images: Immutable templates used to create containers, containing all application code and dependencies.

Docker Containers: Running instances of Docker images, isolated from each other and the host

system. Docker Hub: A public cloud repository where Docker images are stored and shared.

The Docker client communicates with the Docker daemon to execute commands. Images are pulled from Docker Hub (or other registries) to create containers that run applications in isolation.

3. Container Lifecycle

A container undergoes several stages during its lifecycle:

Create: A container is instantiated from a Docker image but not yet

running. Start: The container starts executing its processes.

Pause: Temporarily halts container processes.

Unpause: Resumes a paused container.

Stop: Gracefully stops a running container.

Kill: Forces immediate termination of the

container. Remove: Deletes the container from

the system.

Understanding this lifecycle helps in effectively managing containers.

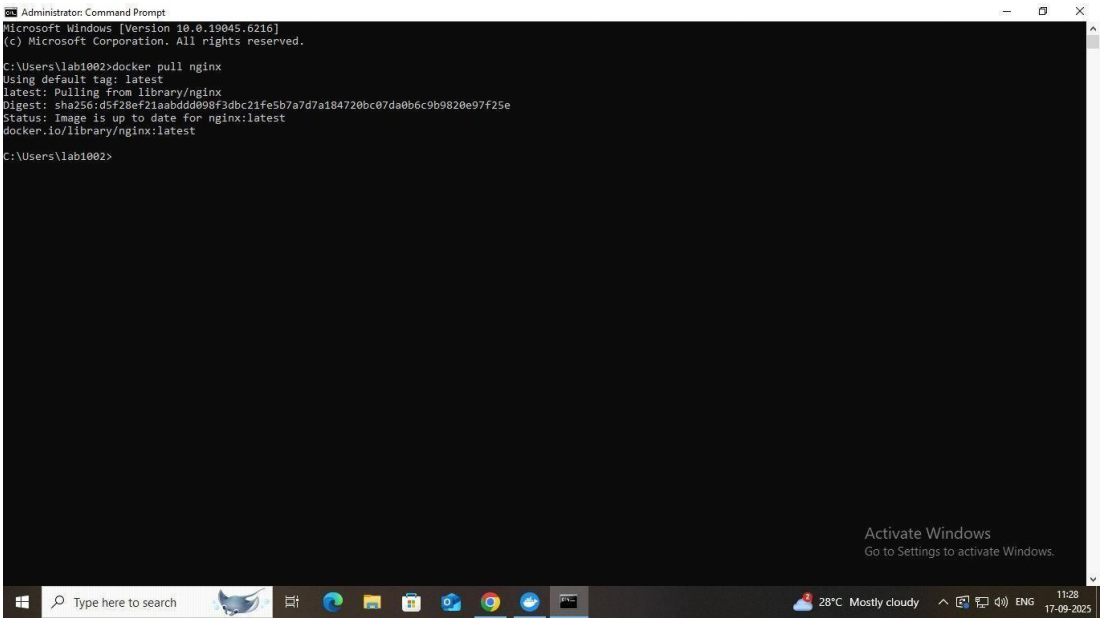
Deploying a Container in Docker

To deploy a container, following are the steps:

Step 1: Pull the Docker Image

Download the official NGINX image from Docker Hub by

running: docker pull nginx



Step 2: Run the Container

Create and start a container named mynginx with port mapping:

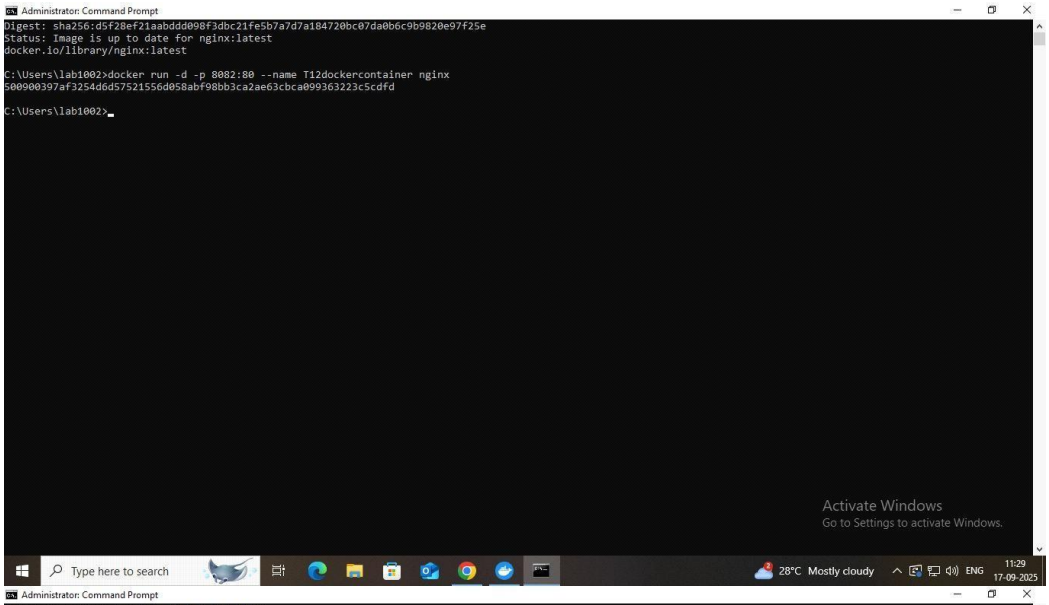
docker run -d -p 8081:80 --name mynginx nginx

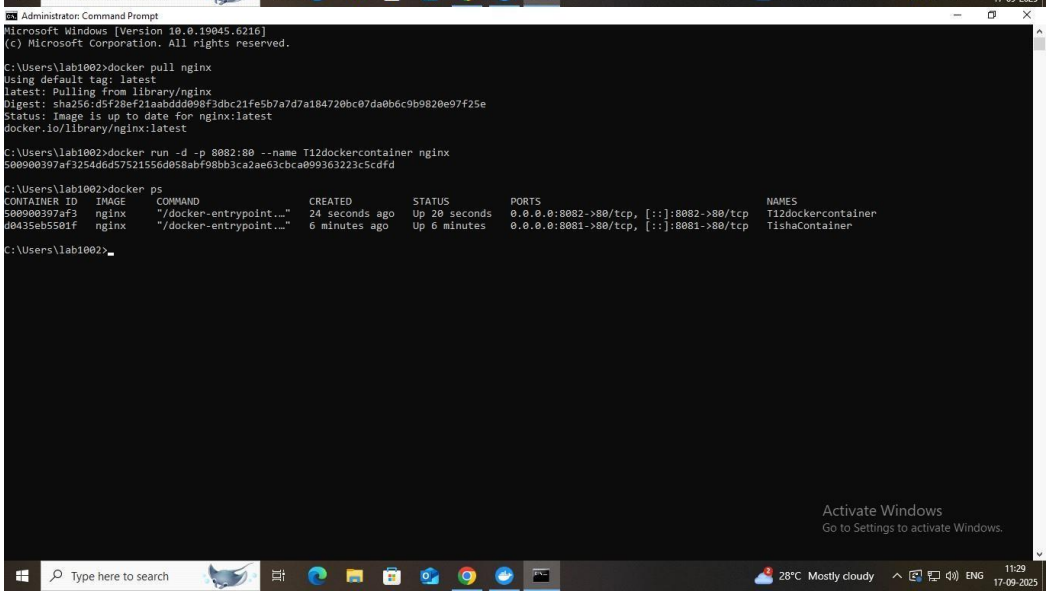
-d runs the container in detached mode.

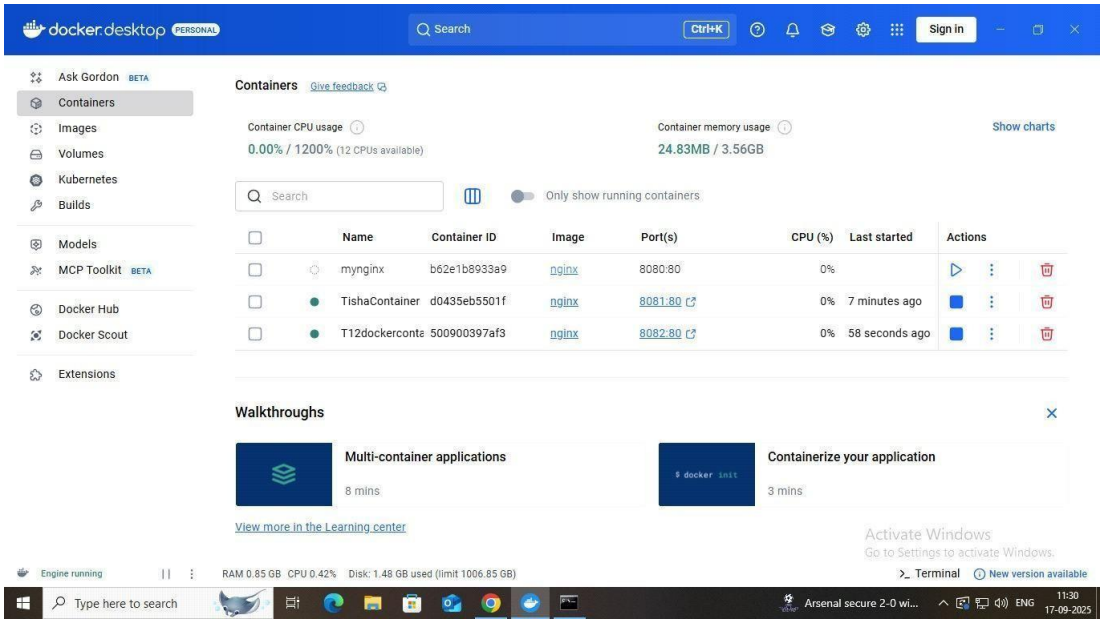
-p 8081:80 maps port 8081 on the host to port 80 in the container.

--name mynginx assigns a name to the container.

Batch / Roll_No – T13 /53







Step 3: Verify the Running Container

Check active containers with:

docker ps

Step 4: Access the Application

Open a web browser and navigate to:

http://localhost:8081

You will see the NGINX welcome page, indicating the container is running successfully.

CONCLUSION:

This assignment demonstrated the basic understanding of Docker's architecture and container lifecycle, as well as practical experience installing Docker and deploying a container. Docker containers provide an efficient and consistent environment to run applications, improving portability and scalability.

LO MAPPING: LO1, LO5 mapped

Name-Pratham Asnani
RollNo.-63
Batch-T13

Assignment 9

Aim-Installation of Nagios on Ubuntu System
Theory-

Introduction

Nagios is a powerful open-source monitoring system designed to monitor hosts, services, and network devices. It provides real-time monitoring, alerting, and reporting of system health and performance. Running Nagios on **Ubuntu**, a popular Linux distribution, allows system administrators to keep track of infrastructure efficiently using an open and flexible platform.

What is Nagios?

Nagios monitors systems, services, and network protocols by executing regular checks, either locally or remotely, and alerts administrators if any issues arise. It helps ensure the availability and performance of servers, applications, and network hardware.

Key features of Nagios:

- Monitoring of host resources: CPU, disk usage, memory, uptime.
- Monitoring network services: HTTP, SMTP, SSH, FTP, DNS, etc.
- Alerting via email, SMS, or custom scripts.
- Web-based interface for real-time monitoring.
- Extensible via plugins and add-ons.
- Supports both local and remote monitoring via agents like NRPE.

Why Use Nagios on Ubuntu?

Ubuntu is a widely-used Linux distribution known for stability and ease of use.

Installing Nagios on Ubuntu provides:

- A reliable and stable environment for monitoring.
- Compatibility with Ubuntu's package management and security features.
- Access to a rich ecosystem of tools, libraries, and plugins.
- Ease of administration via Ubuntu's command-line interface and web tools.

Steps to install

```
lab902@lab902-OptiPlex-390:~$ sudo apt update && sudo apt upgrade -y
[sudo] password for lab902:
Hit:1 https://download.docker.com/linux/ubuntu bionic InRelease
Hit:2 https://dl.google.com/linux/chrome/deb stable InRelease
Hit:3 http://in.archive.ubuntu.com/ubuntu bionic InRelease
Hit:4 http://in.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:5 http://in.archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:6 http://security.ubuntu.com/ubuntu bionic-security InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
4 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  docker-scan-plugin gir1.2-goa-1.0 gir1.2-snapd-1 linux-hwe-5.4-headers-5.4.0-107 linux-hwe-5.4-headers-5.4.0-109
  linux-hwe-5.4-headers-5.4.0-110 linux-hwe-5.4-headers-5.4.0-122 linux-hwe-5.4-headers-5.4.0-124 linux-hwe-5.4-headers-5
  linux-hwe-5.4-headers-5.4.0-126 linux-hwe-5.4-headers-5.4.0-128 linux-hwe-5.4-headers-5.4.0-131 linux-hwe-5.4-headers-5
  linux-hwe-5.4-headers-5.4.0-137 linux-hwe-5.4-headers-5.4.0-84
Use 'sudo apt autoremove' to remove them.
The following security updates require Ubuntu Pro with 'esm-infra' enabled:
```

```
lab902@lab902-OptiPlex-390:~$ sudo apt install -y apache2 php libapache2-mod-php build-essential libgd-dev unzip wget curl openss
daemon
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
build-essential set to manually installed.
curl is already the newest version (7.58.0-2ubuntu3.24).
openssl is already the newest version (1.1.1-1ubuntu2.1~18.04.23).
openssl set to manually installed.
unzip is already the newest version (6.0-21ubuntu1.2).
unzip set to manually installed.
wget is already the newest version (1.19.4-1ubuntu2.2).
wget set to manually installed.
The following packages were automatically installed and are no longer required:
  docker-scan-plugin gir1.2-goa-1.0 gir1.2-snapd-1 linux-hwe-5.4-headers-5.4.0-107 linux-hwe-5.4-headers-5.4.0-109
  linux-hwe-5.4-headers-5.4.0-110 linux-hwe-5.4-headers-5.4.0-122 linux-hwe-5.4-headers-5.4.0-124 linux-hwe-5.4-headers-5.4.0-125
  linux-hwe-5.4-headers-5.4.0-126 linux-hwe-5.4-headers-5.4.0-128 linux-hwe-5.4-headers-5.4.0-131 linux-hwe-5.4-headers-5.4.0-136
  linux-hwe-5.4-headers-5.4.0-137 linux-hwe-5.4-headers-5.4.0-84
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapache2-mod-php7.2 libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap
  libfontconfig1-dev libfreetype6-dev libjbig-dev libjpeg-dev libjpeg-turbo8-dev libjpeg8-dev liblzma-dev libpng-dev libpng-tools
  libtiff-dev libtiff5-dev libtiffxx5 libvpx-dev libxpm-dev php-common php7.2 php7.2-cli php7.2-common php7.2-json php7.2-opcache
  php7.2-readline pkg-config zlib1g-dev
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom php-pear liblzma-doc libssl-doc
The following NEW packages will be installed:
```

```
lab902@lab902-OptiPlex-390:~$ sudo useradd nagios
lab902@lab902-OptiPlex-390:~$ sudo groupadd nagcmd
lab902@lab902-OptiPlex-390:~$ sudo usermod -a -G nagcmd nagios
lab902@lab902-OptiPlex-390:~$ sudo usermod -a -G nagcmd www-data
lab902@lab902-OptiPlex-390:~$ cd /tmp
lab902@lab902-OptiPlex-390:/tmp$ wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.5.1.tar.gz
--2025-09-19 11:22:55--  https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.5.1.tar.gz
Resolving assets.nagios.com (assets.nagios.com)... 45.79.49.120, 2600:3c00::f03c:92ff:fef7:45ce
Connecting to assets.nagios.com (assets.nagios.com)|45.79.49.120|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11540195 (11M) [application/x-gzip]
Saving to: ‘nagios-4.5.1.tar.gz’

nagios-4.5.1.tar.gz                 100%[===================================================================>]  11.00M   114KB/s

2025-09-19 11:24:00 (178 KB/s) - ‘nagios-4.5.1.tar.gz’ saved [11540195/11540195]
```

```
lab902@lab902-OptiPlex-390:/tmp$ tar -xzf nagios-4.5.1.tar.gz
lab902@lab902-OptiPlex-390:/tmp$ cd nagios-4.5.1
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ ./configure --with-command-group=nagcm
checking for a BSD-compatible install... /usr/bin/install -c
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether make sets $(MAKE)... yes
checking whether ln -s works... yes
checking for strip... /usr/bin/strip
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files... yes
checking whether time.h and sys/time.h may both be included... yes
checking for sys/wait.h that is POSIX.1 compatible... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ make all
cd ./base && make
make[1]: Entering directory '/tmp/nagios-4.5.1/base'
gcc -Wall -I.. -I. -I../lib -I../include  -I../include -I..  -g -O2  -DHAVE_CONFIG_H -DNSCORE -c -o nagios.o ./nagios.c
./nagios.c: In function 'main':
./nagios.c:611:4: warning: ignoring return value of 'asprintf', declared with attribute warn_unused_result [-Wunused-result]
    asprintf(&mac->x[MACRO_PROCESSSTARTTIME], "%llu", (unsigned long long)program_start);
    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
./nagios.c:841:4: warning: ignoring return value of 'asprintf', declared with attribute warn_unused_result [-Wunused-result]
    asprintf(&mac->x[MACRO_EVENTSTARTTIME], "%llu", (unsigned long long)event_start);
    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
./nagios.c: In function 'nagios_core_worker':
./nagios.c:176:3: warning: ignoring return value of 'read', declared with attribute warn_unused_result [-Wunused-result]
   read(sd, response + 3, sizeof(response) - 4);
   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
./nagios.c: In function 'test_path_access':
./nagios.c:122:3: warning: ignoring return value of 'asprintf', declared with attribute warn_unused_result [-Wunused-result]
   asprintf(&path, "%s/%s", p, program);
   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
gcc -Wall -I.. -I. -I../lib -I../include  -I../include -I..  -g -O2  -DHAVE_CONFIG_H -DNSCORE -c -o broker.o broker.c
gcc -Wall -I.. -I. -I../lib -I../include  -I../include -I..  -g -O2  -DHAVE_CONFIG_H -DNSCORE -c -o nebmods.o nebmods.c
gcc -Wall -I.. -I. -I../lib -I../include  -I../include -I..  -g -O2  -DHAVE_CONFIG_H -DNSCORE -c -o ../common/shared.o ../commor
gcc -Wall -I.. -I. -I../lib -I../include  -I../include -I..  -g -O2  -DHAVE_CONFIG_H -DNSCORE -c -o query-handler.o query-handle
gcc -Wall -I.. -I. -I../lib -I../include  -I../include -I..  -g -O2  -DHAVE_CONFIG_H -DNSCORE -c -o workers.o workers.c
workers.c: In function 'handle_worker_result':
workers.c:801:4: warning: ignoring return value of 'asprintf', declared with attribute warn_unused_result [-Wunused-result]
    asprintf(&error_reason, "timed out after %.2fs", tv_delta_f(&wpres.start, &wpres.stop));
    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
workers.c:804:4: warning: ignoring return value of 'asprintf', declared with attribute warn_unused_result [-Wunused-result]
    asprintf(&error_reason, "died by signal %d%s after %.2f seconds",
    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
             WTERMSIG(wpres.wait_status),
             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
             WCOREDUMP(wpres.wait_status) ? " (core dumped)" : "",
             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
             tv_delta_f(&wpres.start, &wpres.stop));
             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ sudo make install
cd ./base && make install
make[1]: Entering directory '/tmp/nagios-4.5.1/base'
/usr/bin/install -c -m 775 -o nagios -g nagios -d /usr/local/nagios/bin
/usr/bin/install -c -s -m 774 -o nagios -g nagios nagios /usr/local/nagios/bin
/usr/bin/install -c -s -m 774 -o nagios -g nagios nagiostats /usr/local/nagios/bin
make[1]: Leaving directory '/tmp/nagios-4.5.1/base'
cd ./cgi && make install
make[1]: Entering directory '/tmp/nagios-4.5.1/cgi'
make install-basic
make[2]: Entering directory '/tmp/nagios-4.5.1/cgi'
/usr/bin/install -c -m 775 -o nagios -g nagios -d /usr/local/nagios/sbin
for file in *.cgi; do \
        /usr/bin/install -c -s -m 775 -o nagios -g nagios $file /usr/local/nagios/sb
done
make[2]: Leaving directory '/tmp/nagios-4.5.1/cgi'
make[1]: Leaving directory '/tmp/nagios-4.5.1/cgi'
cd ./html && make install
make[1]: Entering directory '/tmp/nagios-4.5.1/html'
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ sudo make install-init
/usr/bin/install -c -m 755 -d -o root -g root /lib/systemd/system
/usr/bin/install -c -m 755 -o root -g root startup/default-service /lib/systemd/system/nagios.s
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ sudo make install-commandmode
/usr/bin/install -c -m 775 -o nagios -g nagcmd -d /usr/local/nagios/var/rw
chmod g+s /usr/local/nagios/var/rw

*** External command directory configured ***
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ sudo make install-config
/usr/bin/install -c -m 775 -o nagios -g nagios -d /usr/local/nagios/etc
/usr/bin/install -c -m 775 -o nagios -g nagios -d /usr/local/nagios/etc/objects
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/nagios.cfg /usr/local/nagios/etc/nagios.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/cgi.cfg /usr/local/nagios/etc/cgi.cfg
/usr/bin/install -c -b -m 660 -o nagios -g nagios sample-config/resource.cfg /usr/local/nagios/etc/resource.cfg
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/templates.cfg /usr/local/nagios/etc/objects/temp
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/commands.cfg /usr/local/nagios/etc/objects/comma
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/contacts.cfg /usr/local/nagios/etc/objects/conta
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/timeperiods.cfg /usr/local/nagios/etc/objects/ti
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/localhost.cfg /usr/local/nagios/etc/objects/loca
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/windows.cfg /usr/local/nagios/etc/objects/window
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/printer.cfg /usr/local/nagios/etc/objects/printe
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/switch.cfg /usr/local/nagios/etc/objects/switch.

*** Config files installed ***
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ sudo make install-webconf
/usr/bin/install -c -m 644 sample-config/httpd.conf /etc/apache2/sites-available/nagios.conf
if [ 1 -eq 1 ]; then \
        ln -s /etc/apache2/sites-available/nagios.conf /etc/apache2/sites-enabled/nagios.conf;
fi

*** Nagios/Apache conf file installed ***
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
New password:
Re-type new password:
Adding password for user nagiosadmin
lab902@lab902-OptiPlex-390:/tmp/nagios-4.5.1$ cd /tmp
lab902@lab902-OptiPlex-390:/tmp$ wget https://nagios-plugins.org/download/nagios-plugins-2.3.3.tar.gz
--2025-09-19 11:25:29--  https://nagios-plugins.org/download/nagios-plugins-2.3.3.tar.gz
Resolving nagios-plugins.org (nagios-plugins.org)... 45.56.123.251
Connecting to nagios-plugins.org (nagios-plugins.org)|45.56.123.251|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2782610 (2.7M) [application/x-gzip]
Saving to: ‘nagios-plugins-2.3.3.tar.gz’

nagios-plugins-2.3.3.tar.gz         100%[===================================================================>]   2.65M  98.3KB/

2025-09-19 11:25:47 (163 KB/s) - ‘nagios-plugins-2.3.3.tar.gz’ saved [2782610/2782610]
```

```
lab902@lab902-OptiPlex-390:/tmp$ tar -xzf nagios-plugins-2.3.3.tar.gz
lab902@lab902-OptiPlex-390:/tmp$ cd nagios-plugins-2.3.3
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ ./configure --with-nagios-user=nagios --with-nagios-gr
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether to disable maintainer-specific portions of Makefiles... yes
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for Minix Amsterdam compiler... no
checking for ar... ar
checking for ranlib... ranlib
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ make
make  all-recursive
make[1]: Entering directory '/tmp/nagios-plugins-2.3.3'
Making all in gl
make[2]: Entering directory '/tmp/nagios-plugins-2.3.3/gl'
rm -f alloca.h-t alloca.h && \
{ echo '/* DO NOT EDIT! GENERATED AUTOMATICALLY! */'; \
  cat ./alloca.in.h; \
} > alloca.h-t && \
mv -f alloca.h-t alloca.h
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ sudo make in
Making install in gl
make[1]: Entering directory '/tmp/nagios-plugins-2.3.3/gl'
make  install-recursive
make[2]: Entering directory '/tmp/nagios-plugins-2.3.3/gl'
make[3]: Entering directory '/tmp/nagios-plugins-2.3.3/gl'
make[4]: Entering directory '/tmp/nagios-plugins-2.3.3/gl'
if test yes = no; then \
  case 'linux-gnu' in \
    darwin[56]*) \
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ sudo a2enconf nagios
ERROR: Conf nagios does not exist!
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ sudo a2enmod cgi
Enabling module cgi.
To activate the new configuration, you need to run:
  systemctl restart apache2
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ sudo systemctl restart apache2
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ sudo systemctl start nagios
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ sudo systemctl enable nagios
Created symlink /etc/systemd/system/multi-user.target.wants/nagios.service → /lib/systemd/system/nagios
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ curl ifconfig.me
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ curl -s https://ipinfo.io/ip
```

```
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ sudo ufw allow
Rules updated
Rules updated (v6)
lab902@lab902-OptiPlex-390:/tmp/nagios-plugins-2.3.3$ sudo ufw reload
Firewall not enabled (skipping reload)
```

172.17.0.1/nagios/

Nagios

General
- Home
- Documentation

Current Status
- Tactical Overview
- Map (Legacy)
- Hosts
- Services
- Host Groups
  - Summary
  - Grid
- Service Groups
  - Summary
  - Grid
- Problems
  - Services (Unhandled)
  - Hosts (Unhandled)
  - Network Outages

Quick Search:

Reports
- Availability
- Trends (Legacy)
- Alerts
  - History
  - Summary
  - Histogram (Legacy)
- Notifications
- Event Log

System
- Comments
- Downtime
- Process Info

Current Network Status
Last Updated: Fri Sep 19 11:45:03 IST 2025
Updated every 90 seconds
Nagios® Core™ 4.5.1 - www.nagios.org
Logged in as *nagiosadmin*

View Service Status Detail For All Host Groups
View Status Overview For All Host Groups
View Status Summary For All Host Groups
View Status Grid For All Host Groups

Host Status Totals

Up	Down	Unreachable	Pending
1	0	0	0

All Problems	All Types
0	1

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
7	0	0	1	0

All Problems	All Types
1	8

Host Status Details For All Host Groups

Limit Results: 100

Host	Status	Last Check	Duration	Status Information
localhost	UP	09-19-2025 11:41:10	0d 0h 17m 36s	PING OK - Packet loss = 0%, RTA = 0.06 ms

Results 1 - 1 of 1 Matching Hosts

172.17.0.1/nagios/

Nagios

General
- Home
- Documentation

Current Status
- Tactical Overview
- Map (Legacy)
- Hosts
- Services
- Host Groups
  - Summary
  - Grid
- Service Groups
  - Summary
  - Grid
- Problems
  - Services (Unhandled)
  - Hosts (Unhandled)
  - Network Outages

Quick Search:

Reports
- Availability
- Trends (Legacy)
- Alerts
  - History
  - Summary
  - Histogram (Legacy)
- Notifications
- Event Log

System
- Comments
- Downtime
- Process Info

Current Network Status
Last Updated: Fri Sep 19 11:46:39 IST 2025
Updated every 90 seconds
Nagios® Core™ 4.5.1 - www.nagios.org
Logged in as *nagiosadmin*

View History For all hosts
View Notifications For All Hosts
View Host Status Detail For All Hosts

Host Status Totals

Up	Down	Unreachable	Pending
1	0	0	0

All Problems	All Types
0	1

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
7	0	0	1	0

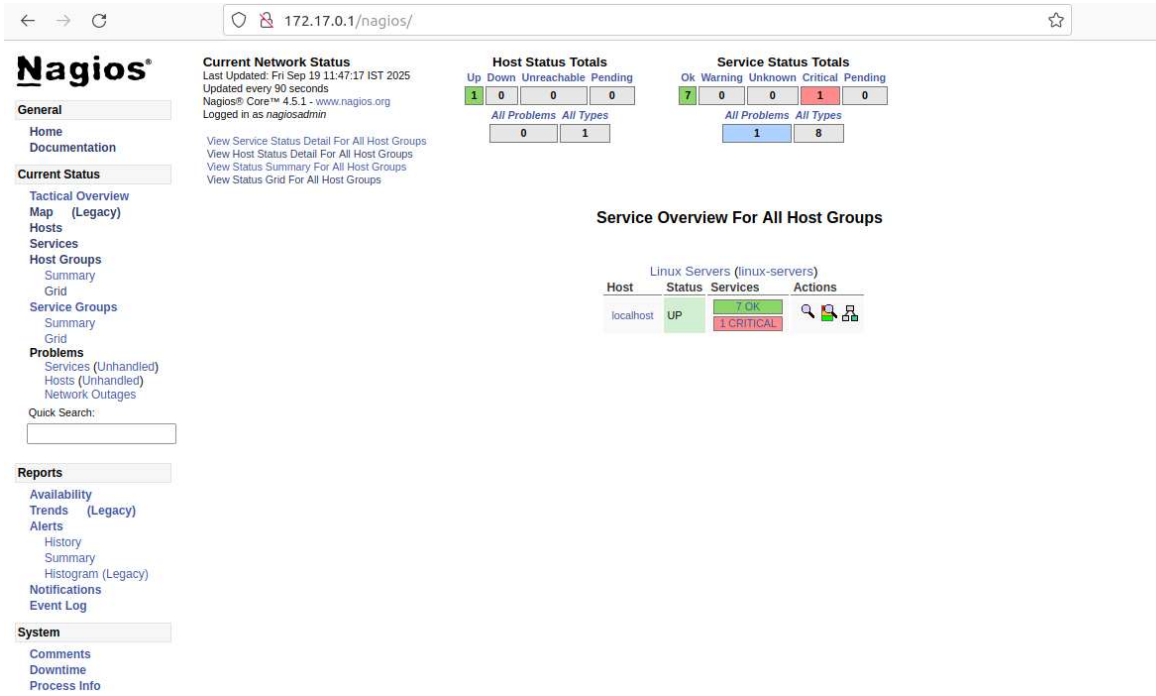
All Problems	All Types
1	8

Service Status Details For All Hosts

Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
localhost	Current Load	OK	09-19-2025 11:43:02	0d 0h 18m 37s	1/4	OK - load average: 2.06, 1.93, 1.62
	Current Users	OK	09-19-2025 11:43:40	0d 0h 17m 59s	1/4	USERS OK - 1 users currently logged in
	HTTP	OK	09-19-2025 11:44:17	0d 0h 17m 22s	1/4	HTTP OK: HTTP/1.1 200 OK - 11192 bytes in 0.001 second response
	PING	OK	09-19-2025 11:44:55	0d 0h 16m 44s	1/4	PING OK - Packet loss = 0%, RTA = 0.05 ms
	Root Partition	OK	09-19-2025 11:45:32	0d 0h 16m 7s	1/4	DISK OK - free space: / 418871 MiB (94.23% inode=98%):
	SSH	CRITICAL	09-19-2025 11:44:10	0d 0h 12m 29s	4/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap Usage	OK	09-19-2025 11:41:47	0d 0h 14m 52s	1/4	SWAP OK - 91% free (1856 MB out of 2047 MB)
	Total Processes	OK	09-19-2025 11:42:25	0d 0h 14m 14s	1/4	PROCS OK: 58 processes with STATE = RSZDT

Results 1 - 8 of 8 Matching Services



Conclusion-

Nagios on Ubuntu is a robust and flexible monitoring solution ideal for keeping track of IT infrastructure health. It helps system administrators maintain uptime, troubleshoot issues, and optimize system performance through detailed monitoring and alerting mechanisms. Ubuntu’s stability and user-friendliness make it an excellent platform to deploy Nagios.

Puppet : Case study

Aim

The aim of this case study is to explore Puppet as a configuration management tool that automates the deployment, configuration, and management of IT infrastructure. This study will illustrate how Puppet can reduce manual intervention, ensure system consistency, and enhance operational efficiency in managing servers and applications at scale.

Theory

What is Puppet?

Puppet is a powerful open-source configuration management and automation tool widely used in DevOps environments. It helps system administrators manage infrastructure as code, enabling automatic configuration and management of servers, applications, and services. Puppet allows you to define the desired state of your systems using a declarative language, ensuring that servers always conform to that state.

Key Concepts of Puppet

- **Manifests:** Puppet configurations are written in manifests using Puppet DSL. These files describe system resources such as packages, files, services, and users.

- **Modules:** These are collections of manifests and related files grouped for reusability and organization. Modules allow teams to share and reuse configurations easily.

- **Puppet Master and Puppet Agent:** Puppet operates using a master-agent architecture. The Puppet Master stores the manifests and modules, while Puppet Agents run on the managed nodes, applying configurations based on instructions received from the Master.

- **Idempotency:** Puppet applies configurations in an idempotent manner, meaning if a resource is already in the desired state, Puppet makes no changes, avoiding unnecessary actions and maintaining system stability.

How Puppet Works (Diagram 1)

```
+-----------------+          +-------------------+          +-----------------+
| Puppet Master   |  <------>  | Puppet Agent      | <-------> | Managed Node    |
| (Manifests &    |          | (Runs on Servers) |          | (Target Systems) |
```

```
| Modules stored)  |          |                |          |               |
+-----------------+          +-----------------+          +-----------------+
        |                    |                        |
        |                    |                        |
        +--------Configuration------> Applies Desired State         |
```

- **Step 1:** The Puppet Master holds all the configuration manifests.
- **Step 2:** Puppet Agents on target nodes periodically check in with the Puppet Master.
- **Step 3:** The Puppet Master compiles a catalog of resources that defines the desired state for each node.
- **Step 4:** Agents apply these configurations, ensuring the systems match the desired state.

Benefits of Using Puppet

- **Automation:** Eliminates repetitive manual configurations.
- **Consistency:** Ensures all systems are configured identically.
- **Scalability:** Manages hundreds or thousands of servers with minimal effort.
- **Flexibility:** Supports many operating systems and cloud platforms.
- **Version Control:** Configuration as code allows for versioning and auditing changes.

Case Scenario: Managing a Web Server Farm

Problem Statement

An organization manages a web server farm of 100 servers. Manually installing, configuring, and maintaining web servers, firewall rules, and application deployments is time-consuming and prone to errors.

Solution Using Puppet

- Create Puppet manifests defining:

    - Installation of Apache web server.
    - Configuration files for the web server.
    - User and permission settings.
    - Firewall rules.
- Use modules to package configurations for reuse.
- Set up Puppet Master to manage all 100 nodes.
- Puppet Agents installed on all servers check in regularly to apply configurations.

Outcome

- Deployment time reduced from days to hours.
- Configuration drift minimized; all servers remain consistent.
- Easy rollback and updates by changing Puppet manifests.
- Reduced manual errors and improved security compliance.

Diagram 2: Puppet Workflow in Web Server Farm

```
+---------------------+     1. Push Manifests      +-------------------+
| DevOps Team         | -------------------------> | Puppet Master     |
| Writes Puppet Code  |                            | (Central Repo)    |
+---------------------+                            +----------+---------+
                                                             |
                                                   2. Catalog Compilation
                                                             |
                                                   +-------v-------+
                                                   | Puppet Agents |
                                                   | (On 100 nodes)|
                                                   +-------+-------+
                                                             |
                                                   3. Configuration Applied
                                                             |
                                             +----------v----------+
                                             | Web Servers (Nodes)  |
                                             | Apache, Firewall etc.|
                                             +---------------------+
```

Conclusion

This case study illustrates that Puppet is a highly effective tool for managing infrastructure through automation and configuration as code. By defining the desired state of systems and automatically enforcing it, Puppet helps organizations reduce manual workload, improve consistency, and scale infrastructure management effortlessly.

For organizations with complex and dynamic IT environments, Puppet provides a robust solution that supports rapid deployment, consistent configuration, and simplified maintenance. Its master-agent architecture, idempotent operations, and modular design make it an essential tool in modern DevOps toolchains.

Pratham Asnani T13
63 Devops

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Written Assignment I

1) What is test automation? What are the advantages of test automation?

Test automation refers to the process of using software tools or scripts to automatically execute test cases on a software applications. The goal is to verify that the software behaves as expected and to identify bugs or issues quickly efficiently and reapeatedly

Types of test automation

1 Unit testing - testing individual components

2 Integration testing - testing combined part of the application

3 UI testing - Automatically brand

Advantages of test automation

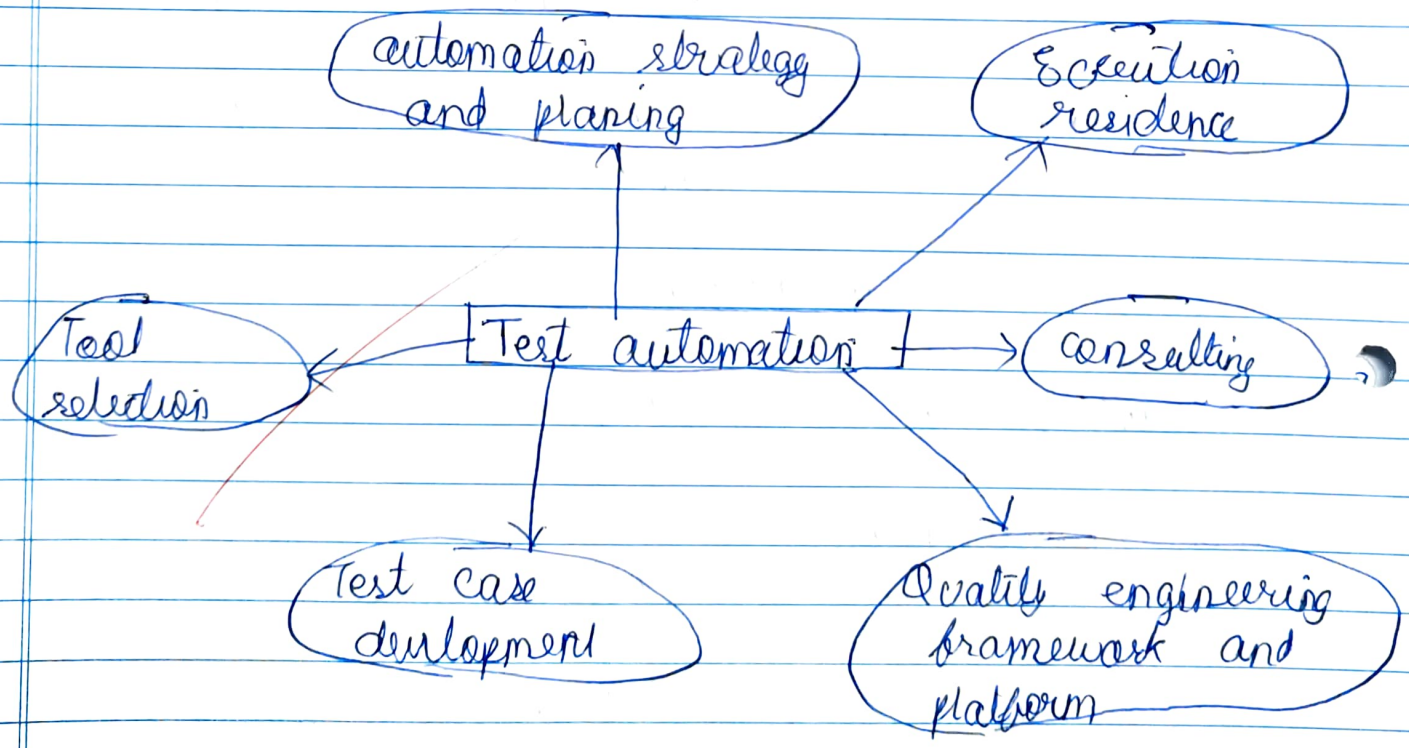
1 Faster execution
Automated test runs much faster than manual testing
Saves time during development and testing cycles

2 Increased test coverage
allows execution of hundreds or thousands test cases across multiple devices platforms and browse

3 Reusability of test scripts
Once written test scripts can be reused for different version of the software

4 Support continuous Integration delivery
essential for automated build and faster deployment

5 Improved accuracy
Reduces human error in repetitive tasks
Ensures consistent and repeatable test execution.



THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

2) What is X path ? Explain difference

X path (XML path language) is a query language used to navigate through elements and attributes in XML and HTML documents. It is widely used in selenium web devices and other automation framework to locate web elements
In web automation elements like button input field or links must be located precisely X path allows testers to allocate such elements
Thus position in the DOM
Thus attributes
Text content

Example code in HTML

<input type = "text", id="username", name"root">
we can use xpath to locate this elements using //* [@ id= "username"]
/- single slash
used when you want to start from the root node
It goes steps by step in hierarchy
fails if any link is incorrect
Selects <input> inside a specific hierarchy.

Double Slash //
Searches anywhere in the document
much more flexible and fault to learn
doesnt care about exact structure

when to use / or // :-
Situation
exact structure is known
testing fixed elements ~~tt~~ like

Pratham Asnani Devops
63 T13

(A) 11/9/25

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Written Assignment - 2

Q1 What is selenium? What are selenium components?

Selenium is an open source automated testing framework used to validate web application across different browsers and platform
It allows testers to write test scripts in various programming language like Java, Python, C#, Ruby and Java Script to automate browsers actions like clicking, typing, navigation or verifying UI elements

Selenium Components
Selenium is not a single tool but a suite of tool, each serving a different purpose

1 Selenium IDE
A browser extension (Chrome / Firefox)
Record and playback tool for quickly creating test scripts
Best for beginners or quick prototyping

2 Selenium RC (Remote control)
Allowed writing test cases in various languags
Has been replaced by Selenium Web Drivers

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

3 Selenium WebDrivers
Most widely used component
Provides API's to interact directly with browsers
Supports multiple browsers
More flexible and faster than Selenium IDE

4 Selenium Grid
Used for parallel test execution across different machines and browsers
Saves time by running multiple tests simultaneously

Q2 What makes Selenium such a widely used?

Selenium is extremly popular because of these advantages

1 Open Source and Free - No license cost

2 Cross-browser Support - Works with chrome Firefox, Edge, Safari

3 Cross platform Support - Runs on windows linux, mac

4 Multilanguage Support - Write tests in Java, Python, C#, Ruby, Java Script

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

5 Integration with tools - Works with Test NG, Junits, Maven, Jenkins, Docker

6 Parallel execution - Selenium Grid saves time.

7 Large community support - A active community for troubleshooting

Q Why Select Selenium for Web Application or System Testing

It is advised to select selenium for web testing becaused

1 Free and open source reducing project cost.
2 Support all major browsers - ensure compatibility testing
3 Runs on multiple operating system - ensures system wide coverage
4 Flexible integration with Devops and CI/CD pipelines - suitable for Agile environments.
5 Support parallel and distributed execution faster test cycles
6 Highly extendable with frameworks like Cucumber, TestNG, JUnit
7 Strong community and documentation easy to find solution.