

SRI BALAJI CHOCKALINGAM ENGINEERING COLLEGE

A.C.S Nagar(Irumbedu), Arni,

T.V.Malai Dt.-632 317.



*Department
Of
Information Technology*

SB8050 - DEVOPS (NAAN MUDHALVAN)



SRI BALAJI CHOCKALINGAM ENGINEERING COLLEGE

A.C.S Nagar(Irumbedu), Arni, T.V.Malai Dt.-632 317.

*Department
of
Information Technology*

BONAFIDE CERTIFICATE

Certified that this is a bonafide record of work done by.....

*Of Final Year / VII Semester **B.Tech Information Technology** in the Anna University Practical Examination during the year 20 - 20 in **SB8050 DEVOPS (NAAN MUDHALVAN)**.*

Register No.:

--	--	--	--	--	--	--	--	--	--	--	--

Staff In-Charge

Head of the Department

Submitted for Practical Examination held on

Internal Examiner

External Examiner

TABLE OF CONTENTS

Sno.	Date	Content	Pg.no.	Signature
1		Introduction to Agile Methodology		
2		Continuous Integration and Continuous Delivery		
3		Building of CI-CD Pipelines in Jenkins		
4		Installation of Jenkins		
5		Installation of SonarQube		
6		Installation of Eclipse IDE		
7		Calculator App in Java using Eclipse IDE		

Exp no: 1

Introduction to Agile Methodology

Date:

AIM:

The aim is to understand the fundamentals of Agile methodology, including its values, principles, and frameworks, and to compare it with traditional software development models.

SOFTWARE REQUIREMENTS:

- No specific software tools are required, as this experiment is theoretical.
- Access to course materials on the Infosys Springboard program.
- Optionally, Agile practice tools like Jira or Trello may be used for demonstration purposes if available.

PROCEDURE:

1. Course Introduction

- The course outline was reviewed to understand the topics covered on Agile methodology and software engineering basics.
- The educators in the course videos were introduced, with their areas of expertise and teaching approaches noted.



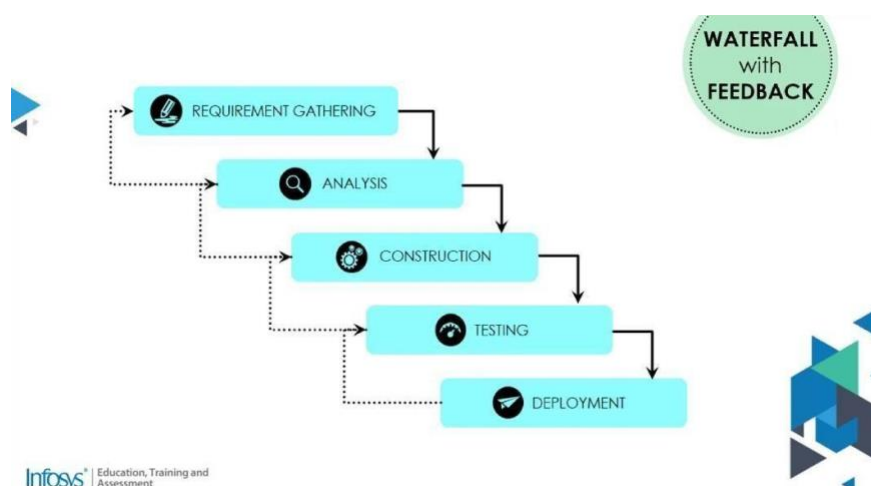
2. Software Engineering Basics

- The Need for Software Engineering was studied to understand the purpose and activities involved in creating reliable software.
- The concept of a software crisis was examined, which led to the development of software engineering.
- Various traditional SDLC (Software Development Life Cycle) models were reviewed, and the roles involved in a software team were discussed.



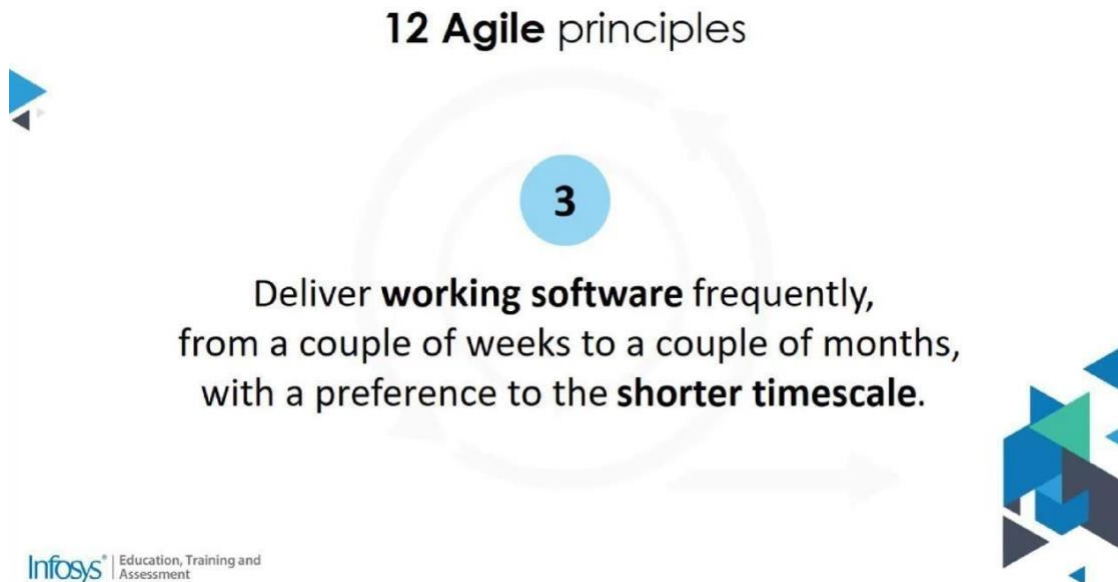
3. Traditional Software Engineering Models

- Different SDLC models were analyzed, including the Waterfall, Prototyping, Spiral, and Rapid Application Development (RAD) models.
- A self-assessment quiz on the Waterfall Model was completed to assess understanding.



4. Agile Manifesto

- The reasons for the rise of the Agile approach in IT were explored, with emphasis on its four core values and 12 guiding principles.
- Understanding was reinforced through a quiz on Values and Principles of Agility.

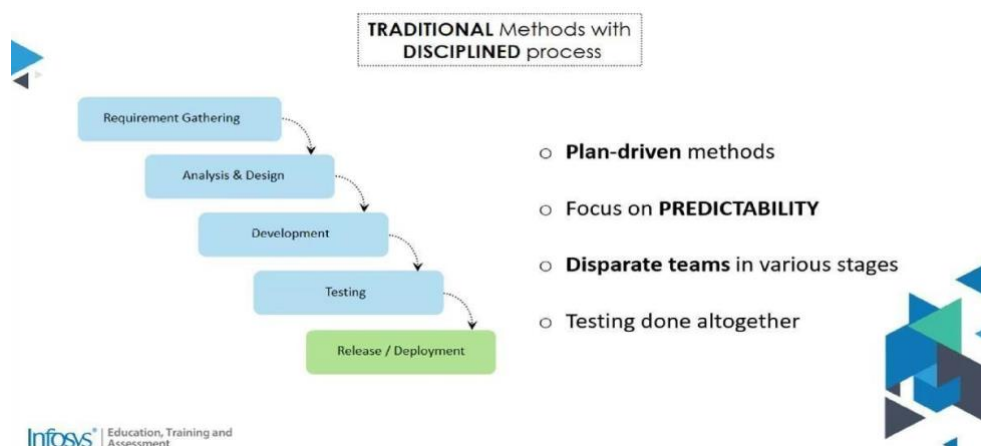


5. Agile Frameworks

- Several Agile frameworks, including Scrum and Kanban, were studied to understand how Agile can be adapted to various project needs.
- FAQs on Agile methods were reviewed, providing insights on applying each framework based on different project requirements.

6. Traditional vs Agile

- A comparison between traditional SDLC methods and Agile was conducted, focusing on how Agile evolved and the flexibility it offers, especially for rapid delivery.



7. Gamification of Scrum and Kanban Activities

- Practical demonstrations of Agile principles were observed through activities such as the Agility in Sprints - Airplane Activity and Kanban Coin Activity.



8. Summary

- A concluding quiz and assessment were completed to validate the understanding of Agile methodology concepts.

THEORY AND APPLICATION:

Agile methodology is a software development approach that emphasizes flexibility, customer collaboration, and incremental delivery. Instead of following rigid planning, Agile prioritizes responding to changes, allowing software to be produced more quickly and to adapt to client needs. The Agile Manifesto introduced four values: prioritizing individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and adapting to change over following a strict plan.

In practical application, Agile frameworks such as Scrum and Kanban are widely utilized. Scrum operates with iterative sprints, each delivering a potentially shippable product increment. Kanban, on the other hand, visualizes work and manages flow by limiting work-in-progress, promoting continuous improvement.

RESULT:

Thus the understand the fundamentals of Agile methodology, including its values, principles, and frameworks, and to compare it with traditional software development models verified successfully

Exp no: 2 Continuous Integration and Continuous Delivery

Date:

AIM:

The aim is to understand the principles of Continuous Integration and Continuous Delivery (CI/CD) in DevOps, including the technology, people, and process aspects, along with practical implementation using Java and an open-source tool stack.

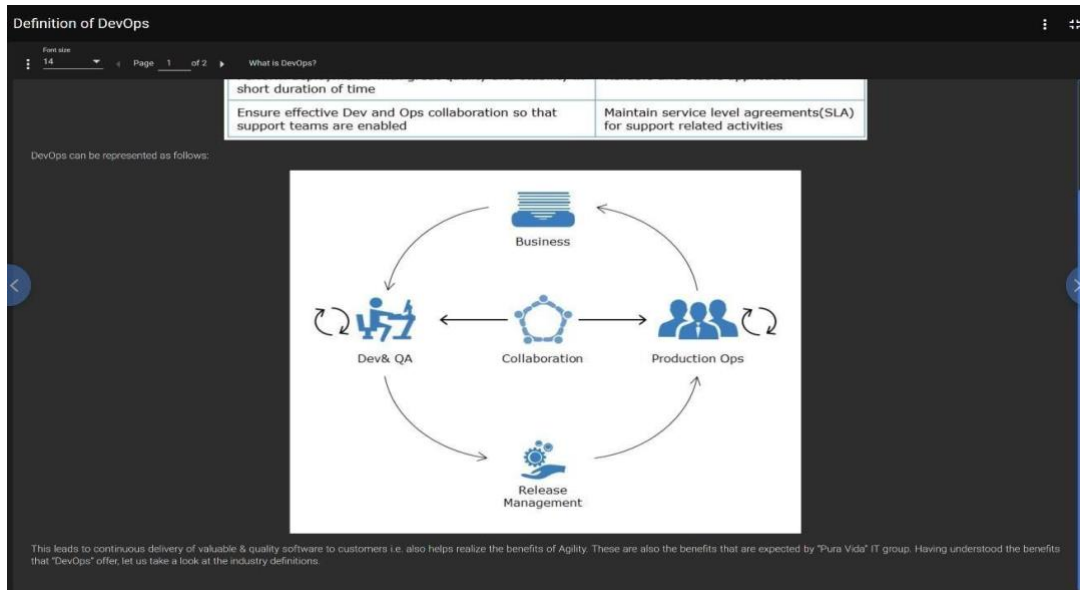
SOFTWARE REQUIREMENTS:

- Java Development Kit (JDK)
- Git for version control
- SonarQube for static code analysis
- JUnit for unit testing and code coverage
- Maven for build automation
- Docker for containerization
- A CI/CD tool (e.g., Jenkins)

PROCEDURE:

1. Prelude

- Pre-requisites for the course were reviewed, including the background knowledge required.
- The learning outcomes were noted, providing an overview of what would be achieved in this course.
- A case study was introduced, which would be referenced throughout the course to contextualize DevOps concepts.



2. Fundamentals of DevOps

- Definitions of DevOps were explored, including popular industry perspectives on the concept.
- A quiz was completed to evaluate understanding of DevOps fundamentals.

Aligning capabilities

Page 15 of 17

Aligning capabilities - Operations team

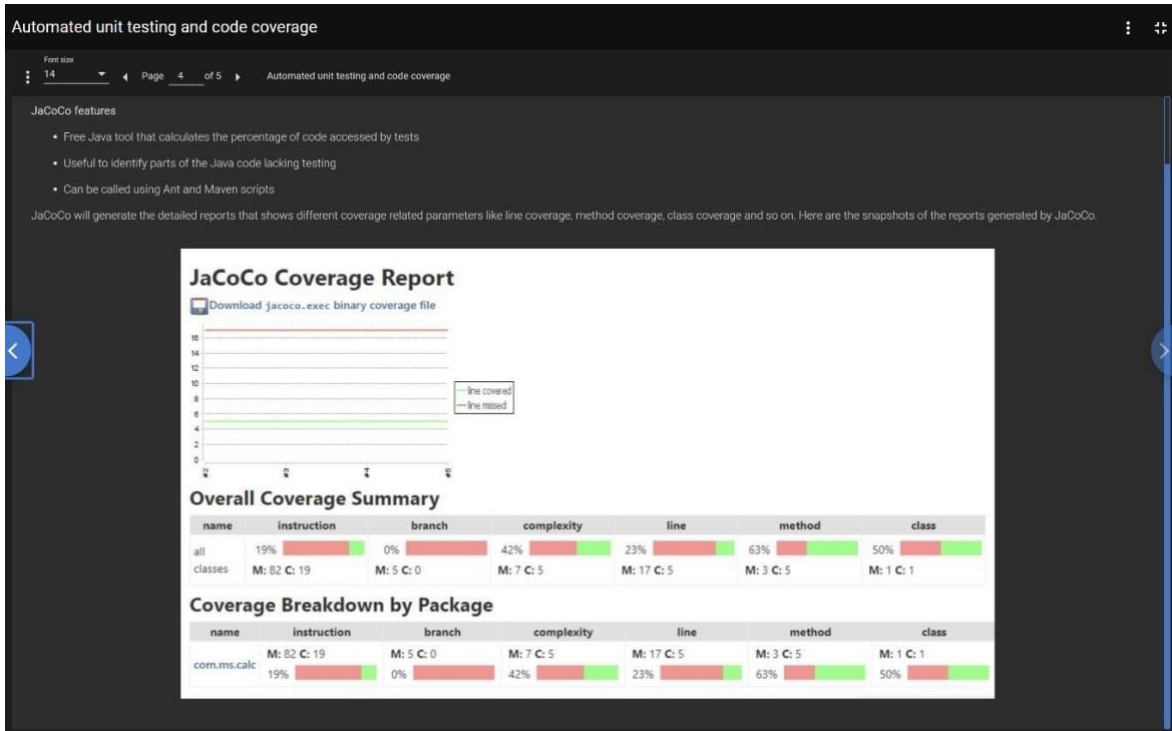
Business	Dev	Test	Infra	Ops
Acceptance Testing	Version Control	Functional / Acceptance Test	Database Deployment	Incident Management Tools
Rapid prototyping	Static / Dynamic Code Quality Analysis	Test and Defect Management	Infrastructure Layer	Support Analytics
Agile	Code Coverage	Test Data Management	Environment Management	Monitoring and Dashboard
Big Room Planning	Code Review	Service Virtualization	CD Automation	Predictive Monitoring
Lean	Unit Testing	Performance Testing	Release Management	Self Healing
	Build Automation	Security Testing	Containerization	
	Baseline in Artifact Repository	Progressive Test Automation	On demand/ Cloud based Infra	
	Continuous Integration		Infra-as-code	
	Agile		Just enough Infra	
	Feature Toggle			
	Incremental Design			
	Micro Services			

☒ Minimum Capabilities / Practices
☐ Good to have Capabilities / Practices
☐ Policies / Procedure / Methods

3. Devops Adoption in Projects

- The technology aspects of DevOps adoption were introduced, covering the tools and infrastructure required.
- Capabilities needed for successful DevOps adoption were aligned with project requirements.
- A quiz was taken to assess knowledge of these aligning capabilities.

- Various tool stacks and their implementation in DevOps were studied.
- The orchestration steps involved in Continuous Integration and Continuous Delivery were examined, followed by a quiz on CI/CD concepts.
- People and process aspects were discussed, focusing on the roles and responsibilities of team members in a DevOps environment.



4. Implementation of CI/CD with Java and Open Source Stack

- The open-source tool stack for creating a CI/CD pipeline was introduced.
- Version control was explained through the use of Git.
- Static code analysis was implemented using SonarQube.
- Automated unit testing was set up using JUnit, and code coverage tools were introduced.
- Build automation was explained, using Maven to streamline the build process.
- The purpose of an artifact repository was discussed.
- The orchestration of build activities to create a continuous integration pipeline was demonstrated.
- Dynamic environment provisioning and management were covered, along with their role in a CI/CD pipeline.
- Release management steps were outlined for an automated CI/CD pipeline.

- The process of continuous delivery and deployment to staging and production environments was explored.
- Containerization was explained to demonstrate its role in hassle-free deployment across multiple environments.
- The concept of gating in a CI/CD pipeline was introduced, describing types and methods of gating conditions.

Metrics to track CICD practices

Font size: 14 Page: 3 of 3 Metrics for measurement and optimization

Here are the common metrics that are tracked for continuous delivery and deployment. These are provided by the various tools which are used for automating the pipeline.

Focus Area	Metric	Formula	Proposed Usage
Release frequency	Deploys / Day	Total # of deployments / # of days	To understand increased collaboration & keep a track of time based improvement
	Releases / Week	Total # of releases / week (by environments)	Improvement in production release agility over time
Agility	Change Lead Time	Time required to release a change in production from the time the development begins on the change request/ development is complete	To know the trend in lead time & help remove any bottlenecks if a degradation is observed
Quality	Code Coverage	% of code covered through the test cycle, (Unit/Regression test)	To know the testing effectiveness through the developed code
	Release Efficiency/ repeatability	% of successful releases over a period of time	Helps measure effectiveness of CD setup & observe trend over a period of time
	Production Defect Matrix	Average # of defects released into production	Helps measure effectiveness of quality control through continuous delivery setup
	Lead time provisioning	Time taken for provisioning	To know the trend in lead time & help remove any bottlenecks if a degradation is observed
Performance	Availability	% availability of servers	Determine stability of environments across the enterprise
	Outage	Outage per day / week / month	To measure availability against SLA
	Capacity	% capacity availability & utilization	Gauge adequate capacity planning & use
Reliability	Mean Time to Recover	Time required to recover from outages	Improvement in MTTR over the time
Efficiency	Cost or effort per release	Initial effort/cost per release compared to post DevOps	To understand how efficient release process is

*MTTR - Mean Time To Recovery

5. Metrics and Measurement

- Common metrics used to evaluate the progress of an automated CI/CD pipeline were explained, focusing on tracking efficiency and success rates.

Metrics to track CICD practices

Font size: 14 Page: 2 of 3 Metrics for measurement and optimization

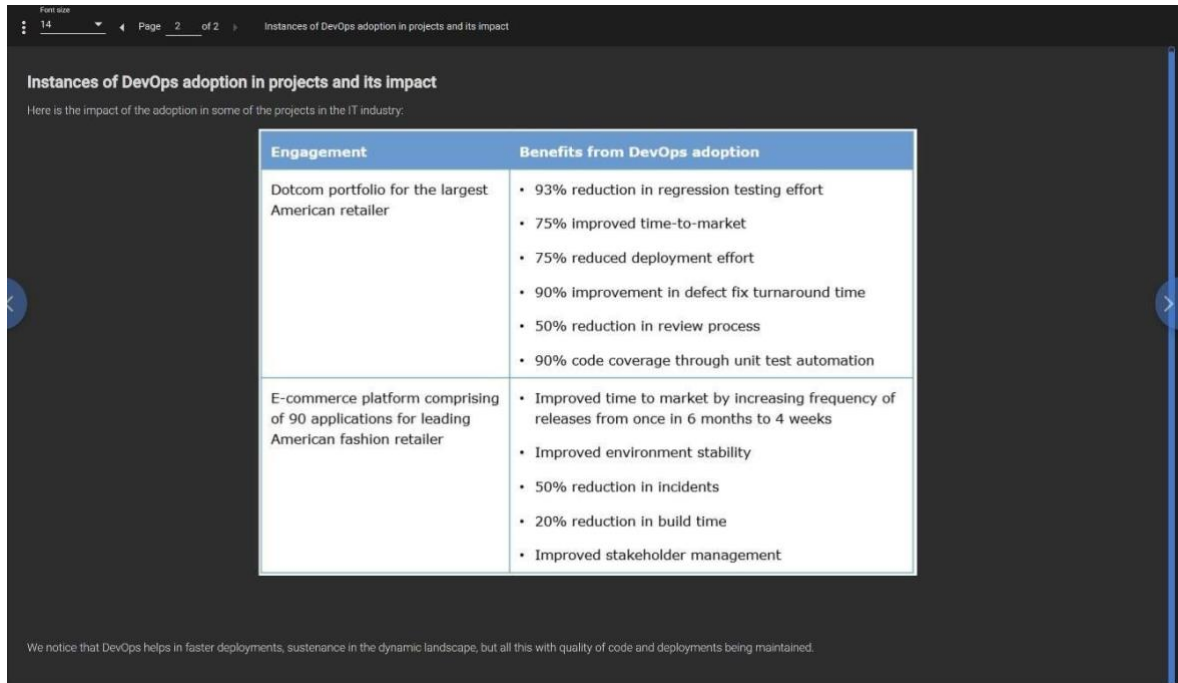
Unified metrics need to be provided to the teams with a common goal identified. This will ensure collaboration between the teams for achieving a common goal of speed.

Here are the common metrics that are tracked for continuous integration. These are provided by the various tools which are used for automating the pipeline.

#	Metric	Purpose
1.	Number of build failures	<ul style="list-style-type: none"> Qualitative analysis of CI process output Help plan improvements to increase build success %
2.	Total time taken for the build	<ul style="list-style-type: none"> Track build processing time which includes compilation, inspecting, testing and deployment Helps in making builds more efficient for faster execution & early feedback
3.	Unit test coverage	<ul style="list-style-type: none"> Track efficiency of test driven development
4.	Unit test success rate	<ul style="list-style-type: none"> Helps ensure code quality
5.	Rule compliance/violations	<ul style="list-style-type: none"> Track non-compliance against set coding conventions
6.	Code complexity	<ul style="list-style-type: none"> Helps developers write better quality code
7.	Duplicate lines of codes	
8.	Percentage of API documented	<ul style="list-style-type: none"> Track code documentation compliance Helps developers provide better understanding of code

6. Practice Exercises

- Practice exercises were provided as a guide, with step-by-step instructions for constructing an automated continuous integration pipeline.
- The course concluded with a self-assessment to evaluate the understanding of continuous integration and delivery concepts in DevOps.



Font size: 14 | Page: 2 of 2 | Instances of DevOps adoption in projects and its impact

Instances of DevOps adoption in projects and its impact

Here is the impact of the adoption in some of the projects in the IT industry:

Engagement	Benefits from DevOps adoption
Dotcom portfolio for the largest American retailer	<ul style="list-style-type: none">• 93% reduction in regression testing effort• 75% improved time-to-market• 75% reduced deployment effort• 90% improvement in defect fix turnaround time• 50% reduction in review process• 90% code coverage through unit test automation
E-commerce platform comprising of 90 applications for leading American fashion retailer	<ul style="list-style-type: none">• Improved time to market by increasing frequency of releases from once in 6 months to 4 weeks• Improved environment stability• 50% reduction in incidents• 20% reduction in build time• Improved stakeholder management

We notice that DevOps helps in faster deployments, sustenance in the dynamic landscape, but all this with quality of code and deployments being maintained.

THEORY AND APPLICATION:

DevOps is an approach that combines software development (Dev) and IT operations (Ops), emphasizing collaboration, automation, and iterative delivery. CI/CD is a key component of DevOps, ensuring that code changes are continuously integrated, tested, and deployed to production environments. CI/CD pipelines automate much of the software delivery process, allowing for more reliable and frequent releases.

A CI/CD pipeline typically involves several steps, including version control, static code analysis, automated testing, build automation, and deployment. Tools like Git (for version control), SonarQube (for code quality analysis), JUnit (for testing), Maven (for build automation), and Docker (for containerization) are widely used in CI/CD implementations.

RESULT:

Thus the understand the principles of Continuous Integration and Continuous Delivery (CI/CD) in DevOps, including the technology, people, and process aspects, along with practical implementation using Java and an open-source tool stack verified successfully.

Exp no: 3

Building of CI-CD Pipelines in Jenkins

Date:

AIM:

The aim of this experiment is to build an understanding of Continuous Integration and Continuous Deployment (CI/CD) pipelines using Jenkins, along with their benefits and application in DevOps.

SOFTWARE REQUIREMENTS:

- Jenkins

PROCEDURE:

Step 1: Login into your Jenkins account as shown below.

Step 2. Once logged in, the user will be redirected to the Jenkins console, here's the reference for the same.



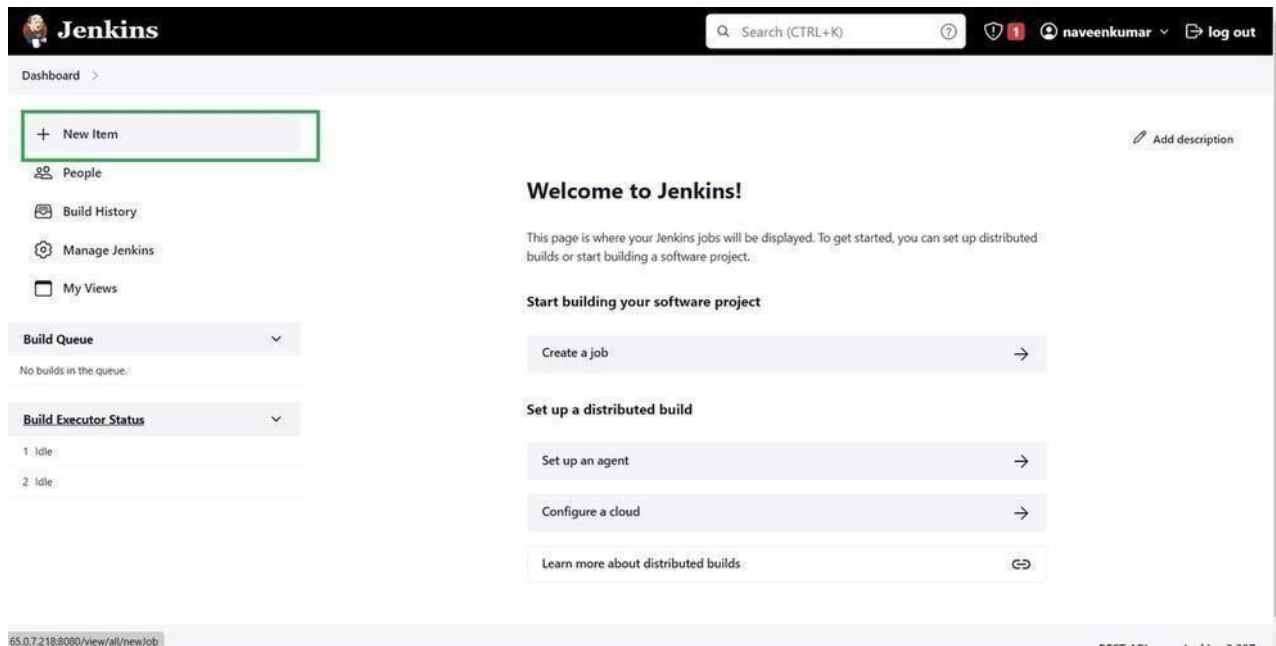
Welcome to Jenkins!

Sign in

☐ Keep me signed in

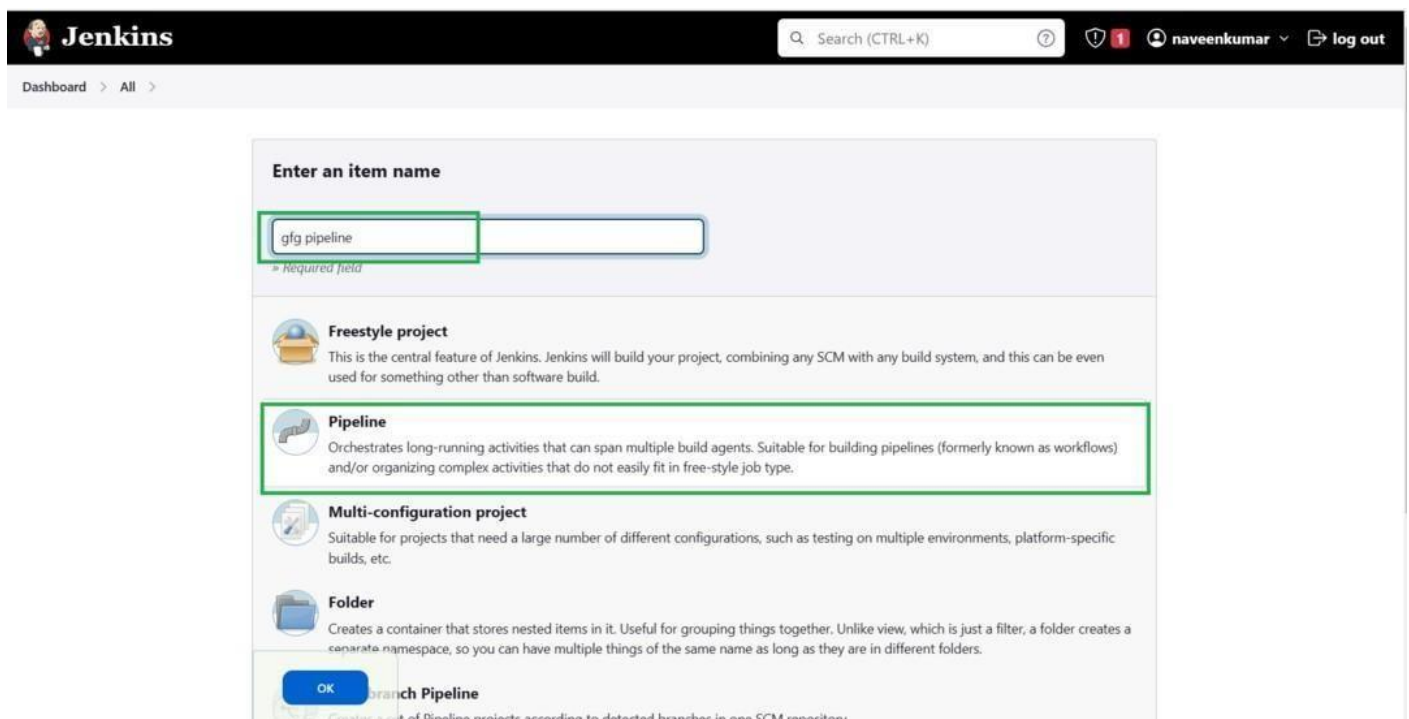
Step 3:

To create a new project, select the option available in the Dashboard which is “New Item” Refer to the image provided below:



Step 4:

Now a list of options will be visible on the screen, along with a field to name the pipeline. Add a suitable name and select the “Pipeline” option to proceed. Refer to this screenshot.



Step 5:

Once redirected, the configuration page will appear. This is the most important page as here all the details will be filled in. At first, there is the General section where the user can add a description based on the project for which the pipeline has to be created. And establish the

connection to compute from where the pipeline will access the project. Refer to the screenshot to understand better.

Dashboard > All > Library Management App1 >

General Build Triggers Advanced Project Options Pipeline

Description

This is the Backend of the Application

[Plain text] [Preview](#)

☐ Discard old builds ?

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☐ GitHub project

[Save](#) [Apply](#)

Step 6:

Now comes the second section, i.e. “Build triggers”. Here, we need to specify the branch and repository and give the credentials too. And add additional behaviours if required so far. Refer to the screenshot to have a better understanding.

Dashboard > All > Library Management App1 >

General Build Triggers Advanced Project Options Pipeline

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☐ Build when a change is pushed to GitLab. GitLab webhook URL: <http://localhost:8282/project/Library%20Management%20App1> ?

☐ Build whenever a SNAPSHOT dependency is built ?

☐ GitHub hook trigger for GITScm polling ?

☒ Poll SCM ?

Schedule ?

[Save](#) [Apply](#)

Step 7:

The next section is “Advanced Project Options”, as the name suggests it is related to the special pipelines only, simpler projects do not require any specifications in this section. Please refer to the screenshot given below for the same.

Dashboard > All > Library Management App1 >

General **Build Triggers** Advanced Project Options Pipeline

⚠ Do you really mean "every minute" when you say "***"? Perhaps you meant "H*****" to poll once per hour**
 Would last have run at Friday, 19 August, 2022 at 5:01:43 PM India Standard Time; would next run at Friday, 19 August, 2022 at 5:01:43 PM India Standard Time.

☐ Ignore post-commit hooks ?

☐ Disable this project ?

☐ Quiet period ?

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

Advanced Project Options

Advanced...

Save **Apply**

Step 8:

This is the last section i.e. “Pipeline”. Here the user specifies from where the scripts will be imported including the path to the file, repository, credentials, etc. Refer to the screenshot attached below for reference.

Dashboard > gfg pipeline > Configuration

Configure

General

Advanced Project Options

Pipeline

Definition

Pipeline script

Script ?

```

1 node
2 {
3   //Mention the maven version that is configured in Jenkins
4   def mavenhome= tool name:"maven3.8.6"
5
6   //Get the source code from Github repository
7
8   stage('checkout code'){
9     git credentialsId: '656dcb6e-d1a6-4c23-bedd-fd1fd29646cc', url: 'https://github.com/MithunTechnologiesDe
10    //Build the code using maven
11    stage('build code'){
12      sh "${mavenhome}/bin/mvn clean package"
13    }
14  }
15 }
  
```

☒ Use Groovy Sandbox ?

Save **Apply**

Sample Pipeline script To Deploy the Web Application Into The Tomcat Server

```

node
{
  //Mention the tools which have been configured

  def mavenhome= tool name:"*****"
  
```


// Mention how to trigger the Pipeline and how many Builds must be there and so on

```
properties([buildDiscarder(logRotator(artifactDaysToKeepStr:
", artifactNumToKeepStr: '5', daysToKeepStr: '
', numToKeepStr: '5')), pipelineTriggers([pollSCM('* * * * *'))]))
```

// Getting the code from the GitHub

```
stage('checkout code'){
    git branch: 'development', credentialsId: '*****', url: '*****'
}
```

//Building the code in to packages by using maven

```
stage('build'){
sh "${mavenhome}/bin/mvn clean package"
```

//Executing the code quality report by using SonarQube

```
}
stage('execute sonarqube package'){
sh "${mavenhome}/bin/mvn clean sonar:sonar"
```

//Uploading the package into nexus

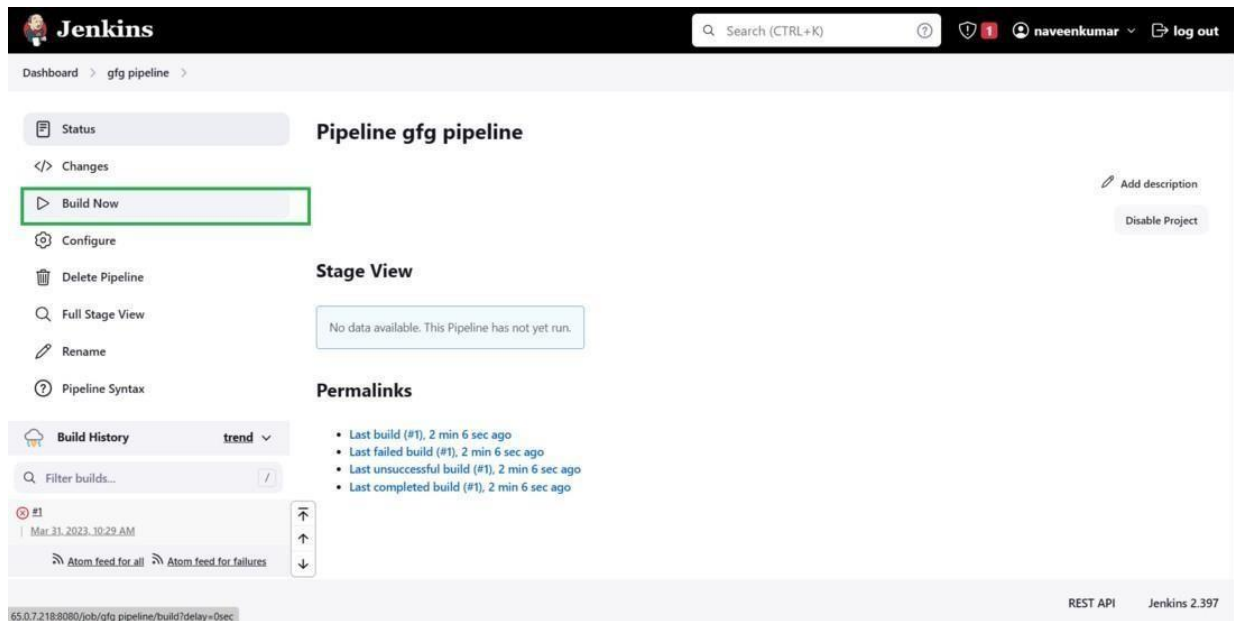
```
}
stage('upload buildartifact'){
sh "${mavenhome}/bin/mvn clean deploy"
```

//Deploying th application into Tomcat

```
}
stage('tomcat'){
sshagent(['myuser']) {
sh "scp -o StrictHostKeyChecking=no target
/maven-web-application.war ec2-user@myuser:/opt/apache-tomcat-9.0.64/webapps/
}
}
```

Step 9:

After writing the pipeline is done click on save it will be directly redirected to the Dashboard of the project there we can use, the “Build Now” option to run the pipeline and check if it is successful or not, by using stage view or console output.



The screenshot shows the Jenkins dashboard for a pipeline named 'gfg pipeline'. The 'Build Now' button is highlighted with a green box. The 'Stage View' section shows a message: 'No data available. This Pipeline has not yet run.' The 'Permalinks' section lists the last build, last failed build, last unsuccessful build, and last completed build, all occurring 2 minutes and 6 seconds ago. The 'Build History' section shows a list of builds, with the first build (#1) selected.

Jenkins Pipeline gfg pipeline

Build Now

Stage View

No data available. This Pipeline has not yet run.

Permalinks

- Last build (#1), 2 min 6 sec ago
- Last failed build (#1), 2 min 6 sec ago
- Last unsuccessful build (#1), 2 min 6 sec ago
- Last completed build (#1), 2 min 6 sec ago

Build History

Filter builds...

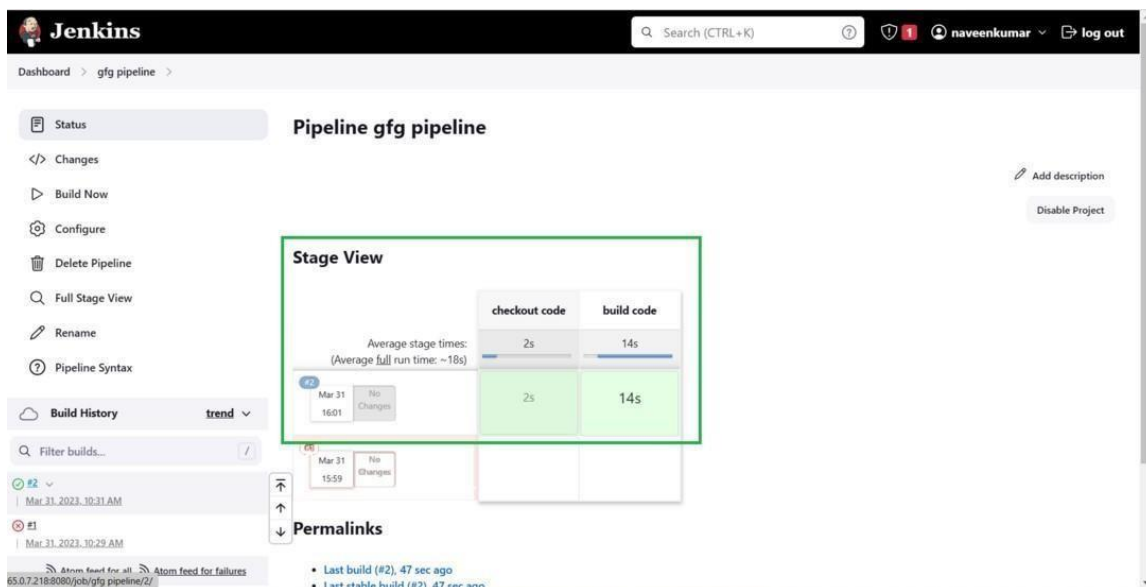
#1 Mar 31, 2023, 10:29 AM

Atom feed for all Atom feed for failures

65.0.7.218:8080/job/gfg pipeline/build?delay=0sec

REST API Jenkins 2.397

We can see the outcome of the pipeline in the stage view where as shown in the image below.



The screenshot shows the Jenkins dashboard for a pipeline named 'gfg pipeline'. The 'Stage View' section is highlighted with a green box, showing a table with columns for 'checkout code' and 'build code'. The table shows the average stage times and the full run time. The 'Build History' section shows a list of builds, with the second build (#2) selected.

Jenkins Pipeline gfg pipeline

Stage View

	checkout code	build code
Average stage times: (Average full run time: ~18s)	2s	14s
Mar 31 16:01 No Changes	2s	14s
Mar 31 15:59 No Changes		

Permalinks

- Last build (#2), 47 sec ago
- Last stable build (#2), 47 sec ago

Build History

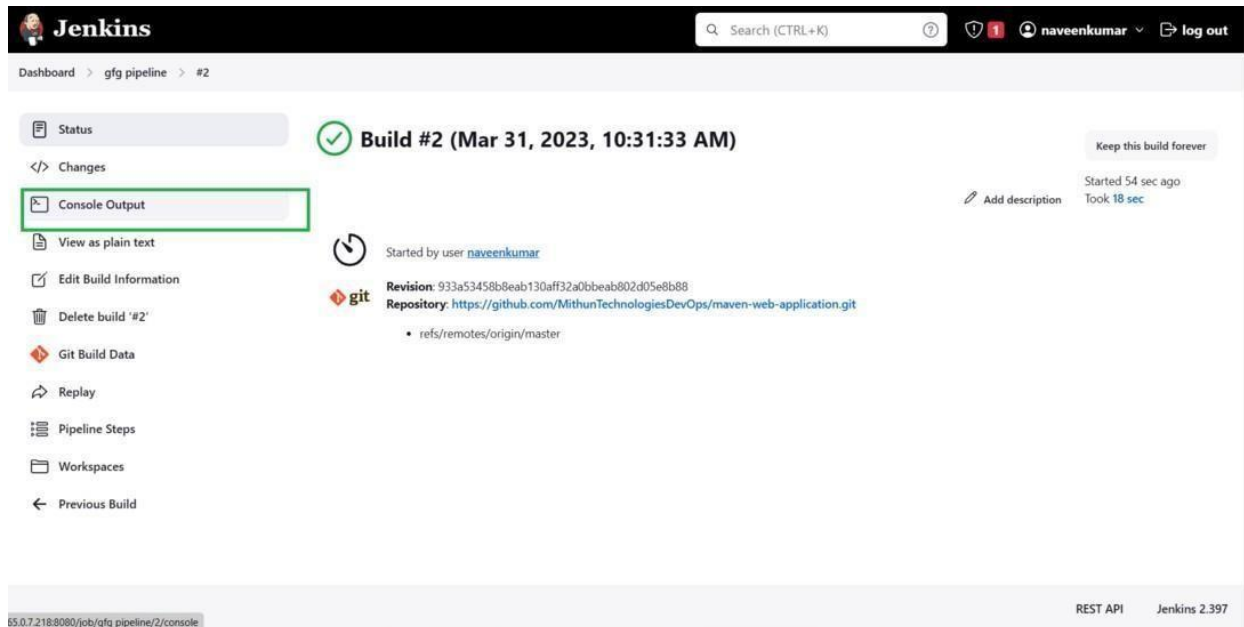
Filter builds...

#2 Mar 31, 2023, 10:31 AM

Atom feed for all Atom feed for failures

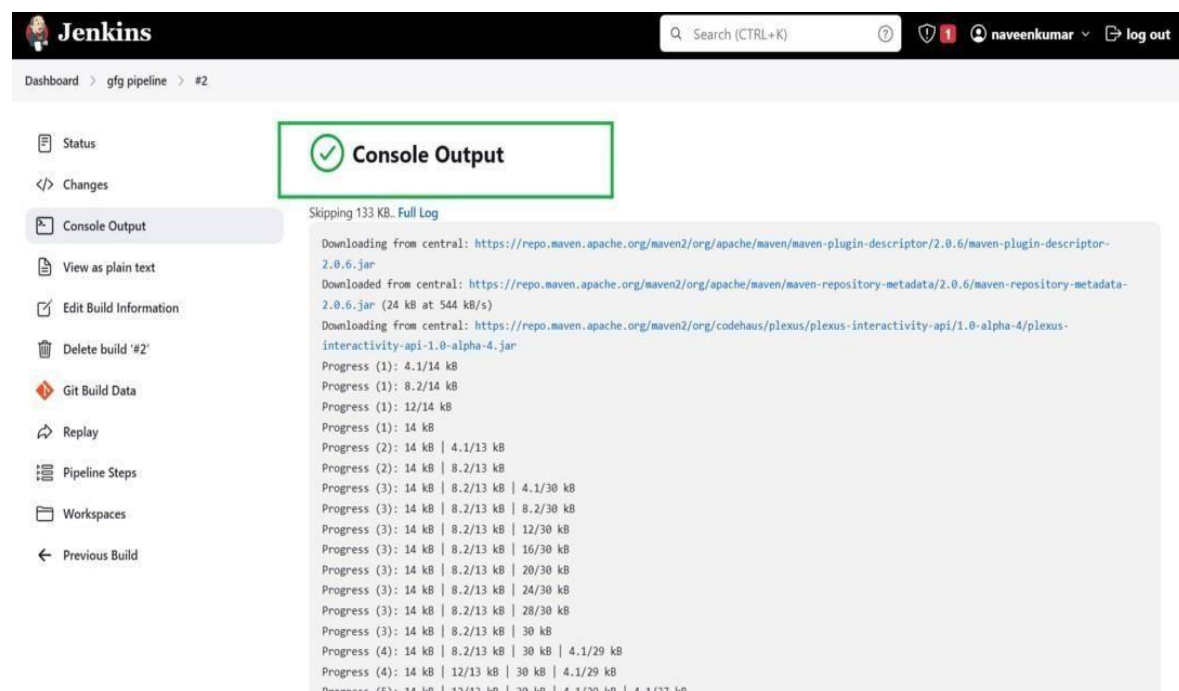
65.0.7.218:8080/job/gfg pipeline/2/

And we can also see the console output where we can see logs of each and every step which is performed.



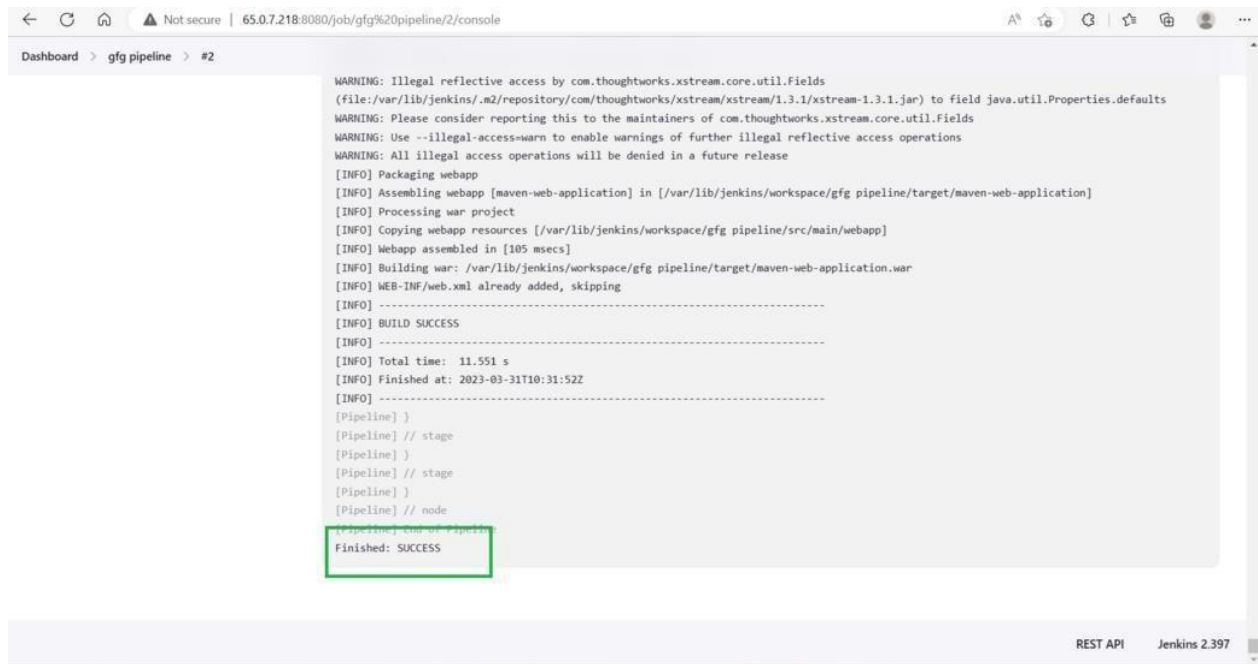
The screenshot shows the Jenkins interface for Build #2 of the 'gfg pipeline'. The build is successful, indicated by a green checkmark. The left sidebar contains a list of actions: Status, Changes, Console Output (highlighted with a green box), View as plain text, Edit Build Information, Delete build '#2', Git Build Data, Replay, Pipeline Steps, Workspaces, and Previous Build. The main area displays the build status 'Build #2 (Mar 31, 2023, 10:31:33 AM)' and a 'Keep this build forever' button. Below this, it shows the build was started by user 'naveenkumar' and provides the Git revision and repository information. The bottom status bar shows the REST API and Jenkins version 2.397.

Click on the console output to see the logs of each and every stage that is performed by using the pipeline.



The screenshot shows the Jenkins interface with the 'Console Output' tab selected. The left sidebar is the same as in the previous image, but the 'Console Output' item is highlighted. The main area displays the console output logs, which include download progress for various Maven artifacts. The logs are as follows:

```
Skipping 133 KB. Full Log
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-descriptor/2.0.6/maven-plugin-descriptor-2.0.6.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-repository-metadata/2.0.6/maven-repository-metadata-2.0.6.jar (24 kB at 544 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interactivity-api/1.0-alpha-4/plexus-interactivity-api-1.0-alpha-4.jar
Progress (1): 4.1/14 kB
Progress (1): 8.2/14 kB
Progress (1): 12/14 kB
Progress (1): 14 kB
Progress (2): 14 kB | 4.1/13 kB
Progress (2): 14 kB | 8.2/13 kB
Progress (3): 14 kB | 8.2/13 kB | 4.1/30 kB
Progress (3): 14 kB | 8.2/13 kB | 8.2/30 kB
Progress (3): 14 kB | 8.2/13 kB | 12/30 kB
Progress (3): 14 kB | 8.2/13 kB | 16/30 kB
Progress (3): 14 kB | 8.2/13 kB | 20/30 kB
Progress (3): 14 kB | 8.2/13 kB | 24/30 kB
Progress (3): 14 kB | 8.2/13 kB | 28/30 kB
Progress (3): 14 kB | 8.2/13 kB | 30 kB
Progress (4): 14 kB | 8.2/13 kB | 30 kB | 4.1/29 kB
Progress (4): 14 kB | 12/13 kB | 30 kB | 4.1/29 kB
Progress (5): 14 kB | 12/13 kB | 30 kB | 4.1/29 kB | 4.1/37 kB
```

The image shows a web browser window displaying the Jenkins console output for a pipeline named 'gfg pipeline'. The browser's address bar shows the URL '65.0.7.218:8080/job/gfg%20pipeline/2/console'. The console output includes several warning messages about illegal reflective access, followed by informational messages about packaging a web application, assembling a webapp, and building a war file. The output concludes with 'BUILD SUCCESS', 'Total time: 11.551 s', and 'Finished at: 2023-03-31T10:31:52Z'. A green box highlights the final status 'Finished: SUCCESS'. The bottom right corner of the console shows 'REST API' and 'Jenkins 2.397'.

Scroll down to the end of the console output there we can see the status of the pipeline if it is “Finished: success” The pipeline which we have written was a success. If it marks as a fail we see the logs in the console we can find the reason why the stage was getting failed.

THEORY AND APPLICATION:

DevOps professionals rely on pipelines to automate stages such as building, testing, and deploying applications. Manually handling these tasks via a user interface consumes significant time and can impact productivity. By using CI/CD pipeline scripts, teams can automate these processes, boosting productivity, minimizing errors, and enhancing delivery speed. CI/CD pipelines help to deliver high-quality applications efficiently to end users.

What is a CI/CD Pipeline?

A CI/CD pipeline, short for Continuous Integration / Continuous Deployment, is a structured, automated process that enables seamless integration and deployment of code changes. In a CI/CD pipeline, multiple stages are linked to create a processing system where each stage receives input, processes it based on predefined rules, and passes it on to the next stage. This linear flow, in which each stage depends on the successful completion of the previous stage, ensures stability and efficiency in application delivery. A typical CI/CD pipeline includes steps like testing code, building the application, pushing to a repository, and deploying to a server.

Each step occurs sequentially, and if any stage fails, subsequent steps halt until the error is resolved, ensuring quality is maintained at every stage. The Devops Engineering – Planning to Production course is a valuable resource, offering real-world examples and best practices for setting up and optimizing CI/CD pipelines using Jenkins.

What is Continuous Integration (CI)?

Continuous Integration (CI) is the practice of automatically building and testing code whenever changes are pushed to a repository, such as GitHub or GitLab. This automation ensures that code remains functional and meets standards before being merged into a shared repository, allowing for rapid detection of issues and increased development speed.

Benefits of Continuous Integration (CI)

- Continuous integration allows for frequent project reports, helping teams to stay informed of the project's health.
- Deployments can be completed within set timeframes.
- Bugs are detected promptly, reducing the time and resources needed for troubleshooting.

What is Continuous Deployment/Delivery (CD)?

Continuous Deployment

Continuous Deployment involves automating the deployment of applications or code to different environments, such as Development, Testing, and Production. With CI/CD pipelines, every build and test stage can be configured to proceed automatically, reducing manual intervention and enabling faster releases.

Continuous Delivery

Continuous Delivery enables teams to deliver builds into production with minimal human intervention. Every build that passes all automated tests can be deployed into production with just a single click, allowing for a rapid and efficient release process while maintaining high quality.

RESULT:

Thus the experiment is to build an understanding of Continuous Integration and Continuous Deployment (CI/CD) pipelines using Jenkins, along with their benefits and application in DevOps verified successfully.

Exp no: 4

Installation of Jenkins

Date:

AIM:

To successfully install Jenkins, a popular open-source automation server, to facilitate continuous integration and continuous deployment (CI/CD) processes.

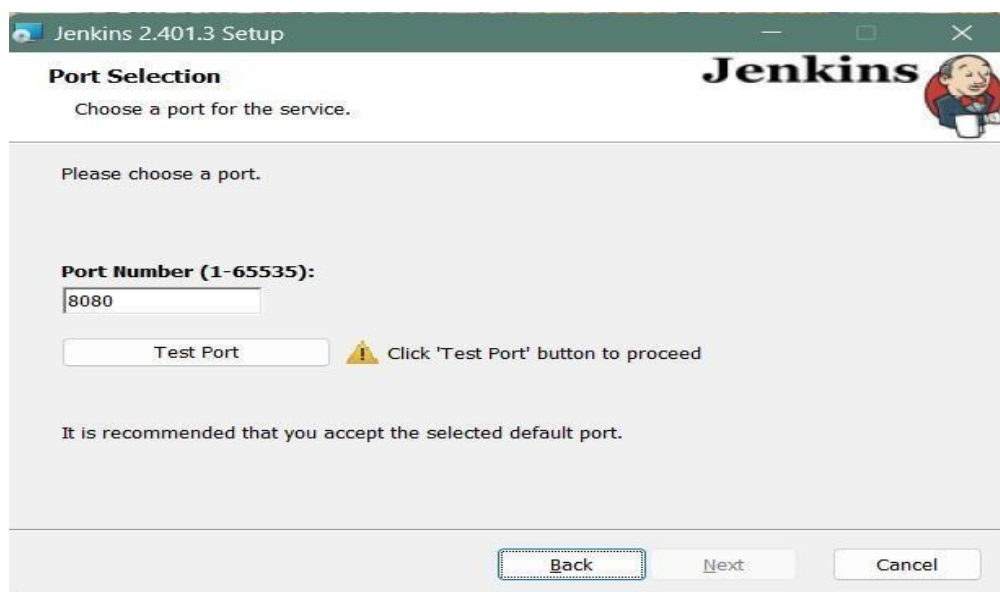
SOFTWARE REQUIREMENTS:

- **Java:** Jenkins requires Java to run. Ensure you have Java Development Kit (JDK) version 8 or higher installed.
- **Web Browser:** A modern web browser for accessing the Jenkins web interface.
- **Operating System:** Jenkins can be installed on various operating systems including Windows, macOS, and Linux.
- **Minimum Hardware Requirements:** RAM: 256 MB (1 GB recommended for better performance). Disk Space: 1 GB (10 GB recommended for Docker installations). Recommended Hardware Configuration: RAM: 4 GB or more, Disk Space: 50 GB or more

PROCEDURE FOR INSTALLATION:

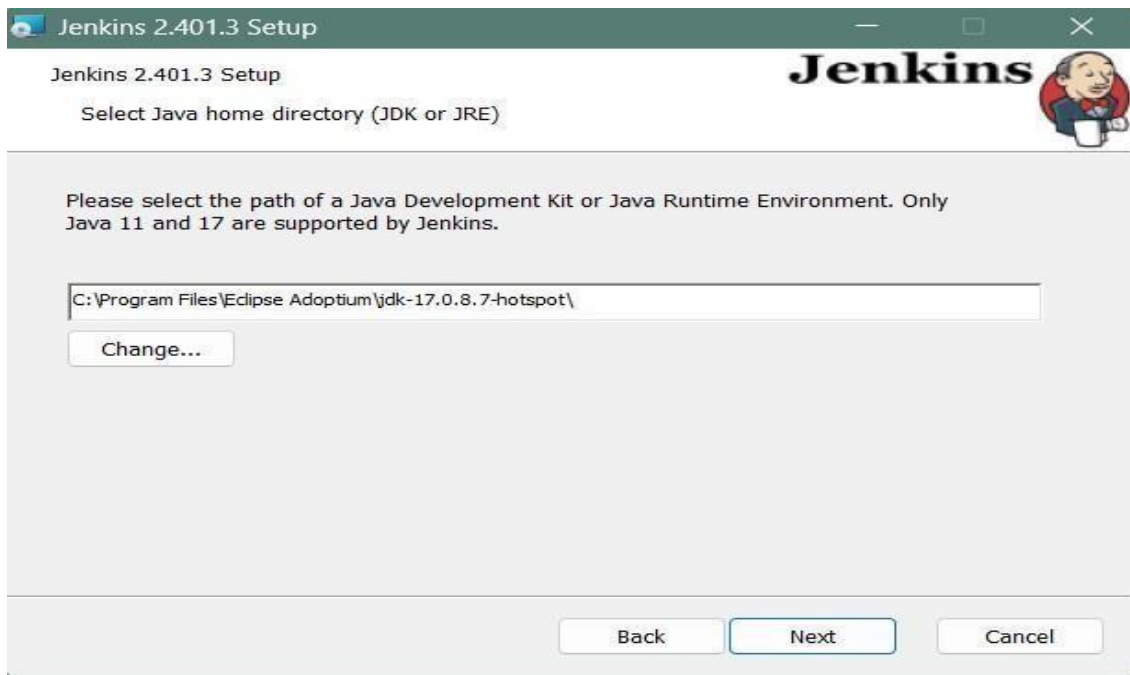
Step 1: Setup wizard

On opening the Windows Installer, an **Installation Setup Wizard** appears, Click **Next** on the Setup Wizard to start your installation.



Step 2: Select destination folder

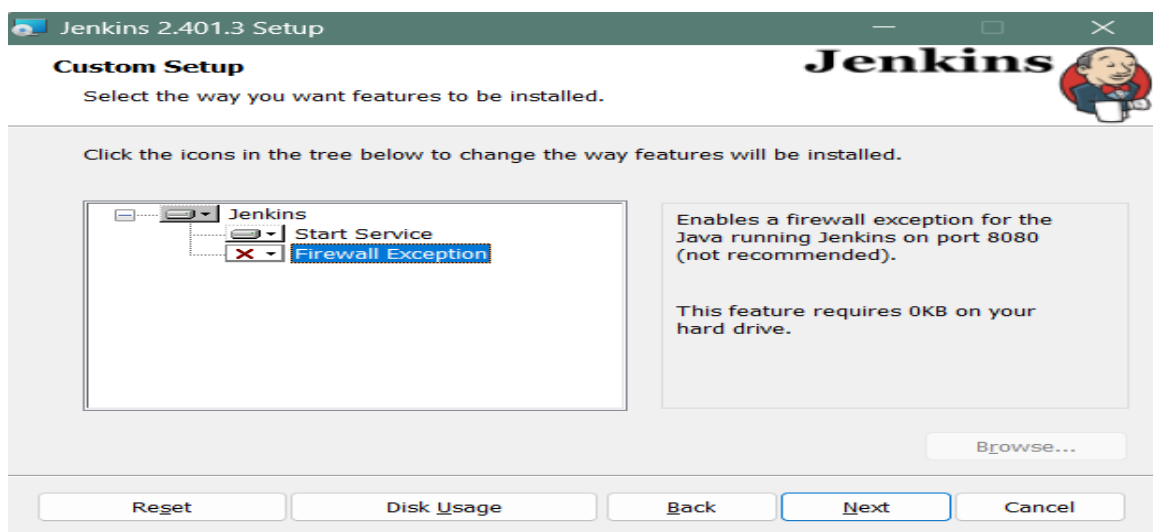
Select the destination folder to store your Jenkins Installation and click **Next** to continue.



Step 3: Service logon credentials

When Installing Jenkins, it is recommended to install and run Jenkins as an independent windows service using a **local or domain user** as it is much safer than running Jenkins using **Local System(Windows equivalent of root)** which will grant Jenkins full access to your machine and services.

To run Jenkins service using a **local or domain user**, specify the domain user name and password with which you want to run Jenkins, click on **Test Credentials** to test your domain credentials and click on **Next**.



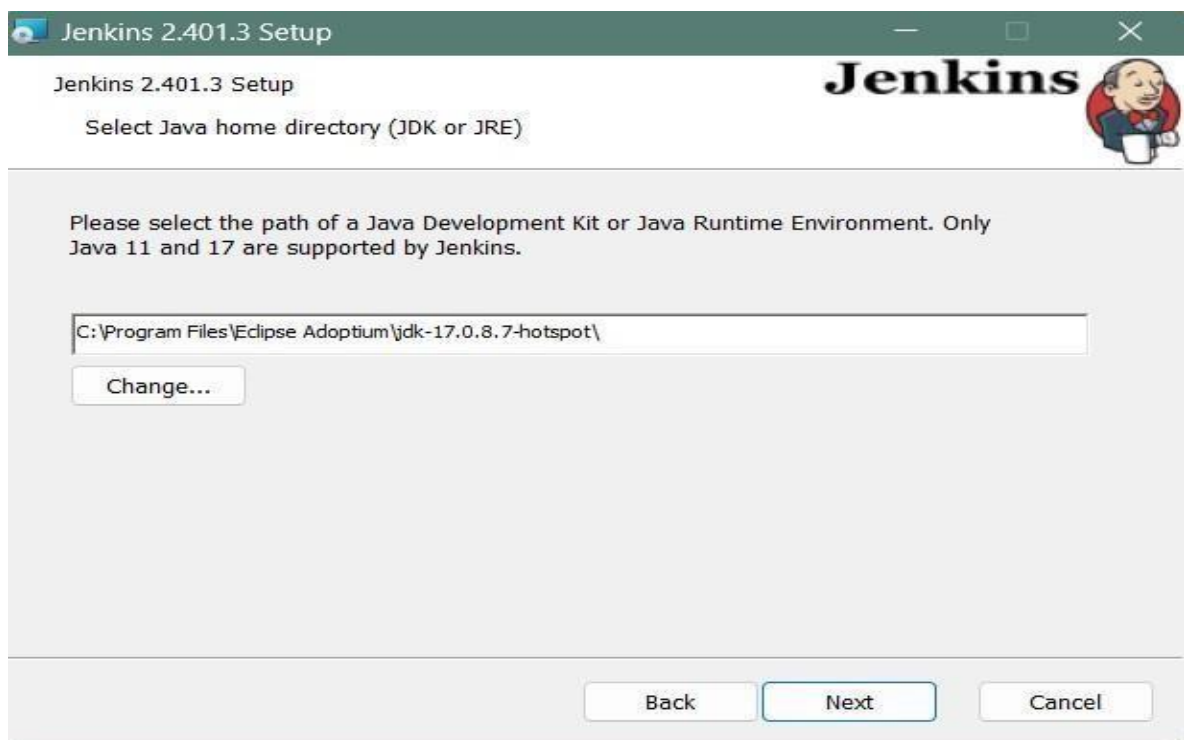
Step 4: Port selection

Specify the port on which Jenkins will be running, **Test Port** button to validate whether the specified port is free on your machine or not. Consequently, if the port is free, it will show a green tick mark as shown below, then click on **Next**.



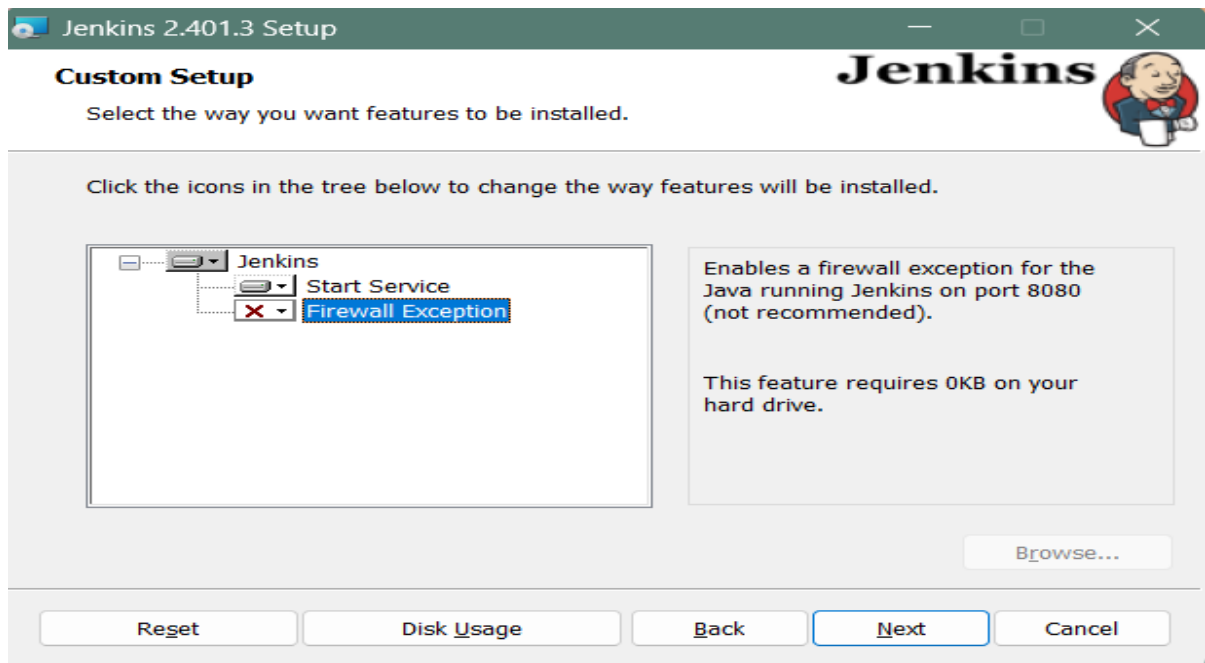
Step 5: Select Java home directory

The installation process checks for Java on your machine and prefills the dialog with the Java home directory. If the needed Java version is not installed on your machine, you will be prompted to install it. Once your Java home directory has been selected, click on **Next** to continue.



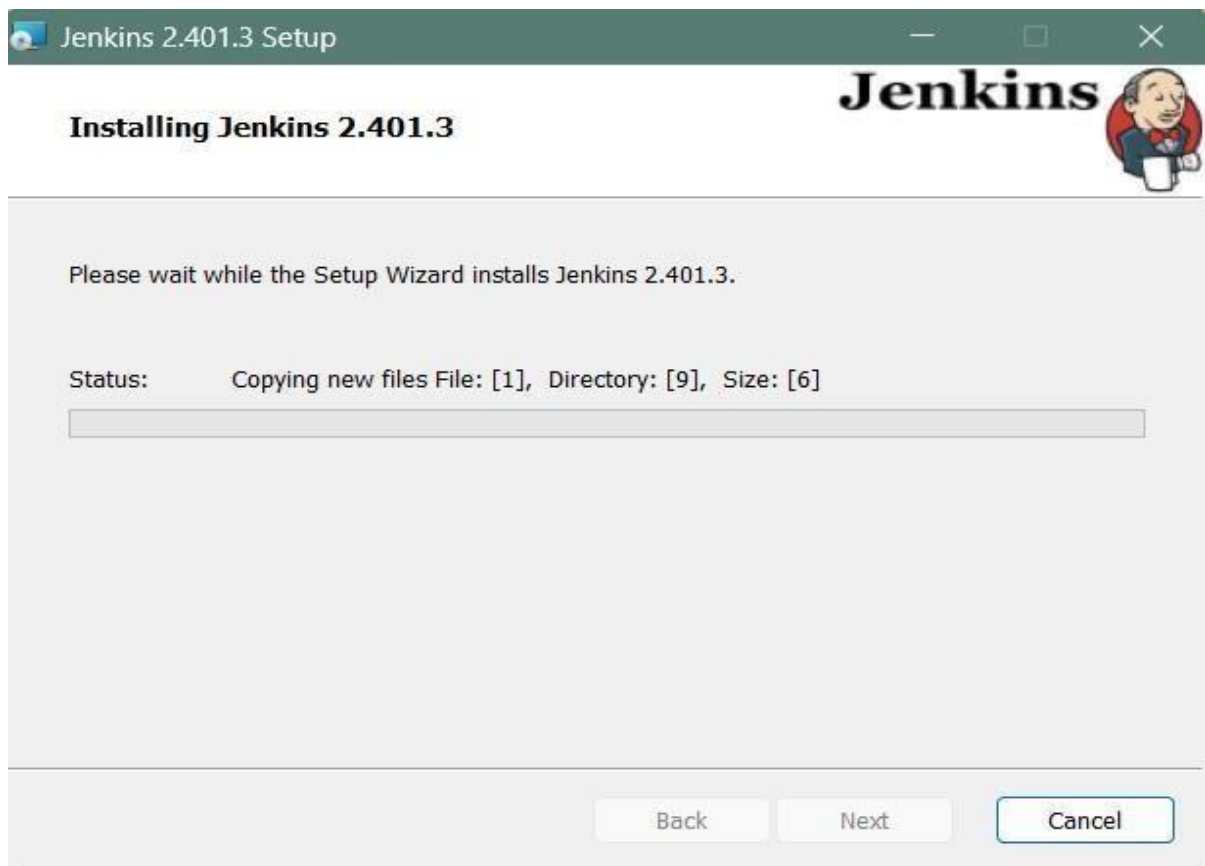
Step 6: Custom setup

Select other services that need to be installed with Jenkins and click on **Next**.



Step 7: Install Jenkins

Click on the **Install** button to start the installation of Jenkins.

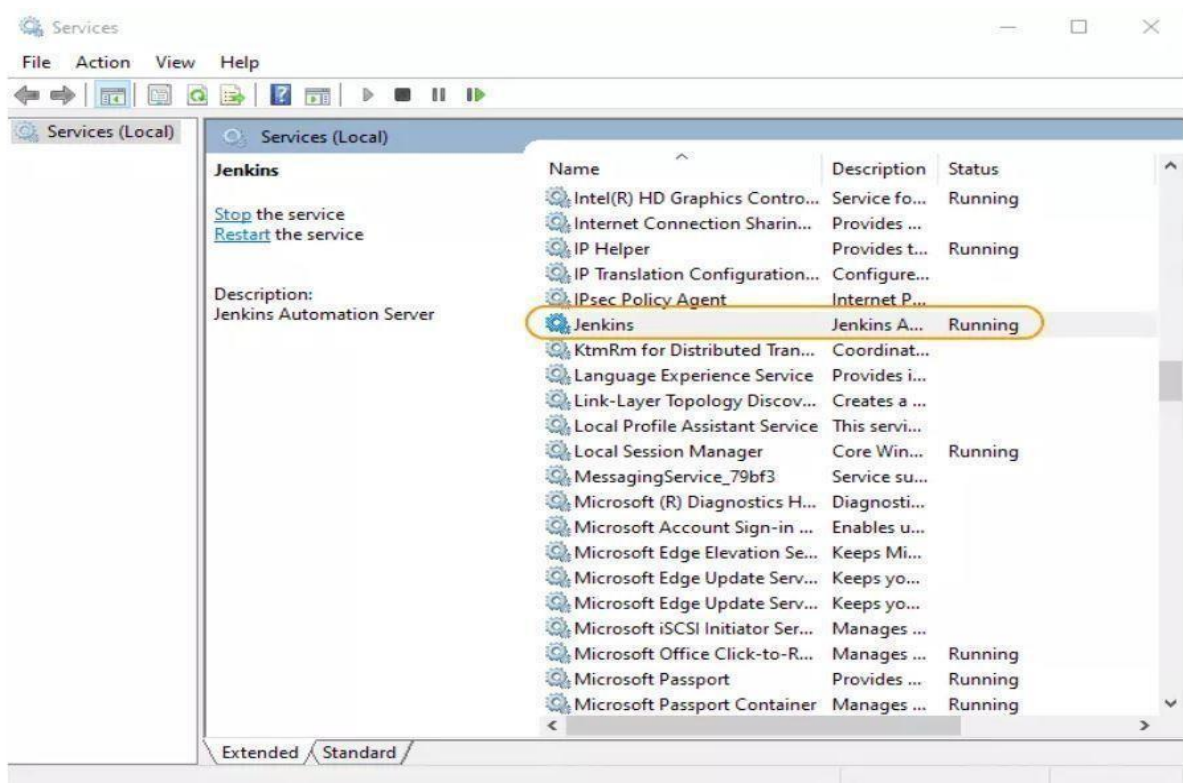


Step 8: Finish Jenkins installation

Once the installation completes, click on **Finish** to complete the installation.



Jenkins will be installed as a **Windows Service**. You can validate this by browsing the **services** section, as shown below:



Post-installation setup wizard

After downloading, installing and running Jenkins, the post-installation setup wizard begins.

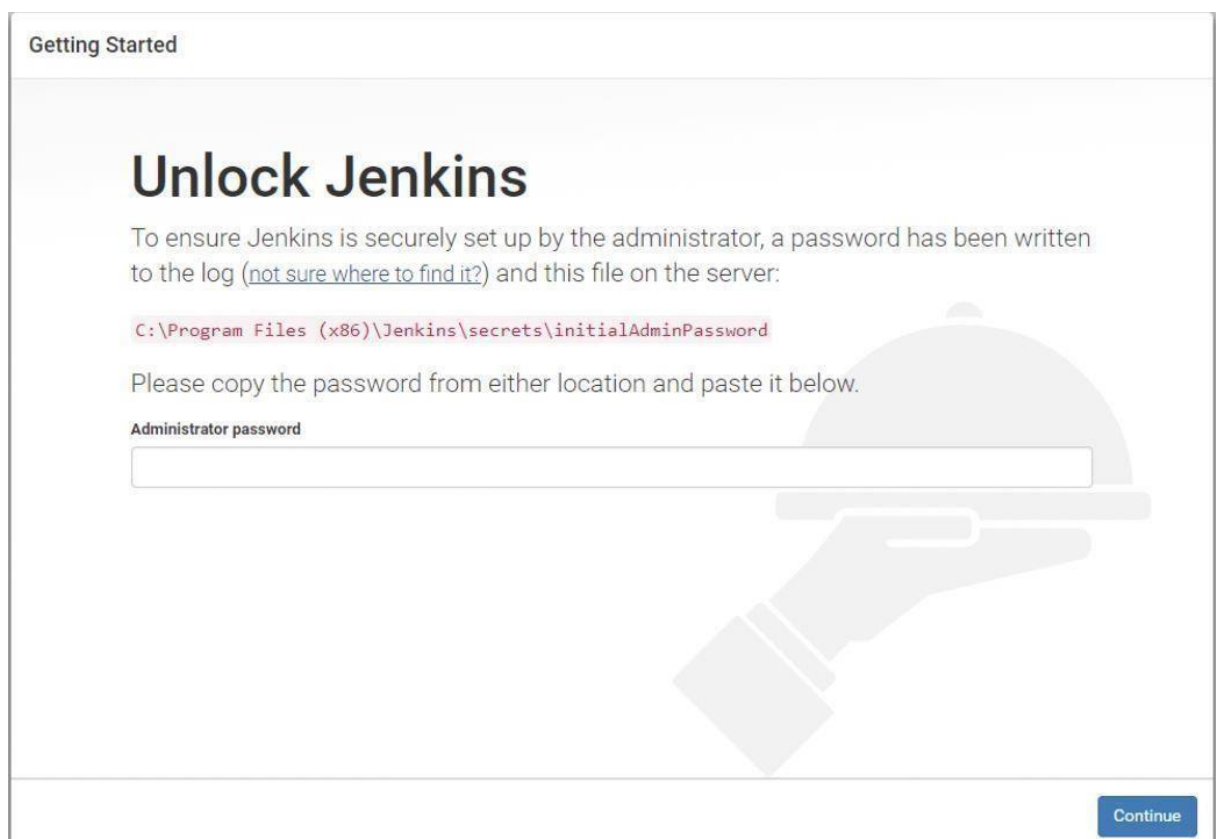
This setup wizard takes you through a few quick "one-off" steps to unlock Jenkins, customize it with plugins and create the first administrator user through which you can continue accessing Jenkins.

Unlocking Jenkins

When you first access a new Jenkins controller, you are asked to unlock it using an automatically-generated password.

Step 1

Browse to <http://localhost:8080> (or whichever port you configured for Jenkins when installing it) and wait until the **Unlock Jenkins** page appears.



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

Step 2

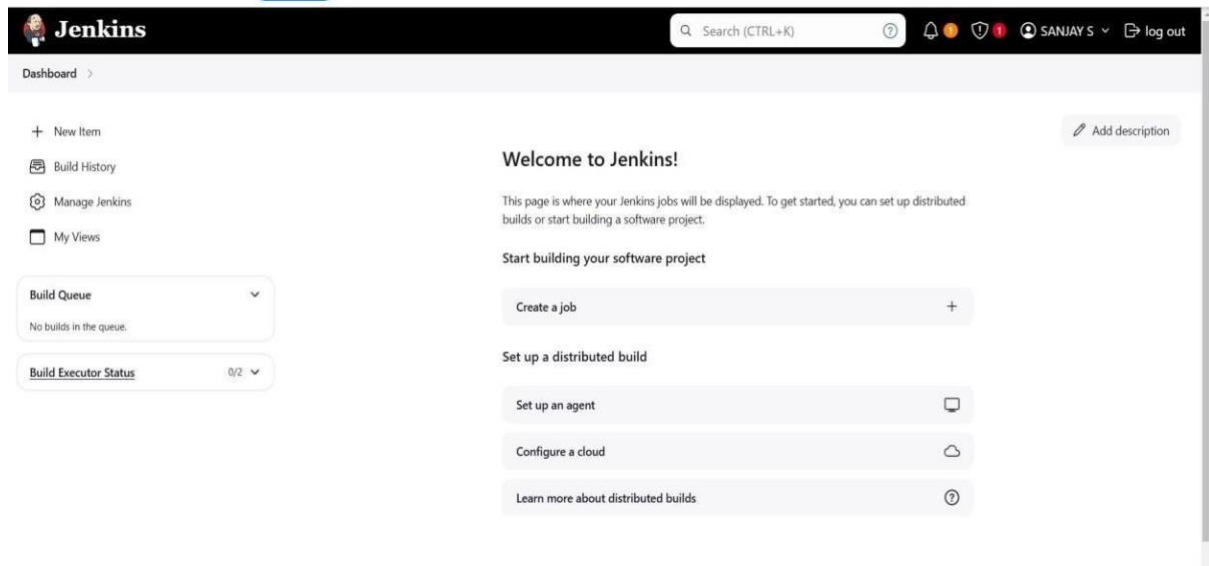
The initial Administrator password should be found under the Jenkins installation path (set at Step 2 in Jenkins Installation).

For default installation location to `C:\Program Files\Jenkins`, a file called **initialAdminPassword** can be found under `C:\Program Files\Jenkins\secrets`.

However, If a custom path for Jenkins installation was selected, then you should check that location for **initialAdminPassword** file.

This password must be entered in the setup wizard on new Jenkins installations before you can access Jenkins's main UI. This password also serves as the default administrator account's password (with username "admin") if you happen to skip the subsequent user-creation step in the setup wizard.

DASHBOARD OF JENKINS:



Theory and Application

Jenkins is used for automating the software development process, allowing teams to build, test, and deploy applications efficiently. It supports various plugins that enhance its functionality, making it adaptable to different workflows and technologies.

RESULT:

Thus the install Jenkins, a popular open-source automation server, to facilitate continuous integration and continuous deployment (CI/CD) processes verified successfully.

Exp no: 5

Installation of SonarQube

Date:

AIM:

To successfully install SonarQube, an open-source platform for continuous inspection of code quality, enabling teams to manage code quality and security in their software development projects.

SOFTWARE REQUIREMENTS:

- Java JDK: SonarQube requires Java 11 or later. Ensure that JDK is installed and configured properly.
- Database: SonarQube requires a database to store analysis results. Supported databases include:
 - PostgreSQL (recommended)
 - MySQL
 - Oracle
 - Microsoft SQL Server
- Web Browser: A modern web browser for accessing the SonarQube web interface.

Minimum Hardware Requirements

- RAM: 2 GB (4 GB recommended for better performance).
- Disk Space: 1 GB (more depending on the size of the projects).

Recommended Hardware Configuration

- RAM: 8 GB or more.
- Disk Space: 20 GB or more.

Step 1: Download and Install Oracle Java 17

Before installing SonarQube, ensure you have Java installed. Follow these steps:

1. Download Oracle Java 17 from Oracle's website.
2. Run the downloaded installer and follow the installation wizard.
3. Set the JAVA_HOME environment variable to C:\Program Files\Java\jdk-17\bin.

Step 2: Download and Install SonarQube

Now, let's install SonarQube:

1. Download the SonarQube zip file from SonarSource.
2. Extract the contents of the zip file to C:\sonarqube-10.4.1.88267.
3. Set the SONAR_JAVA_PATH environment variable using CMD:

```
set "SONAR_JAVA_PATH=C:\Program Files\Java\jdk-17\bin\java.exe"
```

Step 3: Start SonarQube

Start Sonar Qube using the following command in CMD:

```
C:\sonarqube-10.4.1.88267\bin\windows-x86-64>set "SONAR_JAVA_PATH=C:\Program Files\Java\jdk-17\bin\java.exe"
C:\sonarqube-10.4.1.88267\bin\windows-x86-64>echo %SONAR_JAVA_PATH%
C:\Program Files\Java\jdk-17\bin\java.exe
C:\sonarqube-10.4.1.88267\bin\windows-x86-64>StartSonar.bat
Starting SonarQube...
2024.04.03 14:39:29 INFO app[[o.s.a.AppFileSystem] Cleaning or creating temp directory C:\sonarqube-10.4.1.88267\temp
2024.04.03 14:39:29 INFO app[[o.s.a.es.EsSettings] Elasticsearch listening on [HTTP: 127.0.0.1:9001, TCP: 127.0.0.1:21763]
2024.04.03 14:39:29 INFO app[[o.s.a.ProcessLauncherImpl] Launch process[ELASTICSEARCH] from [C:\sonarqube-10.4.1.88267\elasticsearch]: C:\Program Files\Java\jdk-17\bin\java.exe -cp C:\sonarqube-10.4.1.88267\elasticsearch\lib\*;C:\sonarqube-10.4.1.88267\elasticsearch\lib\cli-launcher\* org.elasticsearch.launcher.CliToolLauncher
2024.04.03 14:39:29 INFO app[[o.s.a.SchedulerImpl] Waiting for Elasticsearch to be up and running
2024.04.03 14:40:02 INFO app[[o.s.a.SchedulerImpl] Process(es) is up
2024.04.03 14:40:02 INFO app[[o.s.a.ProcessLauncherImpl] Launch process[WEB_SERVER] from [C:\sonarqube-10.4.1.88267]: C:\Program Files\Java\jdk-17\bin\java -Djava.awt.headless=true -Dsonar.java.util=ALL-UNNAMED -Dsonar.java.lang=ALL-UNNAMED -Dsonar.java.nio=ALL-UNNAMED -Dsonar.java.management=ALL-UNNAMED -Dsonar.jdk.management=com.sun.management.internal=ALL-UNNAMED -Dcom.redhat.fips=false -Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError -cp /lib/sonar-application-10.4.1.88267.jar;C:\sonarqube-10.4.1.88267\lib\jdbc\h2-2.2.224.jar org.sonar.server.app.WebServer C:\sonarqube-10.4.1.88267\temp\sq-process
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by org.sonar.process.BlueBoxSecurityManager (file:/C:/sonarqube-10.4.1.88267/lib/sonar-application-10.4.1.88267.jar)
```

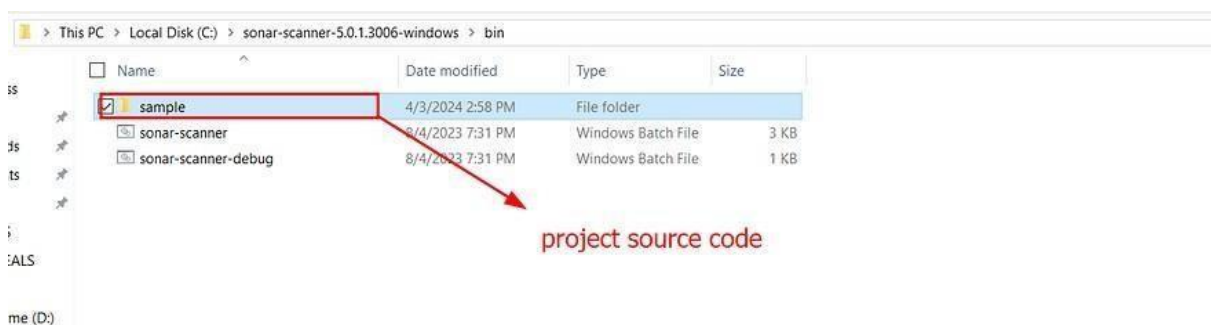
Step 4: Access SonarQube Web Interface

1. Open a web browser and navigate to <http://localhost:9000>.
2. Log in with the default credentials (admin/admin).
3. Follow the prompts to change the password.

Step 5: Install SonarScanner

SonarScanner allows you to analyze your projects for code quality. Here's how to install it:

1. Download SonarScanner CLI from SonarSource.
2. Extract the contents to a directory, e.g., C:\sonar-scanner-5.0.1.3006-windows.



Step 6: Analyze Your Project

1. Open CMD prompt.
2. Run the SonarScanner command:

```
C:\sonar-scanner-5.0.1.3006-windows\bin\sonar-scanner.bat -
D"sonar.projectKey=myproject" -D"sonar.sources=." -
D"sonar.host.url=http://localhost:9000" -D"sonar.login=<your_token_here>"
```

Replace myproject with your project key and <your_token_here> with your generated token from SonarQube.


Step 7: Create and Analyze a Project in SonarQube

1. Log in to SonarQube dashboard.
2. Navigate to Projects tab > Create Project.


1 of 2

Create a local project


Project display name *

 
Up to 255 characters. Some scanners might override the value you provide.

Project key *

 
The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

Main branch name *

The name of your project's default branch [Learn More](#) 

3. Set up the project using global settings.

☒ Use the global setting

Previous version
Any code that has changed since the previous version is considered new code.
Recommended for projects following regular versions or releases.

☐ Define a specific setting for this project

☐ Previous version
Any code that has changed since the previous version is considered new code.
Recommended for projects following regular versions or releases.

☐ Number of days
Any code that has changed in the last x days is considered new code. If no action is taken on a new issue after x days, this issue will become part of the overall code.
Recommended for projects following continuous delivery.

☐ Reference branch
Choose a branch as the baseline for the new code.
Recommended for projects using feature branches.

4. Analyze your project locally using SonarScanner.
5. Provide a token for authentication and execute the SonarScanner command.

1 Provide a token

Analyze "test": sqp_530a8f43417f9fd03c70fc2f4f0e748f825a5f7

2 Run analysis on your project

What option best describes your build?

MavenGradle.NETOther (for JS, TS, Go, Python, PHP, ...)

What is your OS?

LinuxWindowsmacOS

Download and unzip the Scanner for Windows

Visit the official documentation of the Scanner to download the latest version, and add the %PATH% environment variable

Execute the Scanner

Running a SonarQube analysis is straightforward. You just need to execute the following commands in your project's folder.

sonar-scanner.bat -D"sonar.projectKey=test" -D"sonar.sources=." -D"sonar.host.url=http://127.0.0.1:9000" -D"sonar.token=sqp_530a8f43417f9fd03c70fc2f4f0e748f825a5f7"

Copy

```
C:\sonar-scanner-5.0.1.3006-windows\bin>sonar-scanner.bat -h
INFO: usage: sonar-scanner [options]
INFO:
INFO: Options:
INFO: -D, --define <arg>      Define property
INFO: -h, --help              Display help information
INFO: -v, --version            Display version information
INFO: -X, --debug              Produce execution debug output

C:\sonar-scanner-5.0.1.3006-windows\bin>sonar-scanner.bat -D"sonar.projectKey=test" -D"sonar.sources=." -D"sonar.host.url=http://127.0.0.1:9000" -D"sonar.token=sqp_530a8f43417f9fd03c70fc2f4f0e748f825a5f7"
INFO: Scanner configuration file: C:\sonar-scanner-5.0.1.3006-windows\bin\..\conf\sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarScanner 5.0.1.3006
INFO: Java 17.0.7 Eclipse Adoptium (64-bit)
INFO: Windows 10 10.0 amd64
INFO: User cache: C:\Users\Nehru\.sonar\cache
INFO: Analyzing on SonarQube server 10.4.1.88267
INFO: Default locale: "en_US", source code encoding: "windows-1252" (analysis is platform dependent)
INFO: Load global settings
INFO: Load global settings (done) | time=110ms
INFO: Server id: 1478411c-Aw6j0fref-c7BddZiur
INFO: User cache: C:\Users\Nehru\.sonar\cache
WARN: sonar.plugins.downloadOnlyRequired is false, so All available plugins will be downloaded
INFO: Loading all plugins
INFO: Load plugins index
```

Theory and Application

- Theory: SonarQube is a tool that enables developers to continuously inspect the quality of code. It provides reports on code smells, bugs, vulnerabilities, and coverage metrics, helping teams maintain high code quality.
- Application: It is widely used in software development environments to integrate quality checks into the CI/CD pipeline. SonarQube can analyze multiple programming languages and integrates with various CI tools like Jenkins, GitLab CI, and more.

RESULT:

Thus the install SonarQube, an open-source platform for continuous inspection of code quality, enabling teams to manage code quality and security in their software development projects verified successfully.

Exp no: 6

Installation of Eclipse IDE

Date:

AIM:

To successfully install Eclipse IDE, a popular open-source integrated development environment (IDE) used for Java development and other programming languages.

SOFTWARE REQUIREMENTS:

Prerequisites

- **Java Development Kit (JDK):** Eclipse requires Java to run. Ensure you have JDK version 11 or higher installed.
- **Operating System:** Eclipse can be installed on various operating systems including Windows, macOS, and Linux.

Minimum Hardware Requirements

- **RAM:** 4 GB (8 GB recommended for better performance).
- **Disk Space:** 1 GB (more depending on the number of plugins and projects).

Recommended Hardware Configuration

- **RAM:** 8 GB or more.
- **Disk Space:** 5 GB or more.

PROCEDURE:

5 Steps to Install Eclipse

1. Download the Eclipse Installer

Download Eclipse Installer from <http://www.eclipse.org/downloads>

Eclipse is hosted on many mirrors around the world. Please select the one closest to you and start to download the Installer

2. Start the Eclipse Installer executable

For Windows users, after the Eclipse Installer executable has finished downloading it should be available in your download directory. Start the Eclipse Installer executable. You may get a

security warning to run this file. If the Eclipse Foundation is the Publisher, you are good to select Run.

For Mac and Linux users, you will still need to unzip the download to create the Installer. Start the Installer once it is available.



3. Select the package to install

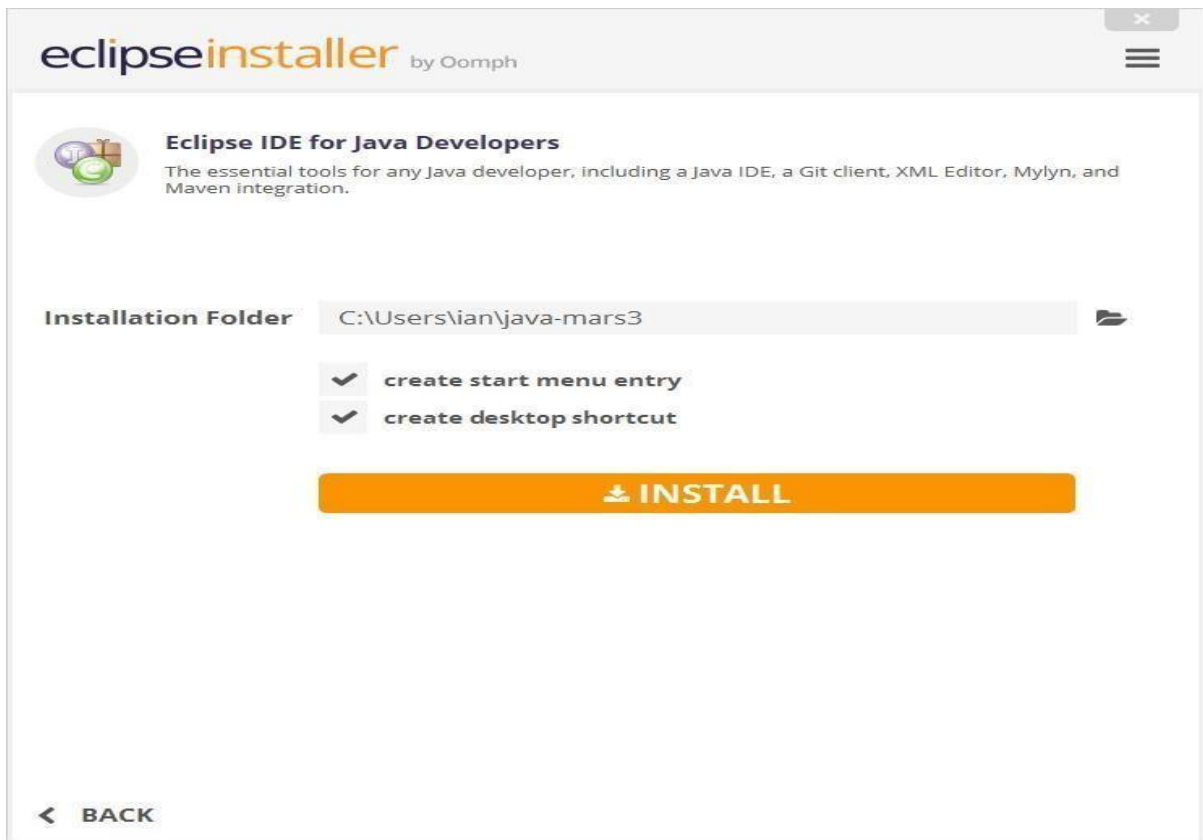
The new Eclipse Installer shows the packages available to Eclipse users. You can search for the package you want to install or scroll through the list.

Select and click on the package you want to install.



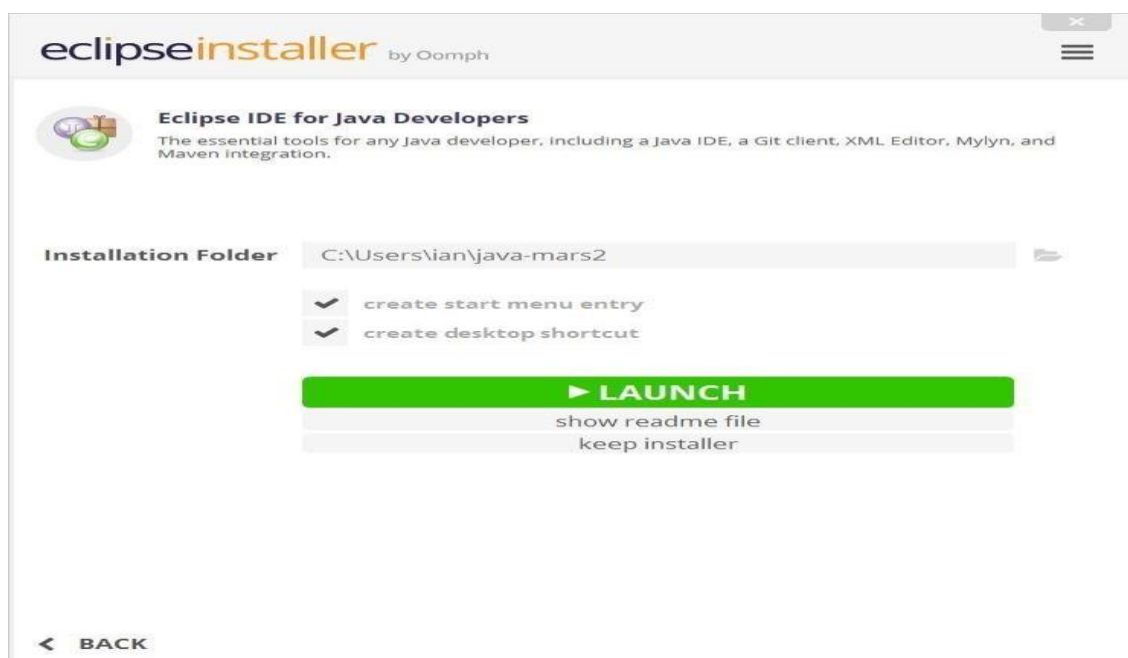
4. Select your installation folder

Specify the folder where you want Eclipse to be installed. The default folder will be in your user directory. Select the 'Install' button to begin the installation.



5. Launch Eclipse

Once the installation is complete you can now launch Eclipse. The Eclipse Installer has done it's work.



Theory and Application

- **Theory:** Eclipse IDE is a powerful tool that provides a comprehensive environment for software development. It supports multiple programming languages through plugins and offers features like code completion, debugging, and version control integration.
- **Application:** Eclipse is widely used in the industry for Java development, but it also supports other languages such as C/C++, PHP, and Python through various plugins. It is especially popular for developing enterprise applications, Android apps, and web applications.

RESULT:

Thus the install Eclipse IDE, a popular open-source integrated development environment (IDE) used for Java development and other programming languages verified successfully.

Exp no: 7

Calculator App in Java using Eclipse IDE

Date:

AIM:

To create a simple calculator application in Eclipse using Java Swing, capable of performing basic arithmetic operations (addition, subtraction, multiplication, and division).

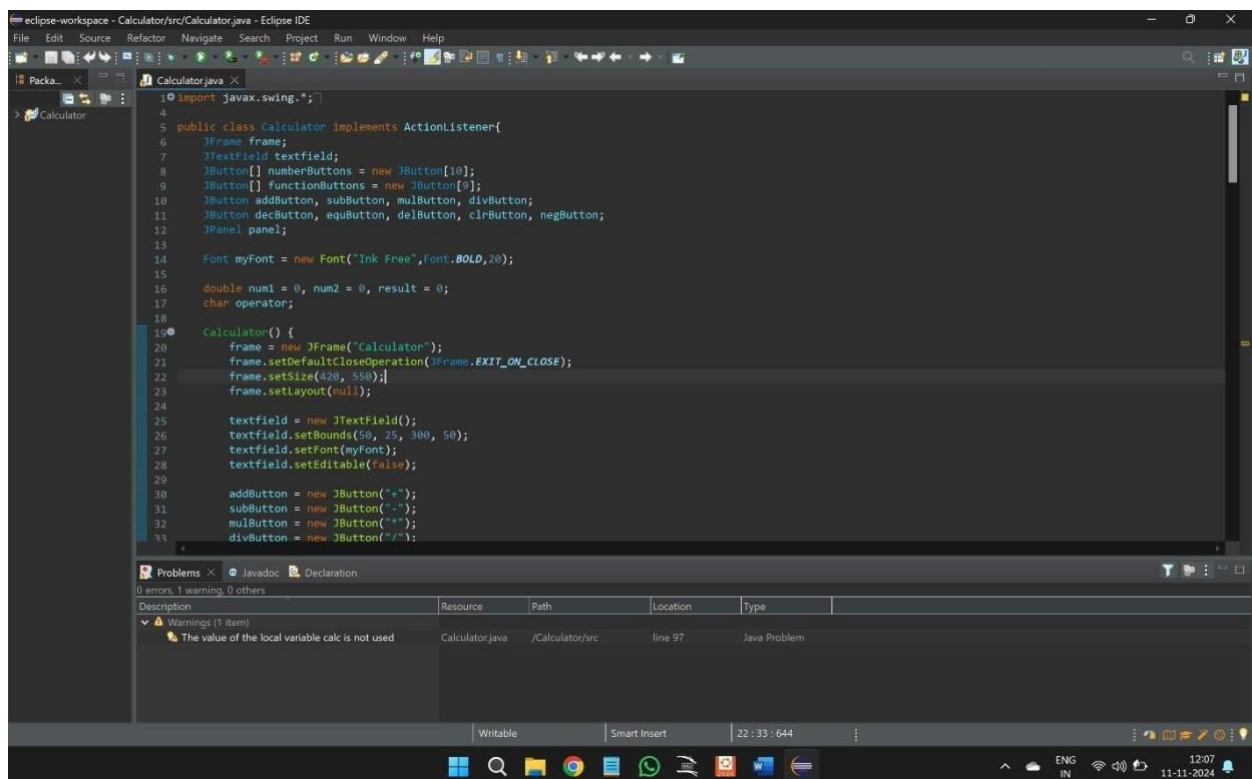
SOFTWARE REQUIREMENTS:

- Java Development Kit (JDK)
- Eclipse IDE
- Java Swing Library

PROCEDURE:

1. Setup and Configuration:

- Open Eclipse IDE and create a new Java Project.
- Name the project as desired (e.g., "CalculatorApp") and set up the workspace.
- Inside the project, create a new Java class named Calculator.



```
1 import javax.swing.*;
4
5 public class Calculator implements ActionListener{
6     JFrame frame;
7     JTextField textField;
8     JButton[] numberButtons = new JButton[10];
9     JButton[] functionButtons = new JButton[9];
10    JButton addButton, subButton, mulButton, divButton;
11    JButton decButton, equButton, delButton, clrButton, negButton;
12    JPanel panel;
13
14    Font myFont = new Font("Ink Free",Font.BOLD,20);
15
16    double num1 = 0, num2 = 0, result = 0;
17    char operator;
18
19    Calculator() {
20        frame = new JFrame("Calculator");
21        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22        frame.setSize(430, 550);
23        frame.setLayout(null);
24
25        textField = new JTextField();
26        textField.setBounds(50, 25, 300, 50);
27        textField.setFont(myFont);
28        textField.setEditable(false);
29
30        addButton = new JButton("+");
31        subButton = new JButton("-");
32        mulButton = new JButton("*");
33        divButton = new JButton("/");
```

2. Import Required Packages:

- Import necessary libraries at the beginning of the Calculator class, including:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

3. Initialize Components:

- Initialize JFrame, JTextField, JButton, and other components inside the constructor of the Calculator class.
- Set properties such as font, size, and layout for each component.
- Use arrays to organize the number and function buttons, then add them to a panel to arrange them in a grid layout.

4. Configure Button Actions:

- Implement ActionListener in the class and override the actionPerformed method.
- Define logic for each button press within actionPerformed, including:
 - Number Buttons: Append numbers to the display text field.
 - Operation Buttons (+, -, *, /): Store the first number, set the operator, and clear the display.
 - Equals Button (=): Perform the calculation based on the selected operator.
 - Clear Button (C): Clear the text field.
 - Delete Button (Del): Remove the last digit entered.





5. Add Components to Frame:

- Add the panel containing number and operation buttons to the frame.
- Set the frame's properties (size, visibility, layout) and make it visible for interaction.

6. Run and Test:

- Compile and run the program in Eclipse.
- Test each function to ensure buttons respond correctly and calculations are accurate.



THEORY AND APPLICATION:

The calculator application is built using Java Swing, a part of Java's Abstract Window Toolkit (AWT) for creating window-based applications. Java Swing provides a rich set of GUI components and features that allow us to create user-friendly interfaces. This project primarily makes use of JFrame for the main window, JTextField for displaying numbers and results, and JButton for interactive buttons.

Key Features Implemented

1. **Basic Arithmetic Operations:** The application supports addition, subtraction, multiplication, and division.

2. User Interface Elements:

- **Display Field:** A non-editable JTextField to show the current input and calculation results.
- **Number and Operation Buttons:** Buttons are created for numbers (0-9) and basic arithmetic operations (+, -, *, /), alongside buttons for decimal input (.), equals (=), clear (C), delete (Del), and negation ((-)).
- **Event Handling:** Implements ActionListener to handle button clicks, capture inputs, and perform calculations based on the selected operation.

RESULT:

Thus the simple calculator application in Eclipse using Java Swing, capable of performing basic arithmetic operations (addition, subtraction, multiplication, and division) verified successfully.