VASANTDADA PATIL PRATISHTHAN'S COLLEGE OF ENGINEERING AND VISUAL ARTS
Department of Information Technology

LAB MANUAL OF

DevOps Lab
Code: ITL503

Class: TE Information Technology
Semester: V (Rev-2019 'C' Scheme)

Lab In-charge
Mr. Pravin Patil
Mr.Vinod Sapkal

H.O.D IT
Dr. Pradip S. Mane

College Vision

- To provide an environment to educate, encourage and explore students by facilitating innovative research, entrepreneurship, opportunities and employability to achieve social andprofessional goals.

College Mission

- To foster entrepreneurship & strengthen industry institute interaction to enhance careeropportunities for the employability of students.
- To encourage collaborations with industries and academic institutes in terms of projects &internships by creating area for Research and Development.
- To build up appropriate moral and ethical skills and to promote holistic development ofstudents through various academic, social and cultural activities

Department Vision

- To impart quality education in the field of Information Technology to meet the challengingneeds of the society and industry.

Department Mission

- To provide quality education to students by including Problem Solving, Teamwork andLeadership Skills to achieve their goals in the field of Information Technology.
- To develop skilled IT professionals with moral principles and empower them in lifelonglearning.
- To educate students for global development including entrepreneurship, employability andthe ability to apply technology to real life problems.

Program Educational Objectives (PEO)

- Graduates will be successful with sound foundation in engineering fundamentals, trendingtechnologies and entrepreneurship.
- Graduates will be able to identify and solve real world problems.
- Graduates will become ingenious and responsible citizens by demonstrating ethics withnurtured professional attitude

Program Specific Outcomes (PSO)

- Develop efficient IT based solutions by applying and integrating various domains likeArtificial Intelligence, IoT, Computer Networks and Security to solve real time problems.
- Apply technical knowledge in the field of Information Technology to achieve successfulcareer and to pursue higher studies for future endeavors.

Program Outcomes (POs)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identity, formulate complex engineering problems reaching substantiatedconclusions using principles of Computer Engineering.
3. **Design / development of solutions:** Design / develop solutions for complex engineering problems and design system components or processes that meet the specified needs with

appropriate consideration for the society.

4. **Conduct investigations of complex problems:** Use knowledge for the design of experiments, analysis, interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select and apply appropriate techniques and modern engineering tools, including predictions and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply the knowledge to assess social issues and the responsibilitiesrelevant to engineering practices.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions insocial and environmental contexts, and demonstrate the knowledge for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and normsof the engineering practice.
9. **Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively such as being able to comprehend and write effective reports and design documentation, make effective presentations.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management skills and apply the skills to manage projects effectively.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Lab Objectives

The Lab experiments aims:

1. To understand DevOps practices which aims to simplify Software Development Life Cycle
2. To be aware of different Version Control tools like GIT, CVS or Mercurial
3. To Integrate and deploy tools like Jenkins and Maven, which is used to build, test and deployapplications in DevOps environment
4. To be familiarized with selenium tool, which is used for continuous testing of applications deployed
5. To use Docker to Build, ship and manage applications using containerization
6. To understand the concept of Infrastructure as a code and install and configure Ansible tool.

Lab Outcome:

On successful completion, of course, learner/student will be able to:

1. To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meetyour business requirements
2. To obtain complete knowledge of the “version control system” to effectively trackchanges augmented with Git and GitHub
3. To understand the importance of Jenkins to Build and deploy SoftwareApplications on server environment
4. Understand the importance of Selenium and Jenkins to test Software Applications
5. To understand concept of containerization and Analyze the Containerization ofOS images and deployment of applications over Docker.
6. To synthesize software configuration and provisioning using Ansible.

Hardware & Software Requirements:

Hardware Requirements	Software Requirements	Other Requirements
PC With following Configuration 1. Intel i3 core or above 2. 4 GB RAM or above 3. 500 GB HDD 4. Network interface card	1. Linux / Windows Operating system 2. VIRTUAL BOX/ VMWARE	1. Internet Connection for installing additional packages 2. GitHub account 3. Docker hub account

DETAILED SYLLABUS:

Sr. No.	Module	Detailed Content	Hours	LO Mapping
0	Prerequisite	Knowledge of Linux Operating system, installation and configuration of services and command line basics, Basics of Computer Networks and Software Development Life cycle.	**00**	LO1
I	Introduction to Devops	Understanding of the process to be followed during the development of an application, from the inception of an idea to its final deployment. Learn about the concept of DevOps and the practices and principles followed to implement it in any company's software development life cycle. Learn about the phases of Software Lifecycle. Get familiar with the concept of Minimum Viable Product (MVP) & Cross-functional Teams. Understand why DevOps evolved as a prominent culture in most of the modern-day startups to achieve agility in the software development process **Self-Learning Topics: Scrum, Kanban, Agile**	**04**	LO1
II	Version Control	In this module you will learn: · GIT Installation, Version Control, Working with remote repository · GIT Cheat sheet · Create and fork repositories in GitHub · Apply branching, merging and rebasing concepts. · Implement different Git workflow strategies in real-time scenarios · Understand Git operations in IDE **Self-Learning Topics: AWS Codecommit, Mercurial, Subversion, Bitbucket, CVS**	**04**	LO1 & LO2

III	Continuous Integration using Jenkins	In this module, you will know how to perform Continuous Integration using Jenkins by building and automating test cases using Maven / Gradle / Ant. · Introduction to Jenkins (With Architecture) · Introduction to Maven / Gradle / Ant.	**04**	LO1 & LO3
		· Jenkins Management Adding a slave node to Jenkins · Build the pipeline of jobs using Maven / Gradle / Ant in Jenkins, create a pipeline script to deploy an application over the tomcat server **Self-Learning Topics: Travis CI, Bamboo, GitLab, AWS CodePipeline**		
IV	Continuous Testing with Selenium	In this module, you will learn about selenium and how to automate your test cases for testing web elements. You will also get introduced to X-Path, TestNG and integrate Selenium with Jenkins and Maven. · Introduction to Selenium · Installing Selenium · Creating Test Cases in Selenium WebDriver · Run Selenium Tests in Jenkins Using Maven **Self-Learning Topics: Junit, Cucumber**	**04**	LO1 , LO3 & LO4
V	Continuous Deployment: Containerizatio n with Docker	In this module, you will be introduced to the core concepts and technology behind Docker. Learn in detail about container and various operations performed on it. · Introduction to Docker Architecture and Container Life Cycle · Understanding images and containers · Create and Implement docker images using Dockerfile. · Container Lifecycle and working with containers. · To Build, deploy and manage web or software application on Docker Engine. · Publishing image on Docker Hub. **Self-Learning Topics: Docker Compose, Docker Swarm.**	**05**	LO1 & LO5

VI	Continuous Deployment: Configuration Management with Puppet	In this module, you will learn to Build and operate a scalable automation system. · Puppet Architecture · Puppet Master Slave Communication · Puppet Blocks · Installation and Configuring Puppet Master and Agent on Linux machines · Use exported resources and forge modules to set up Puppet modules · Create efficient manifests to streamline your deployments **Self-Learning Topics: Ansible, Saltstack**	**05**	LO1 & LO6

List of Experiments

Sr.No	Name Of Experiment	LO'S
1	Case Study on understand DevOps: Principles, Practices and DevOps Engineer Role and Responsibilities.	LO1
2	Study of Version Control System/Source Code Management, install git and create a GitHub Account.	LO2
3	Demonstration of various git operations on local and remote repository using git cheat-sheet.	LO2
4	Install of Jenkins On Windows.	LO3
5	Executing java programming and integration of GitHub with Jenkins.	LO3
6	Executing java pipeline programming and integration of GitHub with Jenkins.	LO3
7	Simple maven project execution in Jenkins from GitHub platform.	LO3
8	Demonstration to Create Maven Pipeline.	LO3
9	Running selenium test cases to test web-based UI components.	LO4
10	Installation of Docker.	LO5
11	Building Docker Image.	LO5
12	Demonstration of ansible automation.	LO6

Sub-in charge
(Mr.Pravin R.Patil)

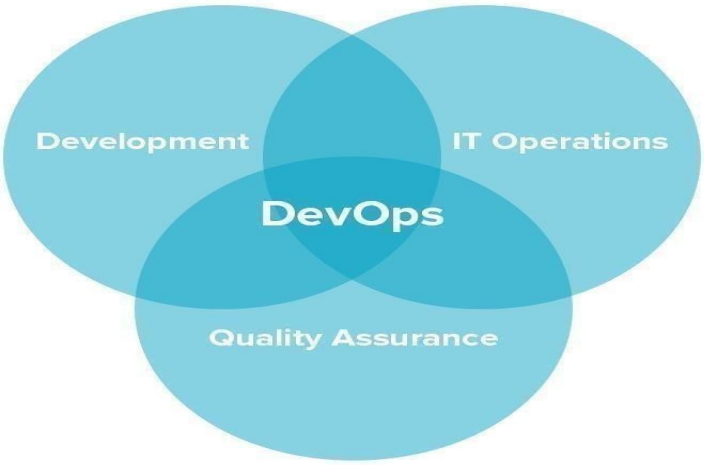
HODIT
(Dr.Pradip Mane)

Experiment No .1

Aim: To Understand DevOps: Principles, Practices and DevOps Engineer Role and Responsibilities.

What is DevOps?

DevOps stands for development and operations. It's a practice that aims at merging development, quality assurance, and operations
(deployment and integration) into a single, continuous set of processes. This methodology is anatural extension of Agile and continuous delivery approaches.



By adopting DevOps companies gain three core advantages that cover technical, business, and culturalaspects of development.

Higher speed and quality of product releases. A DevOps speed up product release by introducing continues delivery, encouraging faster feedback, and allowing developers to fix bugs in the system in the early stages. Practicing DevOps, the team can focus on the quality of the product and automate a number of processes.

Faster responsiveness to customer needs. With DevOps, a team can react to change requests from customer's faster, adding new and updating existing features. As a result, the time-to- market and value delivery rates increase.

Better working environment. DevOps principles and practices lead to better communication between team members, and increased productivity and agility. Teams that practice DevOps are considered to be more productive and cross skilled. Members of a DevOps team, both those who develop and those who operate, act in concert.

DevOps Principles:

The adoption of DevOps as a service has produced several principles that continue to evolve constantly. All these principles have a holistic approach to DevOps, and companies of all nature can adopt them.

Here are key principles that are essential when adopting DevOps:

Culture

DevOps is initially the culture and mind-set forging strong collaborative bonds between software development and infrastructure operations teams. This culture is built upon the following pillars.

Constant collaboration and communication. These have been the building blocks of DevOps since its dawn. Your team should work cohesively with the understanding of the needs and expectations of all members.

Gradual changes. The implementation of gradual rollouts allows delivery teams to release a product to users while having an opportunity to make updates and roll back if something goes wrong.

Shared end-to-end responsibility. When every member of a team moves towards one goal and is equally responsible for a project from beginning to end, they work cohesively and look for ways of facilitating other members' tasks

Early problem-solving. DevOps requires that tasks be performed as early in the project lifecycle as

possible. So, in case of any issues, they will be addressed more quickly.

Automation of processes

Automating as many development, testing, configuration, and deployment procedures as possible is the golden rule of DevOps. It allows specialists to get rid of time-consuming repetitive work and focus on other important activities that can't be automated by their nature.

Measurement of KPIs (Key Performance Indicators)

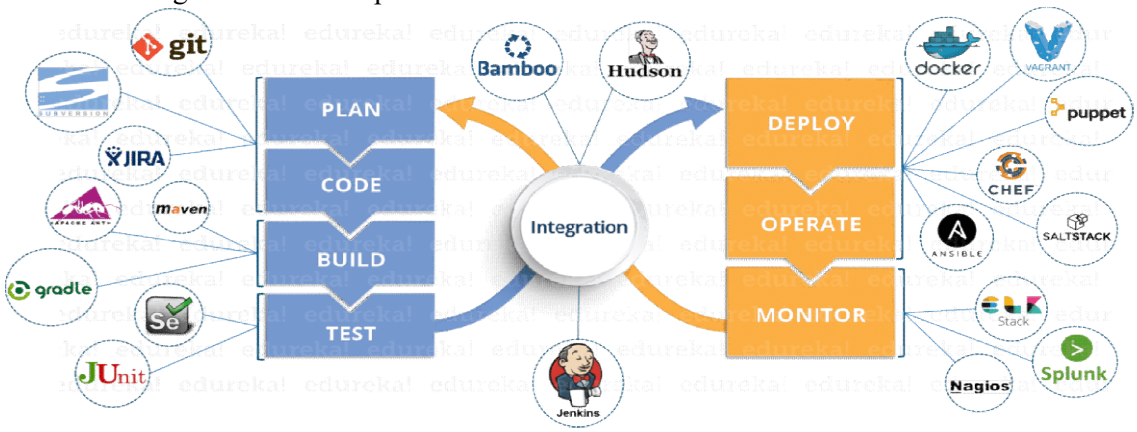
Decision-making should be powered by factual information in the first place. To get optimal performance, it is necessary to keep track of the progress of activities composing the DevOps flow. Measuring various metrics of a system allows for understanding what works well and what can be improved.

Sharing

Sharing is caring. This phrase explains the DevOps philosophy better than anything else as it highlights the importance of collaboration.

DevOps model and practices:

DevOps requires a delivery cycle that comprises planning, development, testing, deployment, release, and monitoring with active cooperation between different members of a team.



To break down the process even more, let's have a look at the core practices that constitute the DevOps:

Agile planning

In contrast to traditional approaches of project management, Agile planning organizes work in short iterations (e.g. sprints) to increase the number of releases. This means that the team has only high-level objectives outlined, while making detailed planning for two iterations in advance. This allows for flexibility and pivots once the ideas are tested on an early product increment.

Continuous development

The concept of continuous "everything" embraces continuous or iterative software development, meaning that all the development work is divided into small portions for better and faster production.

Continuous automated testing

A quality assurance team sets committed code testing using automation tools like Selenium, Ranorex, UFT, etc. If bugs and vulnerabilities are revealed, they are sent back to the engineering team. This stage also entails version control to detect integration problems in advance.

Continuous integration and continuous delivery (CI/CD)

The code that passes automated tests is integrated in a single, shared repository on a server. Frequent code submissions prevent a so-called "integration hell" when the differences between individual code branches and the mainline code become so drastic over time that integration takes more than actual coding.

Continuous deployment

At this stage, the code is deployed to run in production on a public server. Code must be deployed in

"fail fast" approach, meaning that the new features are tested and verified early.

Continuous monitoring

The final stage of the DevOps lifecycle is oriented to the assessment of the whole cycle. The goal of monitoring is detecting the problematic areas of a process and analyzing the feedback from the team and users to report existing inaccuracies and improve the product's functioning.

Infrastructure as a code

Infrastructure as a code (IaC) is an infrastructure management approach that makes continuous delivery and DevOps possible. It entails using scripts to automatically set the deployment environment (networks, virtual machines, etc.) to the needed configuration regardless of its initial state.

Containerization

Virtual machines emulate hardware behavior to share computing resources of a physical machine, which enables running multiple application environments or operating systems (Linux and Windows Server) on a single physical server or distributing an application across multiple physical machines.

Microservices

The microservice architectural approach entails building one application as a set of independent servicesthat communicate with each other, but are configured individually. Building an application this way, you can isolate any arising problems ensuring that a failure in one service doesn't break the rest of theapplication functions. With the high rate of deployment, micro services allow for keeping the whole system stable, while fixing the problems in isolation

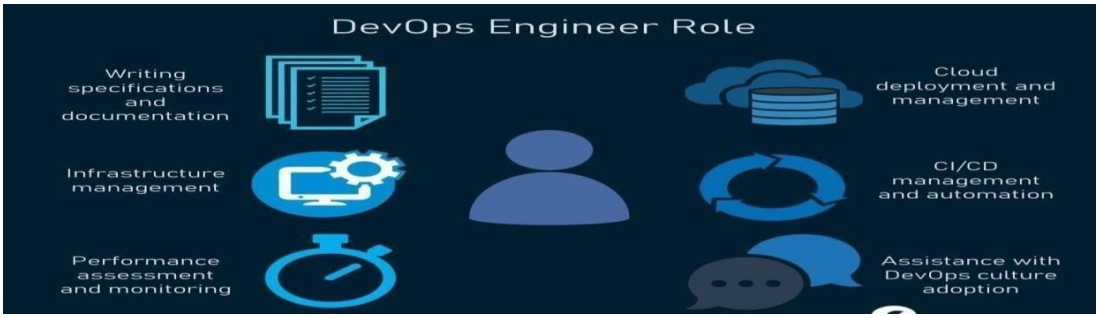
Cloud infrastructure

Today most organizations use Hybrid clouds, a combination of public and private ones. But the shift towards fully public clouds (i.e. managed by an external provider such as AWS or Microsoft Azure) continues. While cloud infrastructure isn't a must for DevOps adoption, it provides flexibility, toolsets, and scalability to applications. DevOps-driven teams can dramatically reduce their effort by basically eliminating server management operations.

A DevOps Engineer: role and responsibilities:

DevOps is a concept that helps to combine two important teams the software development and IT operations in software engineering. The main idea of DevOps is to integrate these two processes with integration tools in order to automate the software development lifecycle which in turn leads to better communication between teams, efficient bug control, better planning more releases, and more RoI for organizations implementing DevOps.

- The fundamental role of a DevOps engineer is to bridge communication the gap between the development and operations teams, which were originally working in silos. Hence, he must be an excellent communicator.
- A DevOps engineer must be a skillful collaborator as he must shape upon the work of other teams. His position is more like a project manager who is assigned to ensure the smooth running of the DevOps as implemented in an organization.
- The role of a DevOps engineer demands to have a 'customer-service' oriented mindset as he usually takes care of the demands of internal clients like the developers (both software and application), internal stakeholders, project managers and members of the DevOps team in an organization.
- DevOps engineer must efficiently implement feedbacks from end-users and have the skill to

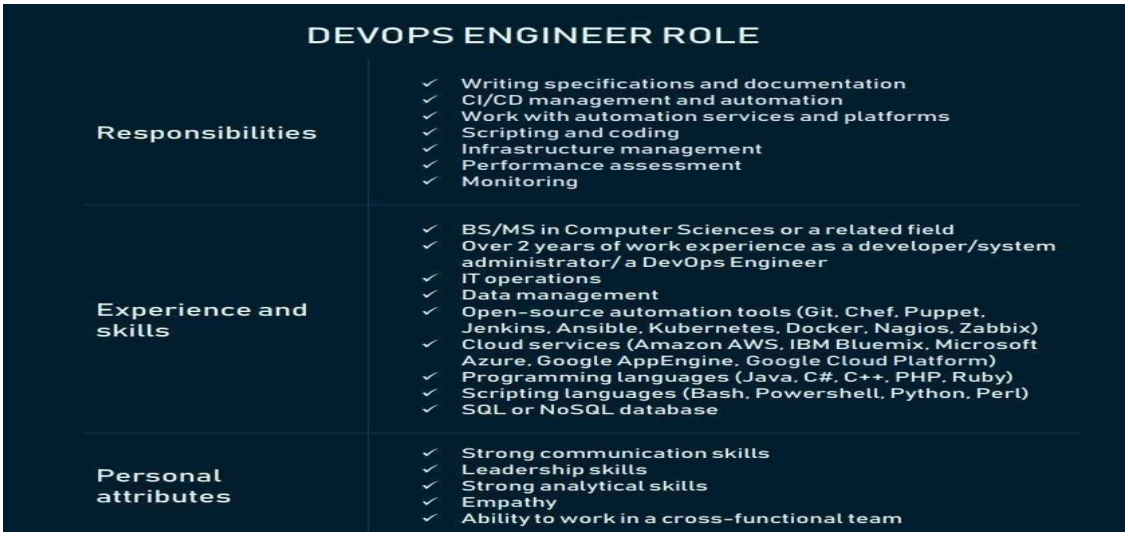


re-engineer approaches as in when needed.

DevOps engineer responsibilities

Out of numerous responsibilities some important tasks of DevOps engineer:

- The first and foremost responsibility is to choose the most appropriate tools and technologies best suited to the business requirements of an organization.
- A DevOps engineer is responsible for guiding teams towards continuous integration and continuous deployment; hence, he must have extensive knowledge of the automation tools and their application like Gradle, Git, Jenkins, Bamboo, Docker, Kubernetes, Puppet Enterprise, Nagios, Chef, and Ansible.
- A DevOps engineer oversees handling the total IT infrastructure. A DevOps engineer is responsible to ensure proper implementation of the automation tools, monitor and re- engineer the existing system based on the client's feedback.
- A DevOps engineer is responsible for conducting system tests in order to ensure optimum data security, availability, and performance. Testing is a crucial part of the transition process from the development phase to the release phase needs effective handling.
- A DevOps engineer is responsible for understanding the language, framing and proper evaluation of code. He must communicate the detailed analysis to the development teams, this ensures overall improvement and completion of the projects within the expected deadline. Hence the knowledge of coding is essential for him.



Conclusion:

Hence, we Understood DevOps its Principles, Practices and DevOps Engineer Role and Responsibilities.

Experiment No.2

Aim: To Understand Version Control System/Source Code Management, install git and create a GitHub Account.

The Fundamentals of a Version Control System (VCS)

The systems that teams use to track changes and different code versions are called Version Control Systems (VCS). Just like with anything else, each VCS has its own unique features and comes with its own set of advantages and disadvantages. There are some basics, though, that make a VCS what it is.
First and foremost, they should retain a long-term history of changes made on a project including creation, deletion, edits and more. This should include the author, date and any notes from the changes as well.

Beyond that, they should have a solution for branching and merging new code changes to the main project to allow concurrent work from multiple members of a team. That is the point, after all. The main codebase for a project is called the Trunk or, simply, Main. Branches are created as independent streams of work that can be merged to the Main. All of this supports traceability in projects. In the event that something goes wrong, it's easier to look back at the changes made over time and connect them to bugs and errors when they appear. Which brings us to the next question, why is version controlso important?

3 Reasons Why VCS is Critical for DevOps

1. **Avoiding dependency issues in modern containerized applications**
   Micro services have essentially become the default for the development of new applications, and more and more teams are containerizing monolithic applications as well. With this trend, dependency issues have entered center stage as something that can make or break a project.

   For DevOps, finding the balance between moving quickly and maintaining application reliability is crucial. Part of that means cultivating traceability, gaining visibility into the changes made to the code and understanding how those changes affected application performance. Access to the different code iterations can reveal where new changes exposed dependencies that clash with other parts of the code.

2. **Version control is tied to higher DevOps performance**
   The annual State of DevOps study found, in 2014, that "version control was consistently one of the highest predictors of performance." Top performing engineering teams were able to achieve higher throughput – 8x deployment frequency and 8000x faster deployment lead times.

   What's causing the high correlation between version control and high performance? Well, it's partially related to the ability to more easily view and understand how changes to one part of the code caused problem across the application. But, it has more to do with how the VCS enables coding practices like Continuous Integration and, as a result, Continuous Delivery/Deployment.

3. **Supports building more reliable applications**

   It's not hard to draw the connections between high performing DevOps teams and reliable applications. DevOps' role in any organization is to enable successful advancements, and because

Although it may seem counterintuitive at first, the first step is usually making smaller changes more frequently (hence the popularity of CI). Just as important, and even more so in an accelerated workflow, is tracking those changes so that everyone is looking at the same thing and so that troubleshooting is easier in the case of failures.

Systems Built for Version Control

The first version (pun intended) of a version control system was the **Concurrent Versions System (CVS)**. It was created to support cooperation between a small team of developers with conflicting schedules. After proving useful on the project, the creator released the code to the public in 1986.

Source Code Management Using Version Control System:

Software is the most vigorous product of Information Technology. The face of software development has changed unprecedentedly and become more custom over a period of time. If we go back 15 years, one can see software were developed for supporting workstations in organizations with centralized database over multiple locations, but rise in international work standard in turn has made software development more competitive and challenging. As a result, software development happens to take place in a collaborative platform – programmers collaborate their code to a central point from multiple locations. How can programmers collaborate? Well, Source code management tools (Version Control System) tackles every single barrier associated with managing source code such as: Integrity, robustness, synchronization, linearity, and revision control. Version control adhere unique functionalities: commit, push/pull to and from code base, snapshots that makes managing source code fairly simple. However conventional version control system can't manage models and there are problems in syncing main repository and local working copy. This work draws shortcoming of 'Git' as a version control system in managing source code and gives insight into Git's association with source code taking line based approach that makes difficult to version control model diagrams with software development. The limitations are identified and the model is proposed to facilitate enhancement in future.

How To Install Git Bash On Windows:

Download Git Bash

- Step 1: Visit the Official Git Bash Website
- Bash Download

Install Git Bash

- Step 3: Run the Installer
- Step 4: Select Destination Location
- Step 5: Select Components
- Step 6: Select Start Menu Folder
- Step 7: Choose the Default Editor used by Git
- Step 8: Adjust your PATH Environment
- Step 9: Choose HTTPS Transport Backend
- Step 10: Configure the Line Ending Conversions
- Step 11: Configure the Terminal Emulator to use with Git Bash
- Step 12: Configuring Extra Options
- Configuring Extra
- Step 14: Wait for Installation
- Step 15: Complete the Git Setup Wizard

Launching Git Bash

Install Git Bash on Windows

In this tutorial, we are going to learn how to install Git Bash on Windows.

Git Bash for Windows is a package that includes git and bash.

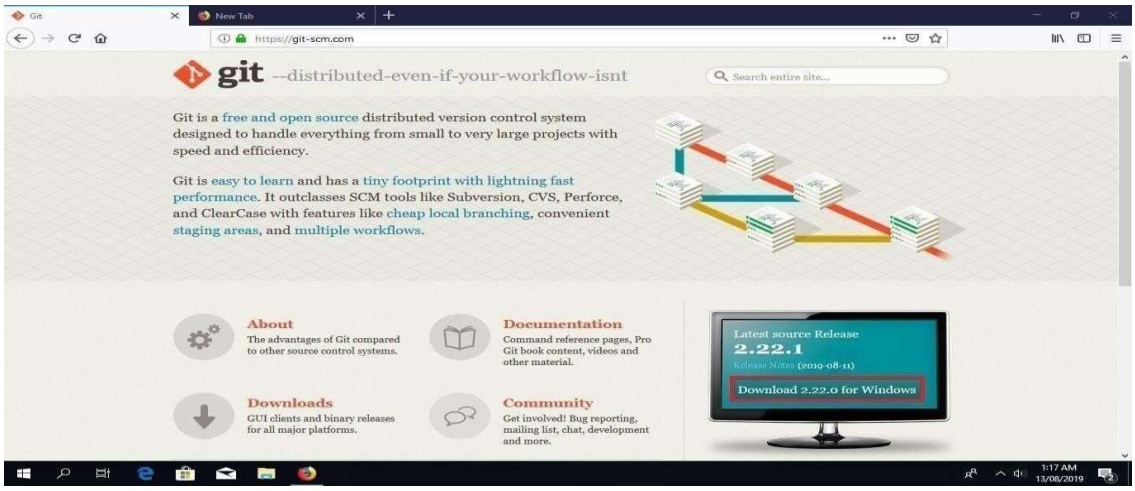
Git is an open-source version control system for tracking source code changes when developing software. It keeps a commit history which allows you to revert to a stable state in case you mess up your code. Gitalso allows multiple developers to collaborate on the same code base.

Bash is a Unix command-line shell. The name is an acronym for the 'Bourne-Again Shell'. It comes withuseful Unix commands like cat, ssh, SCP, etc., which are not usually found on Windows.

Download Git Bash

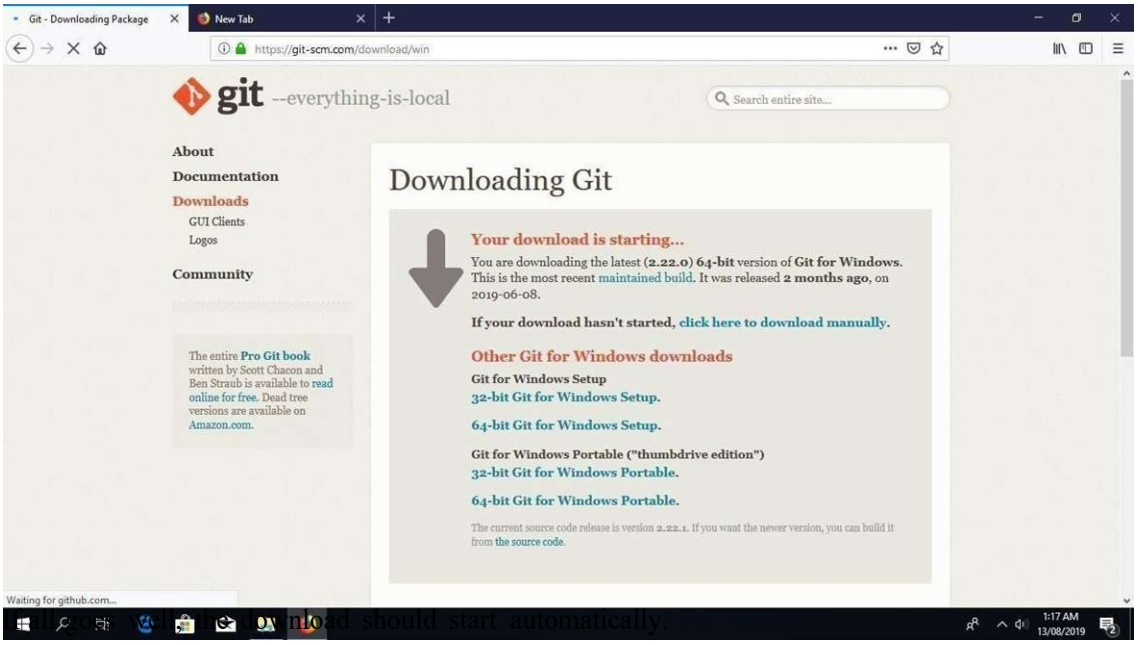
Step 1: Visit the Official Git Bash Website

Download the latest version of Git Bash from their official website: https://git-scm.com/



Click the "Download for windows" button.

Step 2: Start Git Bash Download Next, you will be redirected to a page that lets you know that you are about to start downloading.



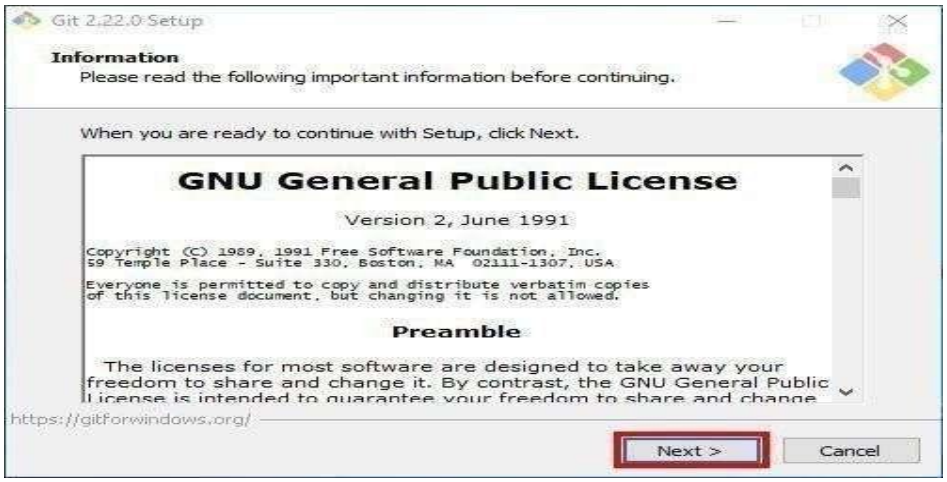
If not, click "download should start automatically.

Click on "Save File" to start downloading the executable.

Install Git Bash

Step 3: Run the Installer

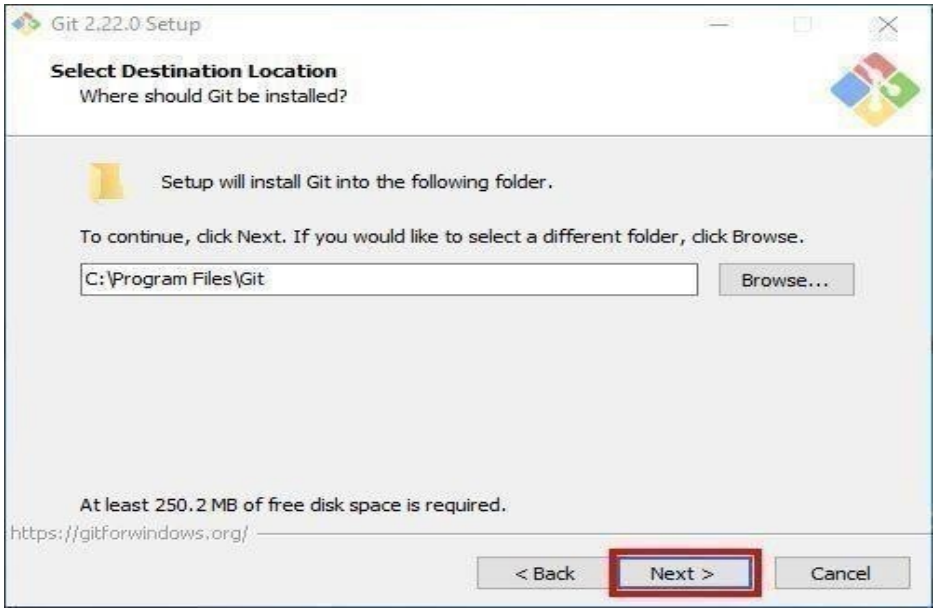
Once you have downloaded the Git Bash executable, click it to run the installer.



Click "Next" after you have read the license.

Step 4: Select Destination Location

Next, select the location you want to install Git Bash. I would recommend you just leave the default option as it is, and click "Next".

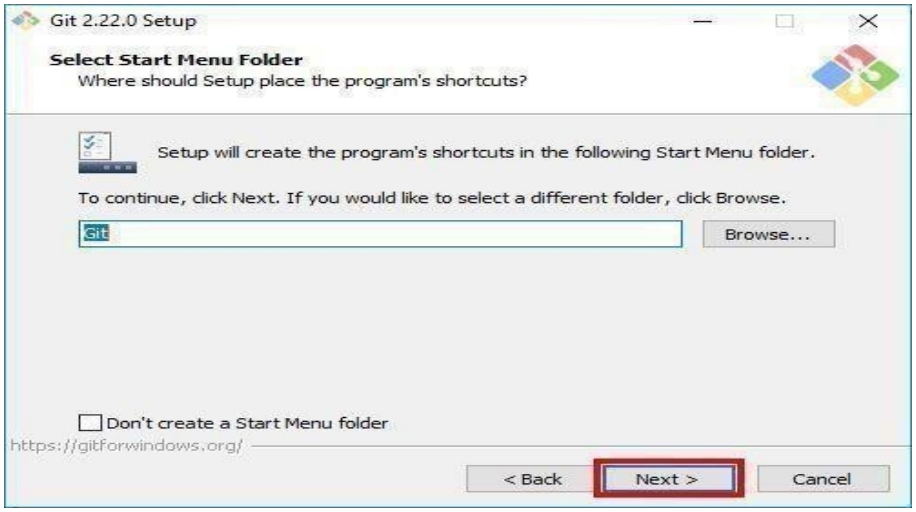


Step 5: Select Components

Choose the components you want to install, or you can just proceed with the default options and click "Next". I prefer selecting the "Additional icons" component which creates a Git Bash shortcut on the desktop.

Step 6: Select Start Menu Folder

You can change the name of start menu folder here if you want, or just leave the default name and click "Next".



Step 7: Choose the Default Editor used by Git

Next, select the default editor for Git to use. Choose the one you like and click "Next". I would recommend you proceed with **Nano** or **Notepad++**. Don't proceed with the default option "Vim" as ithas a steep learning curve.

Step 8: Adjust your PATH Environment

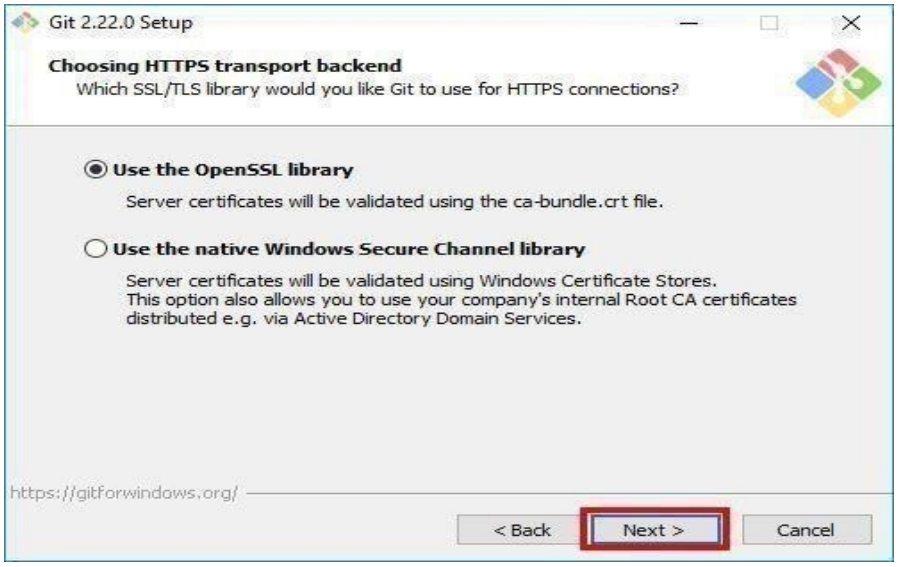
Choose the option you want depending on where you want to use Git and click "Next".

Select **"Use Git from Git Bash only"** option if want to run Git and Bash commands from Git Bash only. This means that you won't be able to run Git commands such as `git status` on Windows CommandPrompt or Power shell. They will only be found on Git Bash.

Select **"Git from the command line and also from 3rd-party software"** option if you want to run Git commands on Windows Command Prompt or Power shell.

Select **"Use Git and optional Unix tools from the Command Prompt"** option if you want to use both Git and Bash commands on Windows Command Prompt or Power shell. This option will override some default Windows Command Prompt tools like find and sort. I don't use CMD or Power shell that much to worry about that. So, I will go ahead with this option by clicking "Next".

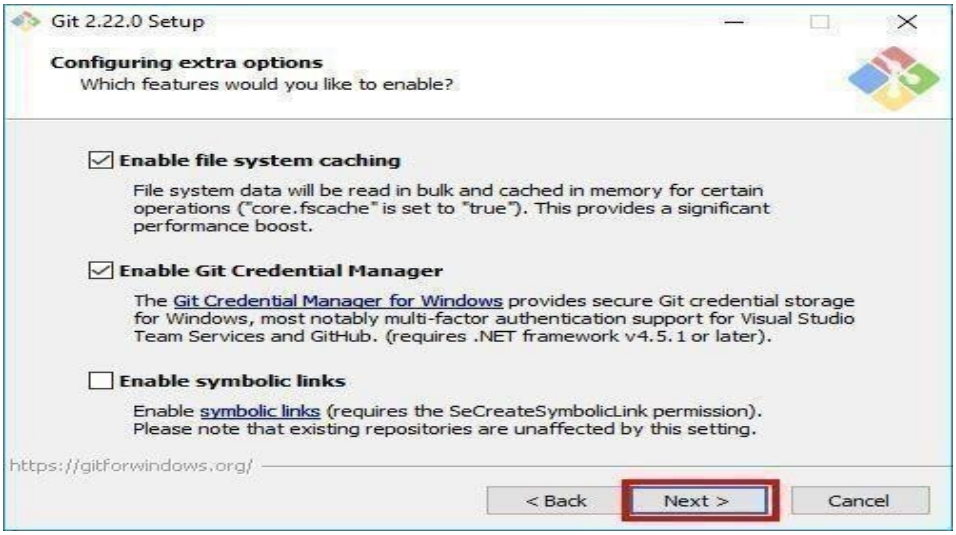
Step 9: Choose HTTPS Transport Backend
Next, select "Use the OpenSSL library" and click "Next".



Step 10: Configure the Line Ending Conversions
Select how Git should treat line endings in text files. It's probably safe to go with the default option "Checkout Windows-Style, commit Unix-style line endings". Click "Next" to proceed.

Step 11: Configure the Terminal Emulator to use with Git Bsh
Next, select the terminal emulator you want Git Bash to use. I will proceed with the default option "Use MinTTY(the default terminal of MSYS2) and click "Next".

Step 12: Configuring Extra Options
Select the features you want(the default options are fine) and click "Next".

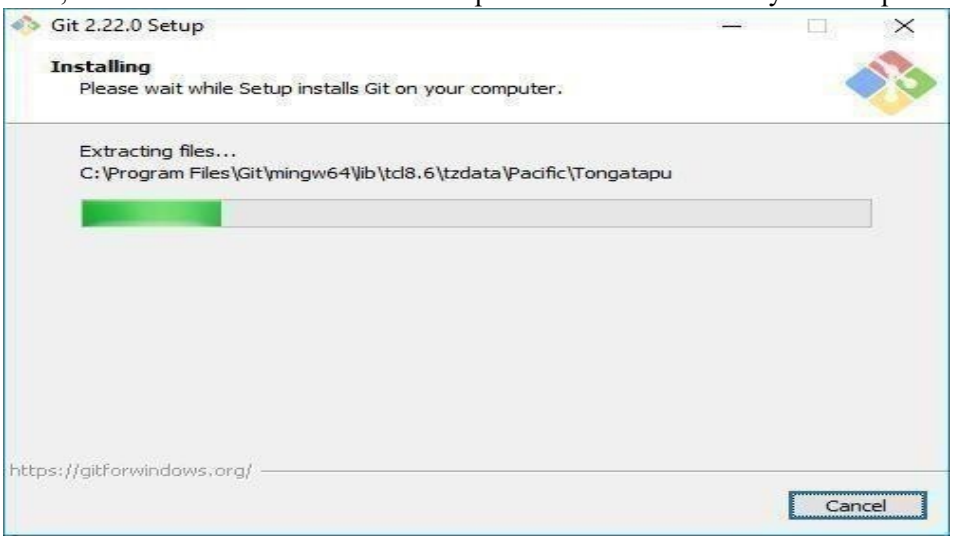


Step 13: Configuring Extra Options

Enable experimental options if you want. Enabling them allows you to try out newer features that are still in development. I don't enable this, so I will just proceed by clicking "Install" to start the installation process.

Step 14: Wait for Installation

Now, wait for a few minutes as the Setup Wizard installs Git on your computer.



Step 15: Complete the Git Setup Wizard

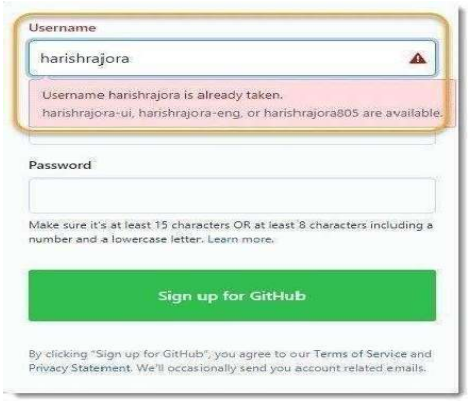
After the installation has finished, check the "Launch Git Bash" and click "Finish" to launch Git Bash.

The Git Bash terminal will now open and you will be able to enter Git and Bash commands.

Congratulations on successfully installing Git Bash.

Setting Up GitHub Account:

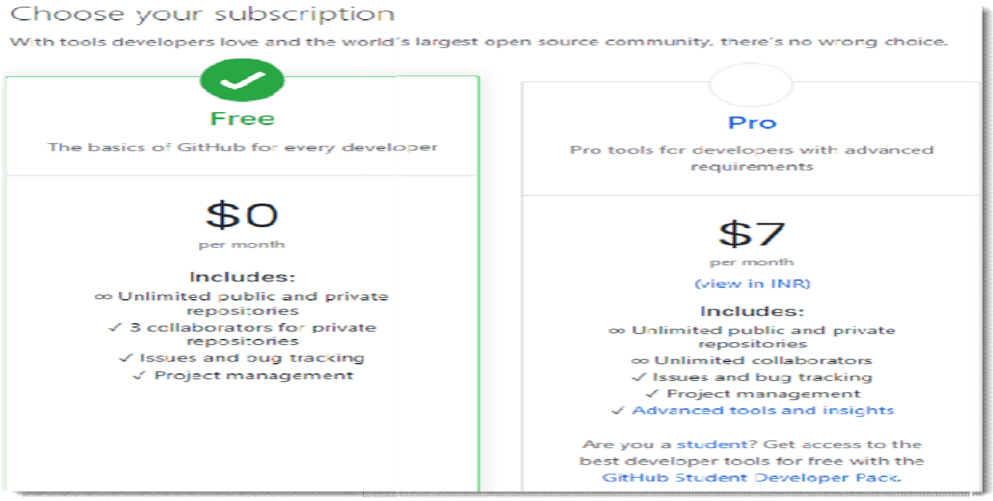
Setting your GitHub account is easy and very simple. To set the account visit **GitHub official** website. The login form will appear on the same page. Fill out the form with your details to create an account on GitHub.



Once you press **Sign up for GitHub** button, you will be prompted to verify that you are not a robot.

Choosing a GitHub Account Plan

Once you have verified your identity, you can choose the GitHub plan you want to subscribe for.

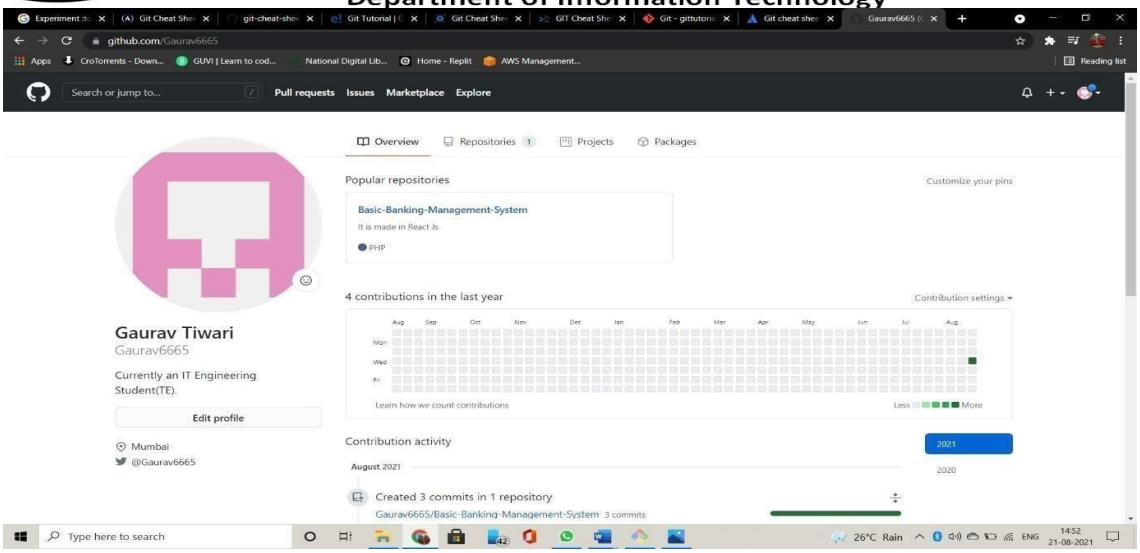


For this tutorial and in general as a beginner, **GitHub Free** plan is more than enough.

As the next step, you would be asked to verify your email address. You can verify it by clicking the link GitHub sent you on your email.

GitHub Account Dashboard

Now that the GitHub account is all set up, you can log in through your credentials on the **GitHub's website**. Logging in will land on the GitHub dashboard which is personalized for everyone according to interests.



Conclusion: Hence, we have Understood Version Control System/Source Code Management and install git and create a GitHub account.

Experiment No.3

Aim: To perform various git operations on local and remote repository using git cheat-sheet.

Git Commands Cheat Sheet – Vocabulary

Sr No.	Git Command	What it Does
1	Bare Repository	Repository that doesn't have a working directory.
2	Branch	An active area of development in Git. The newest commit displays the end of the branch.
3	Blame	Refers to the most recent alteration to every line in the file. Shows Author, Revision, and Time.
4	Checkout	This is talking about the process whereby a particular commit is chosen from the repository and the condition of the file associated with it and the directory tree are reproduced in the working directory.
5	Commit	Record of a moment in Git history containing details of a change set.
6	Diff	The difference in changes between saved changes or two Commits.
7	Detached Head	The state in which a specific commit is checked out rather than a branch.
8	Fetch	Retrieves the most recent changes in the branch and the local or remote repositories.
9	Fork	When you Fork the repository, you can add Commits and add Pull Requests.
10	Hash	A unique SHA1 code for each Commit
11	Head	The name of the Commit at the end of a Branch
12	Index	A group of files that hold state information.
13	Merge	Includes changes from named commits (from when their histories split from the current branch) into the current branch.
14	Master	Git's default development Branch
15	Origin	This is the default Upstream Repository
16	Pull Request	Suggests changes into the Master Branch
17	Push	Pushes new changes once they've been committed
18	Repository	A group of Commits, Branches and Tags to identify Commits.
19	Working Tree	The directory of files that you are currently working on

Git Commands Cheat Sheet – Configuration

Sr.No.	Git Command	Description
1	git config – globaluser.name	Sets the username to be used for every action

2	git config –global user.email	Sets the email to be used for every action.
3	git config –global alias.	Generates a shortcut for the Git command.
4	git config –system core.editor	Sets the text editor for all command actions.
5	git config –global –edit	Opens global configuration file in the text editor to enable manual editing.
6	git config –global color.uiauto	Turns on colour for command line outputs as a visual aid.

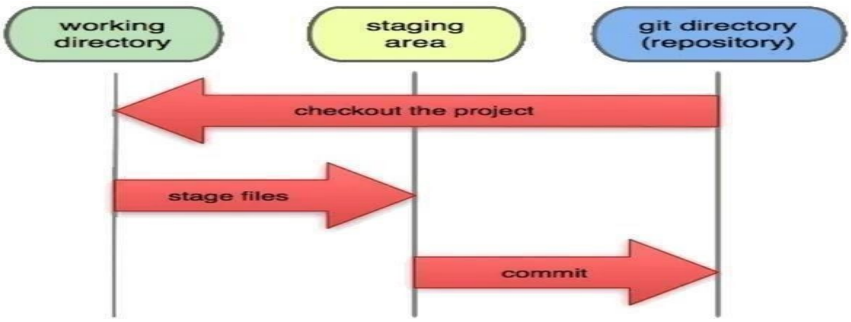
Git Cheat Sheet – Set Up a Git Repository

Sr.No.	Git Command	Description
1	git init	Initializes an emptyGit repository in the current project.
2	git clone (Repo URL)	Clones the repository from GitHub to The project folder.
3	git clone (Repo URL) (Folder)	Clones the repository to a specific folder.
4	git remote add origin https://github.com/username/(repo_name).git	Creates a remote repositorythat points to your current GitHub repository.
5	git remote	Displays the name of remote repositories.
6	git remote -v	Displays the name and URL of remote repositories.
7	git remote rm (remote repo name)	Gets rid of a remote repository.
8	git remote set-url origin (git URL)	Changes a repository URL.
9	git fetch	Obtains the most recent changes from the origin but doesn't merge them.
10	git pull	Obtains the most recent changes from the origin and merges them.

Git Cheat Sheet – Local File Changes

Sr. No.	Git Command	Description
1	git add (file name)	Adds current file changes to staging.
2	git add .	Adds changes for the whole directory to staging butwithout deleting files.
3	git add -A	Adds every new, modified, and deleted file to staging.

5	git rm –cached (file_name)	Stops tracking the current file.
6	git mv (file_name) (new_file_name)	Alters the filename and gets it ready for Commit.
7	git checkout <deleted file name>	Undeletes a file and gets it ready for Commit
8	git status	Displays the status of modified files.
9	git ls-files –other –ignored – excludestandard	Displays a list of each ignored file.
10	git diff	Displays staged changes in the working directory andindex.
11	git diff –staged	Displays differences in files between the most recentversion and staging.
12	git diff (file_name)	Displays changes between a single file and the mostrecent Commit.



Git Commands Cheat Sheet – Declare Commits

Sr. No.	Git Command	Description
1	git commit -m "(message)"	Saves changes along with a custom message.
2	git commit -am "(message)"	Adds all changes to staging and saves them with a custom message.
3	git checkout	Switches to the provided Commit.
4	git show	Outputs content changes and metadata for a particular Commit.
5	git reset –hard	Rolls back all history and changes for a specific Commit.
6	git reset –hard Head	Rolls back all local changes in the working directory.
7	git log	Displays change history.
8	git log -p	Displays the full page for each Commit.

9	git log -oneline	Displays a list of Commits and a simple message.
10	git log –follow (file_name)	Shows the historyof the present file.
11	git blame (file_name)	Displays all changes and the user's name.
12	git stash	Does an Interim save of all tracked files that have been modified.
13	git stash pop	Restores files that were stashed most recently.
14	git stash list	Displays all stash changesets.
15	git stash apply	Applies the most recent stashed contents.
16	git stash drop	Gets rid of the most recently stashed files
17	git stash apply (stash id)	Re-applies content of a particular stash by ID.
18	git stash drop (stash_id)	Drops particular stash content by ID.
19	git push	Pushes changes to the Origin.
16	git push origin (branch_name)	Pushes branch to the Origin.
17	Git push -f origin(branch_name)	Force pushes the changes to the Origin.
18	git tag (tag_name)	Specifies a tag for a version.
19	git push	Pushes changes to the Origin.

Git Commands Cheat Sheet – Branching

Sr. No.	Git Command	Description
1	git branch	Displays a list of every branch.
2	git branch	Makes a new branch.
3	git branch -m	Changes the name of a branch.
4	git branch -a	Lists both local and remote branches.
5	git checkout -b	Creates a branch and switches to it.
6	git checkout	Changes to a particular branch.
7	git checkout -b origin/	Puts a remote branch from the origin in the local directory.
8	git branch -d	Deletes the specified branch.
9	git merge	Merges the current branch with the master (first checkout to master)

10	git rebase	Integrates changes from one branch into another by rewriting the commit historyto produce a linear succession of commits.
11	git rebase	Rebases the current branch onto the base, which can be a Commit ID ora branch name.
12	git fetchremote	Fetches the specified branch from the repository.
13	git diff..	Shows the differences between two branches.
14	git pull –rebase	Fetches the remote copy of the current branch and rebases it into the localcopy.

Conclusion: Hence, we have performed various git operations on local and remote repository using git cheat-sheet.

Experiment No .4

Aim: To Install Jenkins on Windows.

Theory*:*

Jenkins is a Java based open-source application, which is one of the most popular tools for continuous integration and continuous delivery on any platform.

Jenkins is a self-contained server that can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software. It has many plugins for automating almost everything atthe infrastructure level. The use of Jenkins has widely increased due to a rich set of functionalities, which it provides in the form of plugins.

The Jenkins project has two release lines: Stable (LTS) and regular (Weekly). Depending upon the scenario and need, one can be chosen:

Stable (LTS): Long-Term Support (LTS) release baselines are chosen every 12 weeks from the stream of regular releases.

Regular releases (Weekly): This delivers bug fixes and new features rapidly (generally weekly) to users and plugin developers who need them.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

Software requirements:

Jenkins project performs a full test flow with the following JDK/JREs: JDK Installation :

Below are the steps of JDK installation, if not installed initially:

STEP 1: Navigate to the website. Scroll to Java SE11 (LTS) and click on JDK Download.

STEP 2: Scroll to Java SE Development kit 11.0.10 and click on Windows x64 Installer. STEP

3:Accept the Licence Agreement and download it.

STEP 4: You will be navigated to the Sign in page. If you don't have an account on Oracle, click on create account and proceed with the download.

STEP 5: Once downloaded, run the .exe to install JDK by double-clicking it. It might ask for permission before installation. Click on Yes to allow installer to execute. The Welcome Screen

comes first.

STEP 6: Choose the installation path for jdk and click Ok. We can also choose the default path as well.

STEP7: Click on NEXT to start installation. Once it is completed it shows the success message as shown in the image below. It is recommended to restart your system, further click on System Properties and choose Advance System Settings.

STEP 8: Choose the Environment Variable and set the User variable by assigning Variable name as JAVA_HOME and variable value as the path where Java is installed:

Click on Ok and then Apply.

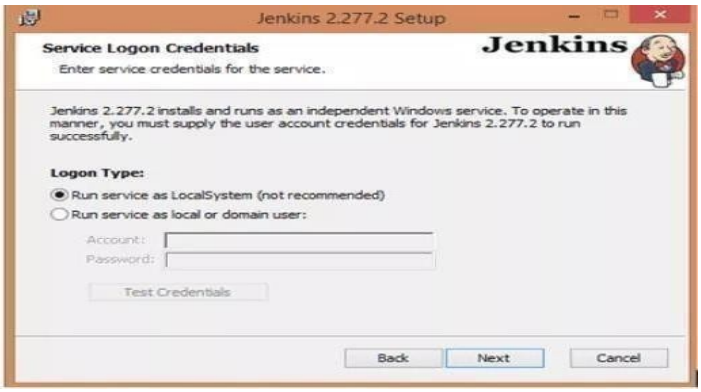
Jenkins Installation:

STEP 1: Navigate to the website and under Jenkins LTS system, click on Windows for the latestJenkins package

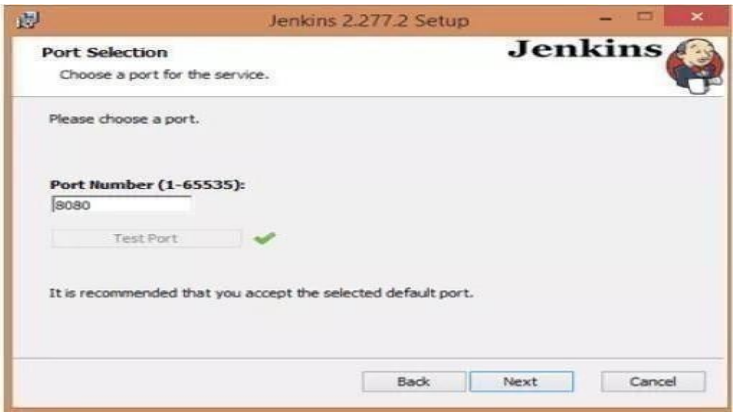
STEP 2: Once the jenkins.msi is downloaded, open it and proceed.

STEP 3: Choose the Jenkins path.

STEP4: For Service logon credentials, choose Logon type as "Run Service as LocalSystem".



STEP 5: Choose the port available on the system and test it (recommended is 8080).



STEP6: Choose the JDK path and click Next.

STEP 7: Click the "Install" button to start the installation process.

STEP8: Please wait till the whole installation is completed. STEP 9: Click on "Finish" after the installation is completed.

STEP 10: You will be redirected to a local Jenkins page by default. If page is not loaded by default, paste the URL http://localhost:8080 , with the chosen port in the browser.

After completing the Jenkins installation phase, we will proceed with its configuration set up.

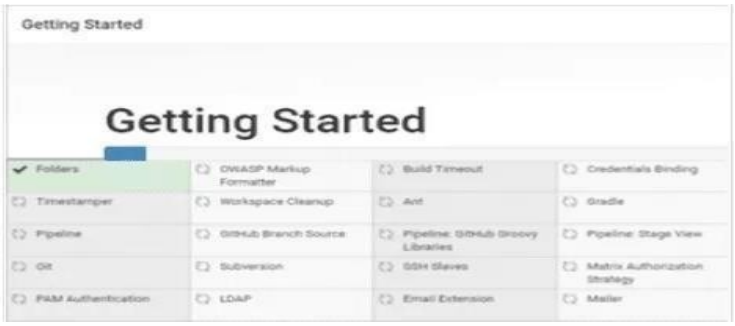
STEP 11: For Unlocking Jenkins, copy the password from the file

at C:\Windows\System32\config\systemprofile\AppData\Local\Jenkins\.jenkins\secrets (shown on

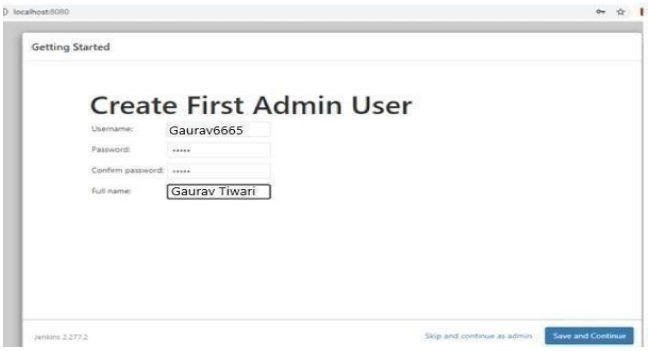
screen) and paste it in the Administrator password field. Click on Continue.

STEP 12: We can either choose the Jenkins suggested plugins or manually select plugins to install. It's preferable to choose "install suggested plugins". If not, we can install this later well.

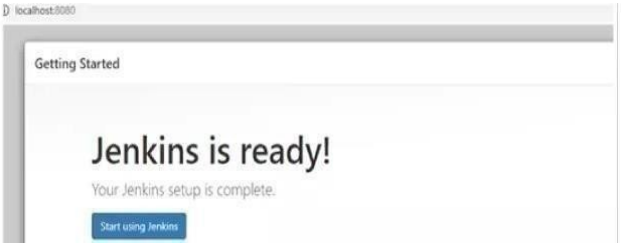
STEP 13: Choose "Install suggested plugins" and let the plugins install completely.



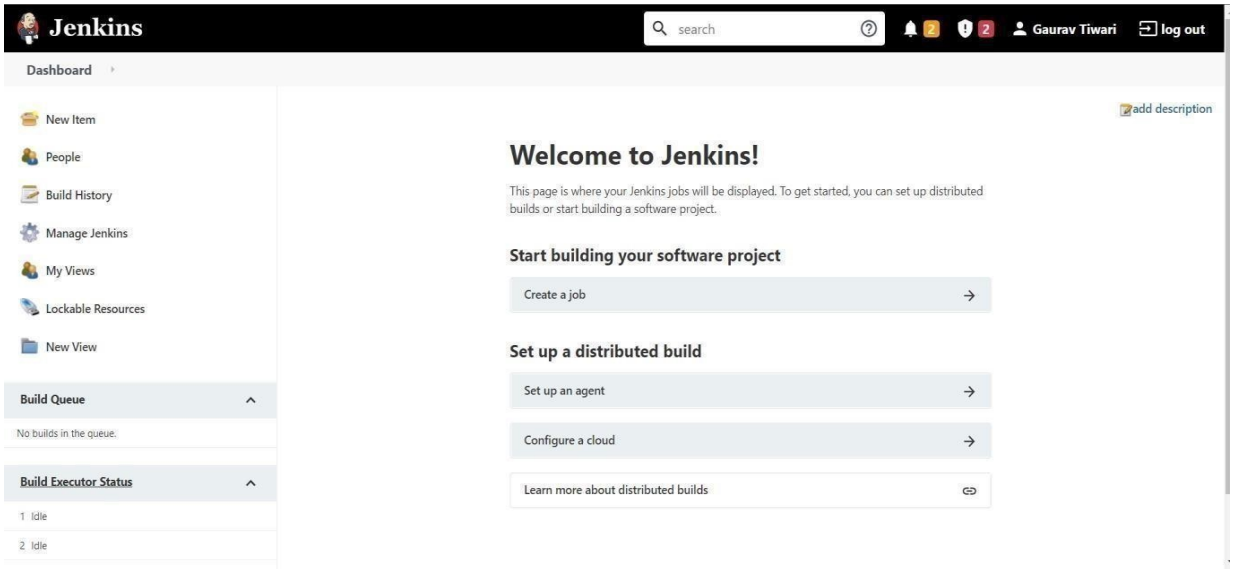
STEP 14: If you choose the Option "Select Plugins to Install", then select the plugins you want to install. STEP 15: Create an admin user account by filling up the details.



STEP 16: Click Save and Finish to complete the Jenkins set up. STEP 17: Click "Start usingJenkins" option.



STEP 18: Finally, the default Jenkins page will be shown as below.



Conclusion:

Hence, we have Seen Installation of Jenkins on Windows.

EXPERIMENT NO: 5 & 6

Aim: Executing java sequence programming and integration of GitHub with Jenkins.

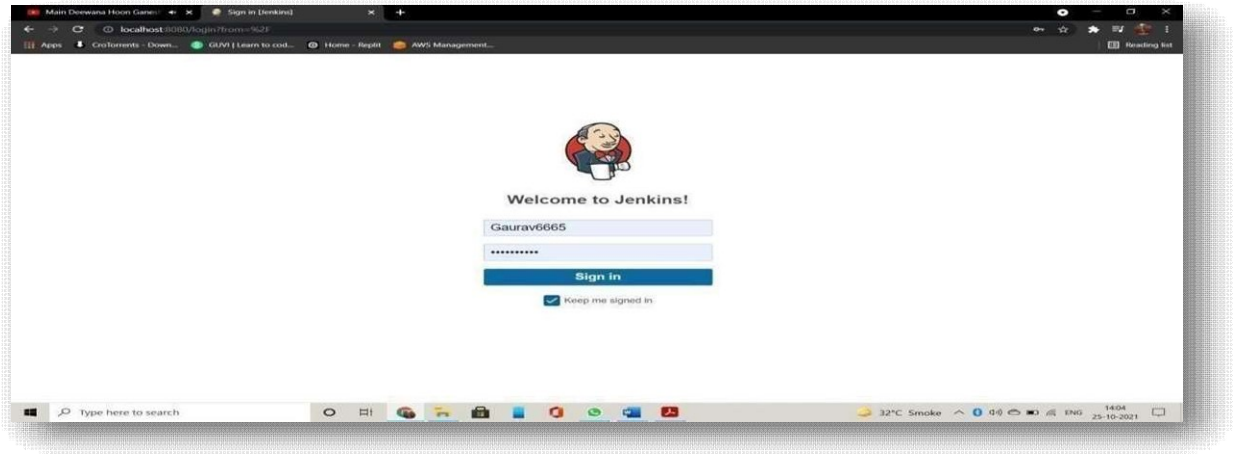
Theory:

Jenkins comes with a pretty basic setup, so you will need to install the required plugins to enable respective third-party application support.

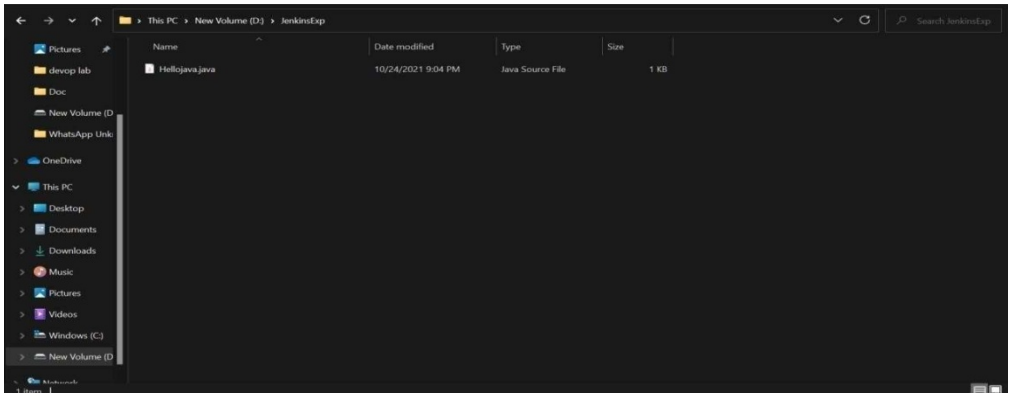
GitHub is a web-based repository of code which plays a major role in DevOps. It provides a common platform for multiple developers working on the same code/project to upload and retrieve updated code, thereby facilitating continuous integration.

Jenkins needs to have GitHub plugin installed to be able to pull code from the GitHub repository.

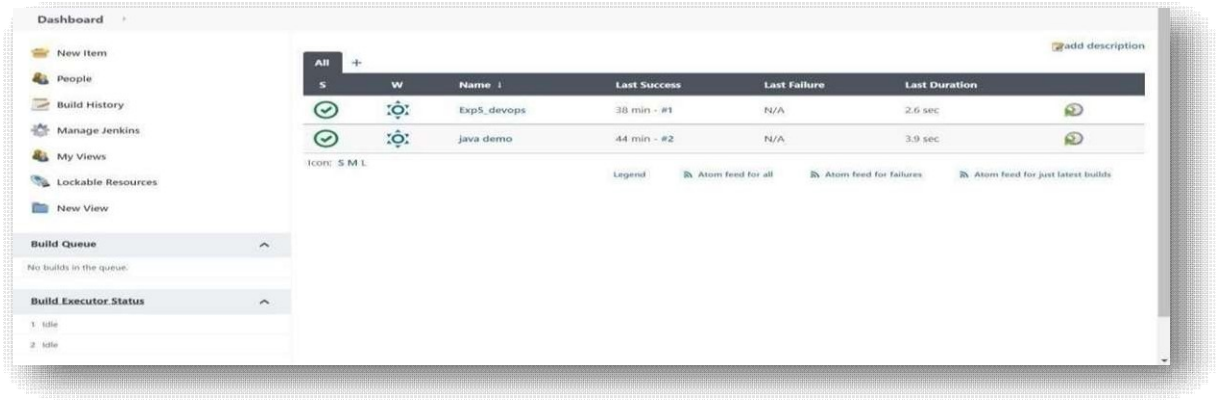
Step 1: Login to Jenkins.



Step 2: Go to any Folder and Create one java File.

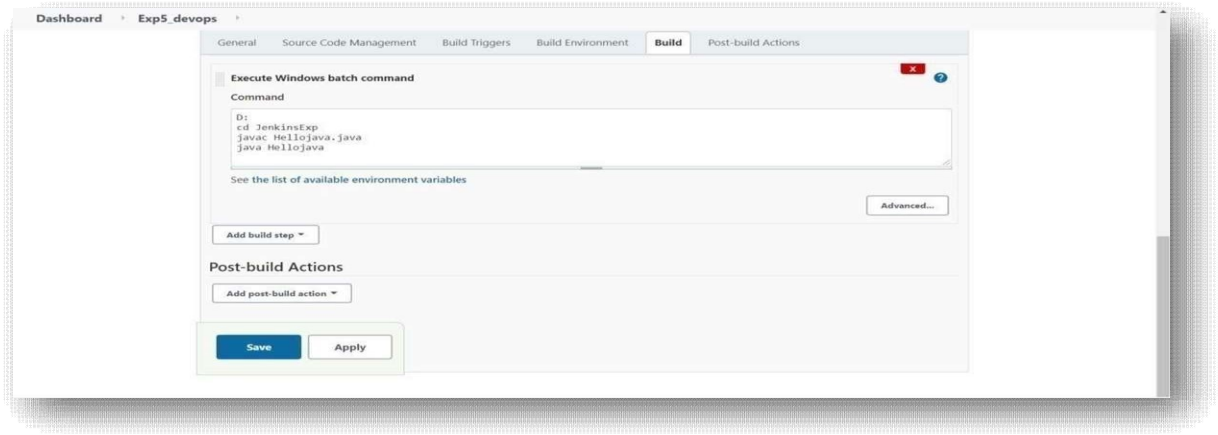


Step 3: Now Create New Job.



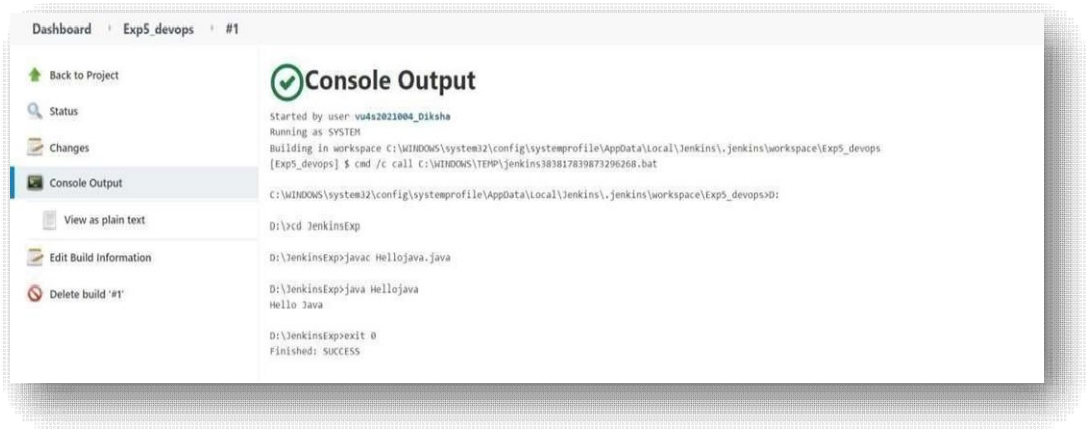
Go to “Build” section and add build step “Execute Windows batch command”

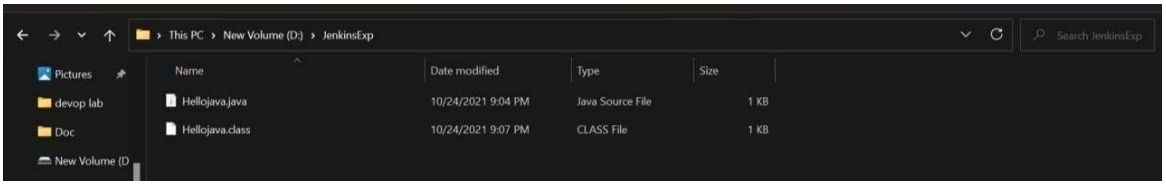
Go to “Build” section and add build step “Execute Windows batch command”



Execute "Hello java" by clicking on "Build Now" link.

If the job is successful, then it will show Blue ball otherwise Red ball under "Build History"

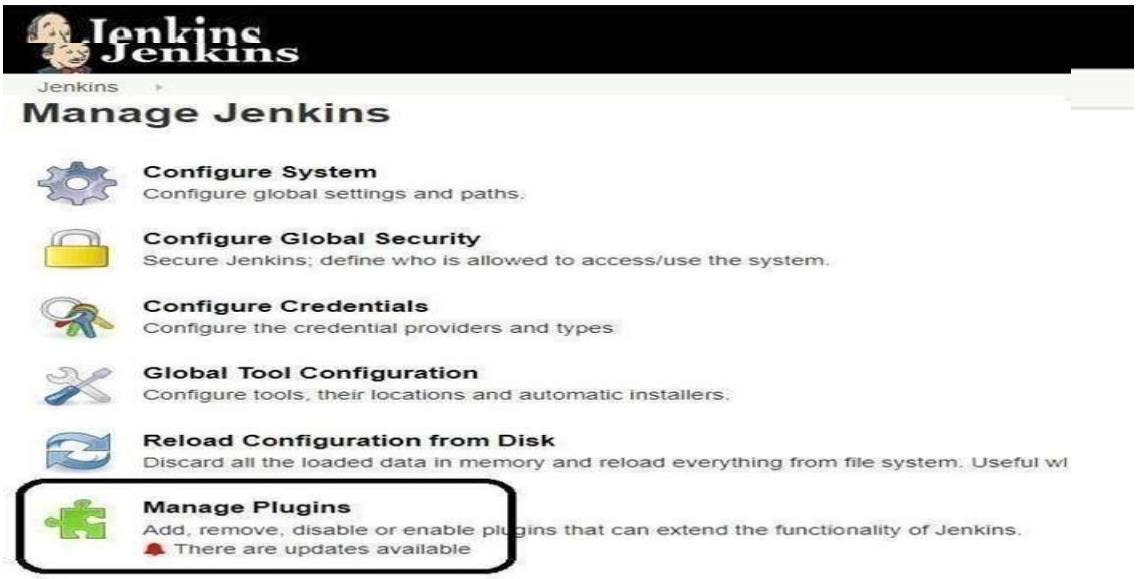




How to Install Git Plugin in Jenkins:

Following is a step-by-step process on how to Install Git plugin in Jenkins:

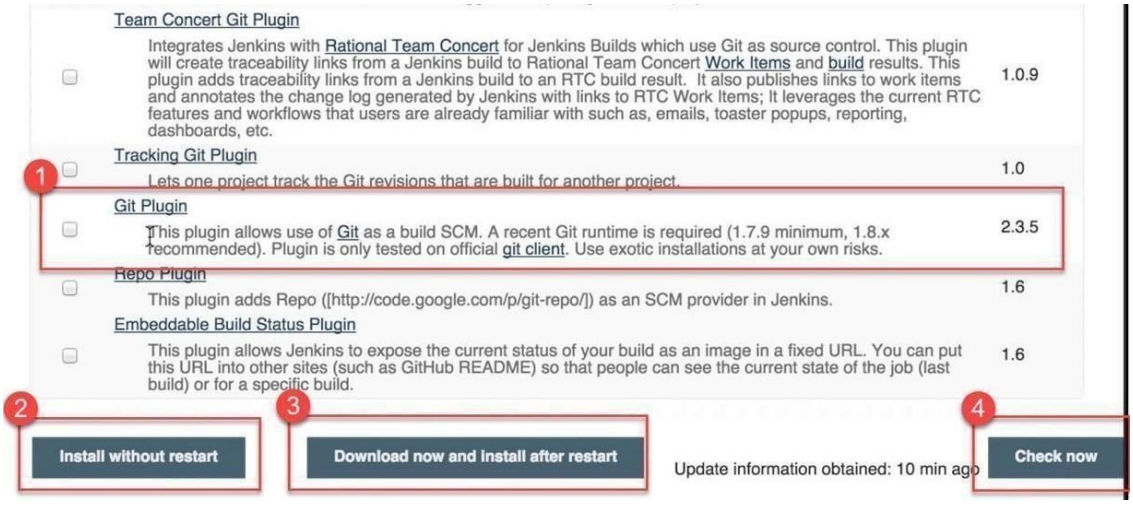
Step 1: Click on the **Manage Jenkins** button on your Jenkins dashboard:



Step 2: Click on **Manage Plugins**:

Step 3: In the Plugins Page

1. Select the GIT Plugin
2. Click on **Install without restart.** The plugin will take a few moments to finish downloading depending on your internet connection, and will be installed automatically.
3. You can also select the option **Download now and Install after restart** button. In which plugin isinstalled after restart
4. You will be shown a "No updates available" message if you already have the Git plugin installed.



Step 4: Once the plugins have been installed, go to **Manage Jenkins** on your Jenkinsdashboard. You will see your plugins listed among the rest.

How to Integrate Jenkins With GitHub:

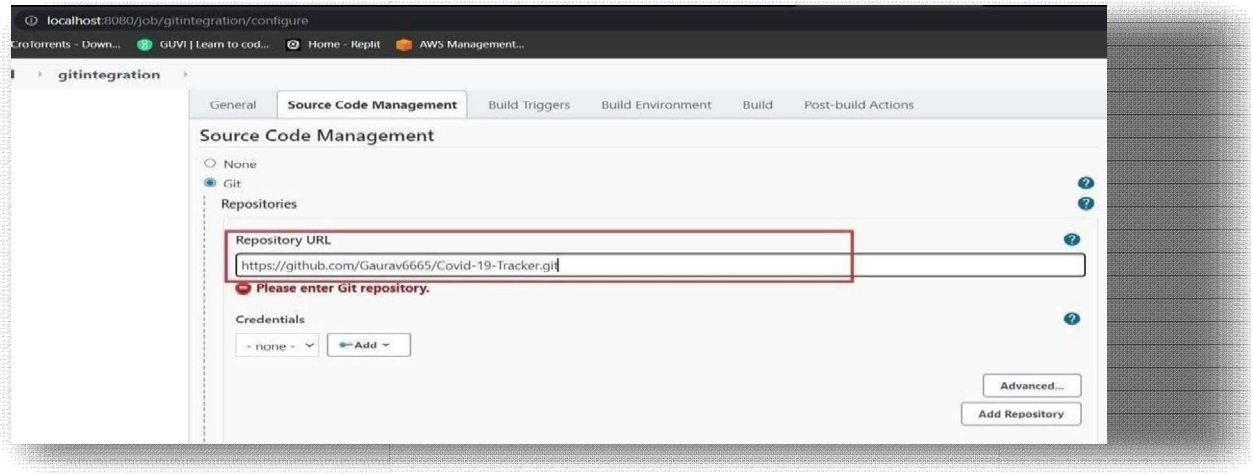
The process of integrating Jenkins and GitHub a Windows system:

Step 1) Create a new job in Jenkins, open the Jenkins dashboard with your Jenkins URL. For example, http://localhost:8080/ Click on **create new jobs**:

Step 2) Enter the item name, select job type and click **OK**. We shall create a Freestyle project as an example.

Step 3) Once you click **OK,** the page will be redirected to its project form. Here you will needto enter the project information:

Step 4) You will see a **Git** option under **Source Code Management** if your Git plugin has been installed in Jenkins:



Step 5) Enter the Git repository URL to pull the code from GitHub.

Conclusion:

Hence, execute java sequence program in Jenkins integrated from Git Hub.

Experiment No .7

Aim: Simple maven project execution in Jenkins from GitHub.

Theory:

Maven is a build automation tool used primarily for Java projects. Maven can als be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project.

Maven's primary goal is to **allow a developer to comprehend the complete state of a development effort in the shortest period of time**. In order to attain this goal, Maven deals with several areas of concern: Making the build process easy. Providing a uniform build system.

POM is an acronym for **Project Object Model**. The pom. xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory,source directory, test source directory, plugin, goals etc. Maven reads the pom.

- **Tools and technologies used :**

- Eclipse Neon
- Maven - 3.5.3
- JDK - 1.8

- **Creating step by step a simple maven project in Eclipse IDE :**

Install Eclipse ID :
Following is a step by step guide to download and install Eclipse IDE:

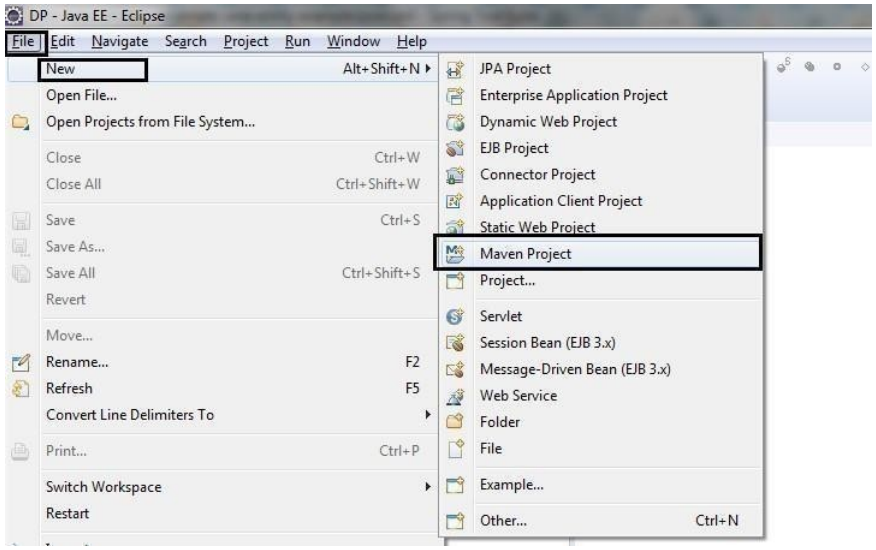
Step 1) Installing Eclipse

Open your browser and type https://www.eclipse.org/

Step 4) Click on "Download" button and Install & Run It.

Step-1

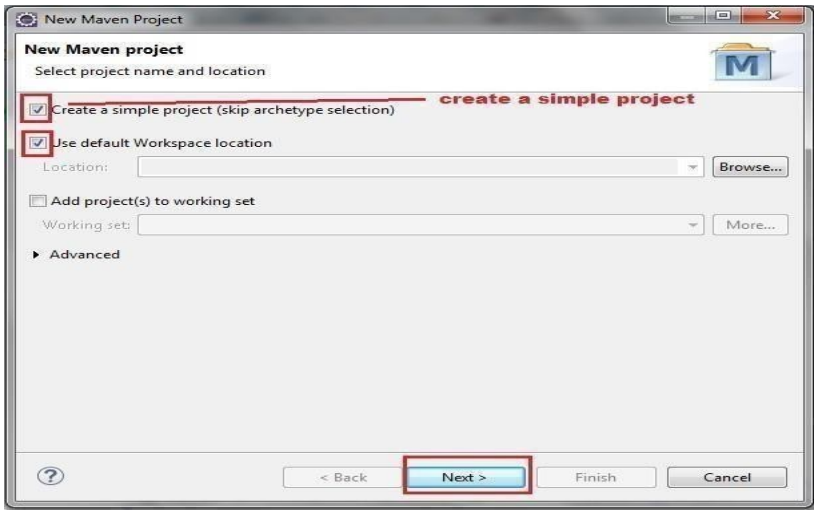
- Open Eclipse
- Click on *File* -> *New* -> *Maven Project*



Step-2

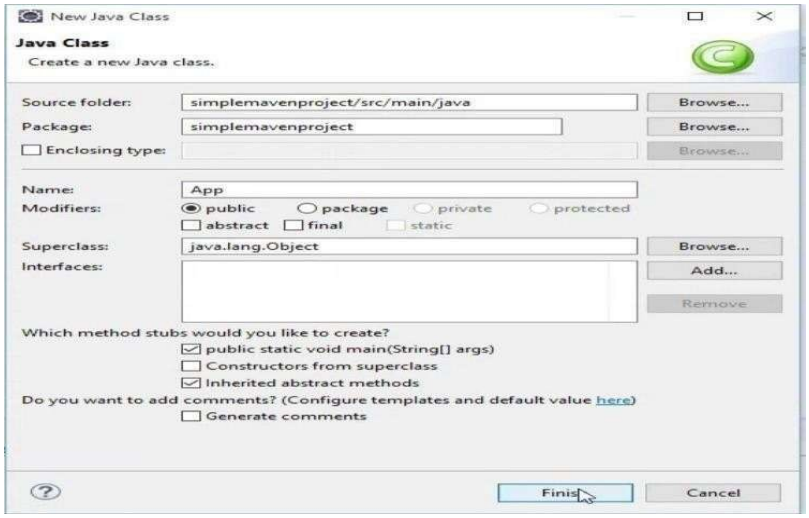
Click on Checkbox for both

- Create a simple project
- Use default Workspace location
- Click on next button



Step-3

Provide GroupId and ArtifactId in next screen.



Step-4

And you are all set. You should see a new Project in Eclipse with below structure.

Step-5

As you can see in the maven project structure, the default java compiler version (i.e. source and target setting) is 1.5. To change the default settings, add the following snippet to **pom.xml**.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.6.0</version>
<configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```
<project
   xmlns="http://maven.apache.org/POM/4.0.0"          xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"                                              xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
   <groupId>net.javaguides.maven-demo</groupId>
   <artifactId>maven-demo-project</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <description>Simple Maven Demo Project</description>
   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.12</version>
       </dependency>
    </dependencies>
   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.6.0</version>
            <configuration>
               <source>1.8</source>
               <target>1.8</target>
            </configuration>
         </plugin>
      </plugins>
   </build>
</project>
```

maven-project → *Maven* → *Update Project*. **Step-6 :**Add some dependencies to **pom.xml** file. Here we are adding *junit* dependency to **pom.xml.**

Let's test our created project by creating a simple JUnit test.

Step-7

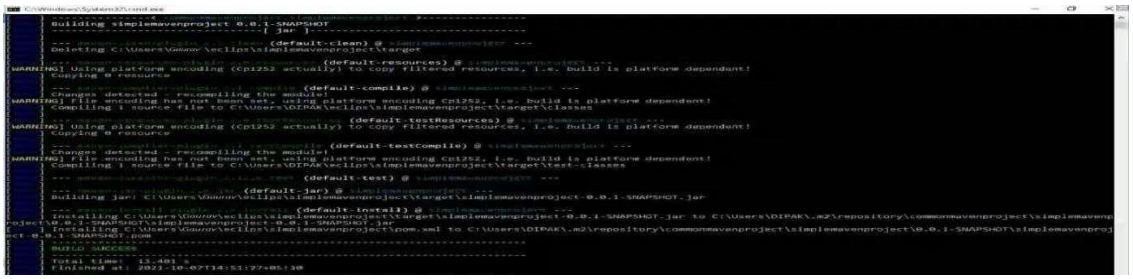
Create a package, named as *net.javaguides.simpleproject*, under *src/test/java* folder. Nowcreate a class *AppTest.java* under *src/test/java* package and write the following code in it.

```
package net.javaguides.simpleproject; import
org.junit.Assert; import org.junit.Test;

public class AppTest {
   @Test
   public void test() {
      Assert.assertEquals("Hello Maven", new String("Hello Maven"));
   }
```

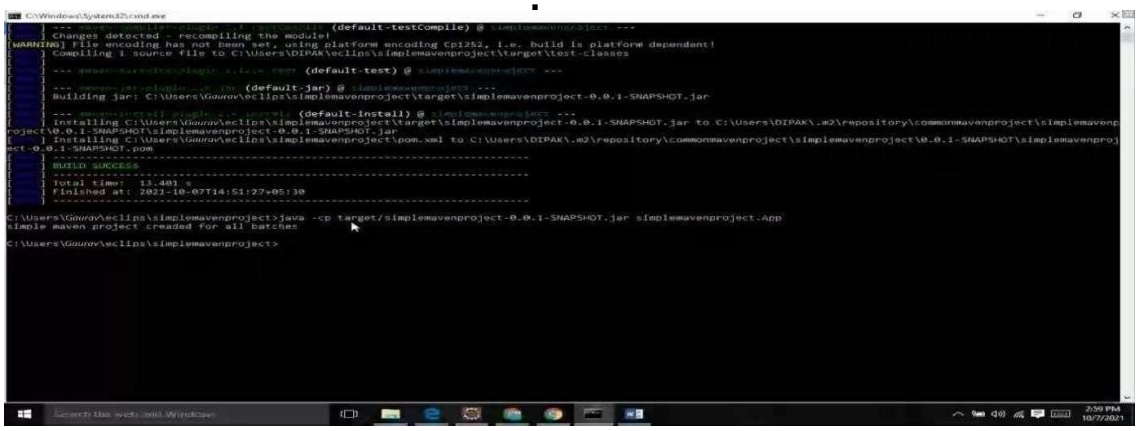
Step-8

Run your first maven project. Right click on *AppTest.java* → Run as → JUnit Test.

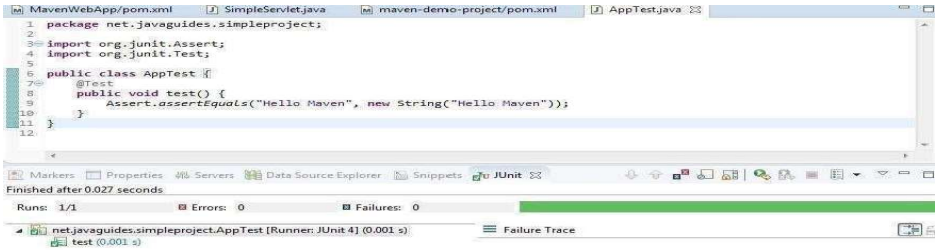
Maven will start building the project. You can see the output in CMD as follows



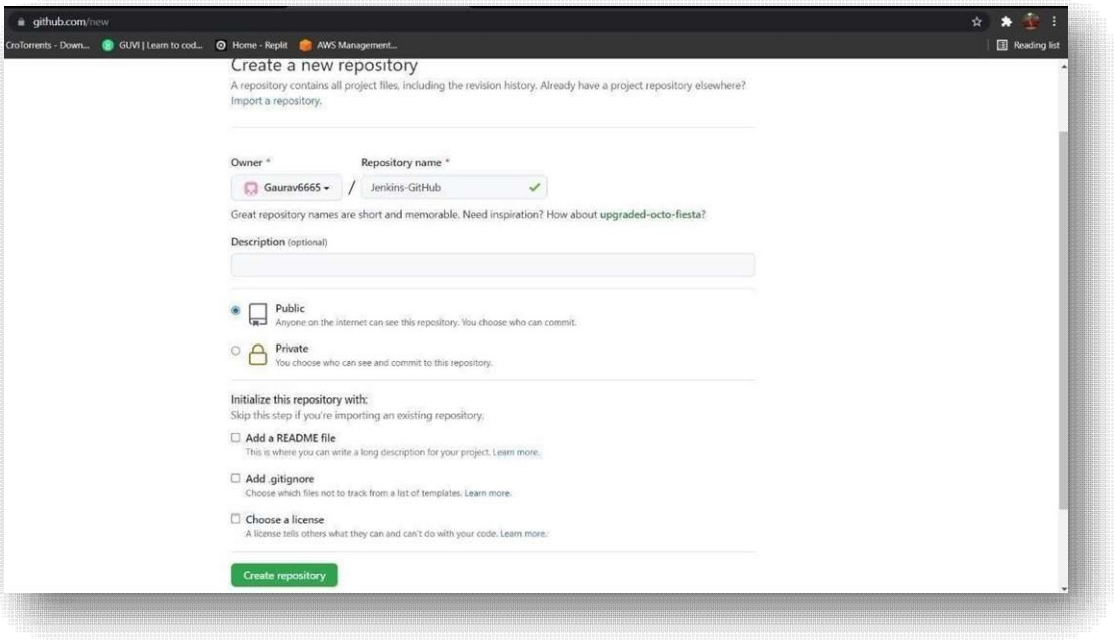
:



As, The Maven Project Is Built Successfully.

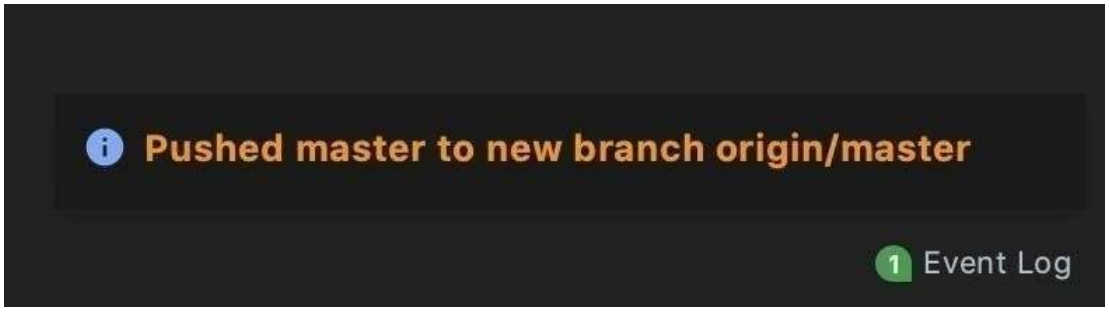


Commit and P ush Maven Projectto Git Hub :



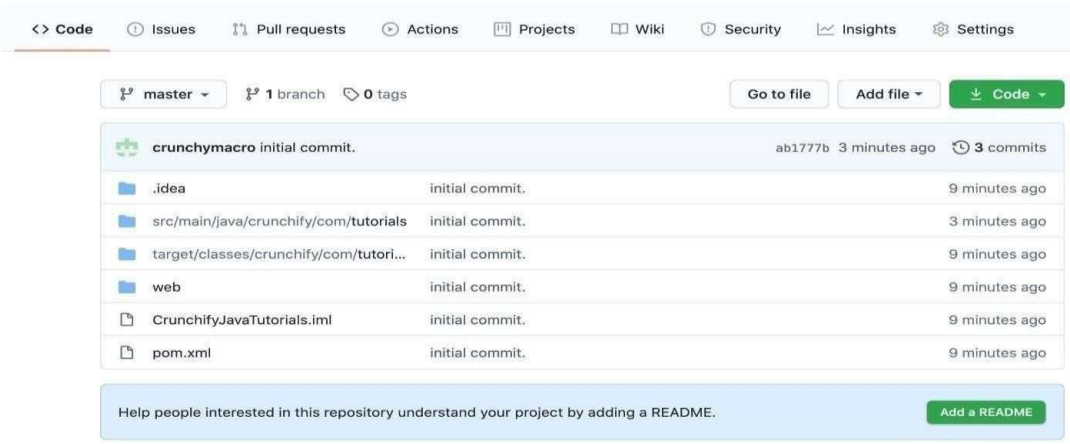
Step-2.

Once you push code, you should see success message as
Pushed to new branch origin/master



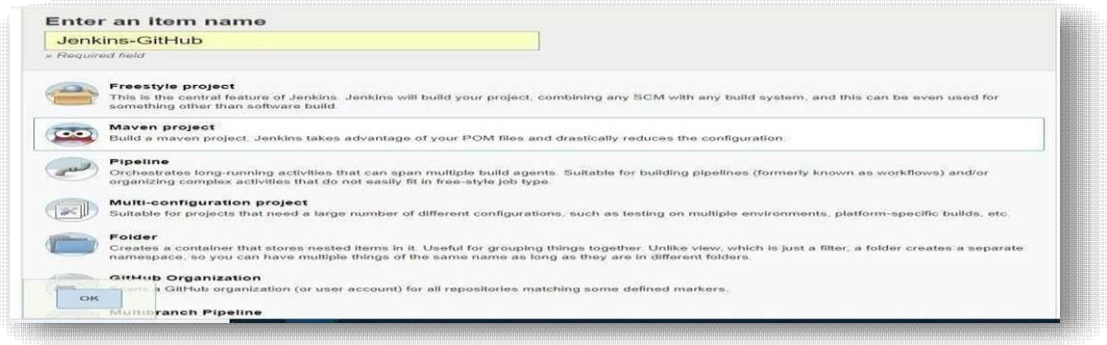
Just visit your Github repository and you should see newly pushed files.

Congratulations. You have successfully integrated Github



Jenkins-GitHub with Java Maven Project:

1) Create a new Job by clicking **New Item**
2) Enter your Job Name **'Jenkins-GitHub'** and select ***Maven Project*** then click **OK**, You will be navigated to configure the Job.



If Maven Project is **not visible** in your Jenkins then install plugin named as **'Maven Integration'**

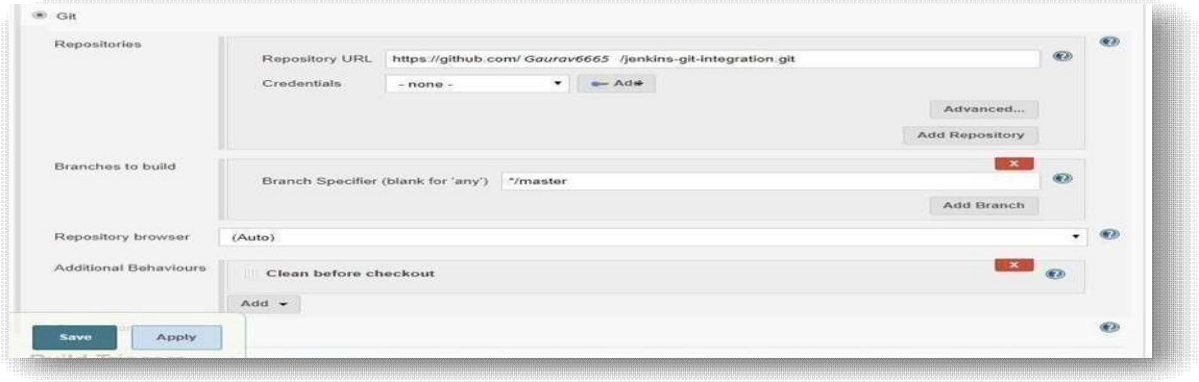
3) In job configuration under **General** Section tick **Github** project and provide your

Github- h***ttps://github.com/Gaurav6665/jenkins-gitintegration.git*** this will provide you link to GitHub from Job dashboard and it is optional.

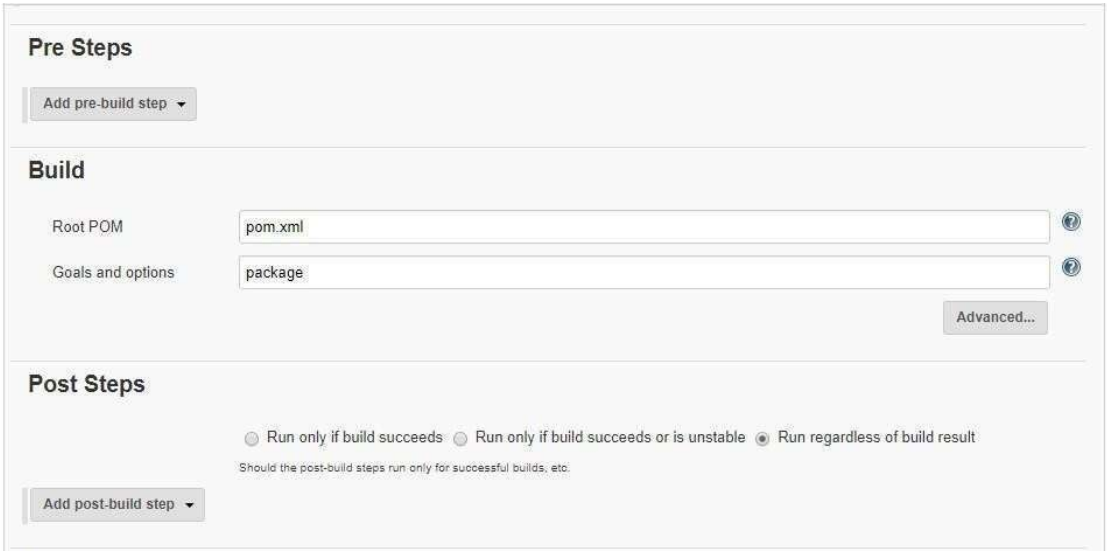
4) Under **Source Code Management** section click on **Git** radio button and provide RepositoryURL- h***ttps://github.com/Gaurav6665/jenkins-gitintegration.git***

5) Select *branch* to build if master- ***'*/master'*** if development- ***'*/development'***

6) You can add *behavior* by selecting value from drop down what to perform in my case I want to clean space before code checkout so I have selected ***Clean before chekout*** (you can configure according to your need this is optional)



7) Navigate to **Build Trigger**- If you want to build your project on *specific time interval* you can configure it under **Build Trigger** > tick **Poll SCM** and in text field * * * * * (five star separated by space for every minute, see bottom of this article for convention). If you will not set up Poll you will have to manually build the Job. Navigate to **Build** Section > provide

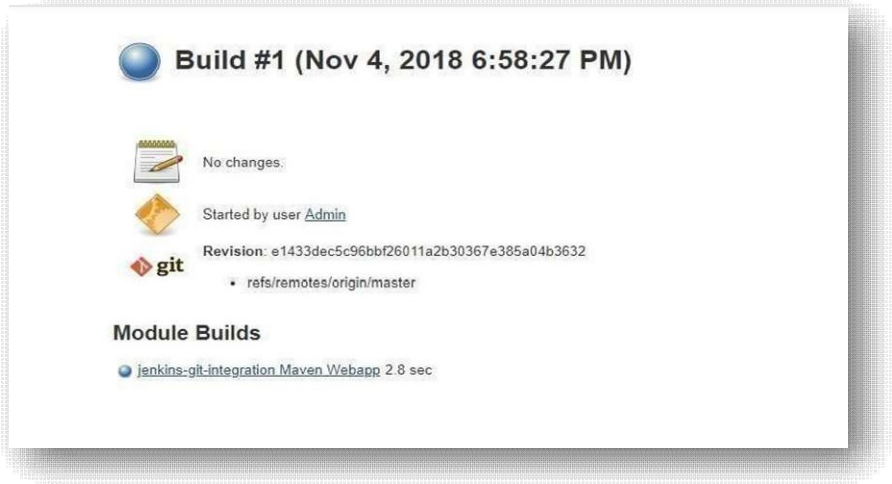


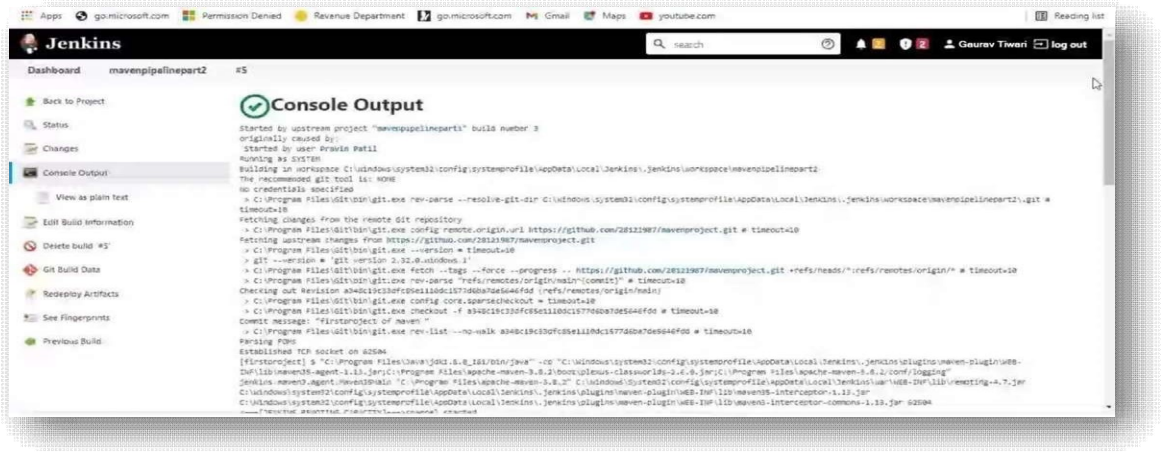
or
ge,
ns-
an,

Navigate to that project. You will have to click on **Build Now** if you have not configured *Poll SCM* else it will automatically trigger every minute (as we have set * * * * *). Your project will build successfully.

Go to project path and under target you will find the ***jar*** or ***war*** of your build project.

By clicking on blue circle you will be able to see logs.





Conclusion:

Hence, Simple maven project execution in Eclipse IDE, Pushed Maven Project into GitHub and

Experiment No .8

Aim: To Create Maven Pipeline.

Theory:

What is Jenkins Pipeline?

Jenkins Pipeline is a combination of Plugins which automates number of tasks and makes the pipeline efficient, high in quality and reliable.

What is Jenkins file?

Jenkins file is nothing but a simple text file which is used to write the Jenkins Pipeline and to automate the Continuous Integration process.

Jenkins file works as a **"Pipeline as a Code".**

Jenkins file is written in couple of way

- Declarative pipeline syntax
- Scripted pipeline syntax

1: Login to Jenkins:

First thing, we will login to our Jenkins account

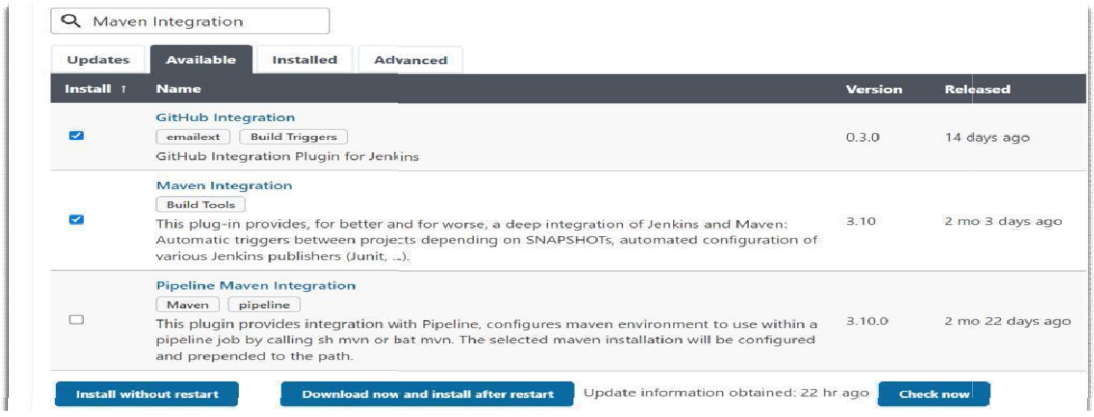
2. Install GitHub Integration and Maven Plugins in Jenkins:

To build Java project with maven we need to install some plugins like,

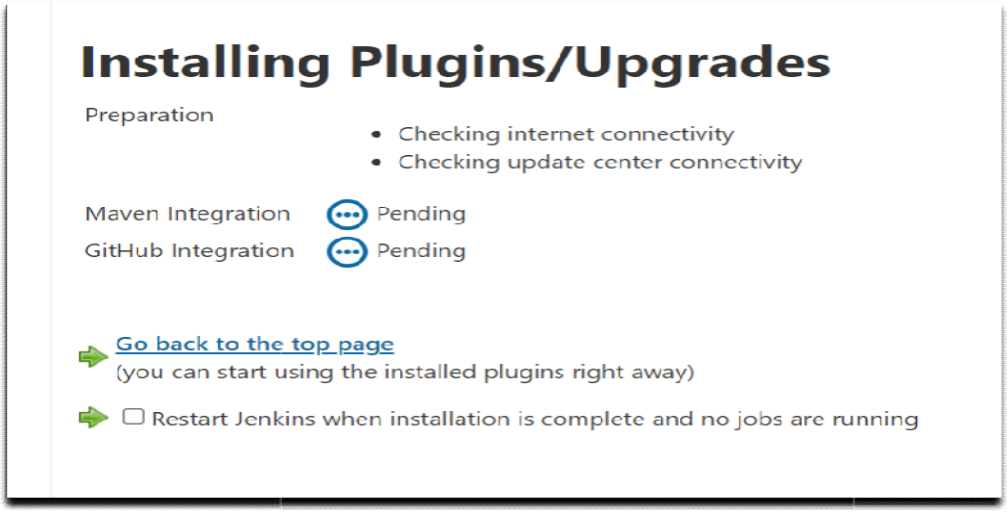
- GitHub Integration Plugin
- Maven Integration Plugin To install plugins follow below

stepsNavigate to **Dashboard->> Manage Jenkins ->> Manage Plugin**

Now search for **"GitHub Integration Plugin"** and click on **Download now and install after restart**.



once the plugins are downloaded successfully. So now click on **Restart Jenkins** checkbox.

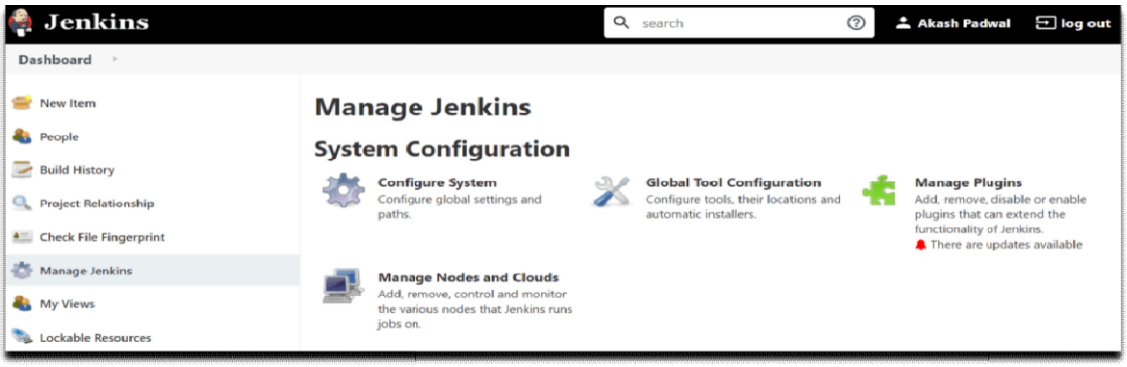


Please wait while Jenkins is restarting

3. Global Tool configuration in Jenkins:

Now we will configure System by adding JDK and Maven installation in Jenkins.

Navigate to **Dashboard->> Manage Jenkins->> Global tool Configuration**



Click on **JDK->> Add JDK**, Give a JDK name and provide your **JAVA_HOME** path where the JDK is present.

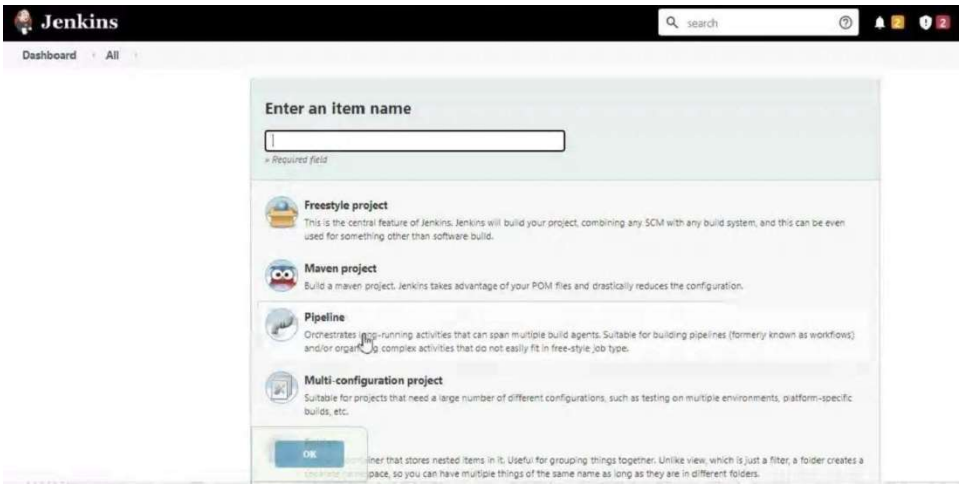
Now below click on **Maven->> Add Maven**, We can provide our Installed Maven path or we can also use Jenkins default one.In this case we are using Jenkins default Maven version.

Click on **apply** and **save.**

4. Create Pipeline Job in Jenkins:

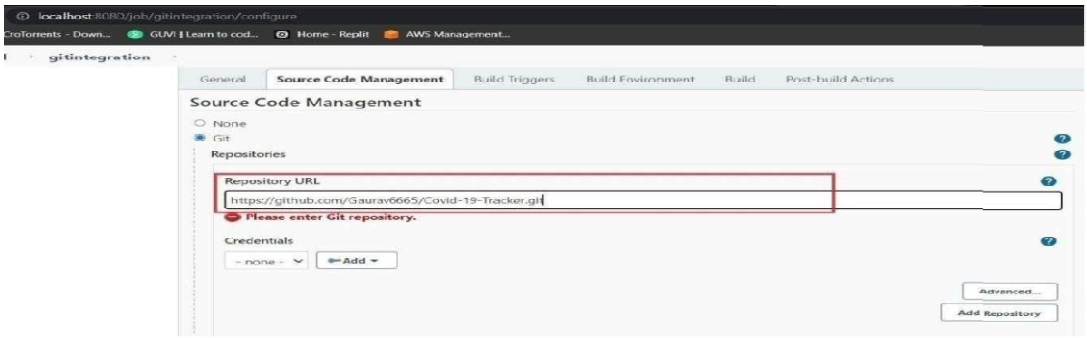
Now we will create a new Pipeline Job in Jenkins, So, click on **"New Item"** on the Jenkins Dashboardas shown below.

Enter Job name and select **"Pipeline"**, Click **OK.**

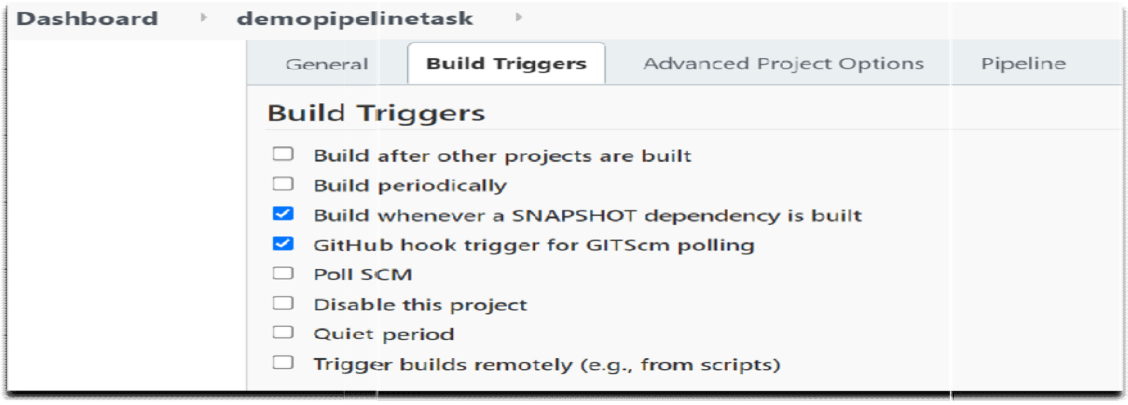


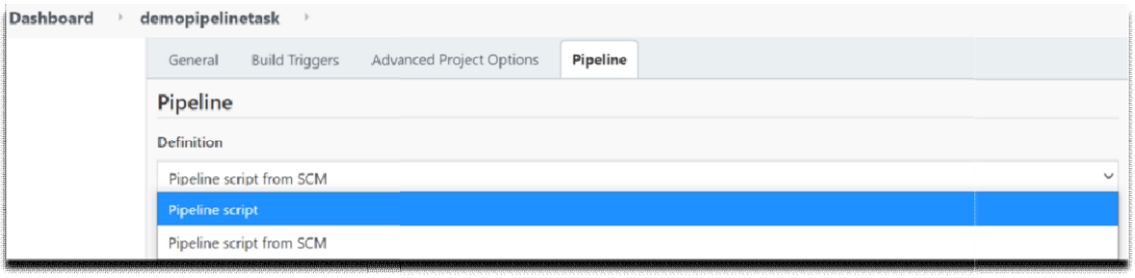
5. Enter Project details in Jenkins Pipeline

Provide some description for the newly created job in **General** Section, Click on **GitHub project** and provide the **GitHub URL**. you can use Sample Java Project on GitHub with jenkinsfile.

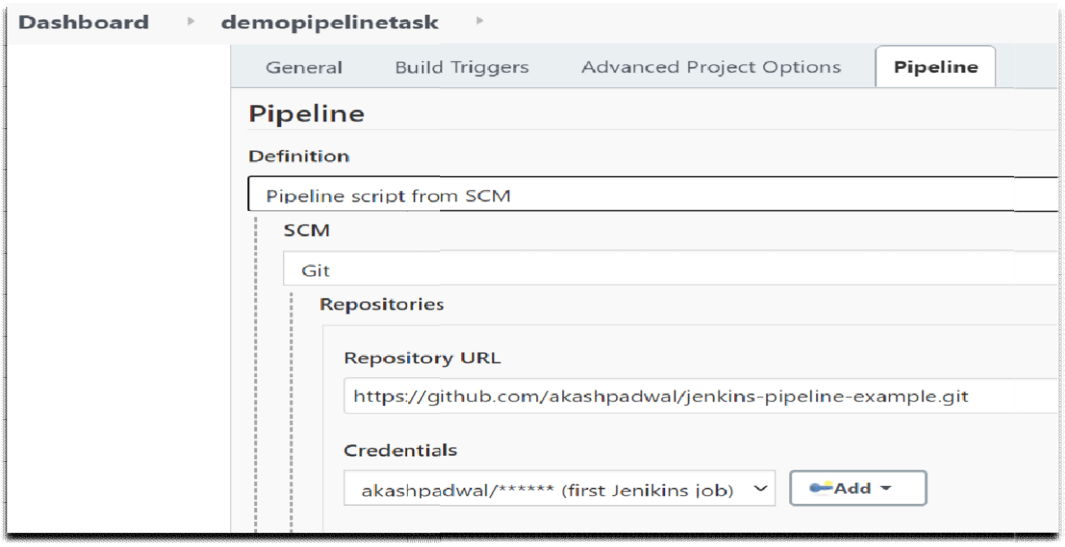


Now we will set the **Build Triggers**, Select **Build whenever a SNAPSHOT** and **GitHub hook trigger for GITScm polling** as triggers.



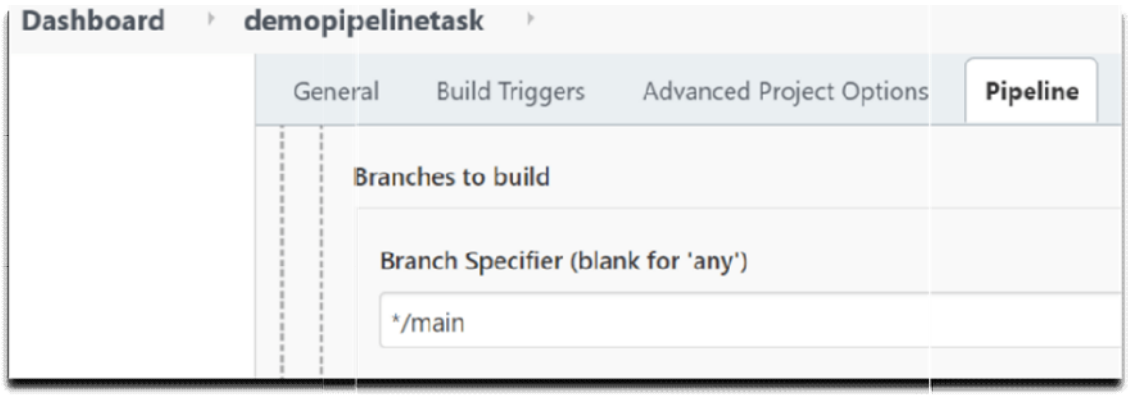


In SCM select **Git** and below it provide your Git repository URL where your project is located and Add your GitHub credentials.



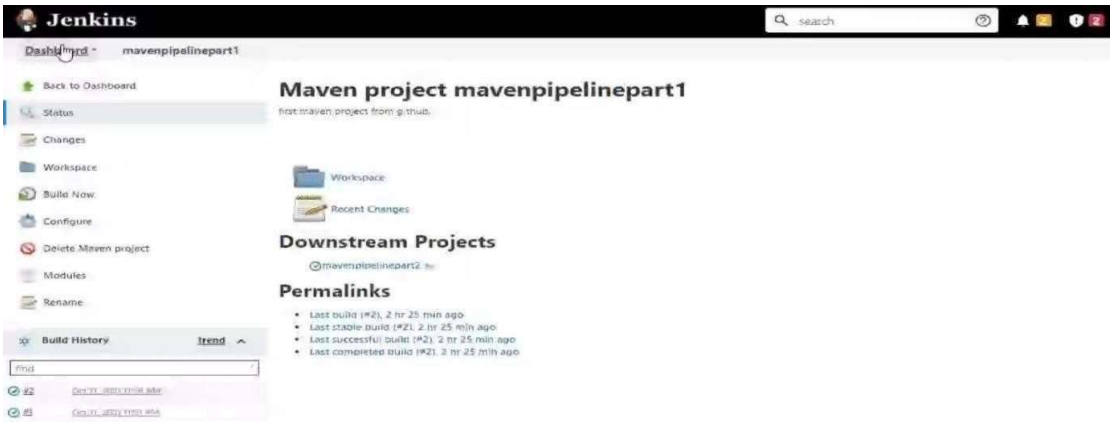
By default Jenkins provide branch as **Master** but you can change it to dev, main, or to what every our branch is.(if you don't know you can check the branch by going to your GitHub repository OR by running **git branch** command on your shell).

Our branch is **main** branch so will change **master** to **main**

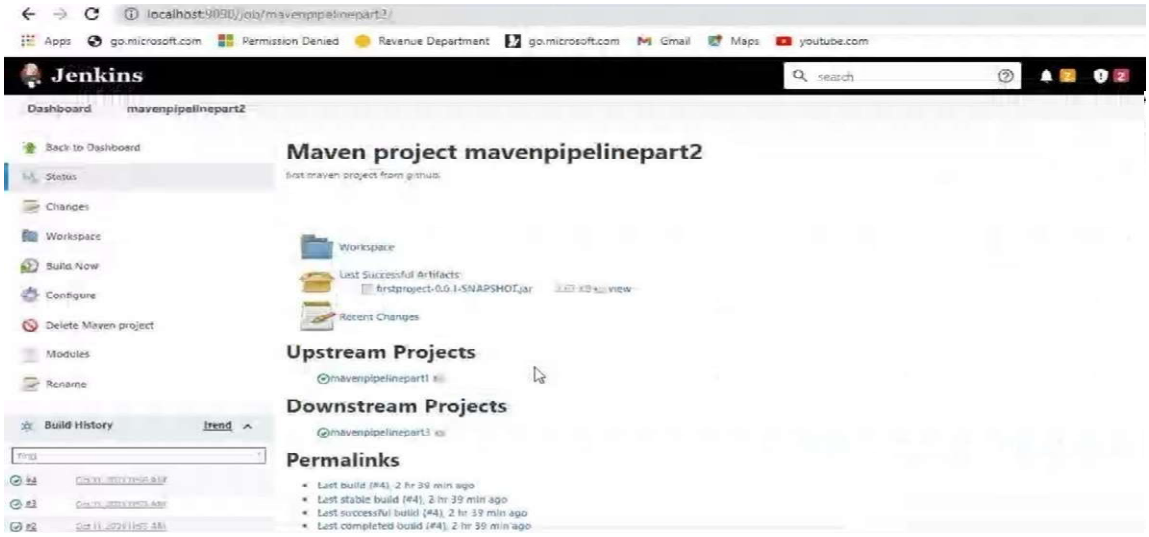


Now comes the most important part of your Project / Job configuration, Provide accurate path of your **Jenkins file.**

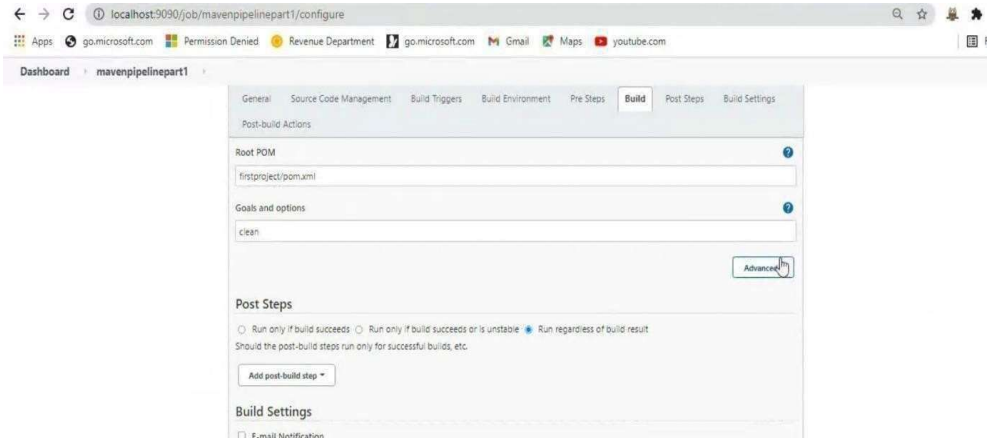
1) Create Maven Pipeline Part 1 as follows:



2) Set The Build Environment and Give Root POM and Goals:

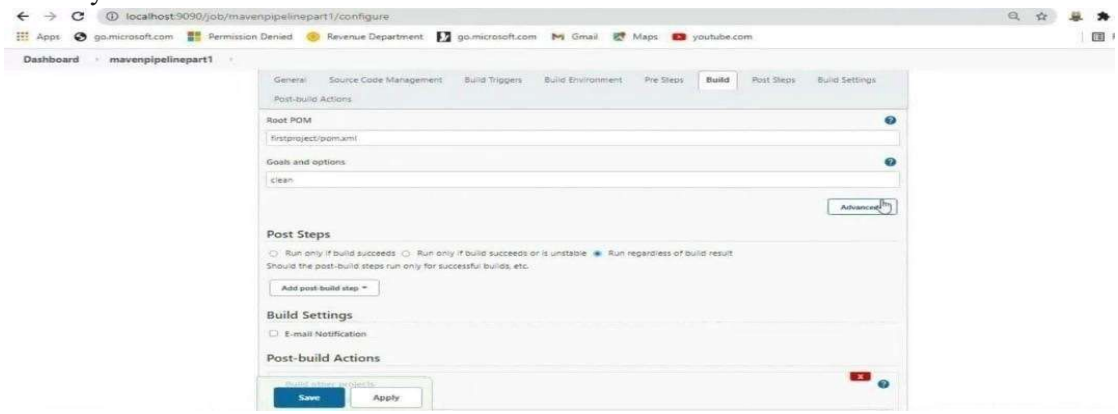


3) Set Up Post Step as Run Regardless of Build Result.

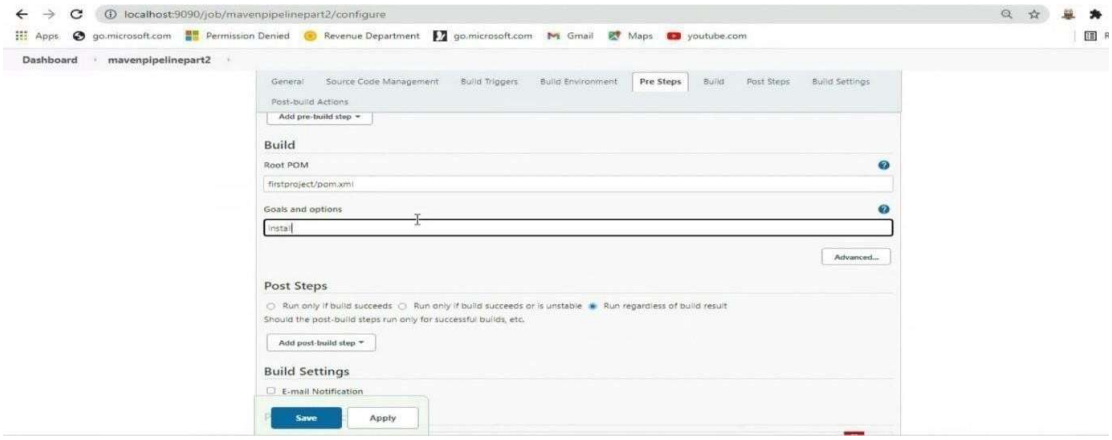


4) Now Similarly Build Maven pipeline part 2.

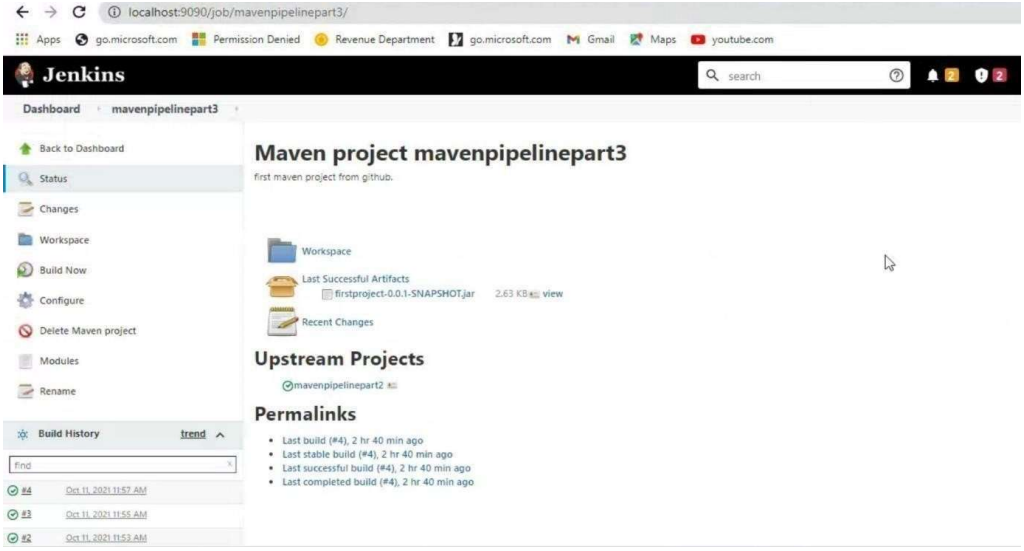
2) similarly set ROOT POM and Goals into BUILD.



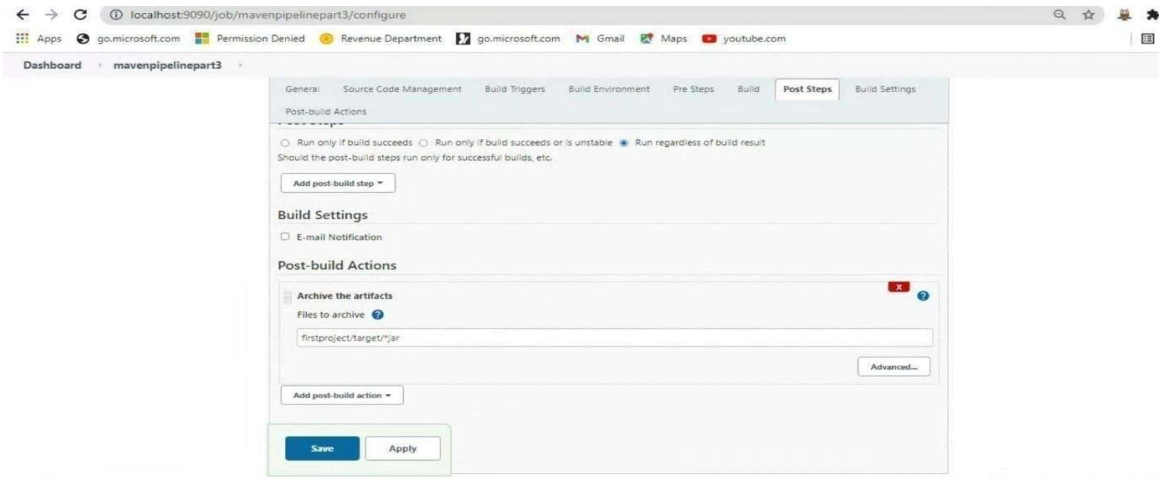
3) Give set ROOT POM and Goals into Pre Steps.



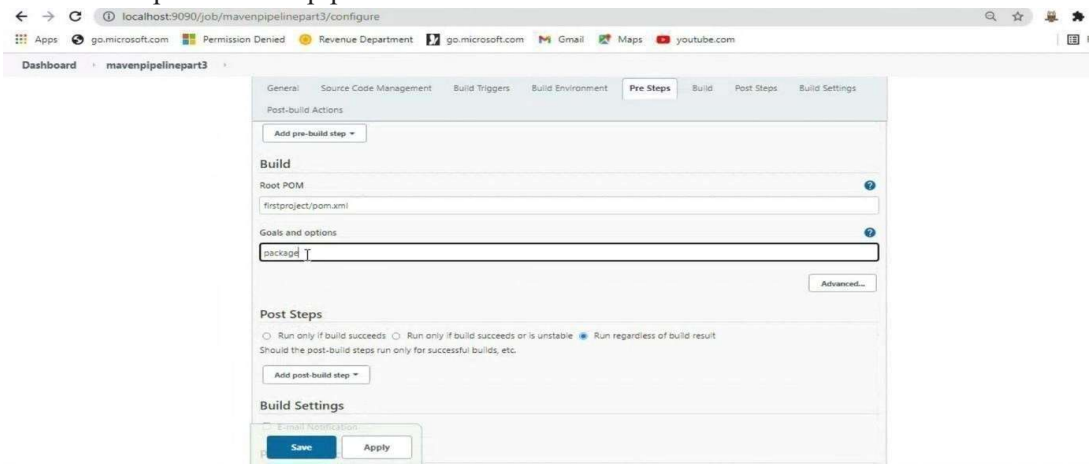
4) Now create maven pipeline part 3 similar to part 1 ans 2.



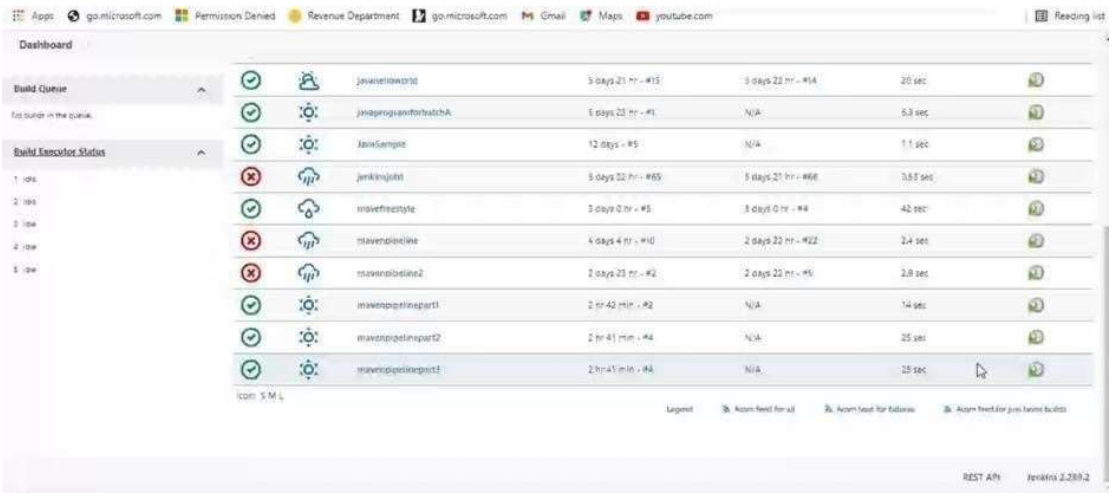
5) Give Post Steps Details and also provide Post Build Actions.



6) Give Pre-Steps Of maven pipeline Part 3.

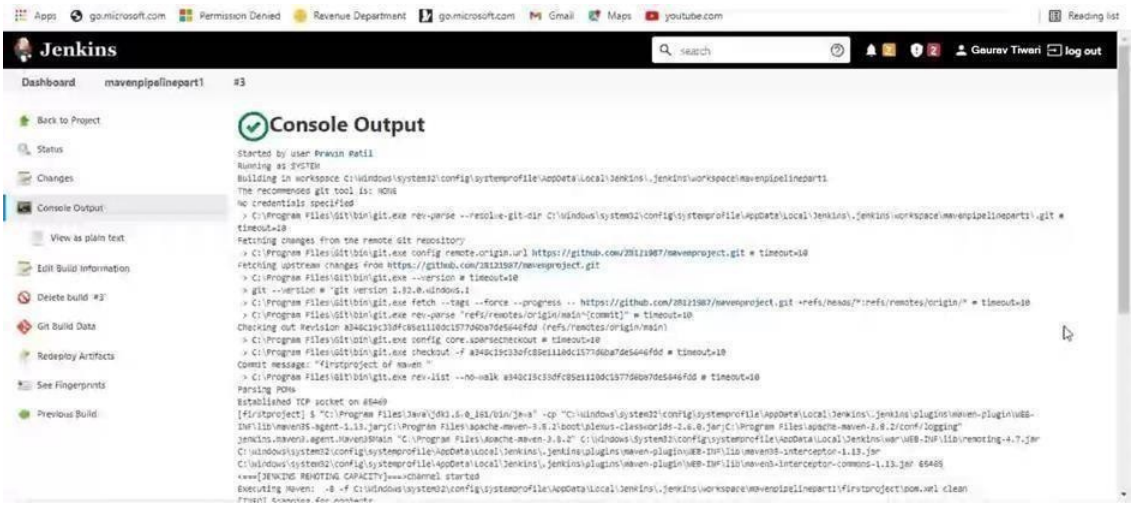


7) After building All 3 Maven Pipeline Part(1,2&3),Goto Dashboard Again to run it.

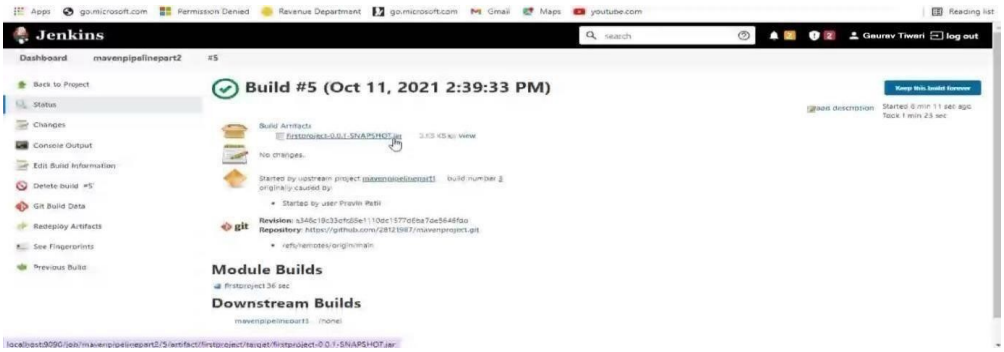


8) Execute & Run All Maven Pipeline part i.e 1,2&3.

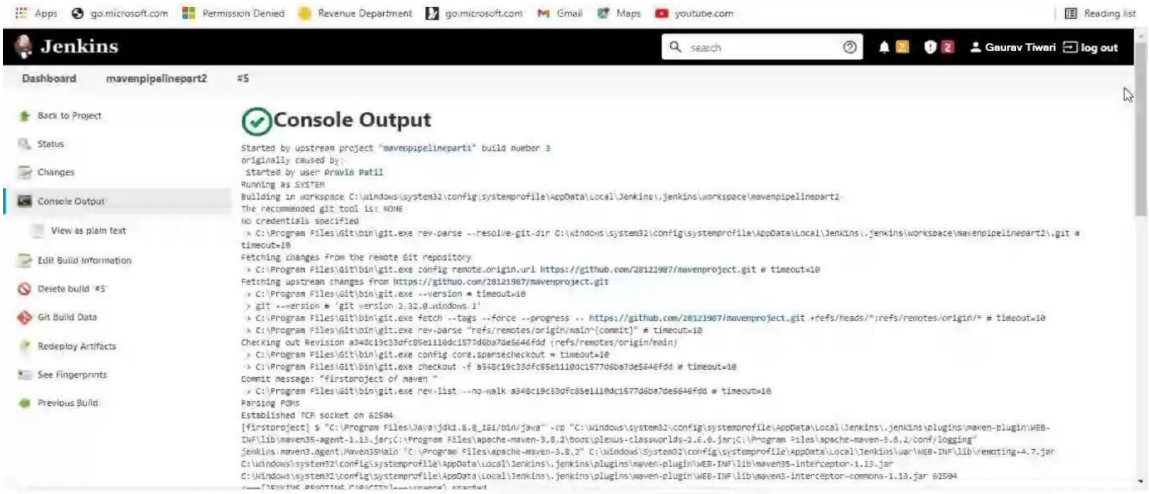
9) After Execution Check Console Output Of Maven Pipeline part 1.



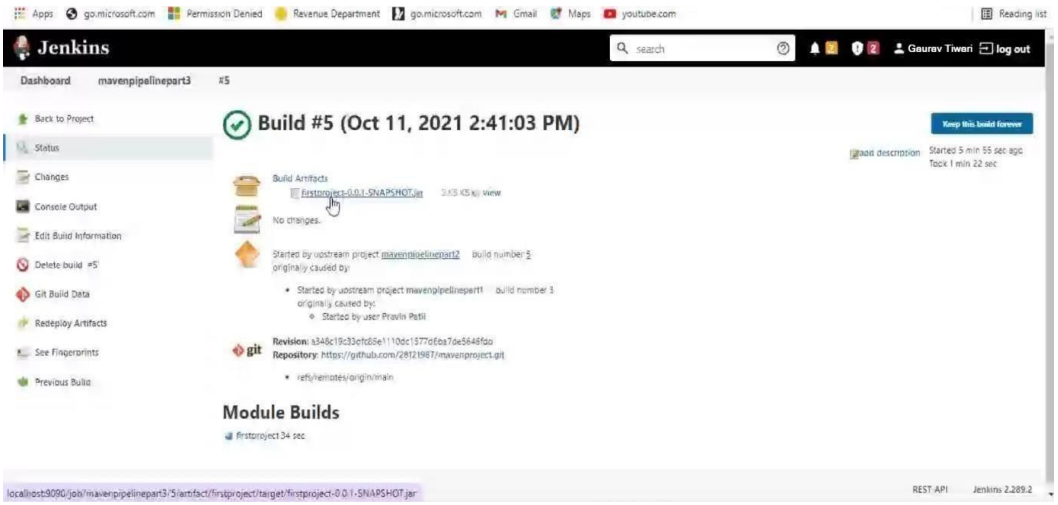
10) similarly, Check Console Output of Maven Pipeline part 2.



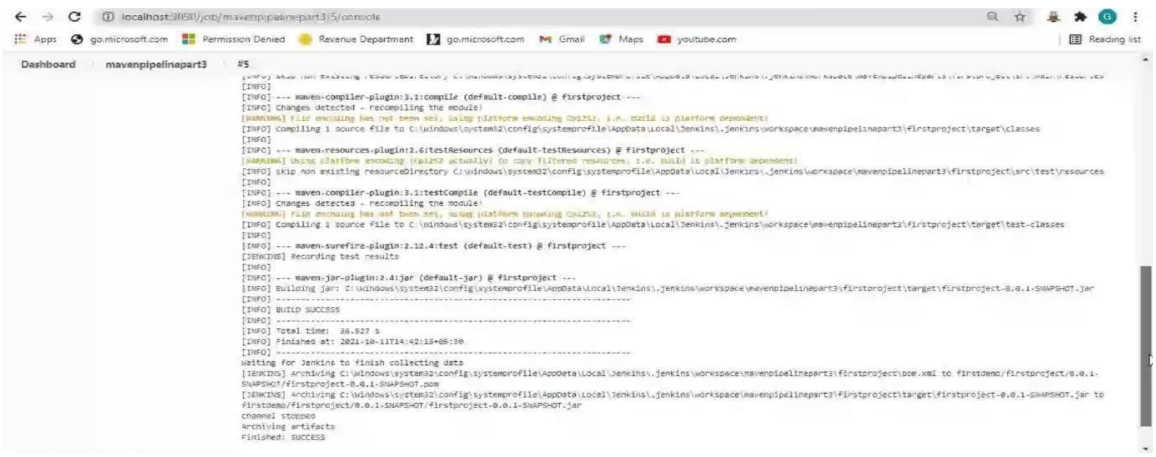
13) Output Of Maven Pipeline part2.



14) similarly, Check Console Output Of Maven Pipeline part 3.



15) As Maven Pipeline Is Built Successfully as Given Below.



Conclusion:

In this Experiment we studied How to Build Java Project Using Maven in Jenkins Pipeline. First we learned how to add a Build Stage and print message. Next we done with adding the tools like JDK and Maven. Then at last we explored how to Run a Maven pipeline parts (i.e. 1,2 & 3) Build Successfully.

Experiment No: 9

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

Theory:

Selenium tutorial provides basic and advanced concepts of Selenium. Our Selenium tutorial is designed for beginners and professionals.

Selenium is one of the most widely used open-source Web UI (User Interface) automation testing suite.

Our Selenium tutorial includes all topics of Selenium such as Features, Selenium vs QTP, Selenium Tool Suits, Selenium IDE, Selenium IDE Locating Strategies, Selenium WebDriver, WebDriver Features, WebDriver vs RC, WebDriver Installation, etc.

Selenium is one of the most widely used open-source Web UI (User Interface) automation testing suite. It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works. Selenium supports automation across different browsers, platforms and programming languages.

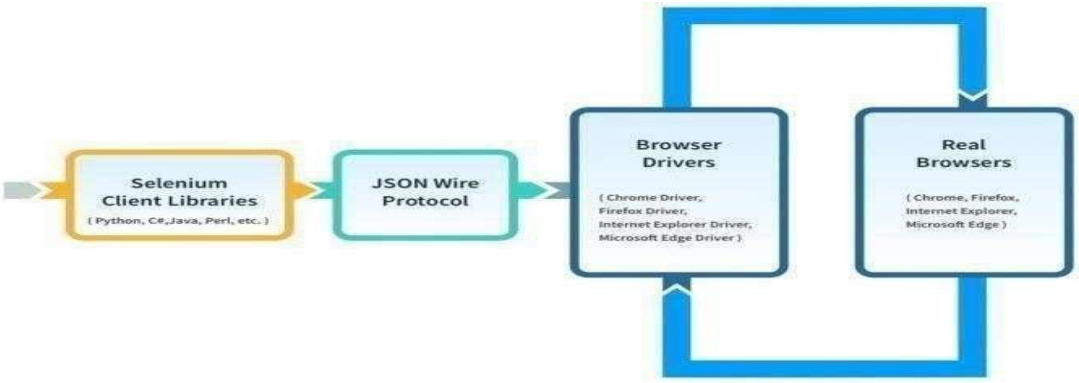
Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows mobile and android.

Selenium supports a variety of programming languages through the use of drivers specific to each language. Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby. Currently, Selenium Web driver is most popular with Java and C#. Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari.

Seven Steps of Selenium Tests

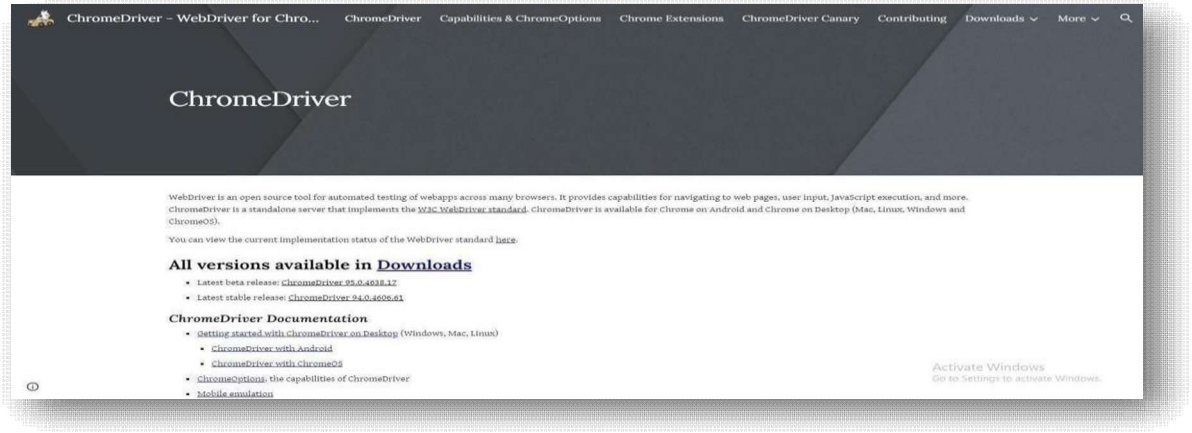
There are seven basic elements of a Selenium test script, which apply to any test case and any application under test (AUT):

1. Create a WebDriver session.
2. Navigate to a Web page.
3. Locate an HTML element on the Web page.
4. Perform an action on the located element.
5. Assert the performed action did the correct thing.
6. Report the result of the assertion.
7. End the session.



- The entire Selenium Software Testing Suite is comprised of four components:
- Selenium IDE, a Firefox add-on that you can only use in creating relatively simple test cases and testsuites.
- Selenium Remote Control, also known as Selenium 1, which is the first Selenium tool that allowed users to use programming languages in creating complex tests.
- WebDriver, the newer breakthrough that allows your test scripts to communicate directly to the browser, thereby controlling it from the OS level.
- Selenium Grid is also a tool that is used with Selenium RC to execute parallel tests across differentbrowsers and operating systems.
- Selenium RC and WebDriver was merged to form Selenium 2.
- Selenium is more advantageous than QTP in terms of costs and flexibility. It also allows you to runtests in parallel, unlike in QTP where you are only allowed to run tests sequentially.

1) Install a Chrome Driver so we can connect our code to the browser through it and run seleniumtestcases



How to set up Selenium for UI tests

Before exploring how to write Selenium UI tests, let's understand how to set up Selenium. First of all, the prerequisites mentioned below have to be met.

- Install NodeJS

  To check if the NodeJS and npm are correctly installed, enter the following commands:

  $ node -v

  $ npm -v

- Install Eclipse IDE
- Install **Selenium Server** Utility using npm, by entering the following command:npm install -g selenium-standalone
- Update Selenium Standalone Server and Browser Drivers to the latest versions, by entering thefollowing command:

  selenium-standalone install
- Run Selenium Standalone Server and the browser before running the test script by entering thefollowing command:

  selenium-standalone start
- Install Selenium WebDriver and Client Language Bindings o Download JavaScriptLanguage Bindings o Changelog o API Docs

  For a detailed demonstration of how to run a Selenium test, go through this SeleniumWebDriver tutorial.

How to write a test script for Selenium UI Testing:
Let's write the first test script using JavaScript. The code will navigate to www.browserstack.com, and fetch its title on the console using the promise function.

Code:

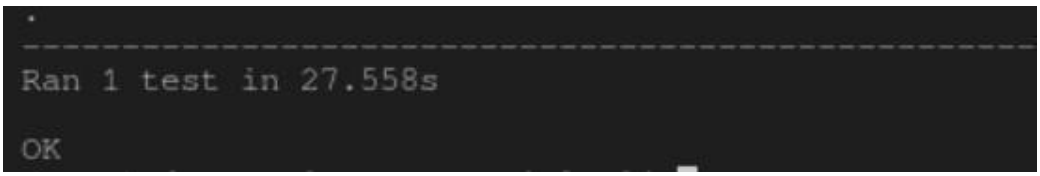
```
var webdriver = require('selenium-webdriver');

var name = new webdriver.Builder().

withCapabilities(webdriver.Capabilities.firefox()).build();

name.get('https://www.browserstack.com'); var promise =

browser_name.getTitle(); promise.then(function(title)

{ console.log(title);

}); browser.quit();
```

(2) Write test cases and simply run. The console will show which test cases passed orfailed.

import unittest from selenium import webdriverfrom selenium.webdriver.common.keys import Keys

```
self.driver = webdriver.Firefox()
def test_search_in_python_org(self):

        driver = self.driver driver.get("http://www.python.org")self.assertIn("Python",
driver.title)

    elem = driver.find_element_by_name("q")elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
assert "No results found." not in driver.page_source

  def tearDown(self): self.driver.close() if __name___==" main ":
  unittest.main()
```

Output:



Conclusion:

Hence, we learned how to automate browser testing.

Experiment No: 10

Aim: Installation of Docker

Theory:

Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere. As Bottomley told me, "Containers gives you instant application portability."

Containers do this by enabling developers to isolate code into a single container. T is makes it easier to modify and update the program. It also lends itself, as Docker points out, for enterprises to break up big development projects among multiple smaller, Agile teams using Jenkins, an open-source CI/CD program, to automate the delivery of new software in containers.

Jay Lyman, senior analyst at 451 Research, added: "Enterprise organizations are seeking and sometimes struggling to make applications and workloads more portable and distributed in an effective, standardized, and repeatable way. Just as GitHub stimulated collaboration and innovation by making source code shareable, Docker Hub, Official Repos, and commercial support are helping enterprises answer this challenge by improving the way they package, deploy, and manage applications."

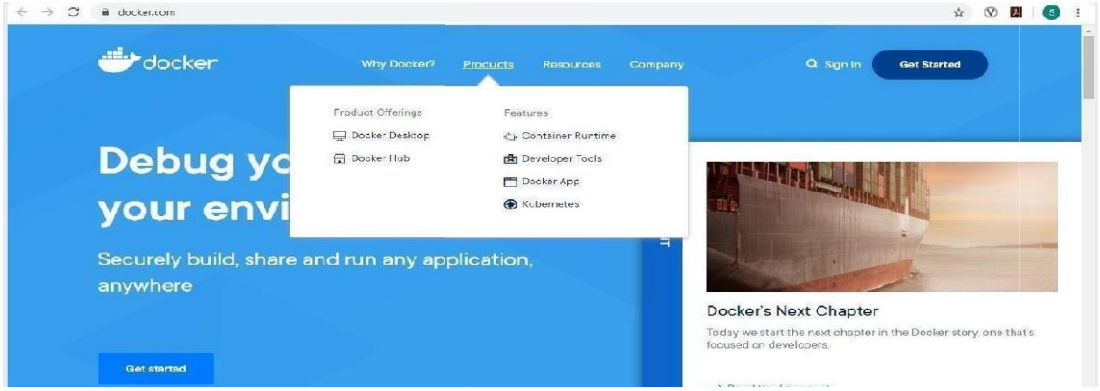
In addition, Docker containers are easy to deploy in a cloud. As Ben Lloyd Pearson wrote in Opensource.com: "Docker has been designed in a way that it can be incorporated into most DevOps applications, including Puppet, Chef, Vagrant, and Ansible, or it can be used on its own to manage development environments."

Specifically, for CI/CD Docker makes it possible to set up local development environments that are exactly like a live server; run multiple development environments from the same host with unique software, operating systems, and configurations; test projects on new or different servers; and allow anyone to work on the same project with the exact same settings, regardless of the local host environment.

Step 1 – Download Docker

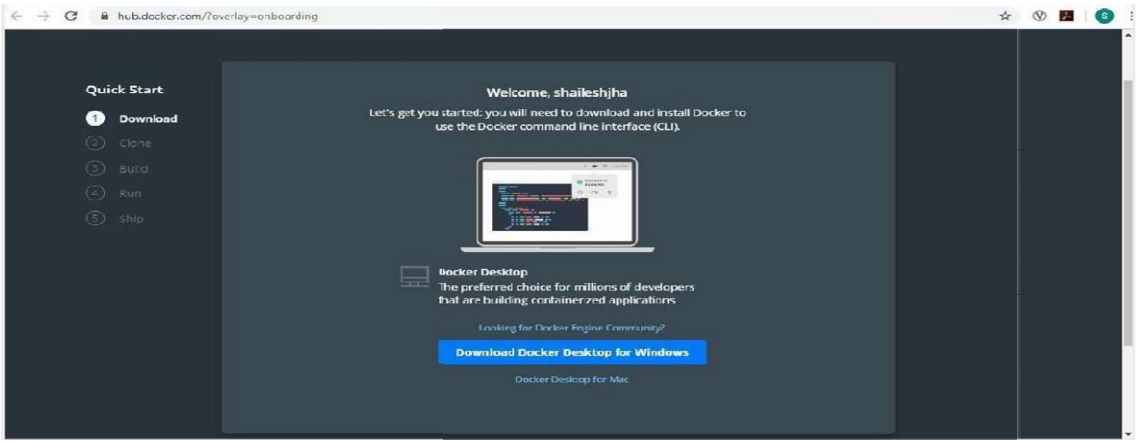
Officially Docker installer, community edition can be downloaded from Docker Store. You will have to create an account to be able to download it. Having an account for Docker is a very good things. It allows you to download docker images in the future.

If you dont want to create an account, this is the **direct link to Download Docker Installer.** You can also reach the Docker Store download page from the Docker official page. Go to the Docker Official home Page.



Click on Products-> Docker Desktop on the menu bar. This will take you to the docker desktop product page.

Click on Download Desktop for Mac and Windows.



Step 2 – Run the installer

The installer file for windows is around 914 MB.

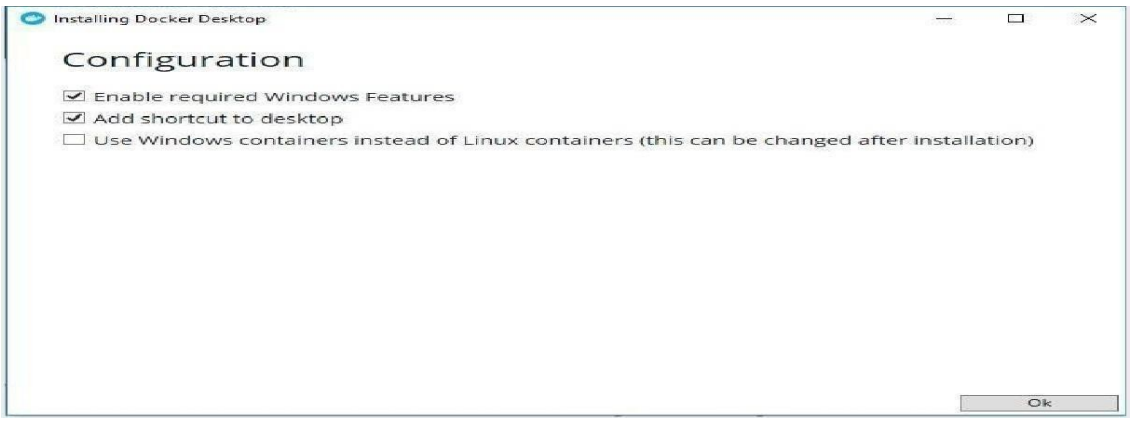
Double click on the downloaded installer file to start the installation wizard. You will see Windows UAC – User Access Control asking for permission to allow the program to run. Click yes to continue.

Now you will see Docker installer downloading additional files required. If you dont have internet connection, installer will move on to the next step. Wait for the process to complete and you will see the configuration screen.

Step 3 Configuration Settings

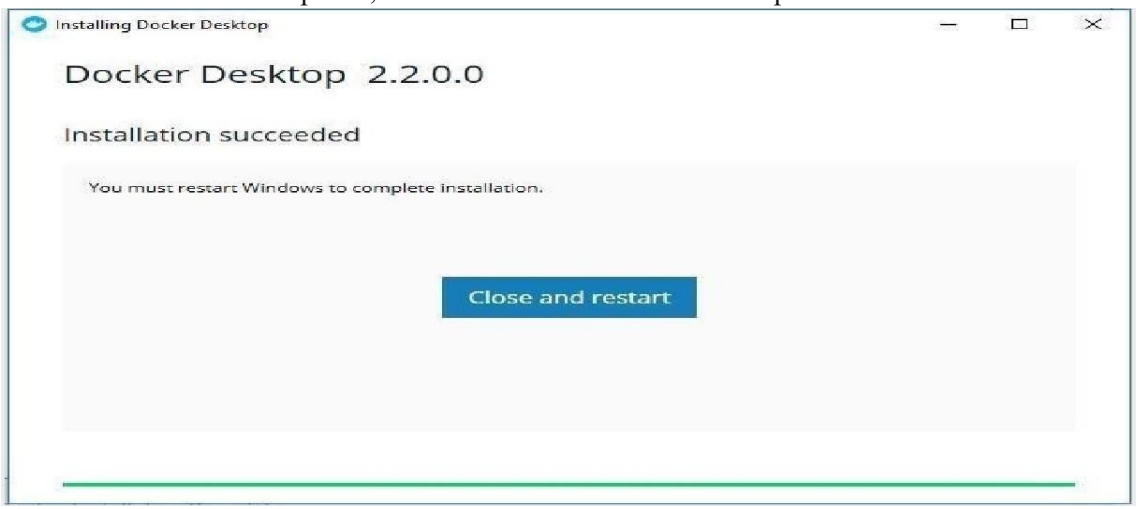
In this dialog box you will be asked if you want to create desktop icon for Docker. I leave this checked. Second option is if you want to use Linux or Windows Container.This option can be changed later on. I leave it as default that is unchecked.
Container run on your host operating system which can be Linux or Windows. Ideally you can run Linux container on Linux OS and Windows container on Windows OS. This could be a problem in development machines if you want to develop Linux container based application on Windows.

To get around this, Docker allow users to create both Linux and windows container on the same windows machine without having to switch between OS platforms. To achieve this, Docker provides a Linux VM for Hyper-V called MobyLinuxVM as a part of installation process. This VM runs Linux container and your Windows 10 host runs Windows Containers. You cannot run both Windows and Linux Containers at the same time. That is why Docker allows you to switch between Windows and Linux container. With this you can run both Linuxand Windows Containers side by side as a part of the same Docker Installation for Windows



Click OK to start the installation. Wait for the installation to complete.

Once the installation completes, click on Close and restart to complete the installation.



Step 4 Run Docker:

Once the system restarts, run Docker by double clicking the icon created on the desktop or from start menu. You will see a docker icon appear on your windows task bar. If you hover your mouse over it, it will say “Docker is Starting”. You will see a warning asking your

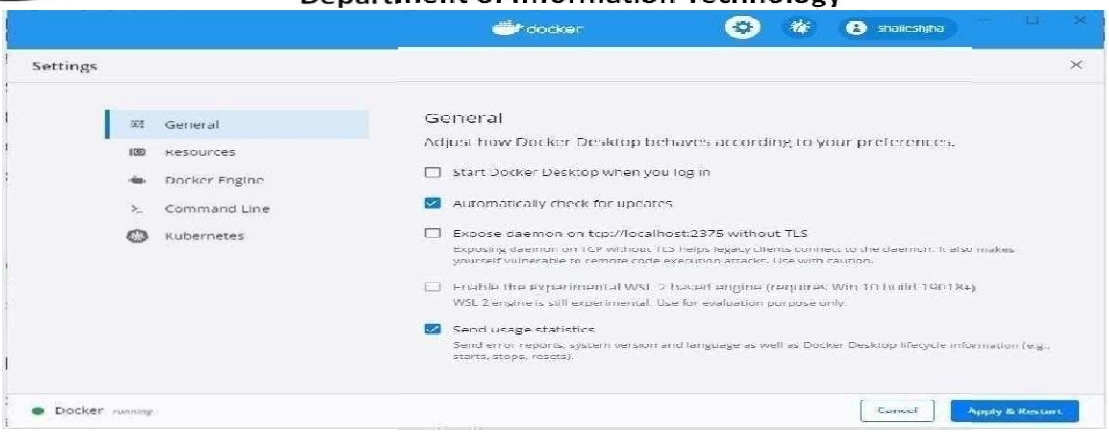
permission to start the docker service. Click Start to continue and wait for docker service to start.

Once docker service starts for the first time, you will see a welcome screen asking you to login to Docker Hub. Enter the docker hub credentials you have created and click on sign in. This isoptional but it is highly recommended that you do it.

Step 5 – Disable Start Docker at Startup

By default, docker will automatically start when you turn on/login to your computer. Since Docker requires a lot of RAM, I don't recommend this. We can start docker manually when we want to use it.

To disable starting docker at startup, right click on the docker Icon in the task bar. Click on Settings, under General Tab, uncheck, Start Docker when you login.



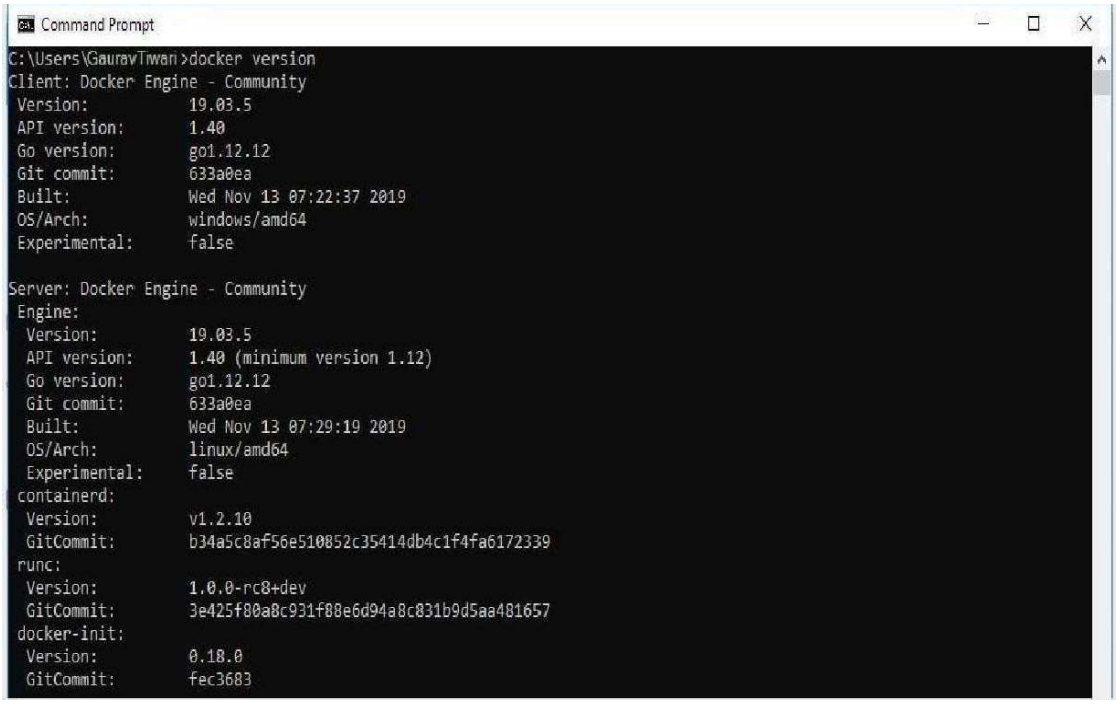
Click on apply and restart to make the changes. This will restart the docker service again (I will not restart Windows, only the docker service).

Step 6 – Check the version of Docker Installed Most of the time you will be working with command line to work with Docker. Let this be your first command to check Docker version and see whats you get.

Open Powershell or command prompt and enter the command:

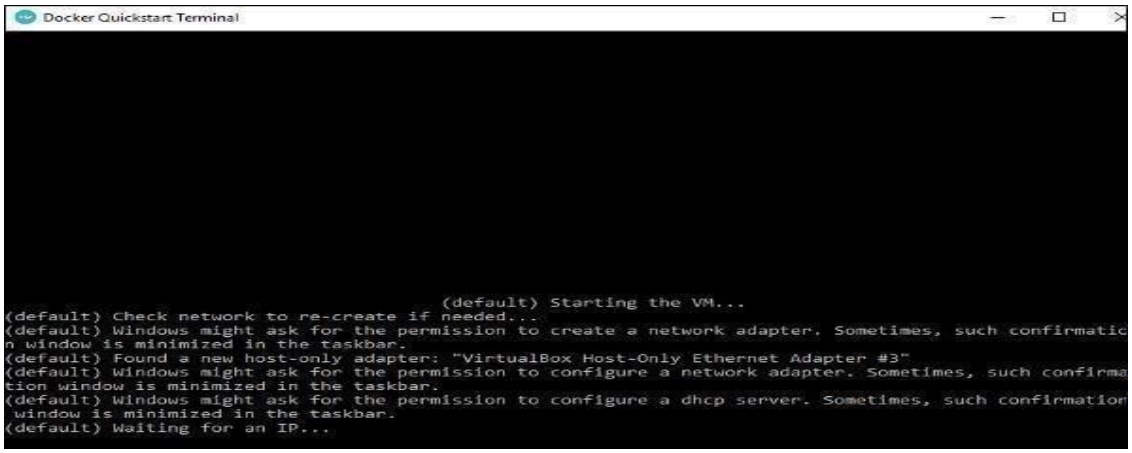
```
docker version
```

You should see something like this.



The first step is to launch the Docker Toolbox application for which the shortcut is created on the desktop when the installation of Docker toolbox is carried out.

Next, you will see the configuration being carried out when Docker toolbox is launched.



Once done, you will see Docker configured and launched. You will get an interactive shell for Docker.
To test that Docker runs properly, we can use the Docker **run command** to download and run a simple

HelloWorld Docker container.

The working of the Docker **run command** is given below −

docker run

This command is used to run a command in a Docker container.

Syntax

docker run image

Options

- **Image** − This is the name of the image which is used to run the container.

Return Value

The output will run the command in the desired container.

Example

This command will download the **hello-world** image, if it is not already present, and run the **hello-world** as a container.

Output

When we run the above command, we will get the following result −

If you want to run the Ubuntu OS on Windows, you can download the Ubuntu Image using the following command −

Docker run –it ubuntu bash

Here you are telling Docker to run the command in the interactive mode via the **–it** option.

In the output you can see that the Ubuntu image is downloaded and run and then you will be logged inas a root user in the Ubuntu container.

To Check if docker is installed or not by writing the command: docker --version

Conclusion: Hence, we studied step of installation of Docker and its installation requirement.

Experiment No: 11

Aim: Building Docker Image using Jenkins.

Docker as we know, is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.

Today we are going to check how to configure Jenkins to build Docker Images based on a Docker file.Below are the steps of how you can use Docker within a CI/CD pipeline, using Images as a build artifact that can be promoted to different environments and finally production.

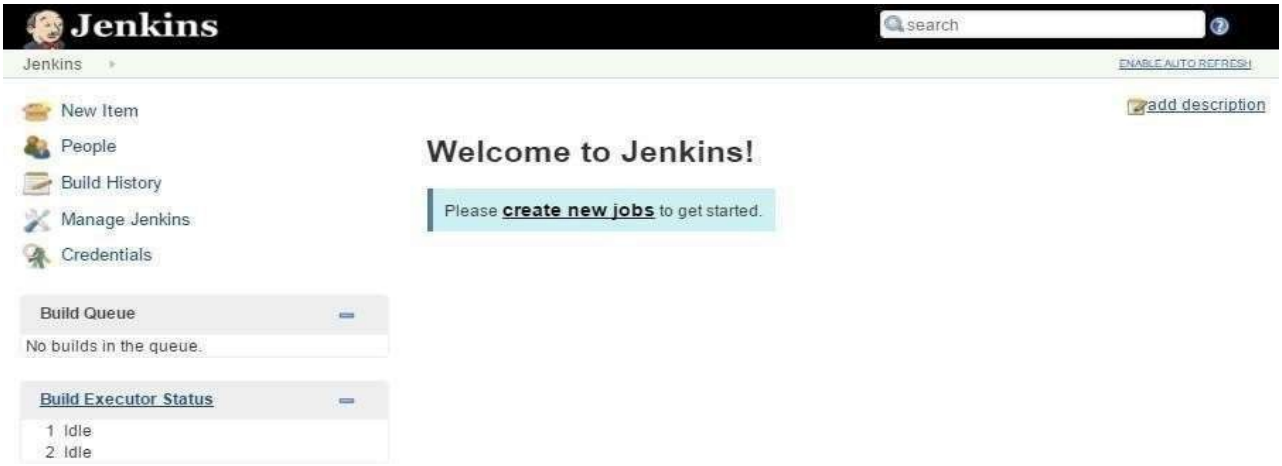
Step #1 : Launch Jenkins

Currently I have Jenkins running on Docker container, if you do `docker ps` command it would show youthe status of the container.

```
  > docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                                  NAMES
bf8a4679a3e3        jenkins:1.651.1-alpine   "/bin/tini -- /usr/lo"   21 months ago       Up 7 minutes        0.0.0.0:8
080->8080/tcp, 0.0.0.0:50000->50000/tcp   jenkins
```

Image — Docker ps command to review status of the containers

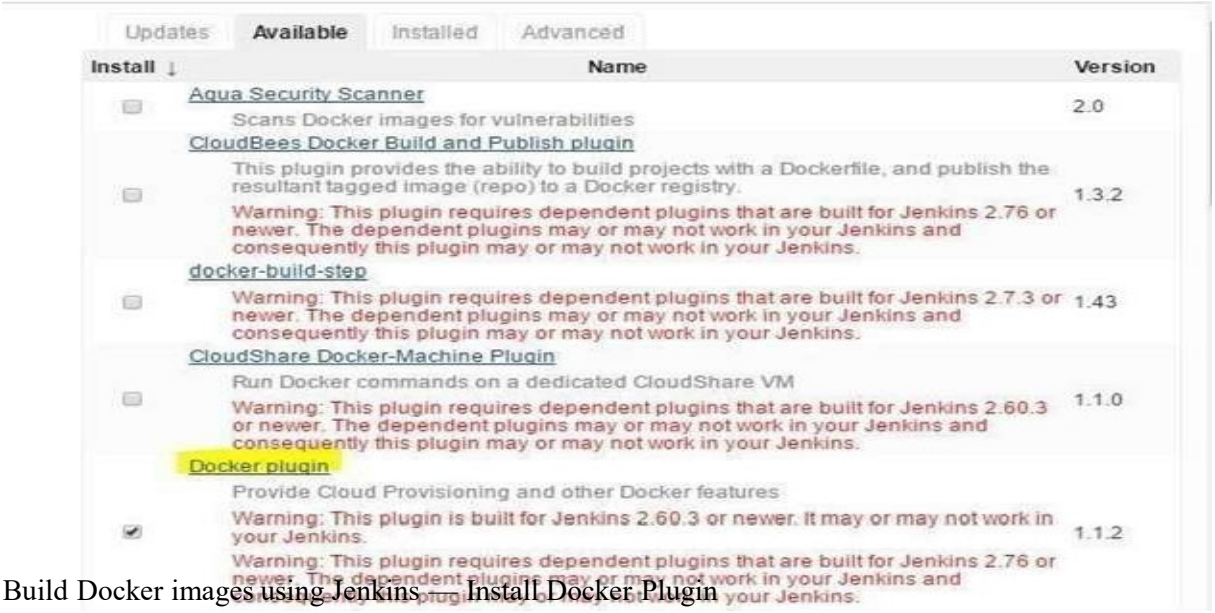
Launch Jenkins' dashboard



Build Docker images using Jenkins — Jenkins dashboard

Step #2: Configure the plugins and start building Docker Images.

Our 1st step is to configure Docker plugin. Whenever a Jenkins build requires Docker, it will create a "Cloud Agent" via the plugin. The agent will be a Docker Container configured to talk to our Docker Daemon. The Jenkins build job will use this container to execute the build and create the image before being stopped. The Docker Image will be stored on the configured Docker Daemon. The Image can then be pushed to a Docker Registry ready for deployment.

1. Once you are inside the Jenkins Dashboard, select **Manage Jenkins** on the left.
2. On the Configuration page, select **Manage Plugins**.
3. Manage Plugins page will give you a tabbed interface. Click **Available** to view all the Jenkins pluginsthat can be installed.
4. Using the search box, search for **Docker plugin**. There are multiple Docker plugins, select **Dockerplugin** using the checkbox.
5. 

Build Docker images using Jenkins — Install Docker Plugin

1. While on this page, install the **Git plugin** for obtaining the source code from a Gi repository.

Build Docker images using Jenkins — Installing plugins

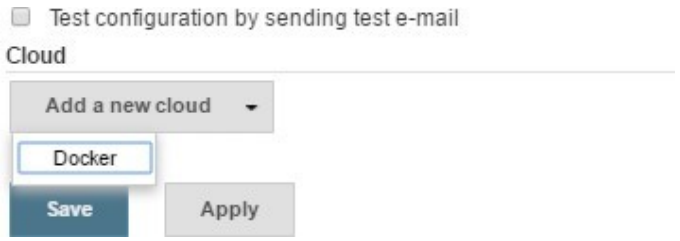
1. Click **Install without Restart** at the bottom.
2. The plugins will now be downloaded and installed. Once complete, click the link **Go back to the toppage**.

Once the Docker & GIT plugins have been installed, now we can go ahead and configure how they launch the Docker Containers.

The configuration would be used by the plugin which Docker Image to use the agent and which Docker daemon to run the containers and builds on. The plugin treats Docker as a cloud provider,spinning up containers as and when the build requires them.

Step #3 : Configure Docker agent

1. On the Jenkins Dashboard, select **Manage Jenkins**.
2. Select **Configure System** to access the main Jenkins settings.
3. At the bottom, there is a dropdown called **Add a new cloud**. Select **Docker** from the list.



Build Docker images using Jenkins — Docker Add new cloud

1. You can now configure the container options. Set the name of the agent to **docker-agent**.

Build Docker images using Jenkins — Set Docker Agent options

1. The "Docker URL" is where Jenkins launches the agent container. In this case, we'll use the same daemon as running Jenkins, but in real world scenario it should be separate instance so that it can scale.

2. Use **Test Connection** to verify Jenkins can talk to the Docker Daemon. You should see the Dockerversion number returned.
   Now plugin can communicate with Docker, next step would be to configure how to launch the Docker Image for the agent.

1. Using the Images dropdown, select **Add Docker Template** dropdown.
2. For the Docker Image, use sample one which has Docker client **benhall/dind-Jenkins-agent**. Thisimage is configured with a Docker client and available at https://hub.docker.com/r/benhall/dind-jenkins-agent/
3. To enable builds to specify Docker as a build agent, set a label of **docker-agent**.
4. Jenkins uses SSH to communicate with agents. Add a new set of "Credentials". The username is **Jenkins** and the password is **Jenkins**.
5. Finally, expand the Container Settings section byclicking the button. In the "Volumes "text box enter **/var/run/docker. Sock:/var/run/docker. Sock**
6. Click Save.

Step #4: Test the setup

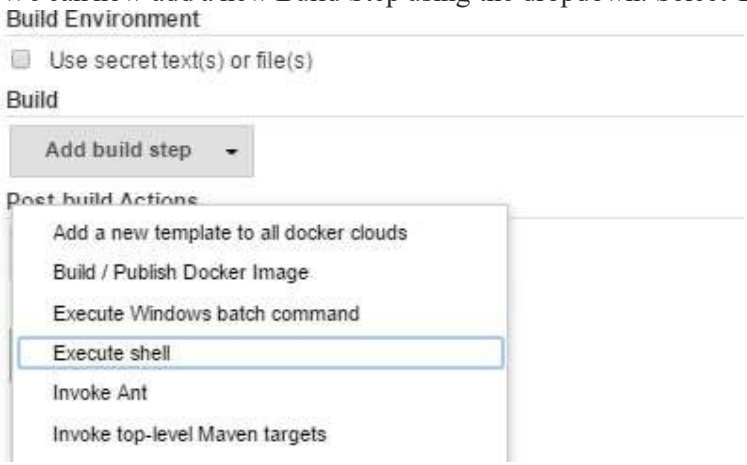
To test the setup, create new job to

new job ex.Jenkins Demo.

Build Docker images using Jenkins — Create New Project

1. The build will depend on having access to Docker. Using the "Restrict where this project can be run" we can define the label we set of our configured Docker agent. The set "Label Expression" to **Docker-agent**. You should have a configuration of "Label is serviced by no nodes and 1 cloud".
2. Select the Repository type as **Git** and set the Repository. I'm using my GIT location **https://github.com/karthi4india/jenkins/.**

3. We can now add a new Build Step using the dropdown. Select **Execute Shell**.



Build Docker images using Jenkins — Add Build Step

1. Docker file takes care of build, Jenkins only needs to call build and specify a friendly name.

Build step:

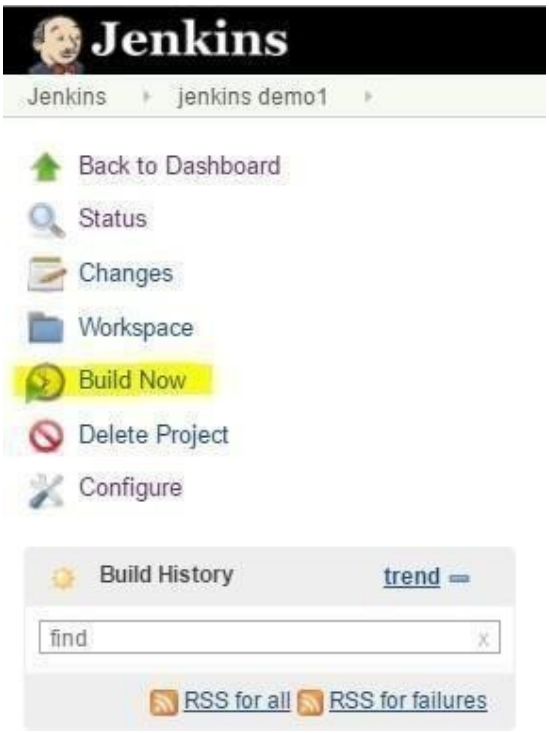
```
ls

docker info

docker build -t jenkins-demo:${BUILD_NUMBER} .

docker tag jenkins-demo:${BUILD_NUMBER} jenkins-demo:latest

docker images
```

The first command lists all the files in the directory which will be built. When calling *docker build* we use the Jenkins build number as the image tag. This allows us to version our Docker Images. We also tag the build with *latest*.

Docker File:

```
FROM scratch
EXPOSE 80
COPY http-server /
CMD ["/http-server"]
```

On the left-hand side, select **Build Now**. You should see a build scheduled with a message“(pending — Waiting for next available executor)”.



Build Docker images using Jenkins — Jenkins Build

Jenkins is launching the container and connecting to it via SSH. Some times this can take a momentortwo.

You can see the progress using`docker logs --tail=10 jenkins`

Once the build has completed you should see the Image and Tags using the Docker CLI

Conclusion: Hence, students are familiar with docker image creation process.

Experiment No:12

Aim: Demonstration of Ansible Automation

Objective: The objective of this experiment is to demonstrate the basics of Ansible automation, including setting up inventory, creating playbooks, and executing tasks on remote hosts.

Materials Required:

A set of computers or virtual machines to act as Ansible control node and managed nodes.
Ansible installed on the control node.
SSH access to the managed nodes from the control node.
Text editor for creating Ansible playbooks (e.g., Vim, Nano, Sublime Text).

Experiment Setup:

Ensure that all required materials are available and accessible.
Set up SSH access between the control node and managed nodes, ensuring passwordless SSH authentication is configured.

Experiment Procedure:

1. Setting Up Inventory:

Explain the concept of inventory in Ansible.
Create an inventory file named hosts on the control node.
Populate the inventory file with the IP addresses or hostnames of managed nodes.

2. Creating a Simple Ansible Playbook:

Introduce the concept of Ansible playbooks.
Create a new playbook file named simple_playbook.yml.
Define a simple playbook structure with hosts and tasks sections.
Write a task to print a message on the console using the debug module.
Execute the playbook using the ansible-playbook command.

3. Writing Playbooks with Tasks:

Explain the anatomy of Ansible tasks.
Create a new playbook file named tasks_playbook.yml.
Define multiple tasks to perform actions such as installing packages, creating files, or configuring services.
Use different Ansible modules to achieve these tasks (e.g., apt, copy, service).
Execute the playbook and observe the changes on managed nodes.

4. Working with Variables:

Discuss the usage of variables in Ansible.
Introduce Ansible facts and user-defined variables.
Create a playbook that utilizes variables for dynamic behavior.
Show examples of variable interpolation and usage within tasks.
Execute the playbook and verify variable substitution on managed nodes.

5. Implementing Conditionals and Loops:

Explain how conditionals and loops can be used in Ansible playbooks.
Create a playbook with conditional statements and loops.
Use conditionals to execute tasks based on specific criteria.
Implement loops to iterate over a list of items and perform repetitive tasks.
Execute the playbook and observe the conditional execution and loop iterations.

6. Securing Ansible Communication:

Discuss the importance of securing Ansible communication.
Introduce Ansible Vault for encrypting sensitive data.
Demonstrate how to encrypt and decrypt files using Ansible Vault.
Modify existing playbooks to use encrypted variables or files.
Execute the playbooks and observe encrypted data usage.

7. Advanced Topics (Optional):

Depending on the audience's proficiency level, cover advanced Ansible topics such as roles, includes, handlers, and Ansible Galaxy integration.
Provide hands-on exercises or examples to reinforce understanding.

Conclusion: Summarize the key learnings from the experiment. Encourage participants to explore further and apply Ansible automation in their environments.