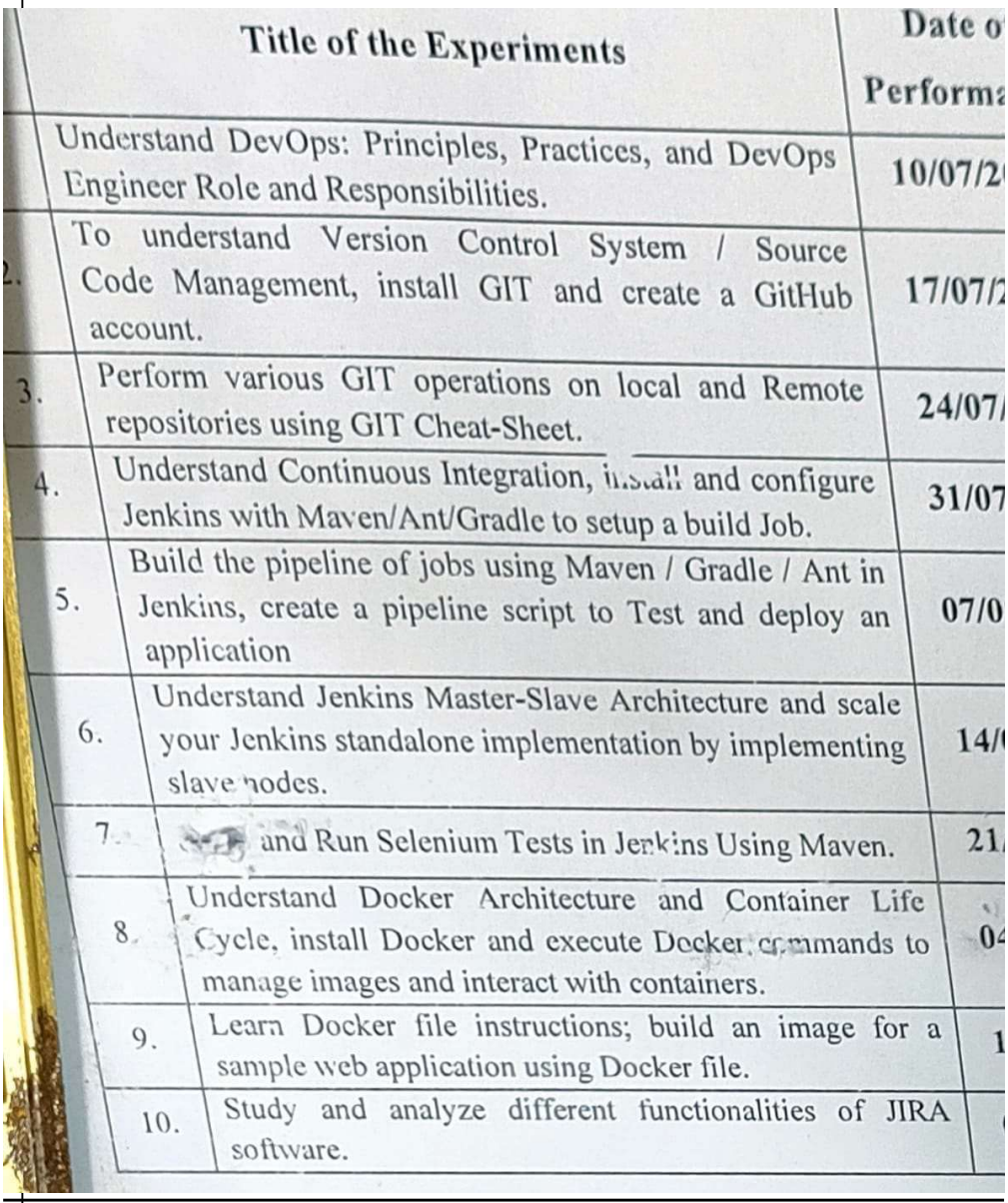


| | Title of the Experiments | Date o Performa |
|---|---|---|
| | Understand DevOps: Principles, Practices, and DevOps Engineer Role and Responsibilities. | 10/07/2 |
| 2. | To understand Version Control System / Source Code Management, install GIT and create a GitHub account. | 17/07/2 |
| 3. | Perform various GIT operations on local and Remote repositories using GIT Cheat-Sheet. | 24/07/ |
| 4. | Understand Continuous Integration, install and configure Jenkins with Maven/Ant/Gradle to setup a build Job. | 31/07 |
| 5. | Build the pipeline of jobs using Maven / Gradle / Ant in Jenkins, create a pipeline script to Test and deploy an application | 07/0 |
| 6. | Understand Jenkins Master-Slave Architecture and scale your Jenkins standalone implementation by implementing slave nodes. | 14/ |
| 7. | and Run Selenium Tests in Jenkins Using Maven. | 21/ |
| 8. | Understand Docker Architecture and Container Life Cycle, install Docker and execute Docker commands to manage images and interact with containers. | 04 |
| 9. | Learn Docker file instructions; build an image for a sample web application using Docker file. | 1 |
| 10. | Study and analyze different functionalities of JIRA software. | |

Additional settings for docker: search in windows for turn windows features on or off and select 1. Windows system for linux and 2. Windows hypervisor platform

Run wsl --update command in cmd if docker is not opening after these steps

Docker --version in cmd to check docker version

shaikhabdulwahid19.atlassian.net

# EXPERIMENT NO 01

## Q1) WHAT IS SDLC ?

a.Software Development Life Cycle (SDLC):

I. A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

II. In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

b.Need of SDLC:

I. The development team must determine a suitable life cycle model for a particular plan and then observe to it.

II. Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representative about when and what to do. Otherwise, it would point to chaos and project failure. This problem can be defined by using an example. Suppose a software development issue is divided into various parts and the parts are assigned to the team members. From then on, suppose the team representative is allowed the freedom to develop the roles assigned to them in whatever way they like. It is possible that one representative might start writing the code for his part, another might choose to prepare the test documents first, and some other engineer might begin with the design phase of the roles assigned to him. This would be one of the perfect methods for project failure.

III. A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.

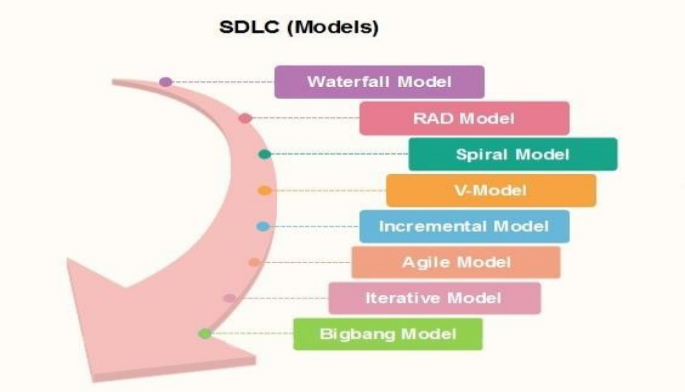
## Q2) TYPES OF SDLC?

SDLC Models:

I. Software Development life cycle (SDLC) is a spiritual model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.

II. There are different software development life cycle models specify and design, which are followed during the software development phase. These models are also called "Software Development Process Models." Each process model follows a series of phase unique to its type to ensure success in the step of software development.

Here, are some important phases of SDLC life cycle:




1. Waterfall Model
   I. The waterfall is a universally accepted SDLC model. In this method, the whole process of software development is divided into various phases.
   II. The waterfall model is a continuous software development model in which development is seen as flowing steadily downwards (like a waterfall) through the steps of requirements analysis, design, implementation, testing (validation), integration, and maintenance.
   III. Linear ordering of activities has some significant consequences. First, to identify the end of a phase and the beginning of the next, some certification techniques have to be employed at the end of each step. Some verification and validation usually do this mean that will ensure that the output of the stage is consistent with its input (which is the output of the previous step), and that the output of the stage is consistent with the overall requirements of the system.

2. RAD Model

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets developing software in a short period. The RAD model is based on the concept that a better system can be developed in lesser time by using focus groups to gather system requirements.

   I. Business Modeling
   II. Data Modeling
   III. Process Modeling
   IV. Application Generation
   V. Testing and Turnover

3. Spiral Model

The spiral model is a risk-driven process model. This SDLC model helps the group to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc. The spiral technique is a combination of rapid prototyping and concurrency in design and development activities. Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the goals, and the constraints that exist. This is the first quadrant of the cycle (upper-left quadrant).

The next step in the cycle is to evaluate these different alternatives based on the objectives and constraints. The focus of evaluation in this step is based on the risk perception for the project.

The next step is to develop strategies that solve uncertainties and risks. This step may involve activities such as benchmarking, simulation, and prototyping.

4. V-Model

   In this type of SDLC model testing and the development, the step is planned in parallel. So, there are verification phases on the side and the validation phase on the other side. V-Model joins by Coding phase.

5. Incremental Model

The incremental model is not a separate model. It is necessarily a series of waterfall cycles. The requirements are divided into groups at the start of the project. For each group, the SDLC model is followed to develop software. The SDLC process is repeated, with each release adding more functionality until all requirements are met. In this method, each cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.

6. Agile Model

I. Agile methodology is a practice which promotes continues interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

II.

III. Any agile software phase is characterized in a manner that addresses several key assumptions about the bulk of software projects:

IV.

V. It is difficult to think in advance which software requirements will persist and which will change. It is equally difficult to predict how user priorities will change as the project proceeds.

VI. For many types of software, design and development are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to think about how much design is necessary before construction is used to test the configuration.

VII. Analysis, design, development, and testing are not as predictable (from a planning point of view) as we might like.

7. Iterative Model

   It is a particular implementation of a software development life cycle that focuses on an initial, simplified implementation, which then progressively gains more complexity and a broader feature set until the final system is complete. In short, iterative development is a way of breaking down the software development of a large application into smaller pieces.

8. Big bang model

I. Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.

II. This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements are either unknown or final release date is not given.

9. Prototype Model

I. The prototyping model starts with the requirements gathering. The developer and the user meet and define the purpose of the software, identify the needs, etc.

II. A 'quick design' is then created. This design focuses on those aspects of the software that will be visible to the user. It then leads to the development of a prototype. The customer then checks the prototype, and any modifications or changes that are needed are made to the prototype.

III. Looping takes place in this step, and better versions of the prototype are created. These are continuously shown to the user so that any new changes can be updated in the prototype. This process continue until the customer is satisfied with the system. Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

## Q.3) Differentiate between WATERFALL & AGILE MODEL.

| Agile Model | Waterfall Model |
|---|---|
| It separates the project development lifecycle into multiple sprints. | The process of software development is divided into distinct phases. |
| It uses an incremental approach | It is a sequential design process. |
| It is flexible. | It is a structured process, and may be quite rigid at times. |
| It can be understood as a collection of multiple different projects. | The software can be developed as one single project. |
| Changes can be made in project development requirements after initial planning has been completed. | The requirements can't be changed once the project development starts. |

| | |
|---|---|
| It follows an iterative development approach. The process of planning, development, prototyping and software development phases can appear multiple times. | The project development phases such as designing, development, testing, are done only once in the waterfall model. |
| The test plan is reviewed after every sprint. | The test plan is not usually discussed during the test phase. |
| The requirements may change and evolve. | This method can be used for projects that have a definite set of requirements and doesn't changes. |
| The process of testing is performed concurrently with software development. | The 'testing' phase comes after the 'build' phase. |
| It uses a product mindset where software product would satisfy the requirements of end customers. | It shows a project mindset, and focuses on completing the project. |
| It would increase the amount of stress in fixed-price scenarios. | The risk is reduced since the price of contracts are fixed by getting the risk agreement in the beginning. |
| It is preferred to work with small and dedicated teams with high amounts of coordination and synchronization. | There is limited team coordination and synchronization. |
| It works well with time and materials, i.e. non-fixed funding. | Business analysis is done to prepare the requirement before beginning it. |
| Products owner with team prepares requirements just about every day during a project. | It is difficult to initiate any changes in the requirements. |
| Project managers are not needed since project can be managed by the entire team. | The process is generally straightforward hence project manager is required. |

**Q.4) Limitations of AGILE.**

First limitation of Agile methodologies is that it is not suitable for maintenance, since there is not much documentation for the system. Developers are not concentrating on the documentation as much because the primary goal when using an Agile Methodology is to write software not documentation. Another limitation is that Agile Methodologies depend heavily on the user involvement, and thus, the success of the project will depend on the cooperation and communication of the user. Another limitation is that agile methodologies concentrate work quality on the skills and behaviors of the developers, as the design of the modules and sub-modules are created mainly by single developer. This is because they focus on building systems that solve specific problems, and not the general ones. Agile methodologies work best for teams with relatively small number of members (no less than 3 and no more than 9) , and hence, they will not work well for teams with large number of members.

## Q5) What is Dev-Ops?

A) DevOps combines development (Dev) and operations (Ops) to unite people, process, and technology in application planning, development, delivery, and operations. DevOps enables coordination and collaboration between formerly siloed roles like development, IT operations, quality engineering, and security. DevOps accelerate the process to deliver applications and software services at high speed and high velocity. So that organizations can learn and Adopt the market at its earliest. DevOps can be best explained as people working together to conceive, build and deliver secure software at top speed.

## Q6) Key principles of Dev-Ops

**A) The key principles of dev-ops are:-**

1. CollaborationDevOps in its purest form is the integration of the development (Dev) and operations (Ops) teams. This means that collaboration is central to the foundation of DevOps.

2. Data-Based Decision MakingAnother central principle of DevOps is informing your decisions with data. Whether it's selecting the right tech stack or selecting tools to streamline your pipeline, you should always collect data around each decision to ensure your choice agrees with your team's metrics and historical data.

3. Customer-Centric Decision MakingThe customer should be a central focus in a DevOps lifecycle. Equally as important as data, decisions should be weighed with the question, "Will this benefit the customer?" Collecting feedback from the customer on the existing product will guide future optimization.

4. Constant ImprovementDevOps focuses on constant improvement, or the idea that the team should continuously focus on new features and upgrades. Another key idea follows the Agile methodology of incremental releases.

5. Responsibility Throughout the LifecycleIn traditional software development models, the development team codes and builds the application. They then hand it to the operations team to test, deploy, and deliver to the customer. Any bugs discovered in the second phase are left to the operations team instead of the developers who wrote the code.

6. AutomationA key benefit of the DevOps approach is speed: speed of software delivery, speed of updates, speed of patches. This momentum is achieved with automation. DevOps teams aim to automate every single phase of the process, from code reviews to handoffs to provisioning and deployment.

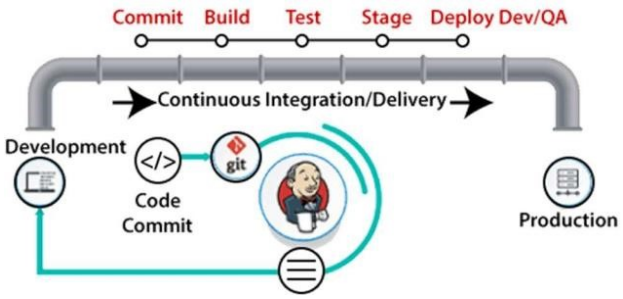
7. Failure as a Learning OpportunityDevOps is a flexible approach to development. Processes are constantly being fine-tuned just as the software itself is continuously improving. Part of maintaining this flexibility is to view failure as an opportunity to learn and improve. Rather than trying to avoid failure at all costs, encourage risk-taking in the right context.

## Q7) Life Cycle of Dev-Ops

A) The DevOps lifecycle includes seven phases as given below:

1) Continuous Development: -This phase involves the planning and coding of the software. The vision of the project is decided during the planning phase. And the developers begin developing the code for the application. There are no DevOps tools that are required for planning, but there are several tools for maintaining the code.

2) Continuous IntegrationThis stage is the heart of the entire DevOps lifecycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or weekly basis. Then every commit is built, and this allows early detection of problems if they are present. The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.



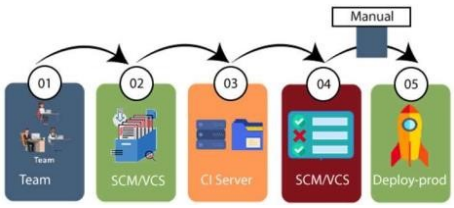

3) Continuous TestingThis phase, where the developed software is continuously testing for bugs. For constant testing, automation testing tools such as **TestNG, JUnit, Selenium**, etc are used. These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that there is no flaw in the functionality. In this phase, **Docker** Containers can be used for simulating the test environment.**Selenium** does the automation testing, and TestNG

generates the reports. Automation testing saves a lot of time and effort for executing the tests instead of doing this manually.

4) Continuous MonitoringMonitoring is a phase that involves all the operational factors of the entire DevOps process, where important information about the use of the software is recorded and carefully processed to find out trends and identify problem areas. Usually, the monitoring is integrated within the operational capabilities of the software application

5) Continuous FeedbackThe application development is consistently improved by analyzing the results from the operations of the software. This is carried out by placing the critical phase of constant feedback between the operations and the development of the next version of the current software application.

6) Continuous DeploymentIn this phase, the code is deployed to the production servers. Also, it is essential to ensure that the code is correctly used on all the servers.




The new code is deployed continuously, and configuration management tools play an essential role in executing tasks frequently and quickly. Here are some popular tools which are used in this phase, such as **Chef, Puppet, Ansible**, and **SaltStack**.Containerization tools like docker help to maintain consistency across the environments where the application is tested, developed, and deployed. There is no chance of errors or failure in the production environment as they package and replicate the same dependencies and packages used in the testing, development, and staging environment. It makes the application easy to run on different computers.

7) Continuous OperationsAll DevOps operations are based on the continuity with complete automation of the release process and allow the organization to accelerate the overall time to market continuingly.

## Q.8 Devops engineer roles and responsibility

A DevOps engineer is an IT generalist who should have a wide-ranging knowledge of both development and operations, including coding, infrastructure management, system administration, and DevOps toolchains. DevOps engineers should also possess interpersonal skills since they work across company silos to create a more collaborative environment. DevOps engineers need to have a strong understanding of common system architecture, provisioning, and administration, but must also have experience with the traditional developer toolset and practices such as using source control, giving and receiving code reviews, writing unit tests, and familiarity with agile principles.

Roles of Devops Engineer: -

- **Infrastructure as Code (IaC):** Design, implement, and manage infrastructure using tools like Terraform, CloudFormation, or Ansible. Automate provisioning of infrastructure to ensure consistency and scalability.
- **Version Control and Configuration Management:** Manage version control systems (e.g., Git) and enforce configuration management practices.
- **CI/CD Pipeline Implementation:** Create automated pipelines for building, testing, and deploying applications. Ensure rapid and reliable delivery of software releases.
- **System administration: -** A DevOps engineer will have experience with system administration, such as provisioning and managing servers, deploying databases, security monitoring, system patching, and managing internal and external network connectivity.
- **Familiarity with coding and scripting: -** Many traditional system administrators have experience writing shell scripts to automate repetitive tasks. A DevOps engineer should go beyond writing automation scripts and understand advanced software development practices and how to implement agile development practices such as code reviews and using source control.

Responsibility of Devops Engineer: -

- **Monitoring:** Set up monitoring tools (e.g., Prometheus, ELK stack) to monitor system performance, detect issues, and ensure high availability.
- **Performance Monitoring:** Monitor system performance metrics and conduct performance tuning to optimize application and infrastructure performance.
- **Training:** Provide training sessions and workshops to educate team members on DevOps practices, tools, and technologies.
- **Mentoring:** Mentor junior team members and share best practices to foster skill development and knowledge sharing within the team.
- **Disaster Recovery Planning:** Develop and test disaster recovery plans to ensure business continuity in the event of system failures or disasters.

## Q.9 Software and tools used for each SDLC Phase

The SDLC is usually broken down into six steps: **Analysis, Planning, Architecture Design, Development, Testing, and Maintenance**


6 Stages of Software Development Process
1
Analysis
2
Planning
3
Design
4
Development
5
Testing
6
Maintenance

# EXPERIMENT NO 02

**What is Version Control?**

Version control is a system that records changes to files over time, allowing multiple users to collaborate on projects without overwriting each other's work. It helps in managing different versions of files, enabling users to revert files to a previous state, compare changes over time, and identify who made specific modifications. Version control is crucial for maintaining a coherent history of the project's evolution, especially in software development where it ensures that changes are systematically tracked and managed.

**Benefits of Version Control Systems**

Version control systems (VCS) offer several benefits, such as enabling collaborative work by multiple developers without overwriting changes, providing the ability to revert to previous versions in case of mistakes, and maintaining a detailed history of the project. VCS helps in tracking changes and identifying contributors, supports parallel development through branching and merging, and enhances overall project management and productivity by providing a structured approach to handling code modifications and updates.

**What is Source Code Management?**

Source Code Management (SCM) involves the practices and tools used to track and control changes in the source code. It ensures that code changes are systematically recorded and managed, facilitating collaboration among developers. SCM includes version control, configuration management, and build management, ensuring that the codebase remains consistent and integrates changes effectively. It is essential in software development for maintaining code quality, managing different versions, and supporting efficient teamwork.

**Importance of Source Code Management**

SCM is vital for maintaining the integrity and consistency of the codebase, facilitating collaboration among developers, and ensuring that changes are tracked and managed efficiently. It allows developers to revert to previous states if needed, supports parallel development through branching and merging, and helps in automating the development, testing, and

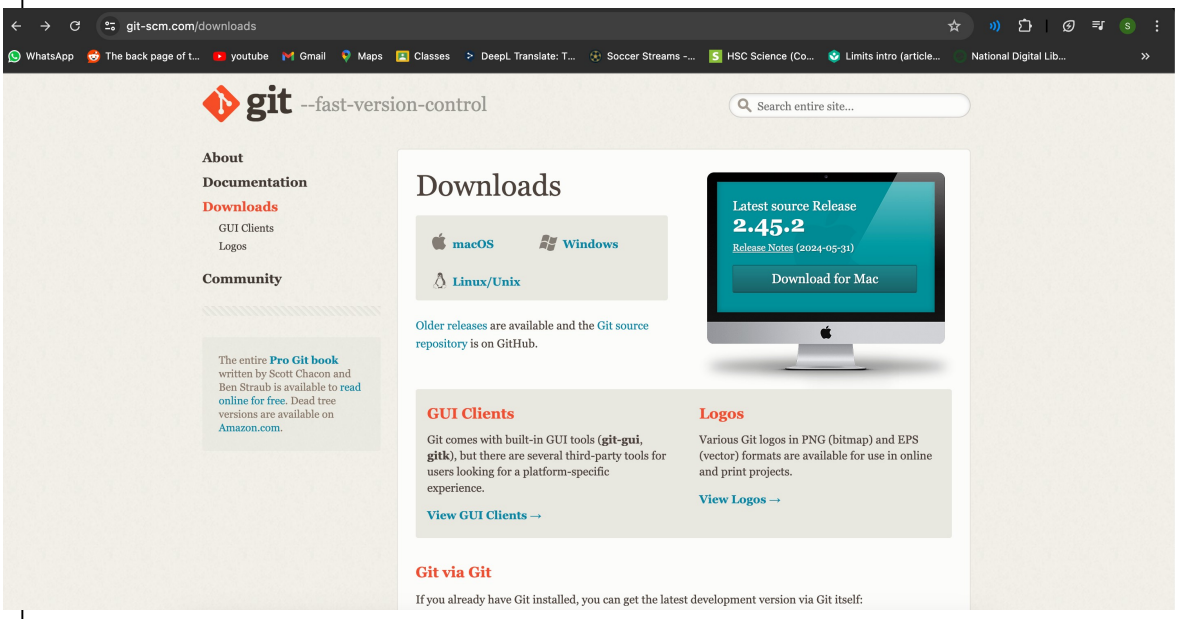
deployment processes. SCM also aids in managing multiple versions of the software, ensuring that the code remains high-quality and reducing the risk of errors.

**Different Tools Used for Version Control System and Source Code Management**

Common tools used for VCS and SCM include Git, Mercurial, Subversion (SVN), and Perforce. Git, often used with platforms like GitHub, GitLab, or Bitbucket, is the most popular due to its distributed nature and robust branching and merging capabilities. These tools provide the infrastructure for version tracking, collaboration, and maintaining the integrity of the source code. They help manage code changes systematically, facilitate teamwork, and ensure that projects are developed efficiently and effectively.

**Git installation:**

**Visit the official git website to install git.**




Command for new installation

```
(base) shayanahmad@Shayans-MacBook-Air ~ % brew install git
==> Auto-updating Homebrew...
Adjust how often this is run with HOMEBREW_AUTO_UPDATE_SECS or disable with
HOMEBREW_NO_AUTO_UPDATE. Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
==> Downloading https://ghcr.io/v2/homebrew/portable-ruby/portable-ruby/blobs/sha256:a37578bcc3b57e02bcf72ad85
3ee493ebf6f042755c4eb922c6cb8c1a905b807
######################################################################## 100.0%
==> Pouring portable-ruby-3.3.4.el_capitan.bottle.tar.gz
==> Auto-updated Homebrew!
Updated 3 taps (homebrew/services, homebrew/core and homebrew/cask).
```

**If already installed, command to update git.**

```
MINGW64:/c/Users/MacBook Air

MacBook Air@DESKTOP-CFC7DEC MINGW64 ~
$ git clone https://github.com/git/git
Cloning into 'git'...
remote: Enumerating objects: 373676, done.
remote: Counting objects: 100% (1736/1736), done.
remote: Compressing objects: 100% (745/745), done.
remote: Total 373676 (delta 1160), reused 1380 (delta 984), pack-reused 3719
40 (from 1)
Receiving objects: 100% (373676/373676), 245.24 MiB | 7.92 MiB/s, done.
Resolving deltas: 100% (281424/281424), done.
Updating files: 100% (4522/4522), done.
```

**Basic git commands:**

git –version:

```
PS C:\Sem 5\AWahid\gitdemo\firstrepo> git --version
git version 2.46.0.windows.1
```

mkdir:

```
PS C:\Sem 5\AWahid\gitdemo\firstrepo> mkdir dop
```

cd:

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo> cd dop
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop>
```

Configure the author name and email address to be used with your commits.

git config

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git config --global user.name Shaikh Wahid
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git config --global user.email shaikhabdulwahid19@gmail.com
```

Login Github credentials into Git

```
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations on this host? HTTPS
[? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: A9B2-3B26
Press Enter to open github.com in your browser...

✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as shayan729
```

git init

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git init
Initialized empty Git repository in C:/Sem 5/AWahid/GitDemo/firstrepo/dop/.git/
```

Git commit

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git commit -m "Initial commit"
On branch main

Initial commit
```

Git status

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

dir

```
PS C:\Sem 5\AWahid\GitDemo> dir


    Directory: C:\Sem 5\AWahid\GitDemo


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         9/25/2024   9:34 PM                firstrepo
d-----          9/6/2024  12:17 AM                git setup
```

git  branch

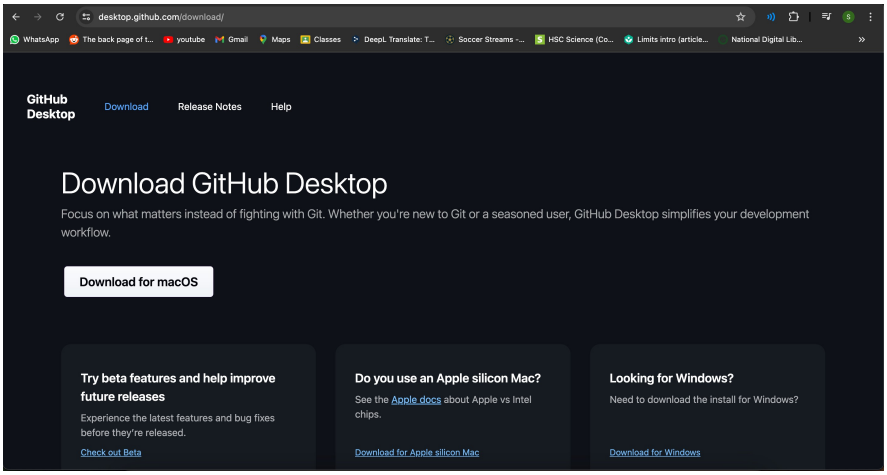
git checkout

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo> git branch exp2
```

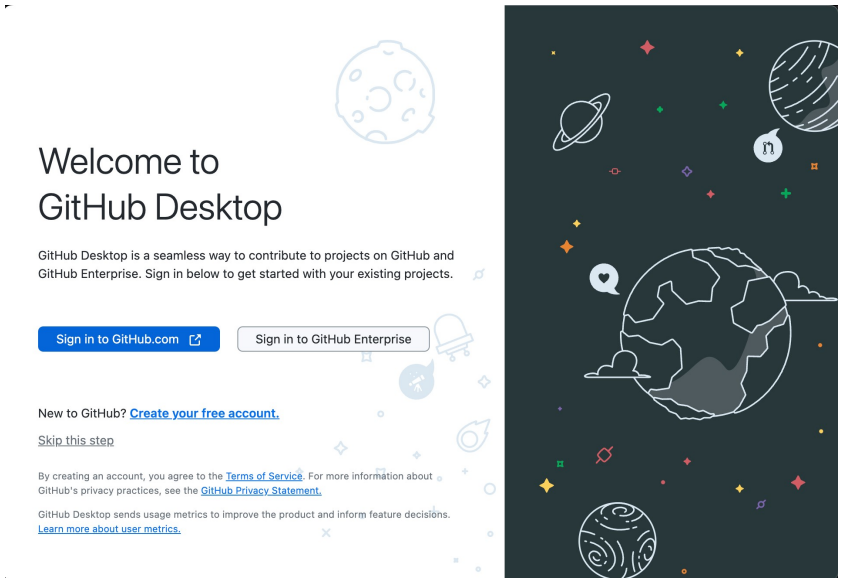
```
PS C:\Sem 5\AWahid\GitDemo\firstrepo> git checkout exp2
M       README.md
M       index.html
Switched to branch 'exp2'
```

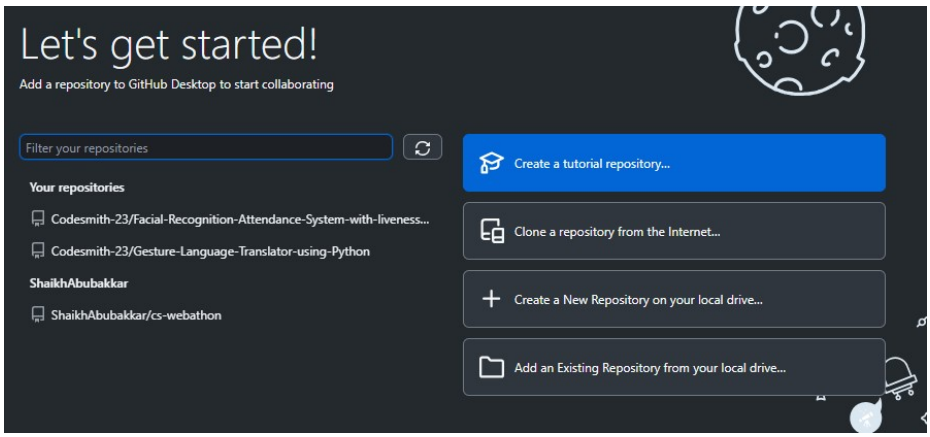
Installation of Github Desktop:

Visit the official Github website:




Sign-in into Github account


Welcome to
GitHub Desktop
GitHub Desktop is a seamless way to contribute to projects on GitHub and GitHub Enterprise. Sign in below to get started with your existing projects.
Sign in to GitHub.com
Sign in to GitHub Enterprise
New to GitHub? Create your free account.
Skip this step
By creating an account, you agree to the Terms of Service. For more information about GitHub's privacy practices, see the GitHub Privacy Statement.
GitHub Desktop sends usage metrics to improve the product and inform feature decisions.
Learn more about user metrics.


Let's get started!
Add a repository to GitHub Desktop to start collaborating
Filter your repositories
Your repositories
Codesmith-23/Facial-Recognition-Attendance-System-with-liveness...
Codesmith-23/Gesture-Language-Translator-using-Python
ShaikhAbubakkar
ShaikhAbubakkar/cs-webathon
Create a tutorial repository...
Clone a repository from the Internet...
Create a New Repository on your local drive...
Add an Existing Repository from your local drive...

# Experiment no 03

**Aim:** Perform various git operation local and remote repository using git cheat sheet.

**git config –global color.ui auto:** set automatic command line coloring for Git for easy reviewing

```
PS C:\Sem 5\AWahid\GitDemo> git config --global color.ui auto
```

**git clone:** This command copies an existing Git repository from a remote server to the local machine. It sets up a new directory, initializes a Git repository, and downloads all the data from the specified remote repository.

```
PS C:\Sem 5\AWahid\GitDemo> git clone https://github.com/ShaikhWahid99/checkingself.git
Cloning into 'checkingself'...
remote: Enumerating objects: 35, done.
```

**git log:** This command displays the commit history for the repository, listing the commits made to the repository in reverse chronological order. It provides details such as commit hashes, author information, dates, and commit messages.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo> git log
commit 0794153bdca0819ce8ba87df3225c2816c50cf8f (HEAD -> exp2, origin/main, origin/HEAD, main)
Author: ShaikhWahid99 <shaikhabdulwahid19@gmail.com>
Date:   Sat Aug 3 23:21:12 2024 +0530

    Added new paragraph
```

**git commit:** This command displays the commit history for the repository, listing the commits made to the repository in reverse chronological order. It provides details such as commit hashes, author information, dates, and commit messages.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git commit -m "Initial commit"
On branch main

Initial commit
```

**git remote add [alias] [url]:** This command adds a new remote repository with a specified alias, allowing the user to easily interact with multiple remote repositories. It associates the given URL with the alias for future Git operations.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git remote add exp2 https://github.com/ShaikhWahid99/checkingself.git
```

**git pull:** fetch and merge any commits from the tracking remote branch

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git pull
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 35 (delta 10), reused 12 (delta 2), pack-reused 0 (from 0)
Unpacking objects: 100% (35/35), 6.63 KiB | 5.00 KiB/s, done.
From https://github.com/ShaikhWahid99/checkingself
 * [new branch]      main       -> exp2/main
```

**git merge [alias]/[branch**]: This command integrates changes from one branch into another, typically merging a remote-tracking branch into the current branch. It combines the commit histories and resolves any conflicts that arise during the merge process.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git merge exp2/main
```

**git stash:** This command temporarily shelves changes made to the working directory, allowing the user to switch contexts without committing the changes.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git stash
No local changes to save
```

**git stash pop:** This command reapplies the most recently stashed changes to the working directory, removing the changes from the stash list. It allows the user to continue working on previously stashed changes.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git stash pop
No stash entries found.
```

**git log –stat -M:** This command reapplies the most recently stashed changes to the working directory, removing the changes from the stash list. It allows the user to continue working on previously stashed changes

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git log --stat -M
commit 715ed731b106e1c98b9ebb8389035d3c21c5d859 (HEAD -> main, exp2/main)
Author: ShaikhWahid99 <shaikhabdulwahid19@gmail.com>
Date:   Fri Sep 6 22:25:59 2024 +0530

    Create 6sep1025pm

 6sep1025pm | 1 +

    Create 6sep1025pm

 6sep1025pm | 1 +
 1 file changed, 1 insertion(+)
```

**git diff –staged:** diff of what is staged but not yet committed

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git diff --staged
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop>
```

**git rm:** This command removes files from the working directory and stages the removal for the next commit. It also removes the file from the repository's history if the changes are committed.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> dir


    Directory: C:\Sem 5\AWahid\GitDemo\firstrepo\dop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/25/2024  10:08 PM             15 6sep1025pm
-a----         9/25/2024  10:08 PM            107 new .js
-a----         9/25/2024  10:08 PM             16 newnew.js
```

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> rm newnew.js
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> dir


    Directory: C:\Sem 5\AWahid\GitDemo\firstrepo\dop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/25/2024  10:08 PM             15 6sep1025pm
-a----         9/25/2024  10:08 PM            107 new .js
```

**git reset:** This command resets the current branch to a specified state, modifying the index and optionally the working directory. It can be used to unstage changes or reset the branch to a previous commit.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git reset
Unstaged changes after reset:
D       newnew.js
D       pully
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop>
```

**git reset –hard:** This command resets the current branch, index, and working directory to a specified commit. It discards all changes in the working directory and index, making it irreversible.

```
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop> git reset --hard
HEAD is now at 715ed73 Create 6sep1025pm
PS C:\Sem 5\AWahid\GitDemo\firstrepo\dop>
```

# EXPERIMENT NO 04

**AIM:** To understand Continuous Integration, install and configure Jenkins with Maven/Ant/Gradle to setup a build Job.

**THEORY:**

***What is Jenkins?***

- Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.
- Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.
- It is a self-contained Java program that is agnostic of the platform on which it is installed, making it available for almost all popular operating systems, including Windows, Unix flavors, and Mac OS.

## Additional Capabilities

- Supports various version control systems, including Git, Subversion, and Mercurial
- Executes Apache Ant, Apache Maven, and sbt-based projects as well as arbitrary shell scripts and Windows batch commands
- Allows storing credentials in Jenkins and provides a standardized API for other plugins to store and retrieve different types of credentials
- Enables monitoring of the result of externally executed jobs and managing agents (formerly known as slaves) running on *nix machines over SSH

## Pros

- Highly customizable through plugins and scripting
- Large and active community with extensive documentation and support
- Supports a wide range of build tools, version control systems, and bug databases
- Scalable and flexible architecture

## Cons

- Steep learning curve due to its complexity and customization options
- Requires significant setup and configuration effort
- Can be resource-intensive, especially with large numbers of jobs and slaves

***What Is Jenkins Used For?***

Jenkins software's popularity stems from its ability to track and monitor repetitive activities that emerge throughout a project's development. For example, if your team is working on a project, Jenkins will continually test your builds and alert you to any mistakes early in the process.

**What is Maven?**

Maven is an automation and management tool developed by the Apache Software Foundation. It simplifies and standardizes the project build process, handling compilation, distribution, documentation, team collaboration, and other tasks seamlessly.

**The following are the key features of Maven in a nutshell:**

- Simple project setup that follows best practices - get a new project or module started in seconds
- Consistent usage across all projects - means no ramp up time for new developers coming onto a project
- Superior dependency management including automatic updating, dependency closures (also known as transitive dependencies)
- Able to easily work with multiple projects at the same time
- A large and growing repository of libraries and metadata to use out of the box, and arrangements in place with the largest Open Source projects for real-time availability of their latest releases
- Extensible, with the ability to easily write plugins in Java or scripting languages
- Instant access to new features with little or no extra configuration
- Ant tasks for dependency management and deployment outside of Maven
- Model based builds: Maven is able to build any number of projects into predefined output types such as a JAR, WAR, or distribution based on metadata about the project, without the need to do any scripting in most cases.
- Coherent site of project information: Using the same metadata as for the build process, Maven is able to generate a web site or PDF including any documentation you care to add, and adds to that standard reports about the state of development of the project. Examples of this information can be seen at the bottom of the left-hand navigation of this site under the "Project Information" and "Project Reports" submenus.
- Release management and distribution publication: Without much additional configuration, Maven will integrate with your source control system (such as Subversion or Git) and manage the release of a project based on a certain tag. It can also publish this to a distribution location for use by other projects. Maven is able to publish individual outputs such as a JAR, an archive including other dependencies and documentation, or as a source distribution.
- Dependency management: Maven encourages the use of a central repository of JARs and other dependencies. Maven comes with a mechanism that your project's clients can use to download any JARs required for building your project from a central JAR repository much like Perl's CPAN. This allows users of Maven to reuse JARs across projects and encourages communication between projects to ensure that backward compatibility issues are dealt with.

**IMPLEMENTATION:**

**CREATE A MAVEN-BASED JAVA PROJECT IN INTELLIJ IDEA:**

**Step 1: Install Maven**

1. **Download Maven:** ○ Go to the Maven download page and download the latest version.
2. **Install Maven:** ○ Unzip the downloaded archive to a directory of your choice.
   - ○ Add the bin directory of the extracted Maven folder to your system's PATH environment variable.
3. **Verify Installation:** ○ Open a terminal or command prompt and type:

   ***mvn -v***

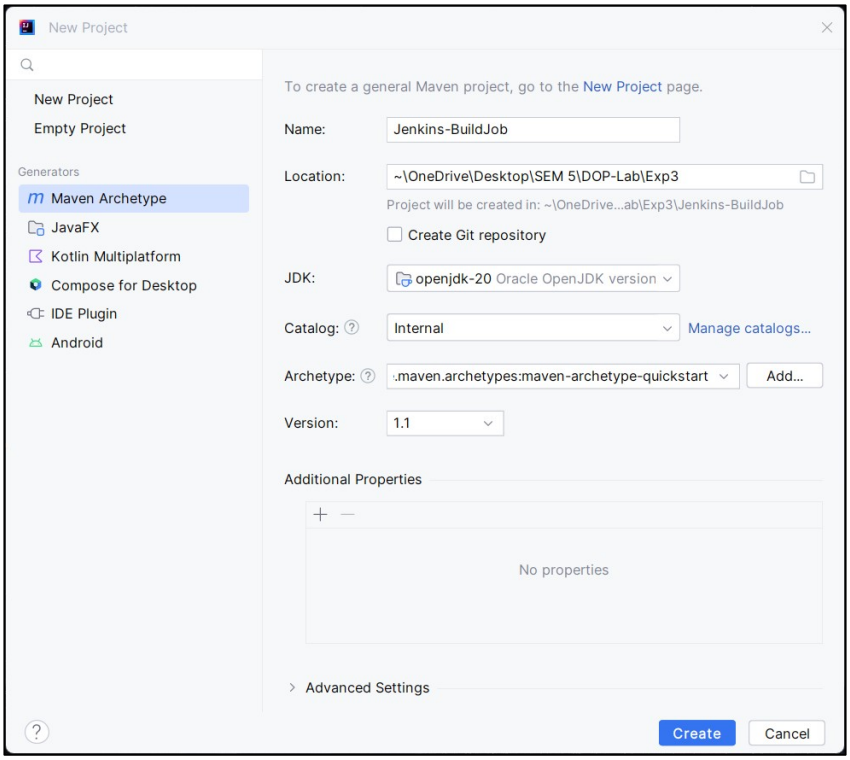
   - ○ You should see output with Maven's version, Java version, and other information.


**Step 2**: Create a New Maven Project in IntelliJ IDEA

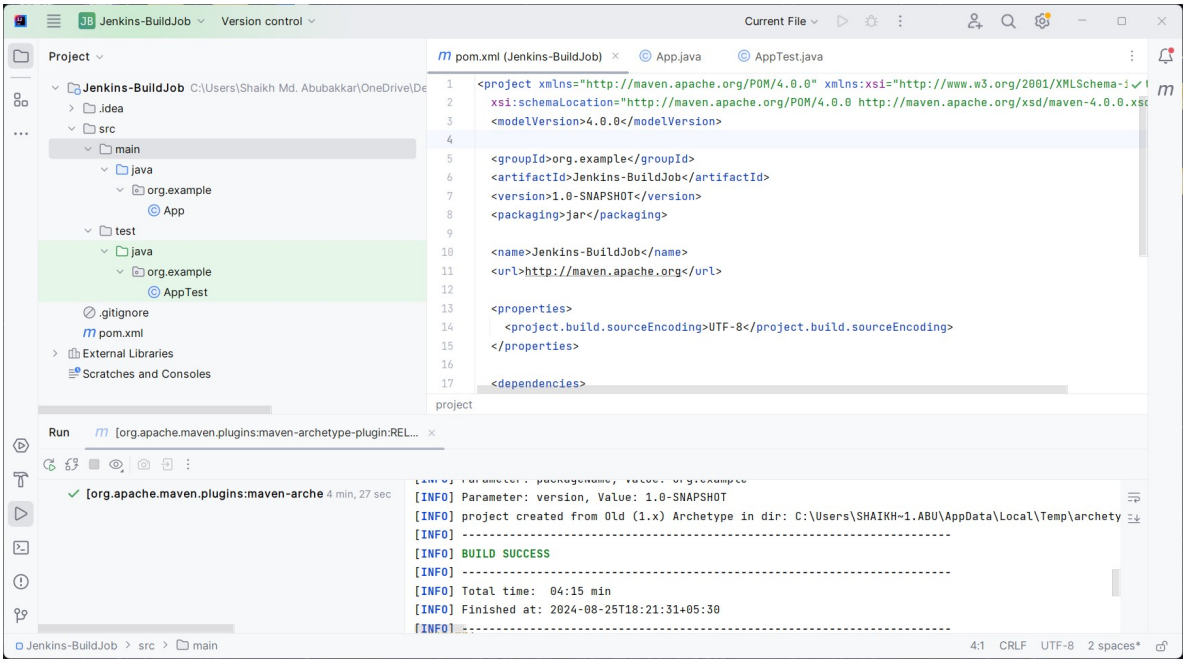
1. Open IntelliJ IDEA: ○ If you don't have a project open, you'll see the Welcome screen. If you do have a project open, go to File > New > Project.
2. Select Maven: ○ In the "New Project" window, select Maven from the left panel.
   - ○ Check Create from archetype if you want to use a specific archetype (for example, the quickstart archetype).
3. Configure Project SDK: ○ Make sure you have a JDK installed and configured. If not, click New and select the JDK installation directory.
   - ○ Click Next.

(The below two steps are executed automatically after creating the project. In case of manual execution, go for them.)

4. Set Project Coordinates:
   - ○ Enter the GroupId (e.g., com.example). ○ Enter the ArtifactId (e.g., my-java-app).
   - ○ The Version is usually set to 1.0-SNAPSHOT by default.
   - ○ Click Next.
5. Specify the Project Name and Location: ○ Enter the Project Name (e.g., my-java-app). ○ Choose the Project Location where you want to save the project on your computer.
   - ○ Click Finish.




Our Maven Project has been created successfully.

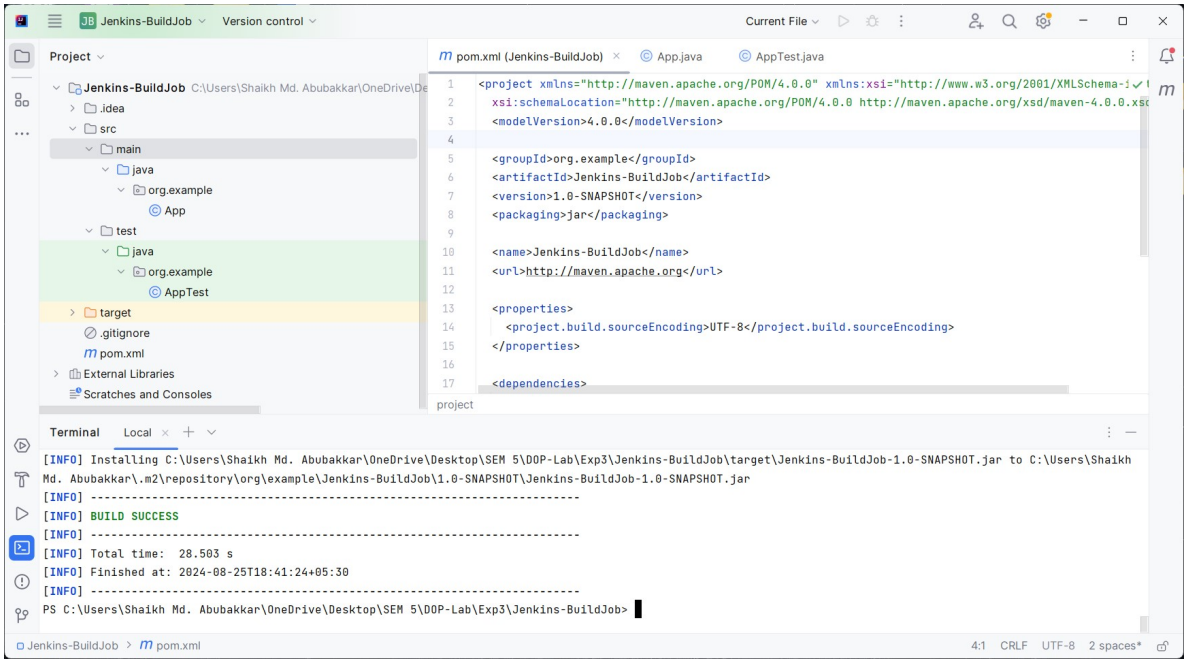



**Step 3**: Build the Project with Maven 1. Navigate to the Project Directory:

2. Build the Project:
    - Run the following Maven command to compile the project and run the tests:

***mvn clean install***

- Maven will compile the App.java, run the AppTest.java, and package the application into a JAR file located in the target/ directory.




**Step 4**: Run the Application

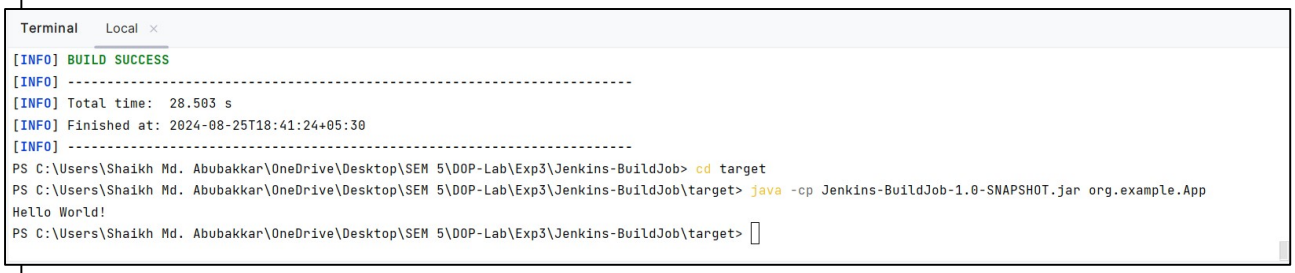
1. Navigate to the target Directory:

   ***cd target***

2. Run the JAR File:

   - Use the following command to run your application: ***java -cp Jenkins-BuildJob-1.0-SNAPSHOT.jar org.example.App*** ○ This should output:

***Hello World!***

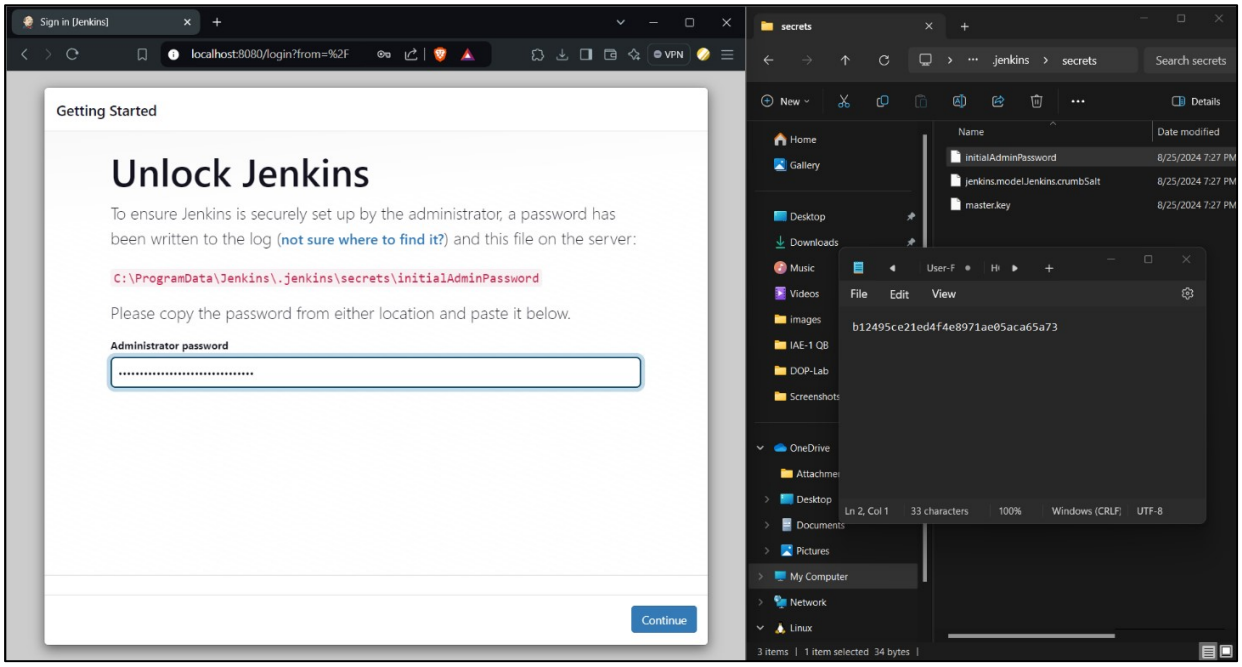


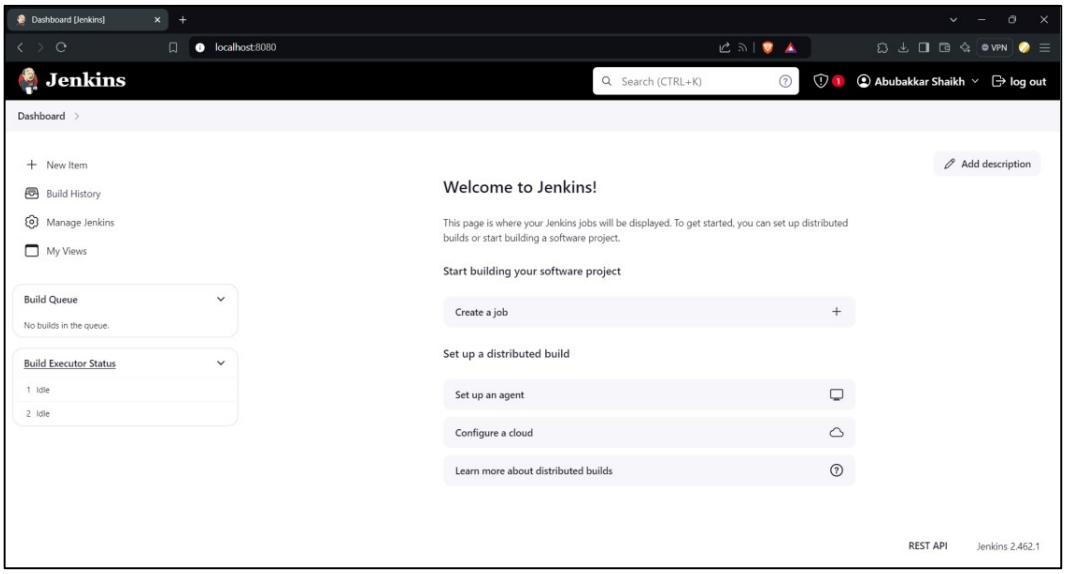

We've now created a basic Java application using Maven. This project structure is ready to be integrated into Jenkins for automated builds. We can further customize the pom.xml to include more dependencies, plugins, or configurations as needed.

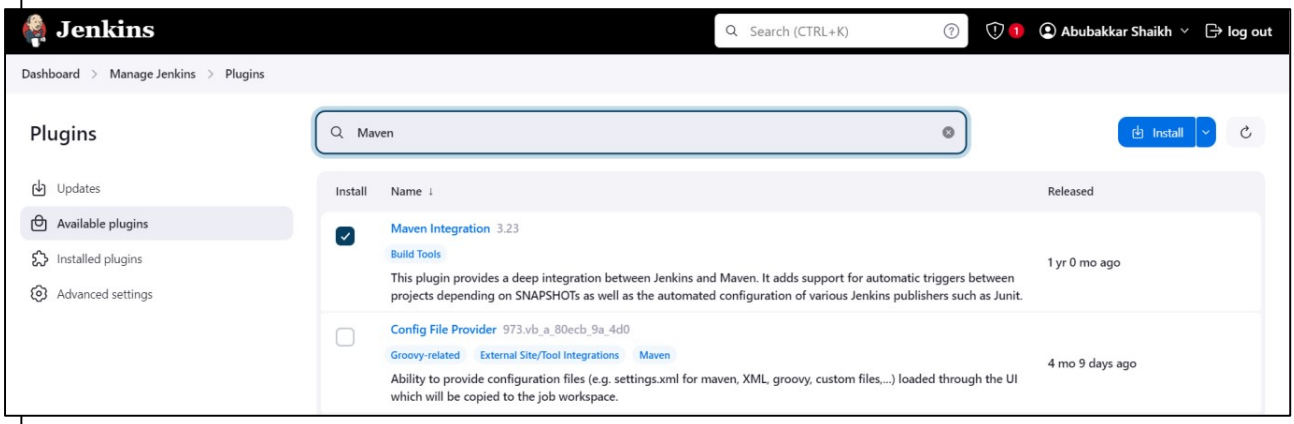
**SET UP A JENKINS BUILD JOB FOR A JAVA APPLICATION USING MAVEN WITH THE SOURCE CODE ON YOUR LOCAL MACHINE:**

**Step 1:** Install Jenkins and Plugin.

1. Install Jenkins:
    - Ensure Jenkins is installed and running on your server.
2. Install Required Plugins:
    - Install the Maven Integration Plugin and JUnit Plugin via Manage Jenkins > Manage Plugins.



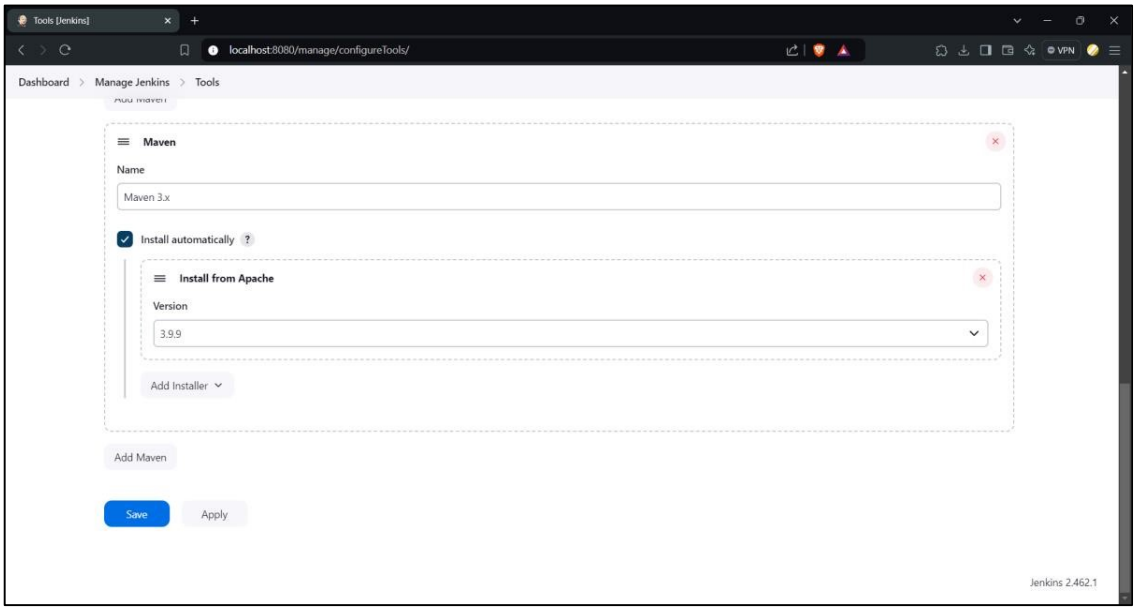






**Step 2:** Configure Maven in Jenkins

1. Add Maven to Jenkins:

- Go to Manage Jenkins > Global Tool Configuration. ○ Under Maven, click Add Maven.
- Name it (e.g., Maven 3.x), and either install automatically or provide the path to your existing Maven installation.

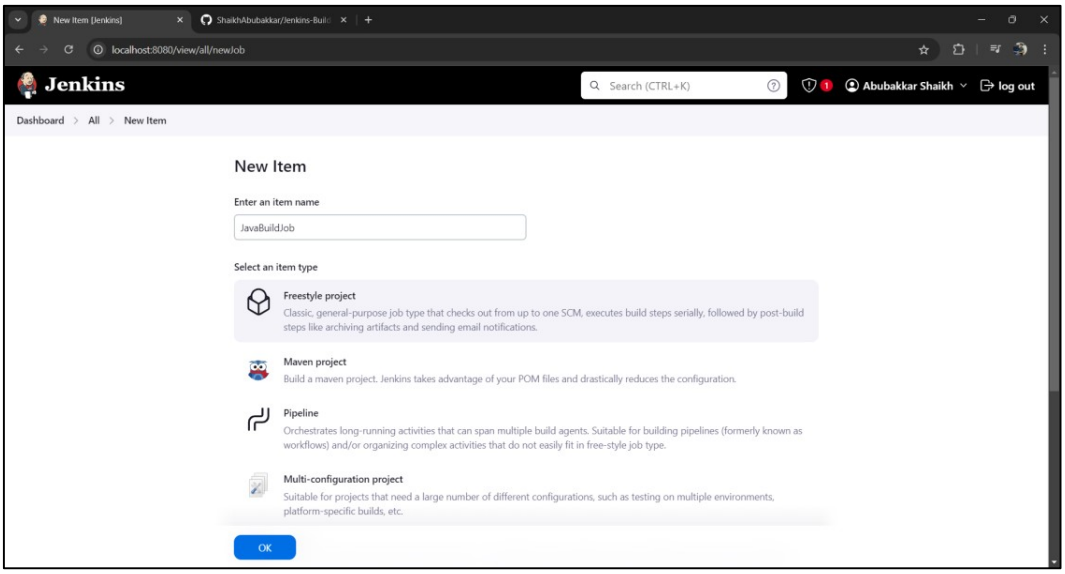



**Step 3:** Create a New Jenkins Job

1. Create a New Job:

- From the Jenkins dashboard, click New Item.
- Enter a name for your job (e.g., JavaBuildJob).

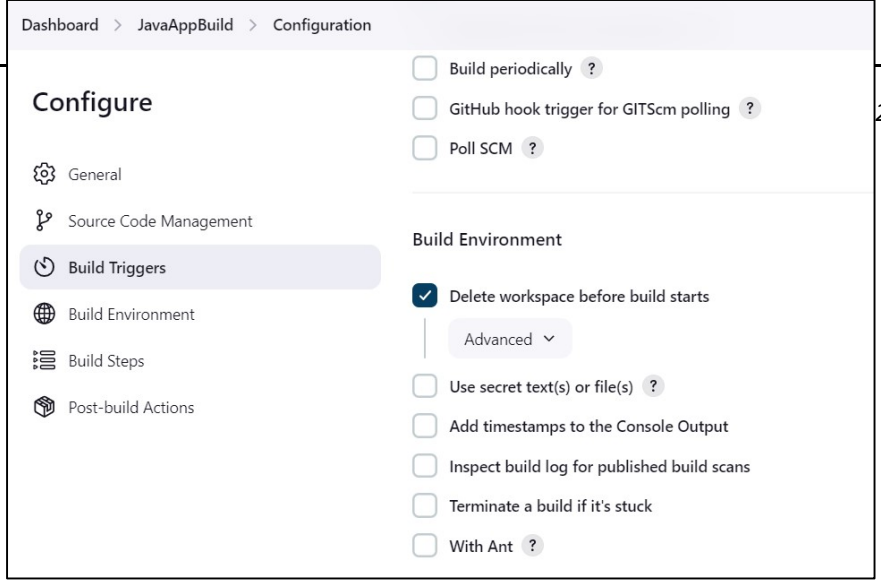
- Select Freestyle Project and click OK.




**Step 4:** Configure the Build Environment

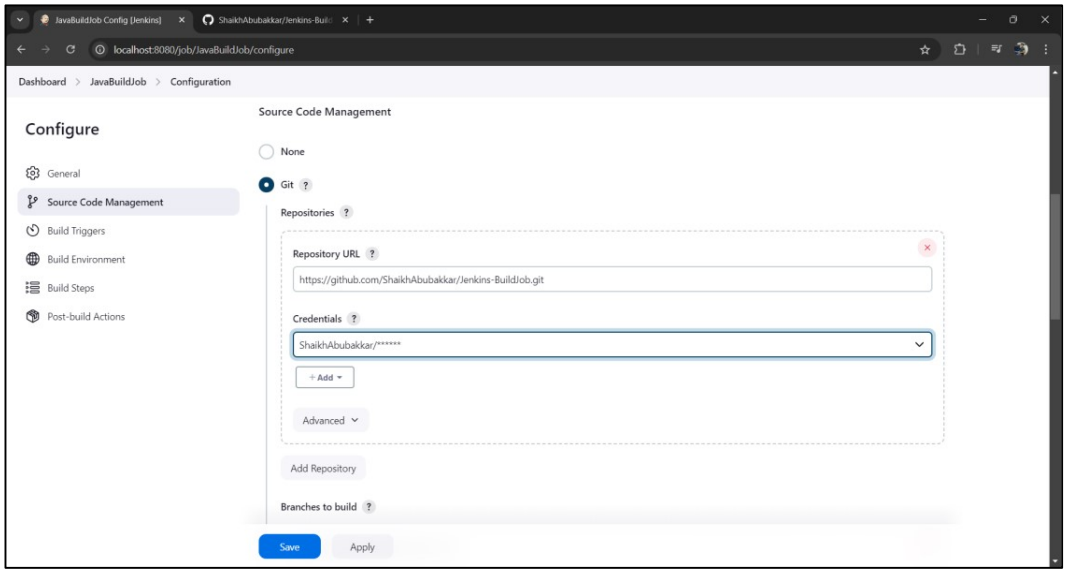
1. Configure the Build Environment:

   - Scroll to the Build Environment section.
   - If necessary, check the option Delete workspace before build starts to ensure a clean build.



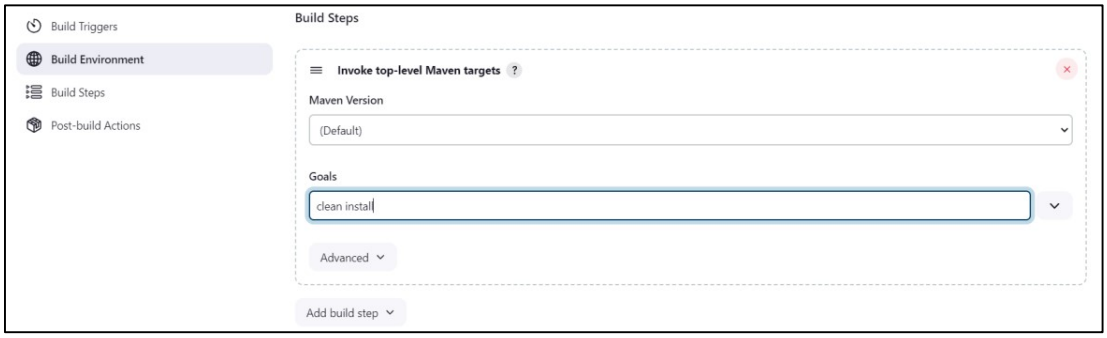

**Step 5:** Add Build Steps

1. **Specify Source Code Location**:
   - Go to the source code management section
   - Check the 'Git' radio button
   - Enter Repository URL and Git credentials
   - Specify the branch (eg. ***/master***)



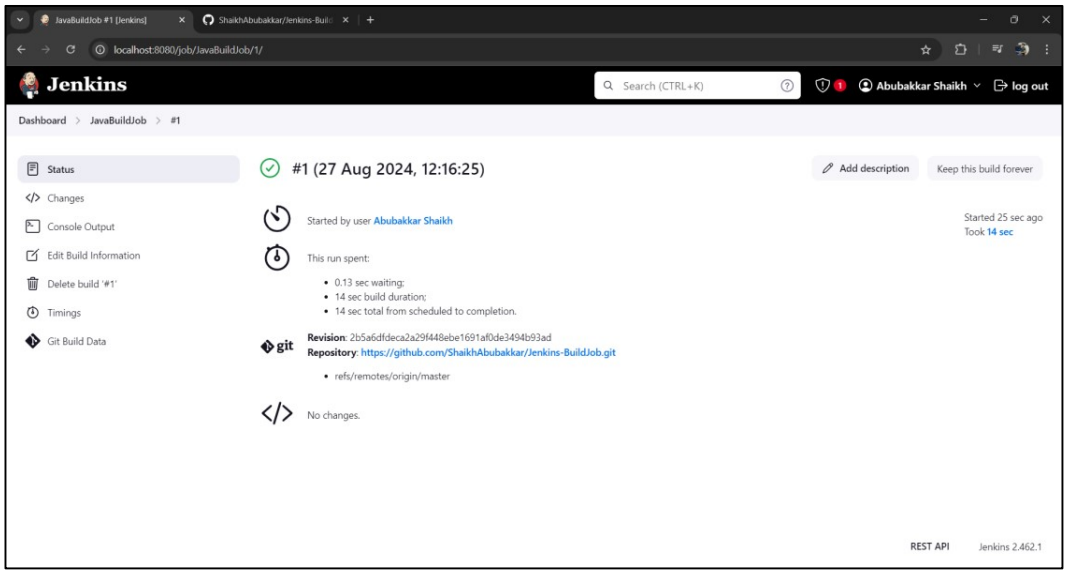

2. **Add Build Steps** :

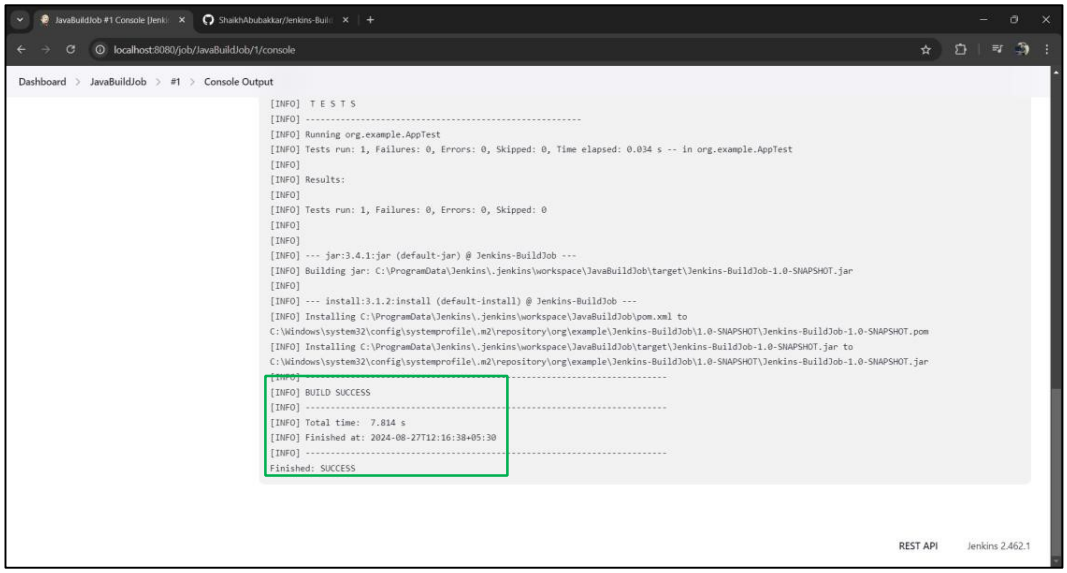
○ In the Build section, click Add build step. ○ Choose Invoke top-level Maven targets. ○ Goals: Enter clean install or the specific Maven goals you want to run.




**Step 6:** Check results ○ Apply & Save all the changes ○ Click on 'Build Now' to initialize the build

- Check for output in the console




**CONCLUSION:** In this experiment, we installed Maven in our local machine. We created a Java Maven project using the IntelliJ IDE and integrated this Maven project on Jenkins. On Jenkins, we installed the required plugins for Maven and after setting up the Build Environment our Java Maven project was successfully integrated on Jenkins.

*Abubakkar-221448 B-Batch*

**DOP Lab Experiment No 05**

**Theory:**

A Jenkins pipeline is a series of automated steps that you can define to build, test, and deploy your application. It's used to implement Continuous Integration (CI) and Continuous Deployment (CD), ensuring software updates are delivered more reliably and efficiently. When deploying to a server like Apache Tomcat, Jenkins can handle the process by executing the necessary build, package, and deployment steps.

Jenkins Pipeline

There are two main types of Jenkins pipelines:

1. Declarative Pipeline: More structured and simple to read. It follows a specific syntax.

2. Scripted Pipeline: More flexible and provides powerful control over the pipeline.

A typical Jenkins pipeline has stages like:

Build: Compiling the code (for Java applications, this could be done using tools like Maven or Gradle).

Test: Running unit or integration tests.

Package: Bundling the application, typically into a .war file for Java web applications.

Deploy: Uploading the package to a server like Apache Tomcat.

Jenkins Deployment on Apache Tomcat

1. Pre-requisites:

Tomcat server must be installed and running on a machine (local or remote).

Jenkins must have the Deploy to Container plugin installed.

Jenkins should have credentials to access the Tomcat Manager (configured in Jenkins as credentials).

2. Pipeline steps for deploying to Tomcat:

Build the project using Maven or any build tool you are using.

Package the project as a .war file (standard for Java web apps).

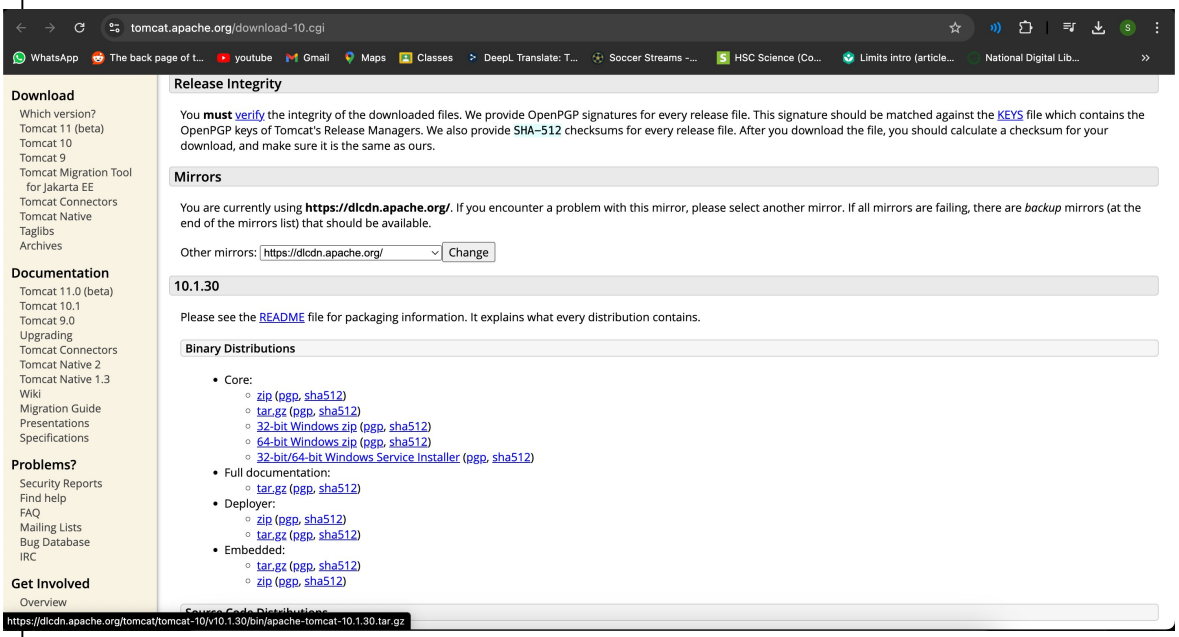
Transfer the .war file to Apache Tomcat using Jenkins' integration with Tomcat Manager.

3. Tomcat Configuration:

Make sure that the Tomcat Manager app is enabled and the user credentials are available in tomcat-users.xml file for deployment.

**Installation and setup of Apache Tomcat**

From tomcat.apache.org/download install the binary distribution




Launch terminal cd into the installed directory, cd into bin directory

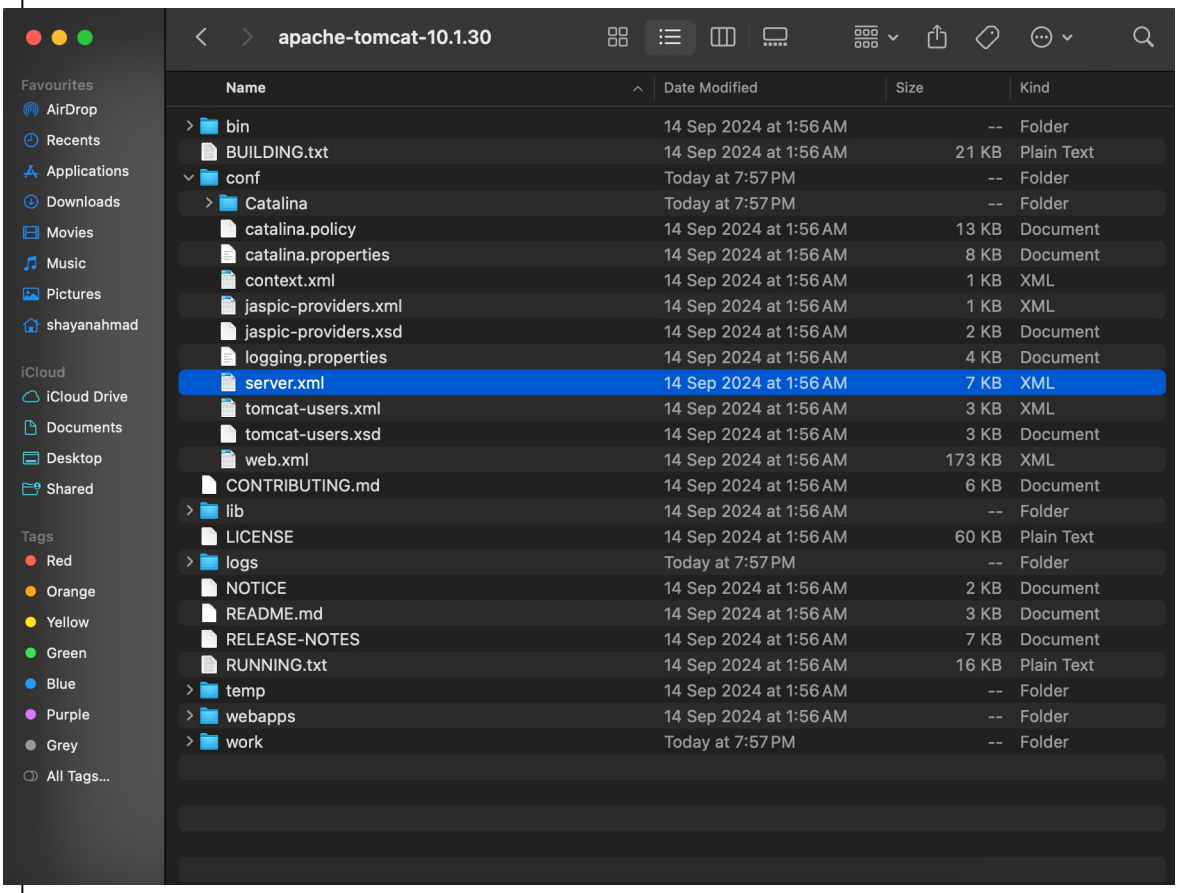
```
(base) shayanahmad@Shayans-MacBook-Air ~ % cd apache-tomcat-10.1.30
```

```
(base) shayanahmad@Shayans-MacBook-Air apache-tomcat-10.1.30 % cd bin
```

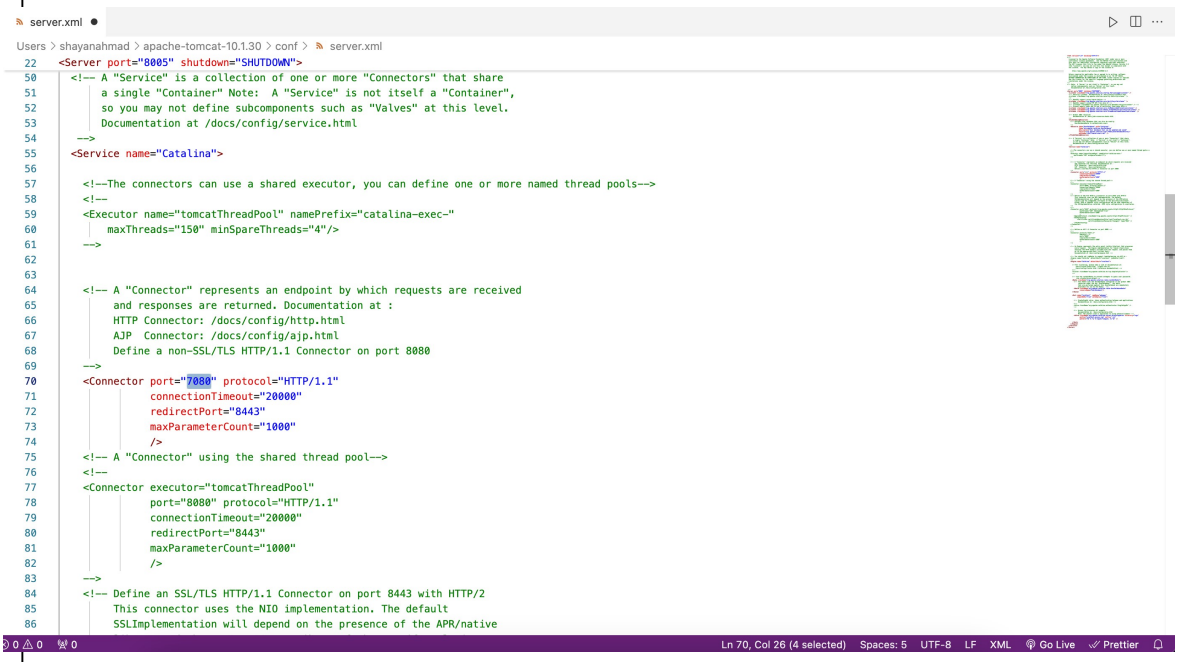
List all .sh files, give execute privilege to these files

```
(base) shayanahmad@Shayans-MacBook-Air bin % ls -al *.sh
-rwxr-x---@ 1 shayanahmad  staff  23445 Sep 14 01:56 catalina.sh
-rwxr-x---@ 1 shayanahmad  staff   1997 Sep 14 01:56 ciphers.sh
-rwxr-x---@ 1 shayanahmad  staff   1922 Sep 14 01:56 configtest.sh
-rwxr-x---@ 1 shayanahmad  staff   8319 Sep 14 01:56 daemon.sh
-rwxr-x---@ 1 shayanahmad  staff   1965 Sep 14 01:56 digest.sh
-rwxr-x---@ 1 shayanahmad  staff   3382 Sep 14 01:56 makebase.sh
-rwxr-x---@ 1 shayanahmad  staff   1970 Sep 14 01:56 migrate.sh
-rwxr-x---@ 1 shayanahmad  staff   3941 Sep 14 01:56 setclasspath.sh
-rwxr-x---@ 1 shayanahmad  staff   1902 Sep 14 01:56 shutdown.sh
-rwxr-x---@ 1 shayanahmad  staff   1904 Sep 14 01:56 startup.sh
-rwxr-x---@ 1 shayanahmad  staff   4600 Sep 14 01:56 tool-wrapper.sh
-rwxr-x---@ 1 shayanahmad  staff   1908 Sep 14 01:56 version.sh
(base) shayanahmad@Shayans-MacBook-Air bin % chmod +x *.sh
(base) shayanahmad@Shayans-MacBook-Air bin % ls -al *.sh
-rwxr-x--x@ 1 shayanahmad  staff  23445 Sep 14 01:56 catalina.sh
-rwxr-x--x@ 1 shayanahmad  staff   1997 Sep 14 01:56 ciphers.sh
-rwxr-x--x@ 1 shayanahmad  staff   1922 Sep 14 01:56 configtest.sh
-rwxr-x--x@ 1 shayanahmad  staff   8319 Sep 14 01:56 daemon.sh
-rwxr-x--x@ 1 shayanahmad  staff   1965 Sep 14 01:56 digest.sh
-rwxr-x--x@ 1 shayanahmad  staff   3382 Sep 14 01:56 makebase.sh
-rwxr-x--x@ 1 shayanahmad  staff   1970 Sep 14 01:56 migrate.sh
-rwxr-x--x@ 1 shayanahmad  staff   3941 Sep 14 01:56 setclasspath.sh
-rwxr-x--x@ 1 shayanahmad  staff   1902 Sep 14 01:56 shutdown.sh
-rwxr-x--x@ 1 shayanahmad  staff   1904 Sep 14 01:56 startup.sh
-rwxr-x--x@ 1 shayanahmad  staff   4600 Sep 14 01:56 tool-wrapper.sh
-rwxr-x--x@ 1 shayanahmad  staff   1908 Sep 14 01:56 version.sh
```

As Jenkins is already running on port 8080, setup Tomcat on 7080 port. Navigate to conf directory, open server.xml



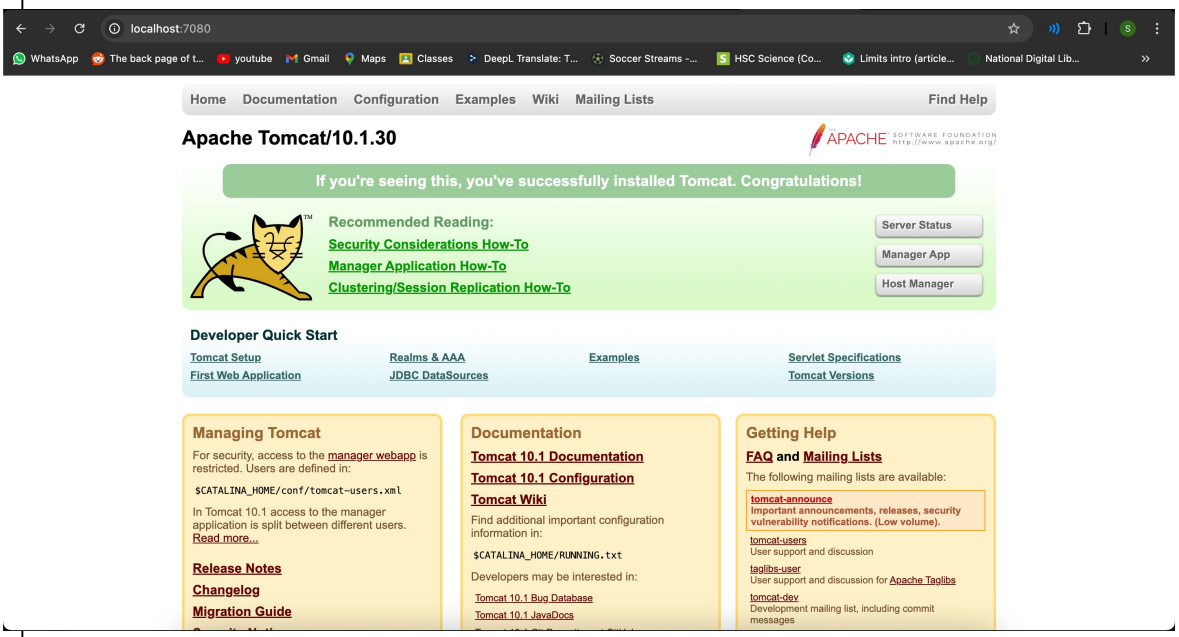

Change the port no to 7080 and save the file



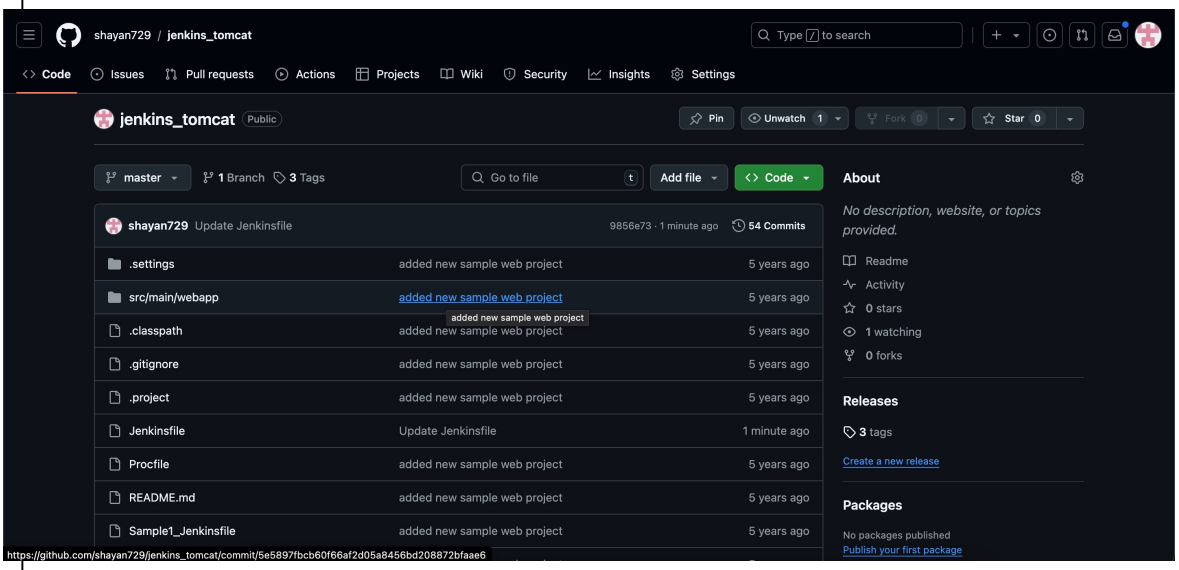
cd to bin directory and start tomcat server using “./startup.sh”

```
(base) shayanahmad@Shayans-MacBook-Air bin % ./startup.sh
Using CATALINA_BASE:   /Users/shayanahmad/apache-tomcat-10.1.30
Using CATALINA_HOME:   /Users/shayanahmad/apache-tomcat-10.1.30
Using CATALINA_TMPDIR: /Users/shayanahmad/apache-tomcat-10.1.30/temp
Using JRE_HOME:        /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home
Using CLASSPATH:       /Users/shayanahmad/apache-tomcat-10.1.30/bin/bootstrap.jar:/Users/shayanahm
ad/apache-tomcat-10.1.30/bin/tomcat-juli.jar
Using CATALINA_OPTS:
Tomcat started.
```

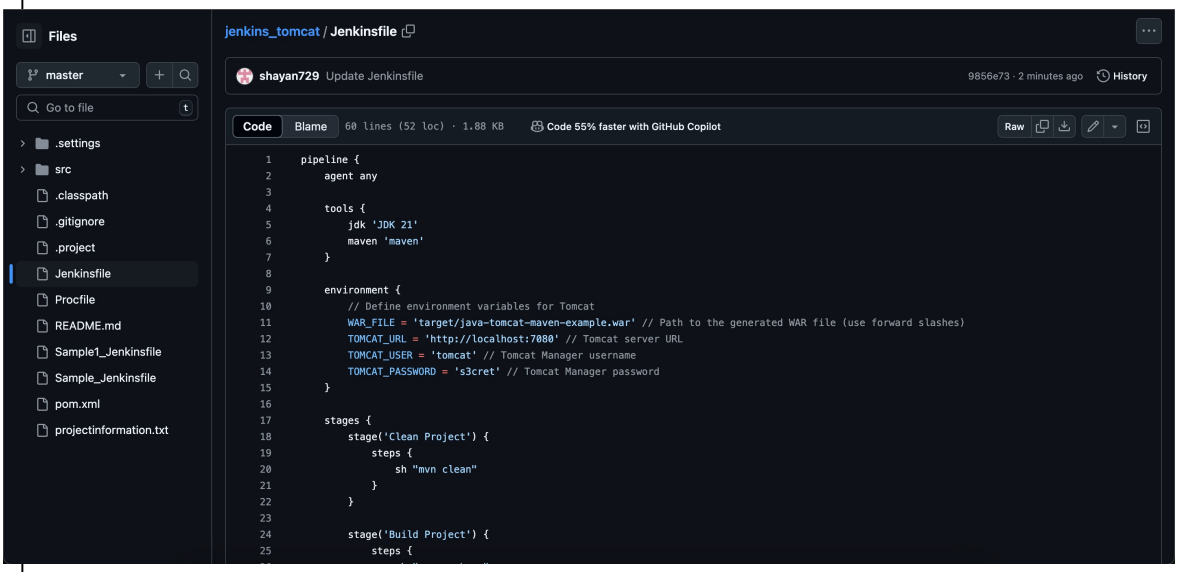
Tomcat dashboard



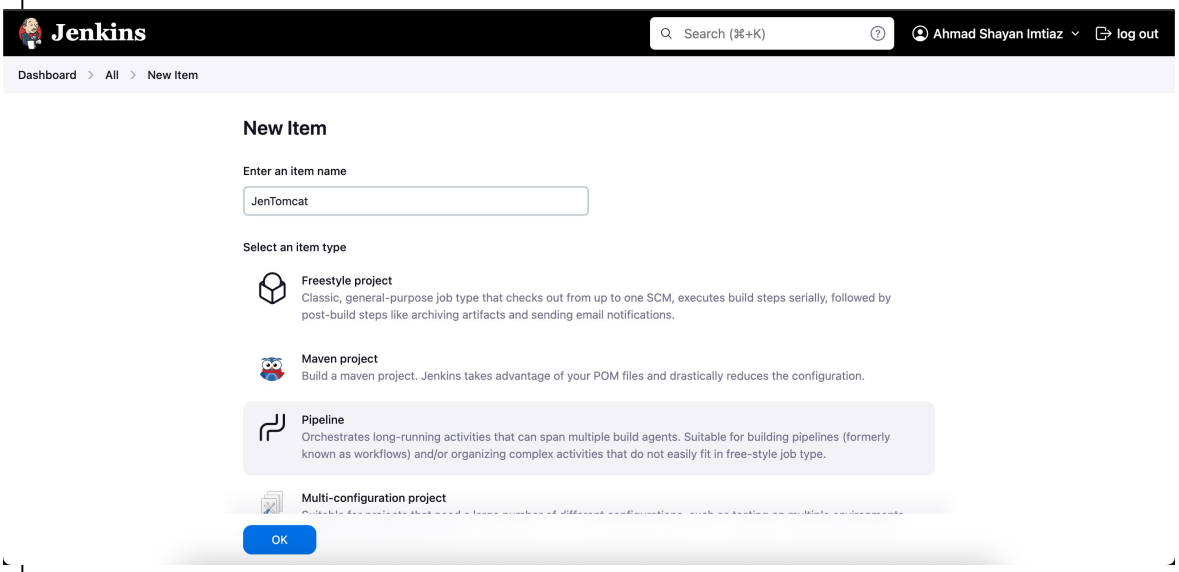

Create or use a GitHub web project  repository with Jenkins file



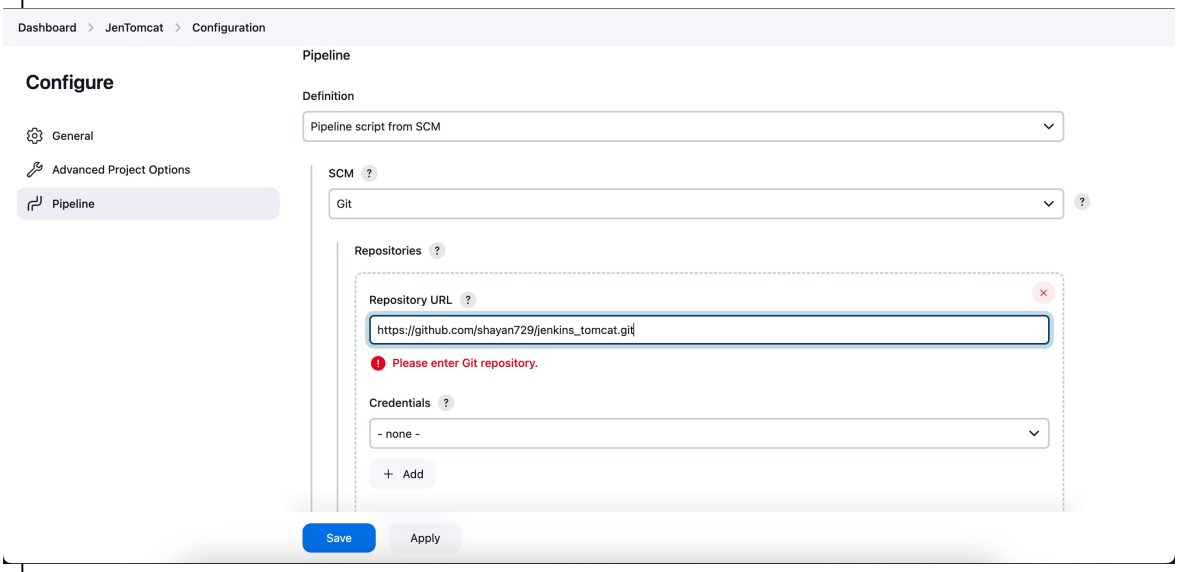

Add path to the WAR file, tomcat username, password. All executing command will be sh for  Linus or OSX systems while bat for Windows systems.



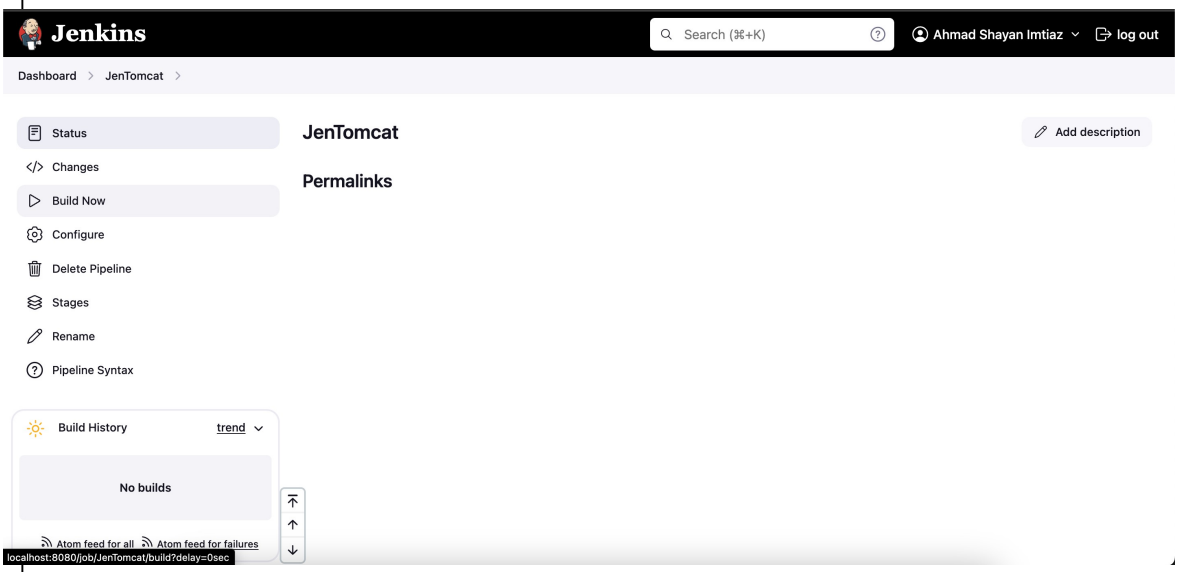

Create a new Jenkins pipeline

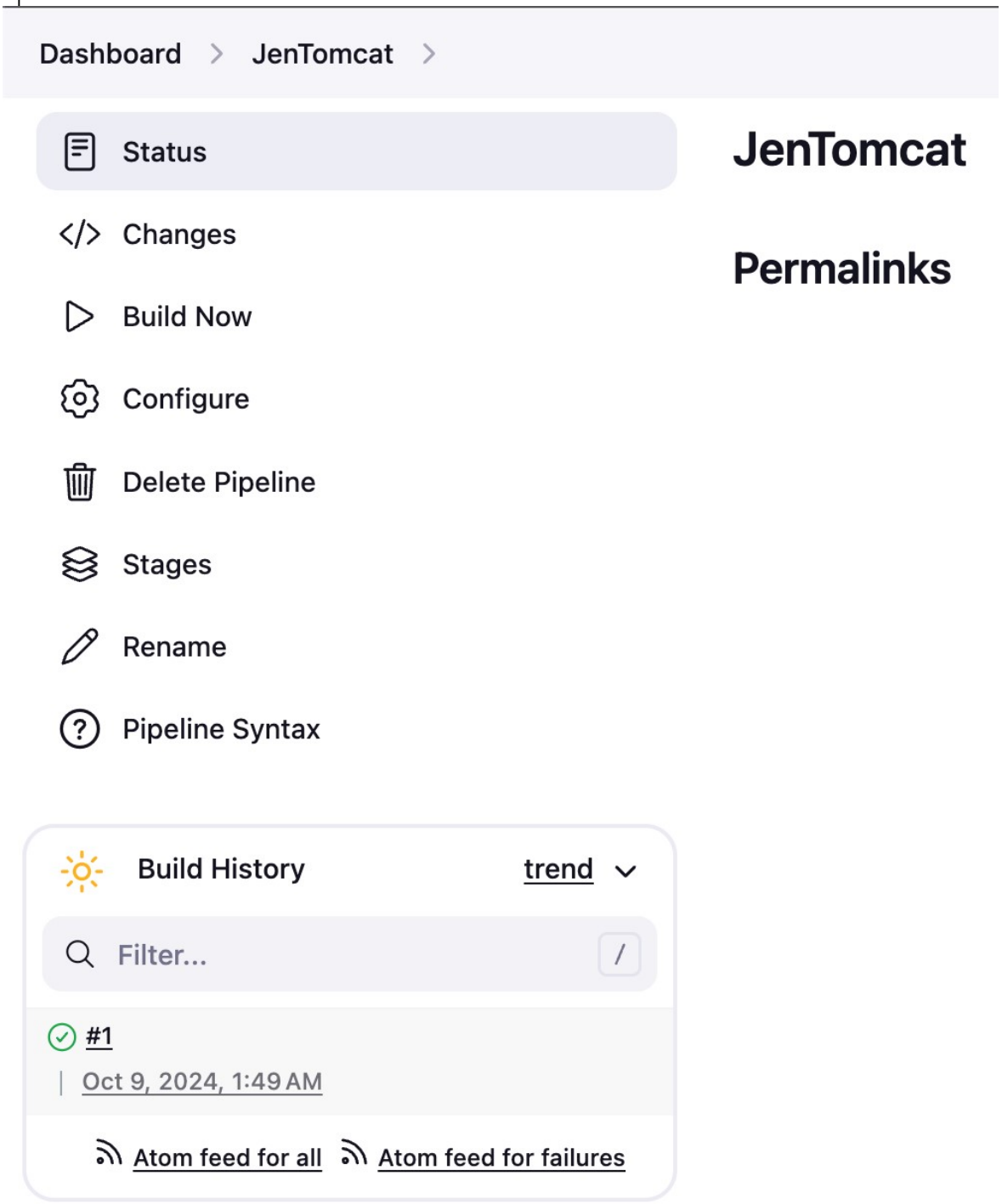



Choose pipeline script from SCM,  paste the URL of the repository, then apply and save

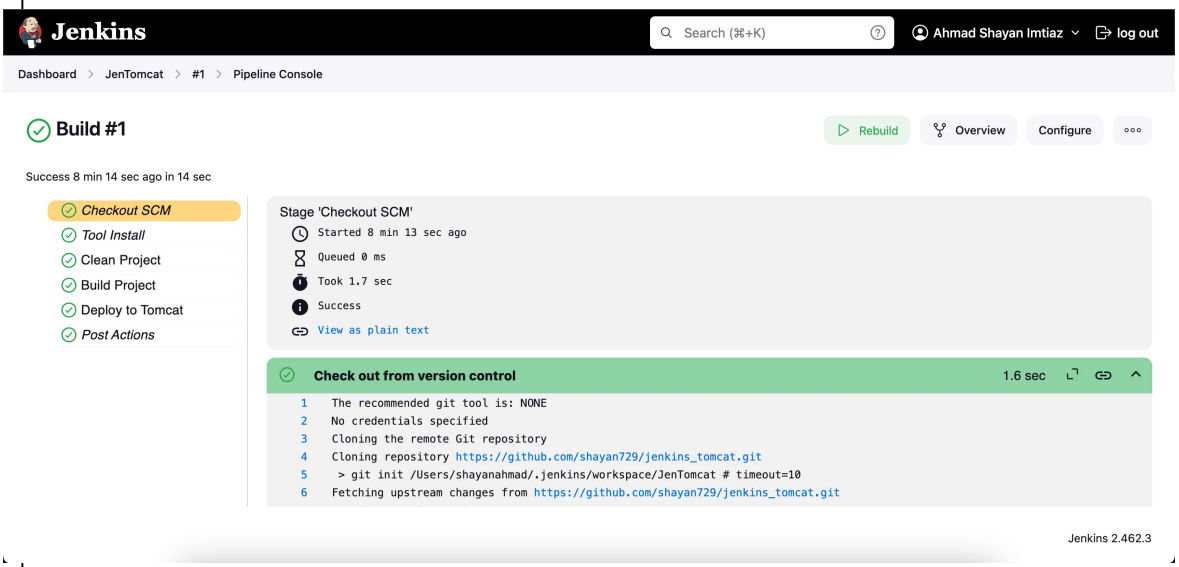


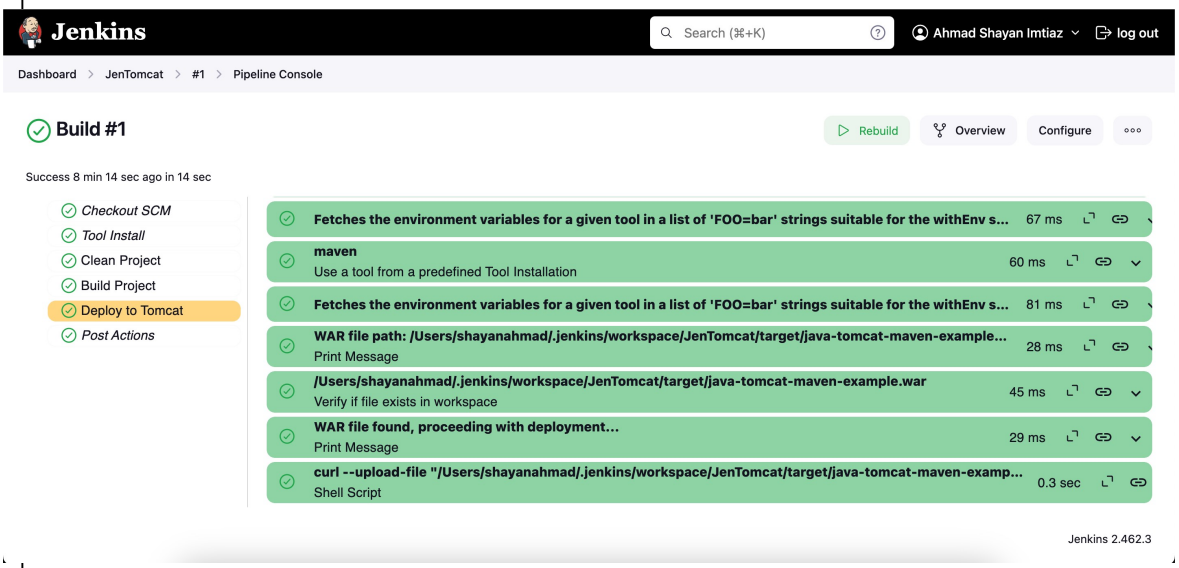

Start a build


Jenkins
Search (⌘+K)
Ahmad Shayan Imtiaz
log out
Dashboard
JenTomcat
Status
Changes
Build Now
Configure
Delete Pipeline
Stages
Rename
Pipeline Syntax
JenTomcat
Add description
Permalinks
Build History
trend
No builds
Atom feed for all
Atom feed for failures
localhost:8080/job/JenTomcat/build?delay=0sec

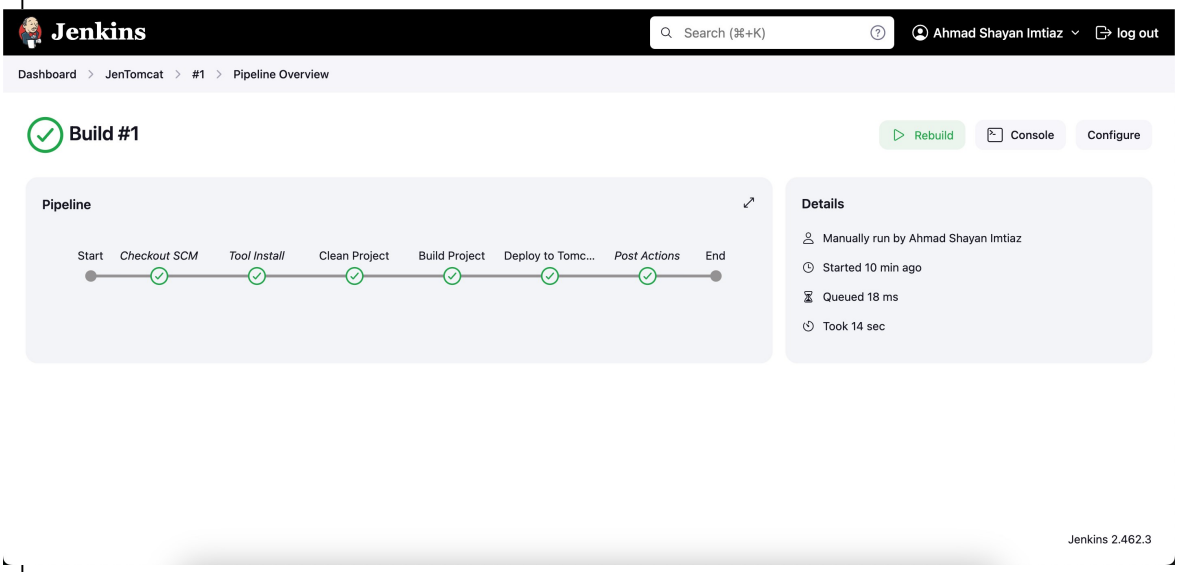

Dashboard
JenTomcat
Status
Changes
Build Now
Configure
Delete Pipeline
Stages
Rename
Pipeline Syntax
JenTomcat
Permalinks
Build History
trend
Filter...
/
#1
Oct 9, 2024, 1:49 AM
Atom feed for all
Atom feed for failures

Pipeline Console

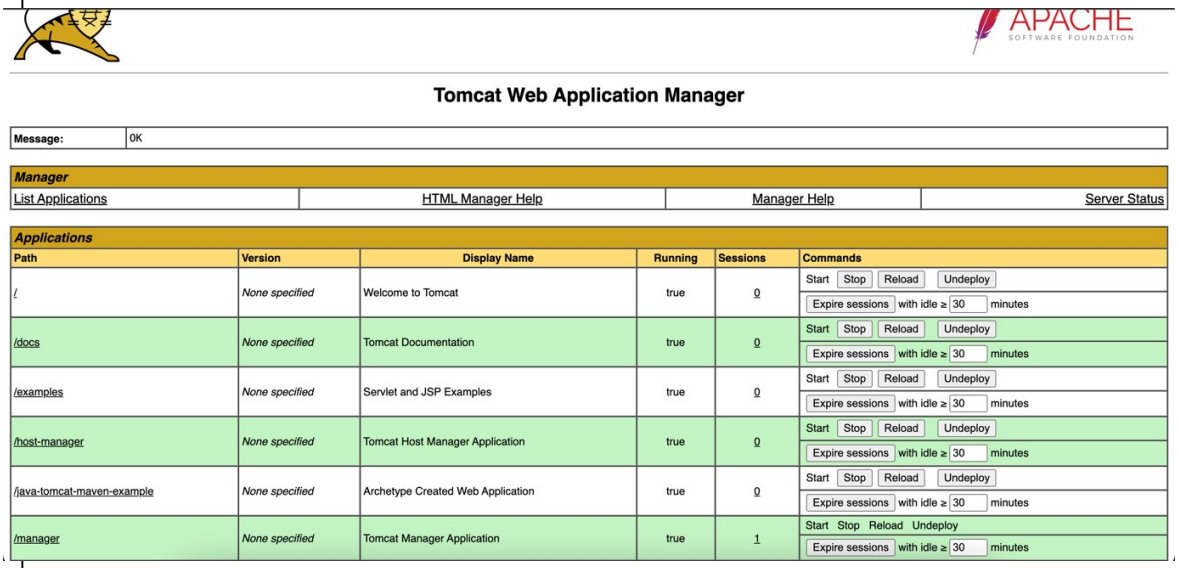



Pipeline overview




From tomcat dashboard, go to Manager App select the path to the WAR file



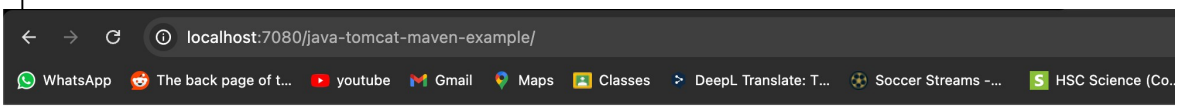
**Tomcat Web Application Manager**

| Message: | OK |
|---|---|

| Manager | | | |
|---|---|---|---|
| List Applications | HTML Manager Help | Manager Help | Server Status |

| Applications | | | | | |
|---|---|---|---|---|---|
| Path | Version | Display Name | Running | Sessions | Commands |
| / | None specified | Welcome to Tomcat | true | 0 | Start Stop Reload Undeploy<br>Expire sessions with idle ≥ 30 minutes |
| /docs | None specified | Tomcat Documentation | true | 0 | Start Stop Reload Undeploy<br>Expire sessions with idle ≥ 30 minutes |
| /examples | None specified | Servlet and JSP Examples | true | 0 | Start Stop Reload Undeploy<br>Expire sessions with idle ≥ 30 minutes |
| /host-manager | None specified | Tomcat Host Manager Application | true | 0 | Start Stop Reload Undeploy<br>Expire sessions with idle ≥ 30 minutes |
| /java-tomcat-maven-example | None specified | Archetype Created Web Application | true | 0 | Start Stop Reload Undeploy<br>Expire sessions with idle ≥ 30 minutes |
| /manager | None specified | Tomcat Manager Application | true | 1 | Start Stop Reload Undeploy<br>Expire sessions with idle ≥ 30 minutes |

Webapp deployed on tomcat server




**Hello World!**

**WelCome to Jenkins! We are working on Code Pipeline version 2**

This is change is to verify the Stage vs Production

>

**Conclusion**: Apache Tomcat was successfully installed and a web application using Jenkins pipeline was deployed.

# EXPERIMENT NO 06

**AIM**: -To understand and implement the Jenkins master-slave architecture and scale stand-alone implementa on by implemen ng slave nodes

Theory 1. Overview of Jenkins Master-Slave Architecture

- Jenkins Master: The master node serves as the central control unit for the Jenkins environment. It is responsible for managing the user interface, scheduling builds, and dispatching tasks to slave nodes. The master also maintains the configura on and history of all projects.
- Jenkins Slave: Slave nodes (also known as agents) are worker nodes that execute the build tasks as instructed by the master. They can be configured to run different environments and so ware versions, allowing for flexible and diverse build setups.

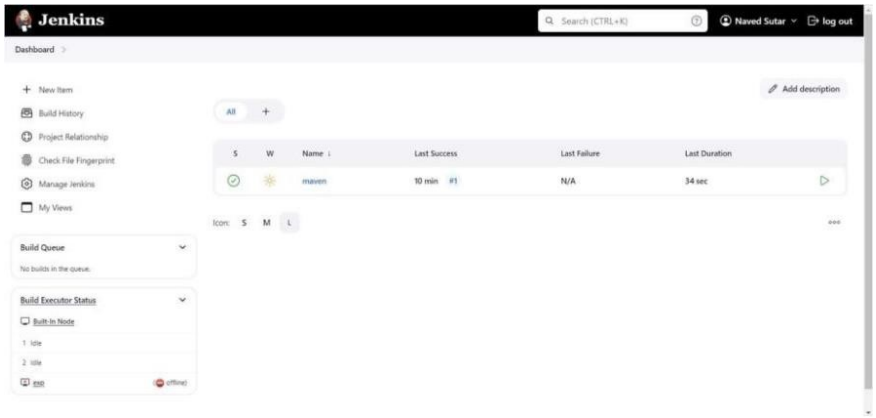
2. Benefits of Master-Slave Architecture

- Load Distribu on: By distribu ng the workload across mul ple nodes, Jenkins can handle mul ple build jobs simultaneously, reducing the overall me taken for builds.
- Resource U liza on: Different projects may require different configura ons or so ware stacks. Slave nodes can be tailored to specific needs, improving resource u liza on and efficiency.
- Fault Tolerance: If one slave node fails, the master can reroute the build jobs to other available nodes, providing redundancy and reliability in the CI/CD process.
- Scalability: New slave nodes can be easily added to the architecture, allowing organiza ons to scale their Jenkins setup in response to growing demands.

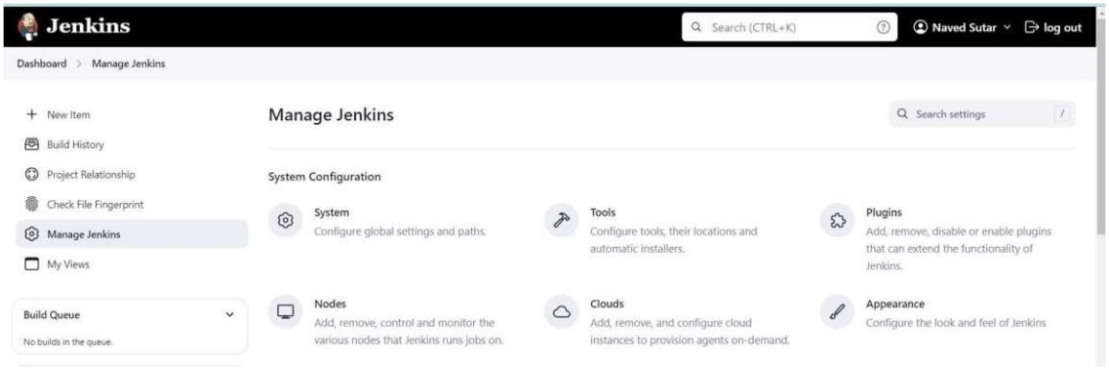
3. Implemen ng Slave Nodes

- Setup: Slave nodes can be set up on physical machines, virtual machines, or containers, allowing for flexibility in deployment.
- Connec on Methods: Jenkins supports various methods for connec ng slaves, including JNLP (Java Network Launch Protocol) and SSH. JNLP is commonly used for Windows-based slaves, while SSH is preferred for Unix/Linux systems.
- Configura on: Each slave node needs to be configured in the Jenkins master interface, where it can be assigned specific roles, labels, and resource limits to op mize its usage.

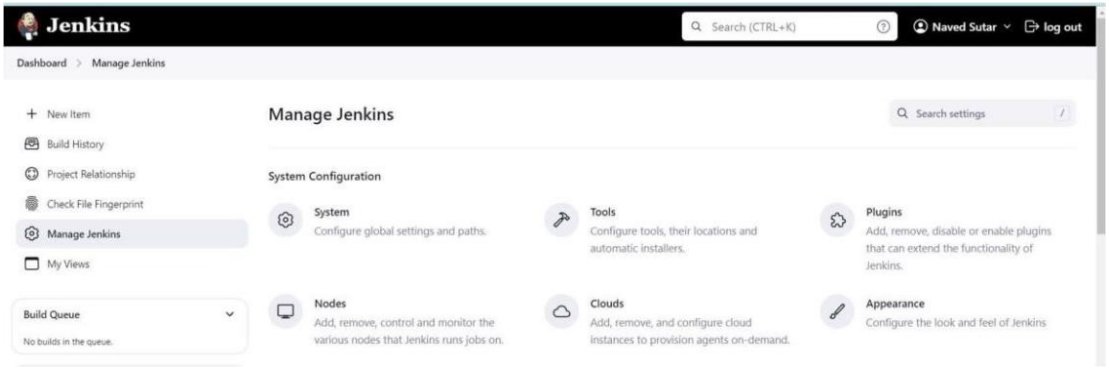
Go to the Jenkins dashboard



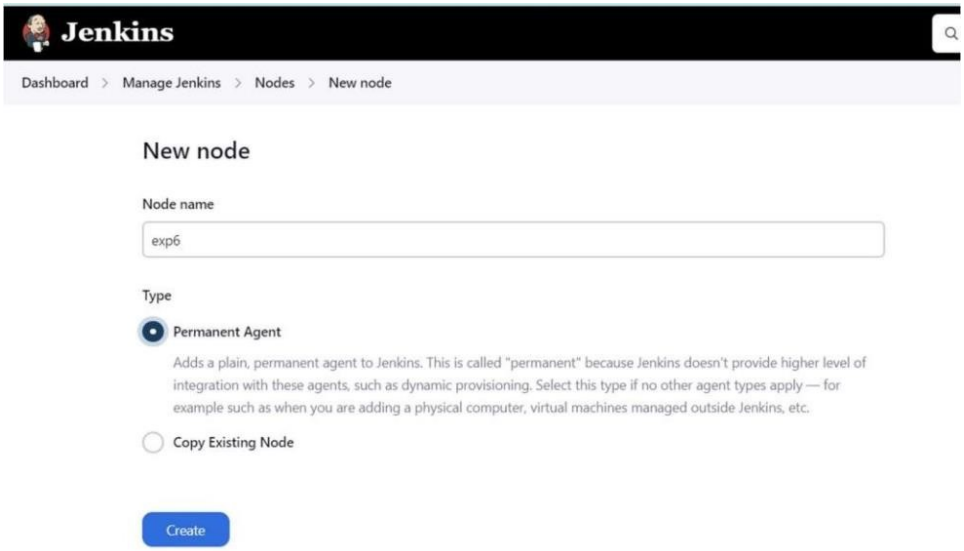

Go to manage Jenkins




Go to nodes sec on

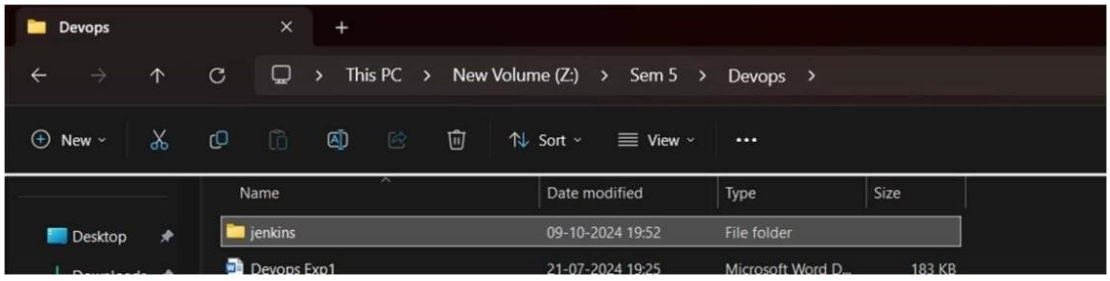



Create a new node





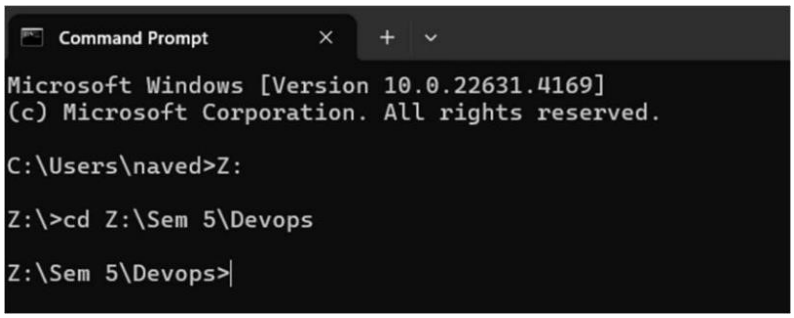
For root directory create a empty folder




Give the root of directory of the newly created folder and then give a label which we need it later, then save


As the node is created but not connected, for this go to newly created node


Now, open terminal and go to the root directory of the new folder you have created

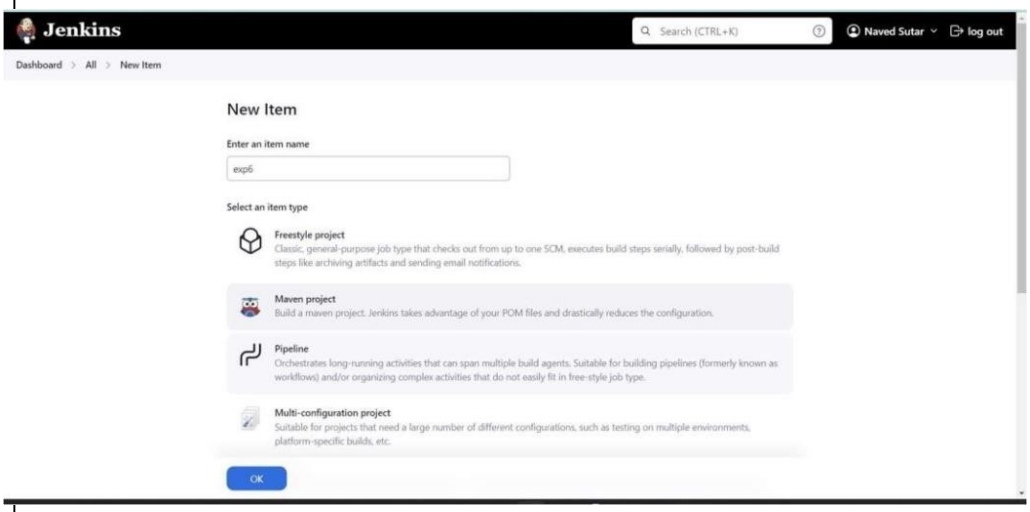



Now run these two commands one by one which is shown on your node agent

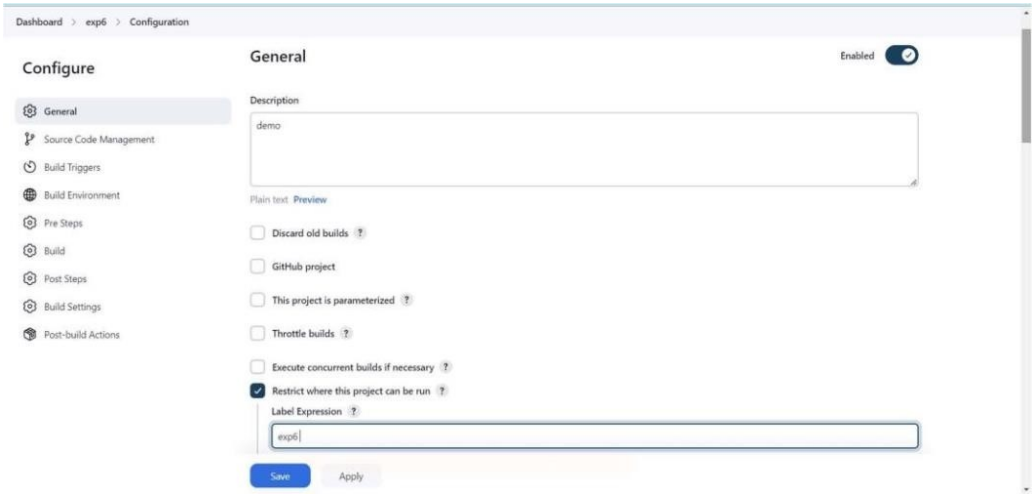
```
Z:\Sem 5\Devops>java -jar agent.jar -url http://localhost:8080/ -secret 4030ab5f1316cd5afe102983d7be7a8406d4a4f447a1d583
e2376fa415b70c31 -name exp -workDir "C:\SLAVENODE"
Oct 09, 2024 7:58:24 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using C:\SLAVENODE\remoting as a remoting work directory
Oct 09, 2024 7:58:24 PM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to C:\SLAVENODE\remoting
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher createEngine
INFO: Setting up agent: exp
Oct 09, 2024 7:58:24 PM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 3248.3250.v3277a_8e88c9b_
Oct 09, 2024 7:58:24 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using C:\SLAVENODE\remoting as a remoting work directory
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher$CuiListener status
INFO: Locating server among [http://localhost:8080/]
Oct 09, 2024 7:58:24 PM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, Ping]
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher$CuiListener status
INFO: Agent discovery successful
  Agent address: localhost
  Agent port:    55507
  Identity:      72:fd:ae:ae:53:8d:0f:08:91:69:b5:26:88:9e:89:69
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher$CuiListener status
INFO: Handshaking
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher$CuiListener status
INFO: Connecting to localhost:55507
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher$CuiListener status
INFO: Server reports protocol JNLP4-connect-proxy not supported, skipping
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher$CuiListener status
INFO: Trying protocol: JNLP4-connect
Oct 09, 2024 7:58:24 PM org.jenkinsci.remoting.protocol.impl.BIONetworkLayer$Reader run
INFO: Waiting for ProtocolStack to start.
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher$CuiListener status
INFO: Remote identity confirmed: 72:fd:ae:ae:53:8d:0f:08:91:69:b5:26:88:9e:89:69
Oct 09, 2024 7:58:24 PM hudson.remoting.Launcher$CuiListener status
INFO: Connected
```

Now the node is connected, to check the status go to node dashboard

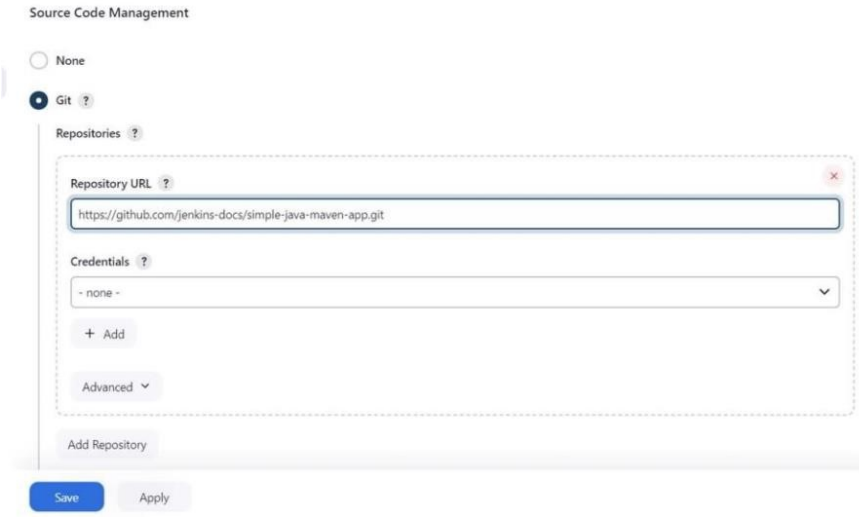

Now create a new build item to run the maven project



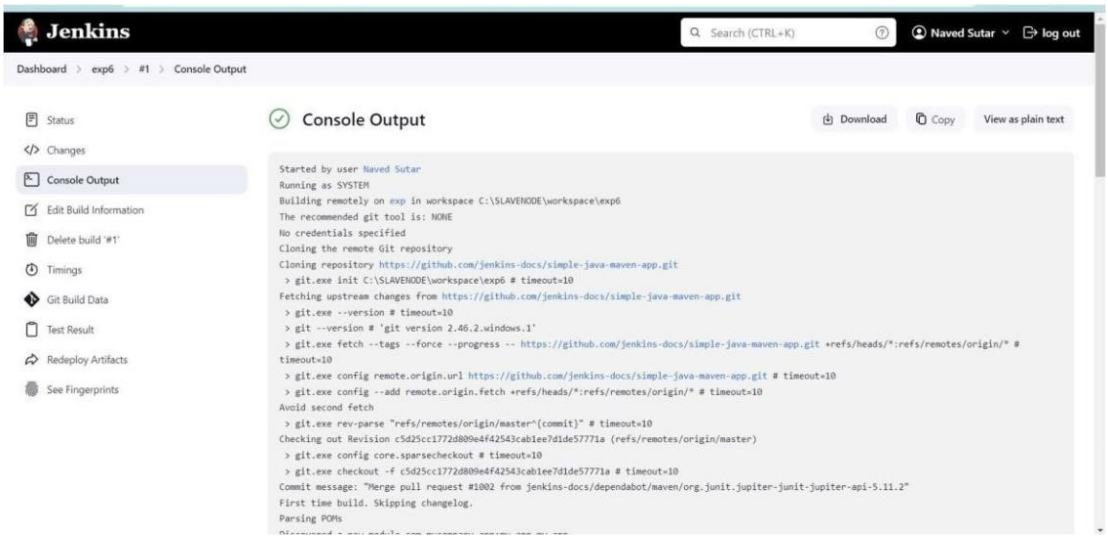

Select the restrict where the project can be run, and add the label of your node which was created




Add any sample maven project git project link, then click apply and then save




Now the build now the created item



**CONCLUSION:**

Understanding the Jenkins master-slave architecture is crucial for efficiently scaling a standalone Jenkins implementa on. By implemen ng slave nodes, organiza ons can achieve be er resource alloca on, enhanced build performance, and greater flexibility in their con nuous integra on and delivery pipelines. The master-slave setup not only improves overall produc vity by allowing concurrent execu on of builds but also ensures a robust and fault-tolerant CI/CD environment. Through careful configura on and management of both master and slave nodes, teams can effec vely harness the power of Jenkins to support their so ware development lifecycle.

# Experiment No 07

**Name: Chaudhary Mohd Arshad**
**Roll No: 221424 Batch: A**

**Aim:** Setup and run selenium test in Jenkins using Maven

**THEORY:**

What is Selenium?
Selenium is a powerful tool for automating web browsers, supporting major browsers such as Chrome, Firefox, Safari, and Internet Explorer. Selenium allows developers and testers to simulate user interactions like clicking buttons, filling forms, or navigating pages, making it an essential tool for automated testing of web applications.

Selenium Components:
Selenium WebDriver:

The WebDriver is the core component of Selenium, allowing you to automate browser actions at a more granular level by directly interacting with the browser.
Java is one of the many supported programming languages, making it easy to integrate with Java-based test frameworks.
Selenium IDE:

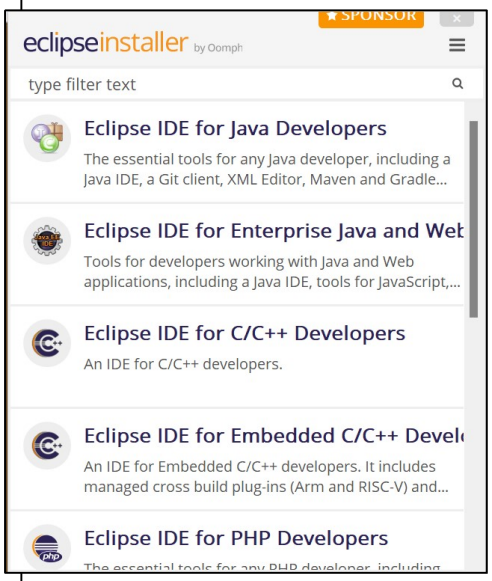
This is a browser extension used to record and playback interactions with web applications. However, it is limited in terms of customization compared to WebDriver.
Selenium Grid:

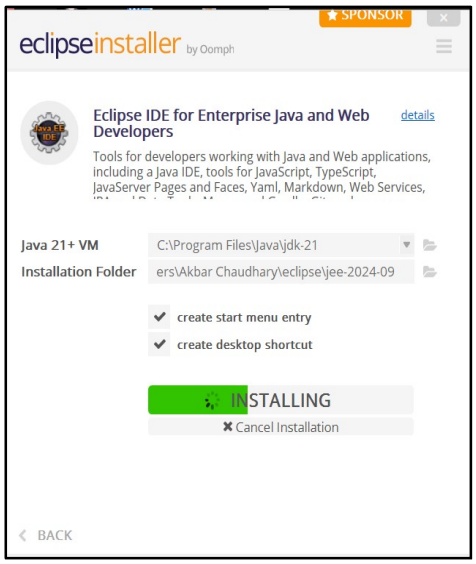
Selenium Grid allows you to run tests in parallel across different machines and browsers, helping to scale your tests.

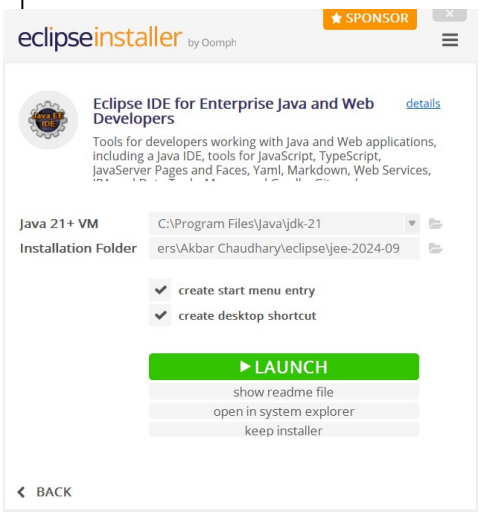
Advantages of Selenium with Java:
Cross-browser testing: Automate tests across different browsers.
Parallel execution: With TestNG and Selenium Grid, you can run tests in parallel.
Flexible: You can create highly customizable test scripts.
Integration with CI/CD: Selenium tests can be integrated into Jenkins pipelines for Continuous Testing.

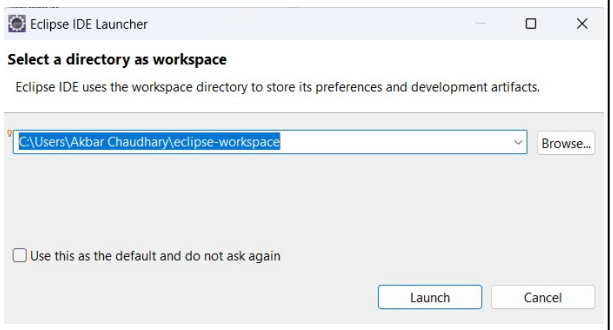
1. **Install Eclipse IDE**

- **Download Eclipse**:
  - Go to the Eclipse official website.
  - Download the "Eclipse IDE for Java Developers" package.
- **Install Eclipse**:
  - Once downloaded, run the installer and choose the default installation options.
- **Launch Eclipse**:
  - After installation, open Eclipse and set up a workspace (a directory where all your projects will be stored).



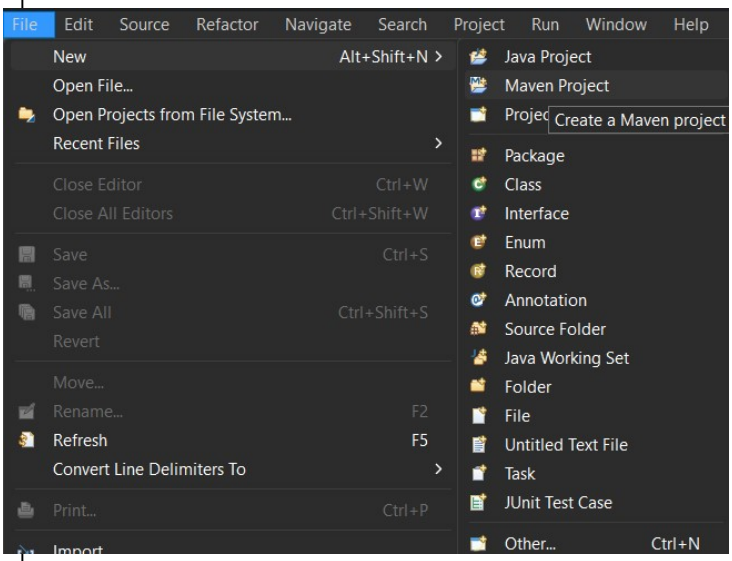



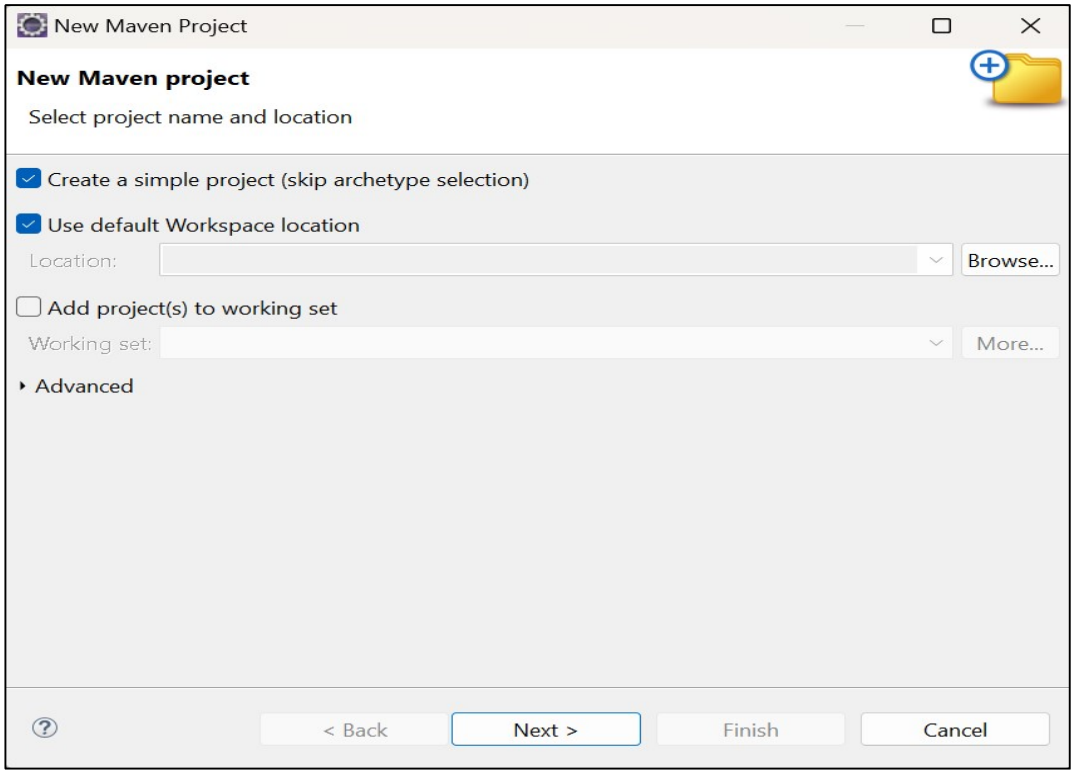



Install JDK if not installed

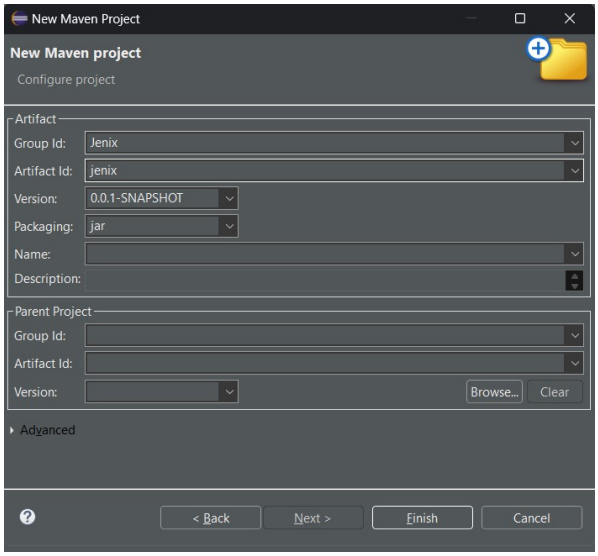
Go to File click on New and then click on Maven Project



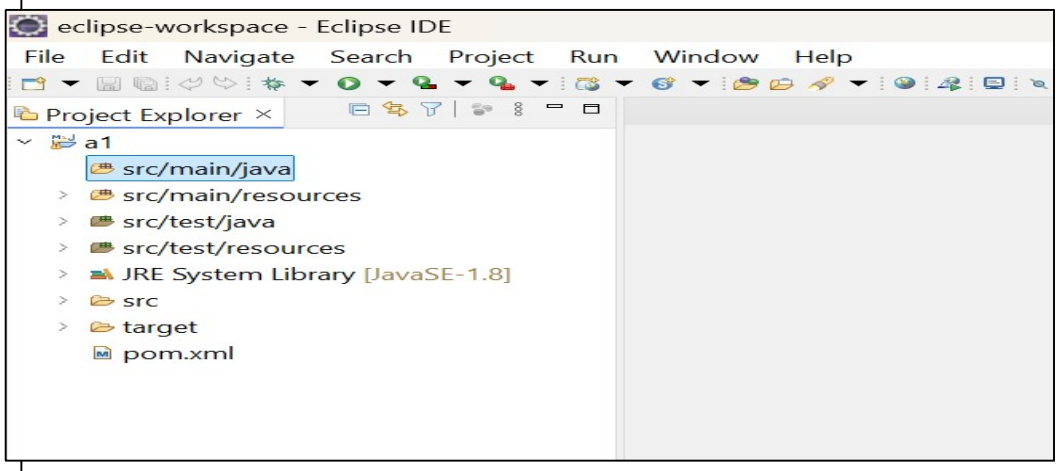

Click on Create a simple project (skip archetype selection)



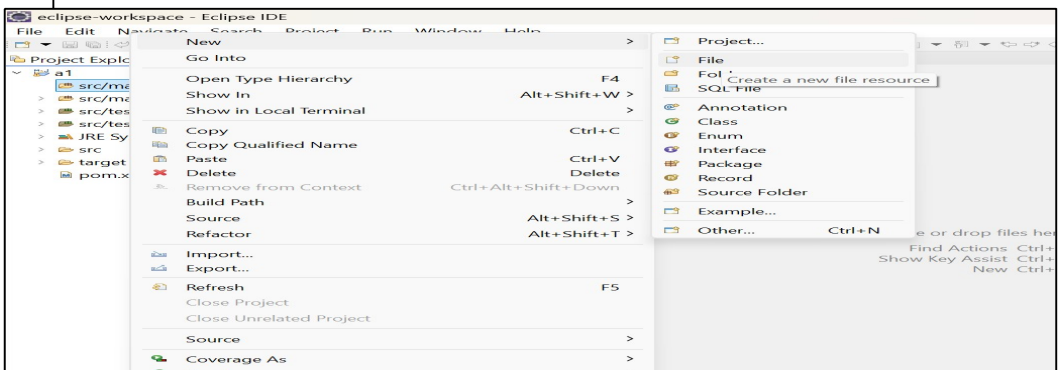
Give Group Id and Artifact Id and click on **Finish**



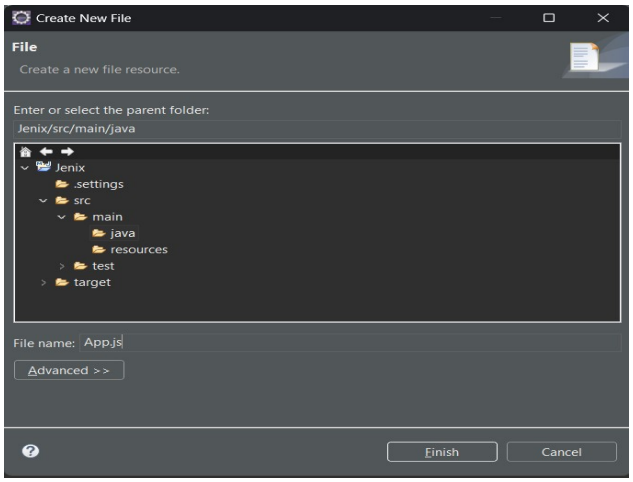

Go to the artifact and then go to src/main/java and right click to create and new file




Then click on New and click on File to create a new file

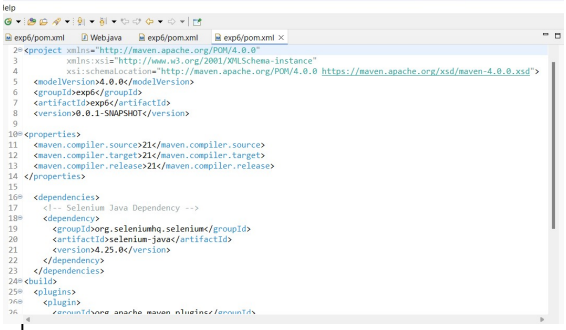


Name the Java File




Now, go to the browser search for Selenium Java Dependency and copy the link


Go back to eclipse, Go to pom.xml and paste the dependencies and save the file

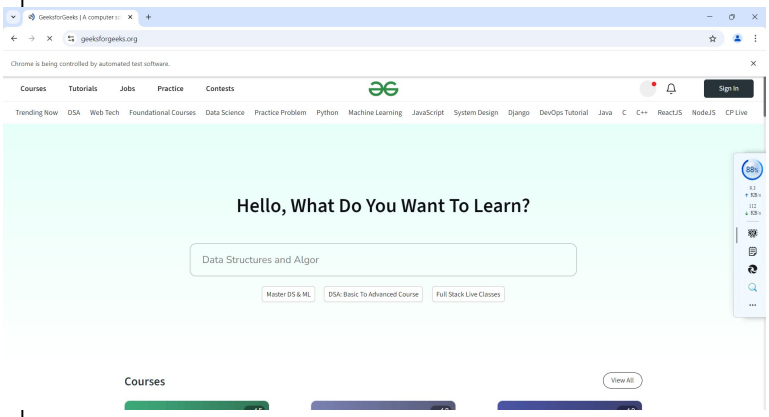



Write the test code in the Java file which you created (Web.java) and save the file

```java
import org.openqa.selenium.WebDriver;

public class Web {

    public static void main(String args[])
    {

        // Instantiate a ChromeDriver class.
        WebDriver driver = new ChromeDriver();

        // Maximize the browser
        driver.manage().window().maximize();

        // Launch Website
        driver.get("https://www.geeksforgeeks.org/");
    }
}
```

Click on the **Run Web** button to run the  selenium test code

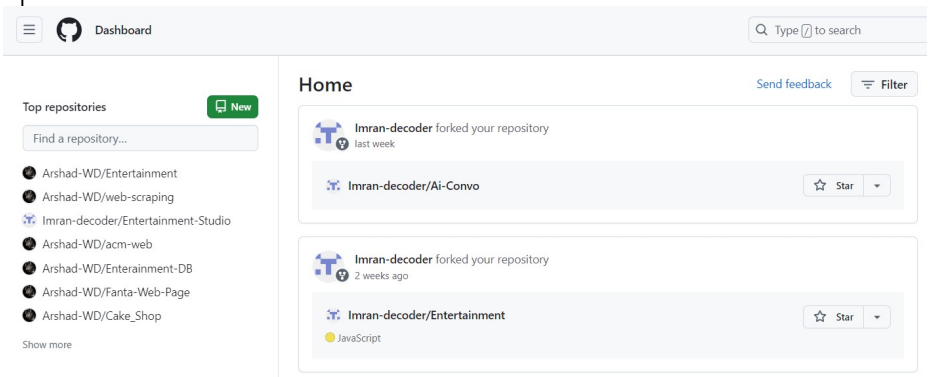
It automatically opens the web page



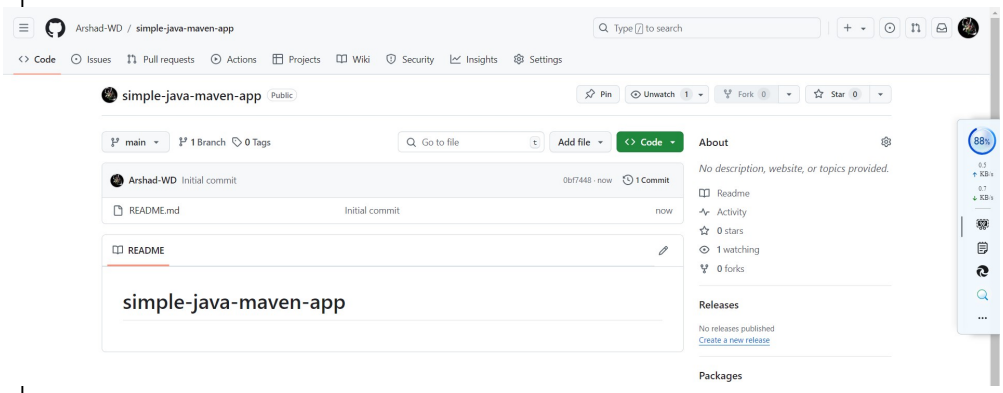

## 2. Create a GitHub Repository

### 1. Create a New Repository:

- On the GitHub dashboard, click the "New" button to create a new repository.
- Name the repository (e.g., SeleniumTestProject), choose whether you want it public or private, and click **Create repository**.




Repository has been created successfully




## 2. Initialize Git in Your Local Project

- Open the command prompt or terminal.
- Navigate to your local project folder that contains the Selenium test files.
- Initialize the project as a Git repository
- Add the GitHub repository you created as the remote origin

```
Arshad Chaudhary@ArshadChaudhary MINGW64 ~/eclipse-workspace/exp6
$ git init
Initialized empty Git repository in C:/Users/Akbar Chaudhary/eclipse-workspace/e
xp6/.git/

Arshad Chaudhary@ArshadChaudhary MINGW64 ~/eclipse-workspace/exp6 (master)
$ git branch -m main

Arshad Chaudhary@ArshadChaudhary MINGW64 ~/eclipse-workspace/exp6 (main)
$ git remote add origin https://github.com/Arshad-WD/simple-java-maven-app.git

Arshad Chaudhary@ArshadChaudhary MINGW64 ~/eclipse-workspace/exp6 (main)
$
```

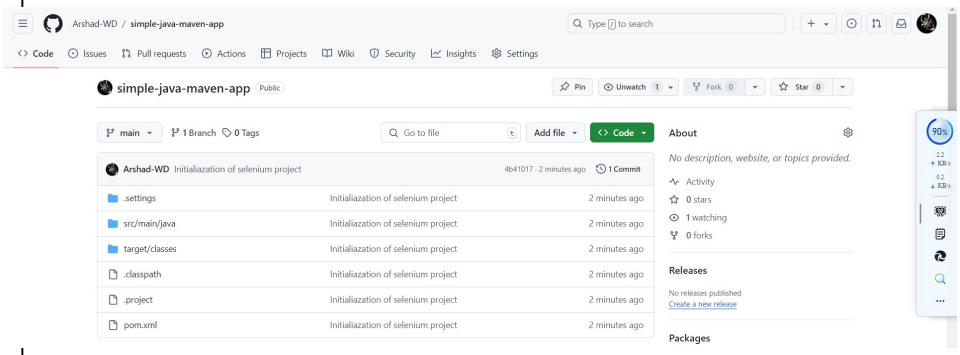
## 3. Push the Project to GitHub

- Add all files to the Git repository
- Commit the files
- Push the project to GitHub

```
Arshad Chaudhary@ArshadChaudhary MINGW64 ~/eclipse-workspace/exp6 (main)
$ git add .

Arshad Chaudhary@ArshadChaudhary MINGW64 ~/eclipse-workspace/exp6 (main)
$ git commit -m "Initialiazation of selenium project"
[main (root-commit) 4b41017] Initialiazation of selenium project
 10 files changed, 178 insertions(+)
 create mode 100644 .classpath
 create mode 100644 .project
 create mode 100644 .settings/org.eclipse.jdt.core.prefs
 create mode 100644 .settings/org.eclipse.m2e.core.prefs
 create mode 100644 pom.xml
 create mode 100644 src/main/java/Web.java
 create mode 100644 target/classes/META-INF/MANIFEST.MF
 create mode 100644 target/classes/META-INF/maven/exp6/exp6/pom.properties
 create mode 100644 target/classes/META-INF/maven/exp6/exp6/pom.xml
 create mode 100644 target/classes/Web.class
```

```
Arshad Chaudhary@ArshadChaudhary MINGW64 ~/eclipse-workspace/exp6 (main)
$ git push -u origin main --force
Enumerating objects: 21, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 8 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (21/21), 3.16 KiB | 404.00 KiB/s, done.
Total 21 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Arshad-WD/simple-java-maven-app.git
 + 0bf7448...4b41017 main -> main (forced update)
branch 'main' set up to track 'origin/main'.
```

Now the code has been pushed to the Git Hub




## 3. Set Up Jenkins to Run the Selenium Test from GitHub

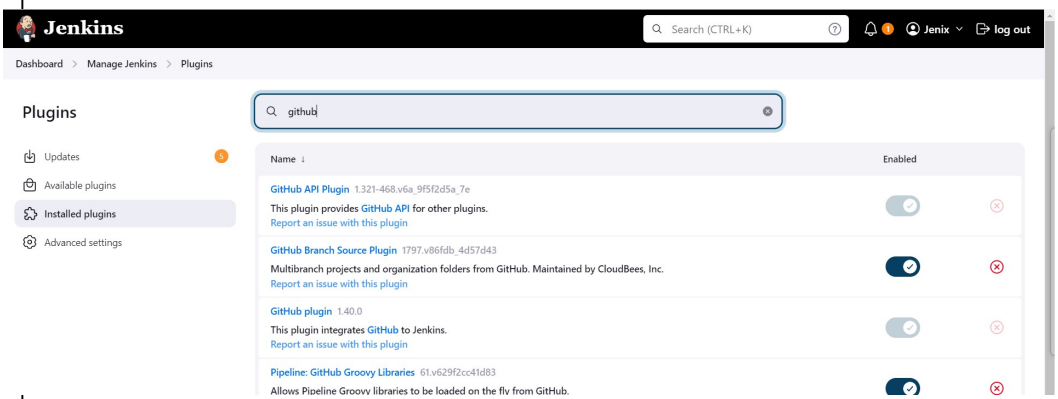
### 1. Install Jenkins Plugins

**Install Git Plugin:**

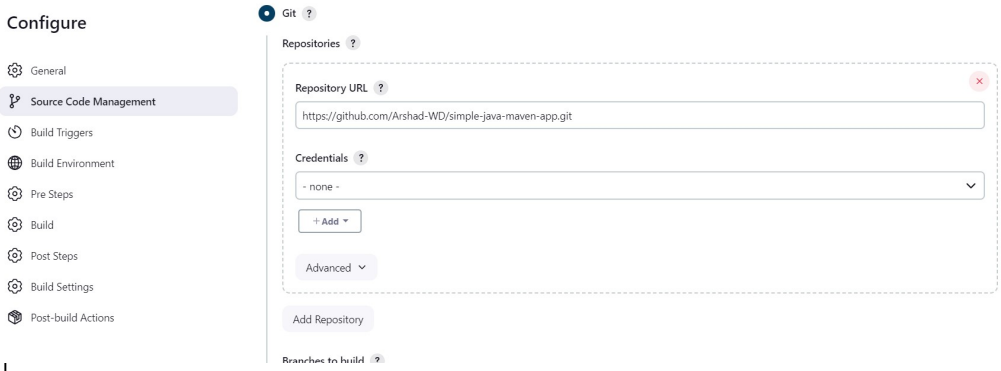
- Go to **Manage Jenkins > Manage Plugins** and install the **Git Plugin** if it is not already installed.
- **Install Maven Plugin** (if not installed):
- **Install the Maven Integration Plugin** to run Maven commands from Jenkins.

Now, go to **Manage Jenkins** and go to **Tools**

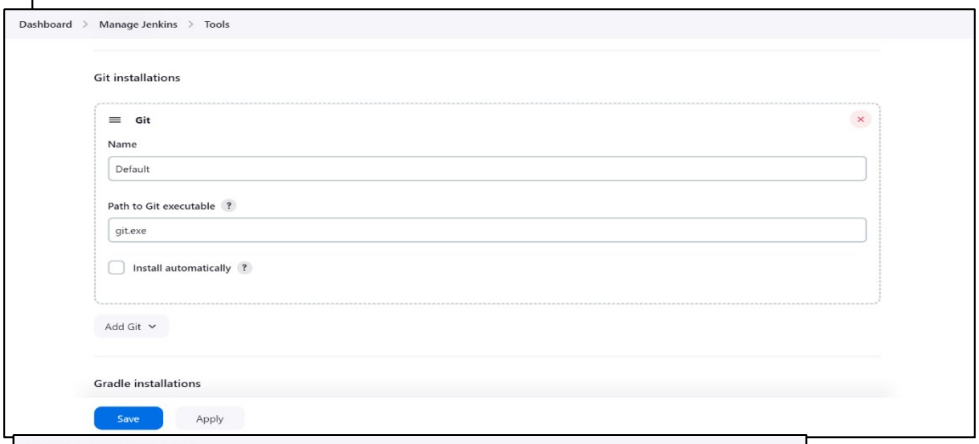
Keep the default settings for the first two options then click on **Add JDK**



Give the Name and add the path




Keep the rest of the things default and click on **Apply** and **Save**

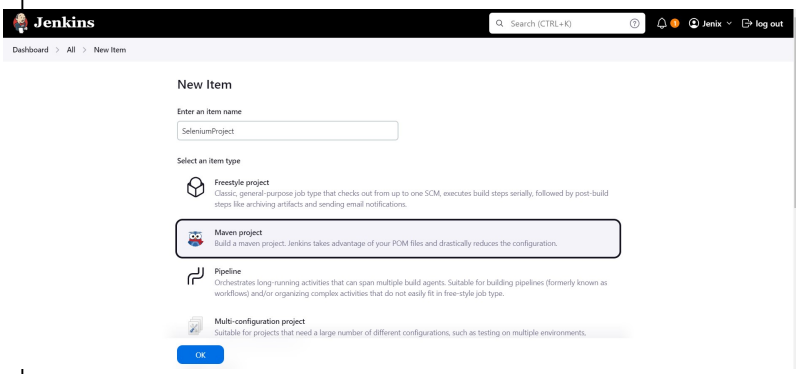




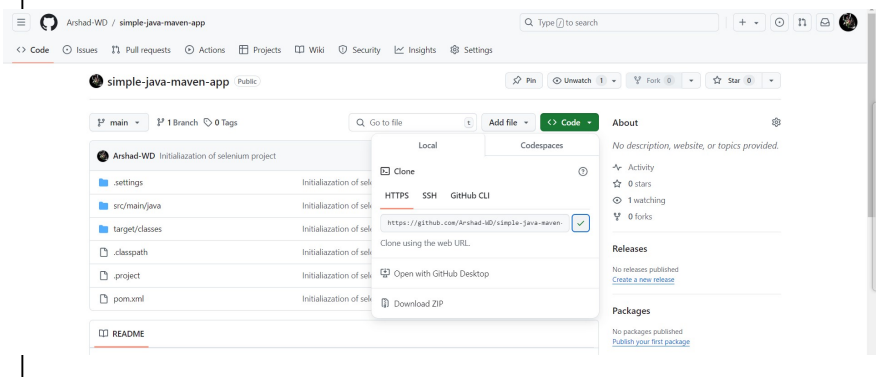
Click on **Apply** and **Save**

Now, Create a new  job:

- From the Jenkins dashboard, click New Item.
- Enter a name for your job (e.g., SeleniumProject).
- Select Maven Project and click OK.

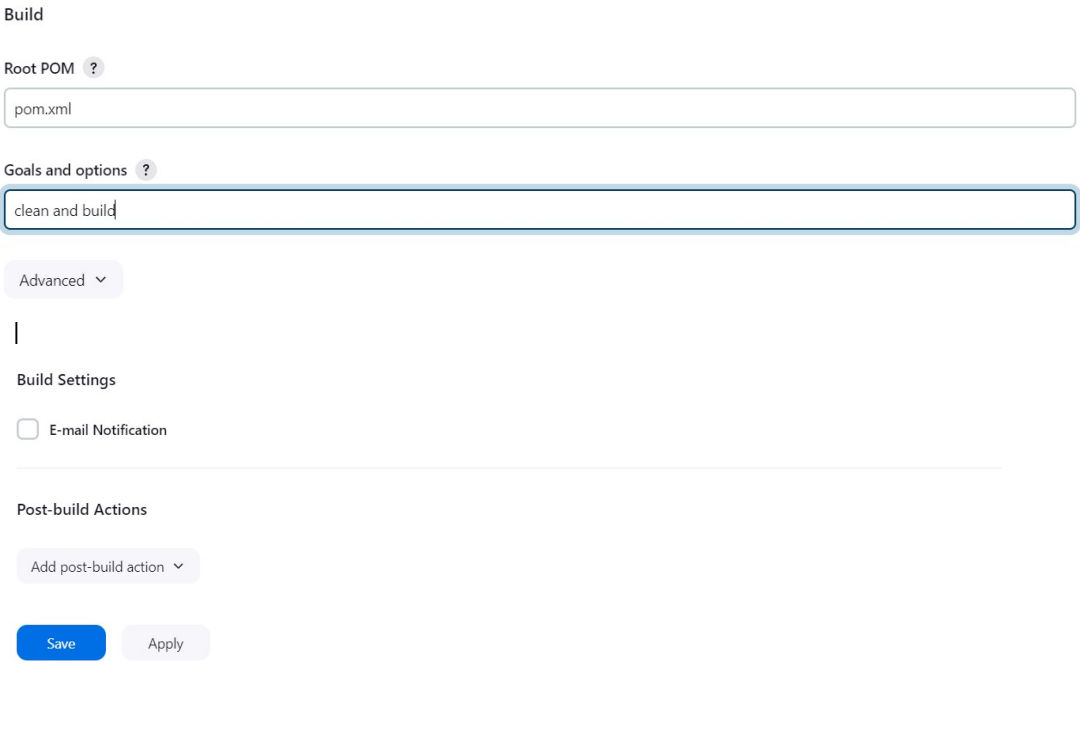



Give description (e.g., This is my first built job). Then go to source code Management and select Git. Now, go to git copy the Repository URl and paste it on Repository URL

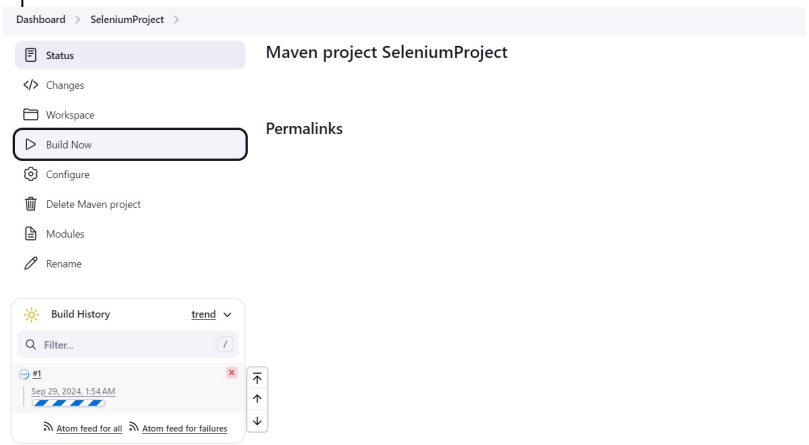




Change the branch main and keep rest of the things as default and click on **Apply** and **Save**

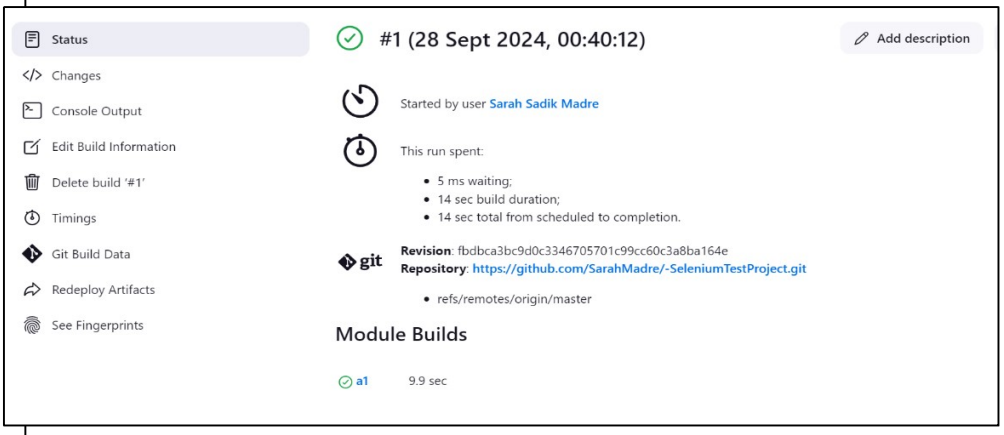



After this click on **Build Now**

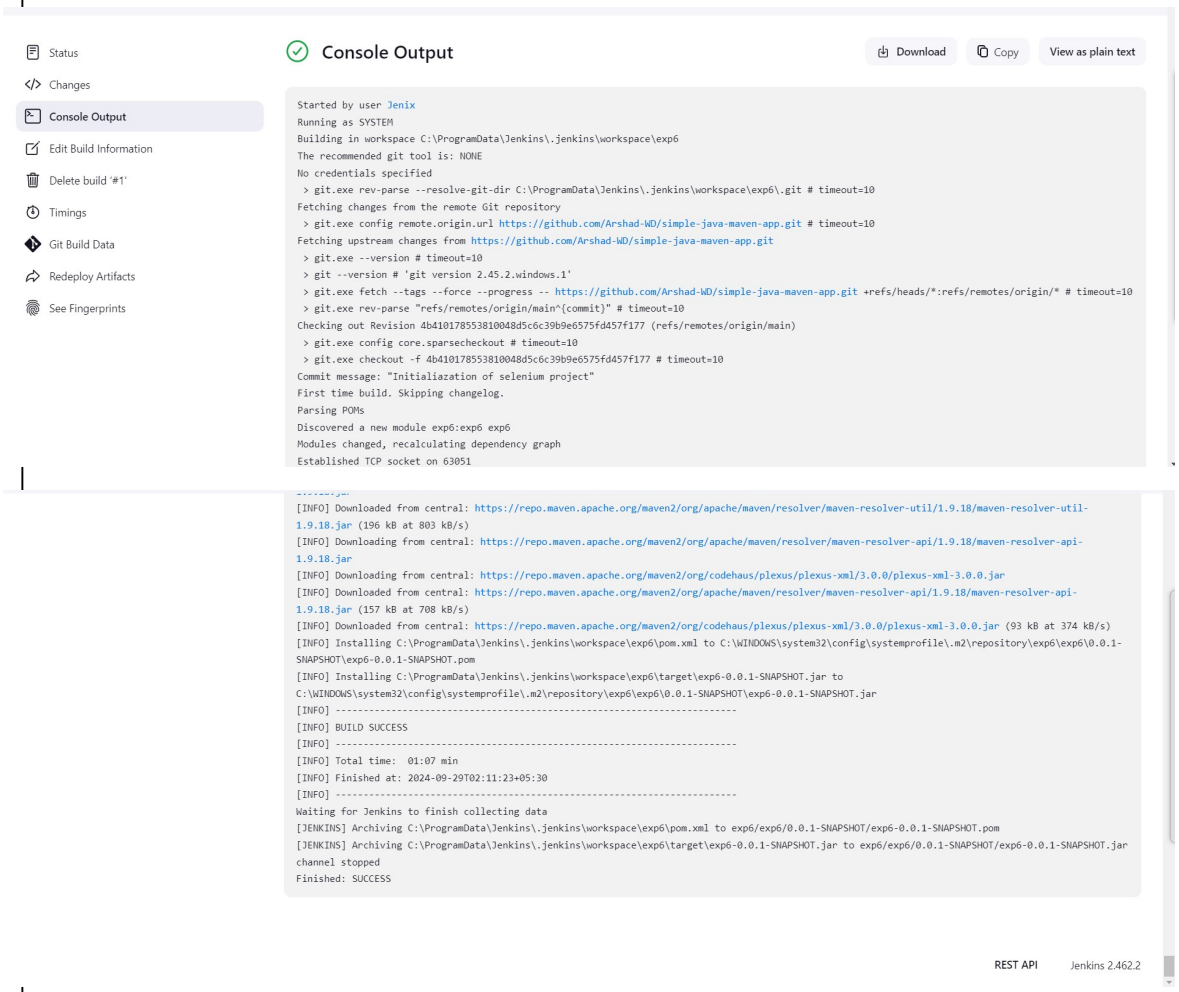



Now the job has been created successfully

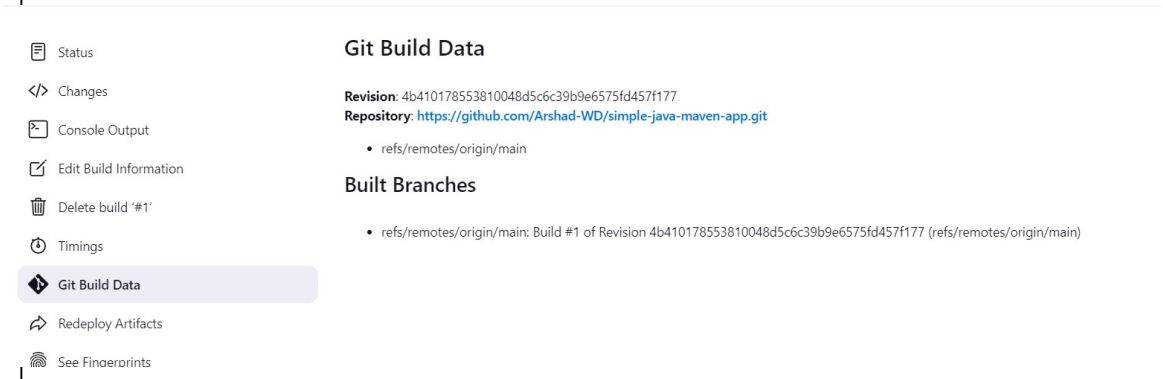
Status:



Console Output:




Git Build Data:




**Conclusion:** In this Experiment, we set up an automated testing framework with Selenium, Maven, Eclipse, and Jenkins. We wrote a Selenium test, pushed it to GitHub, and configured Jenkins for continuous integration. By using webhooks, Jenkins automatically runs tests with each code push, streamlining our development workflow and enhancing reliability.

**Experiment no 08**

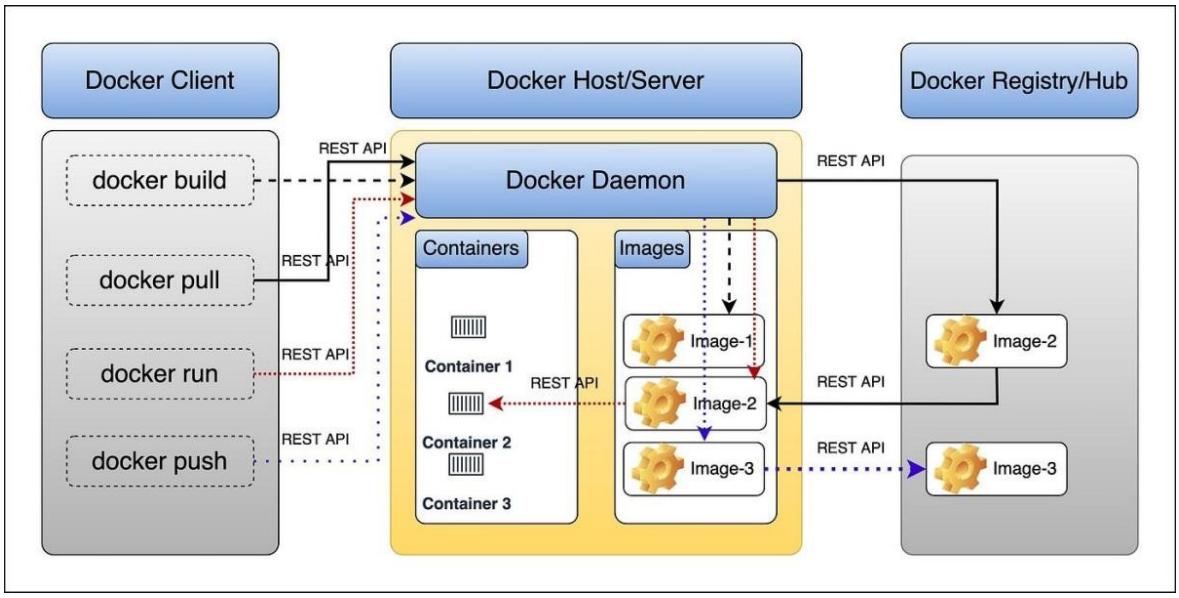
**Roll no: 221436**

**Aim:** Understand Docker Architecture and Container Life Cycle, install Docker und execute Docker commands to manage images and interact with containers.

**Docker Architecture**

Docker's architecture consists of several key components:

1. **Docker Daemon (dockerd)**: The server-side component that runs on the host machine. It manages Docker containers, images, networks, and volumes.
2. **Docker CLI (Command-Line Interface)**: The client-side tool that allows users to interact with the Docker daemon. Commands are executed in the terminal or command prompt.
3. **Docker Images**: Read-only templates used to create containers. They contain the application code, libraries, and dependencies.
4. **Docker Containers**: Instances of Docker images that can be executed. They are lightweight and portable.
5. **Docker Registry**: A repository for storing Docker images, such as Docker Hub, where users can share and distribute images. **Diagram of Docker Architecture**




Docker container lifecycle:

**Create**: Container is created but not started.

**Start**: Container starts running.

**Running**: Container is executing tasks.

**Stop**: Gracefully stops the container.

**Paused**: Freezes container processes (can be resumed).

**Restart**: Stops and restarts the container.

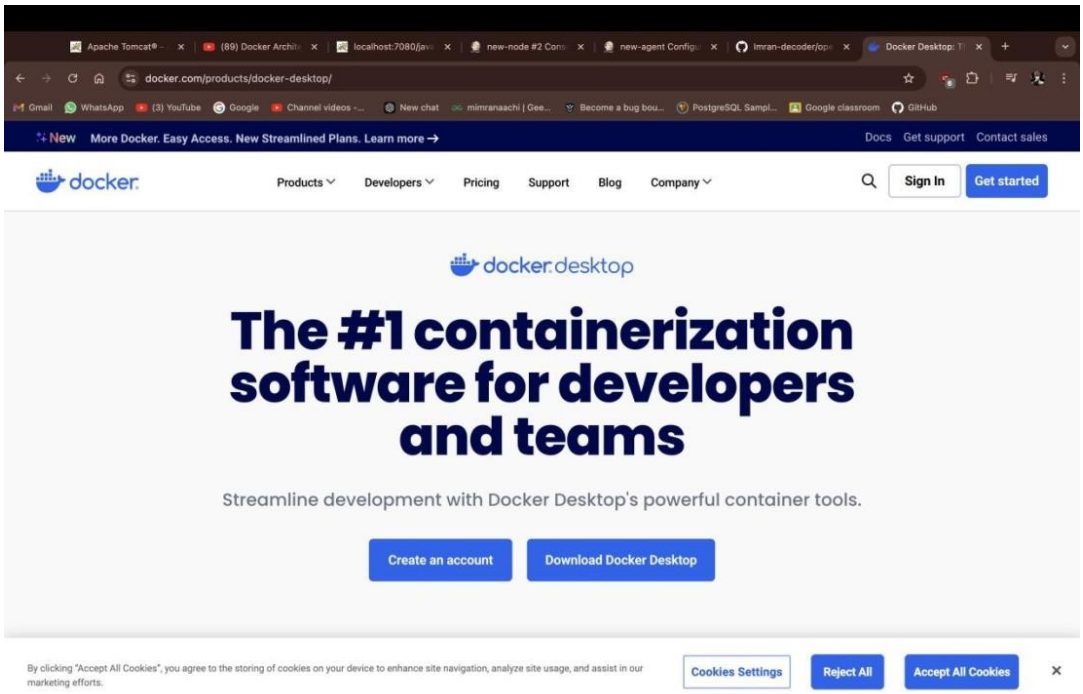
**Stopped**: Container is stopped and inactive.

**Kill**: Forcefully stops the container.

**Remove**: Deletes the container.

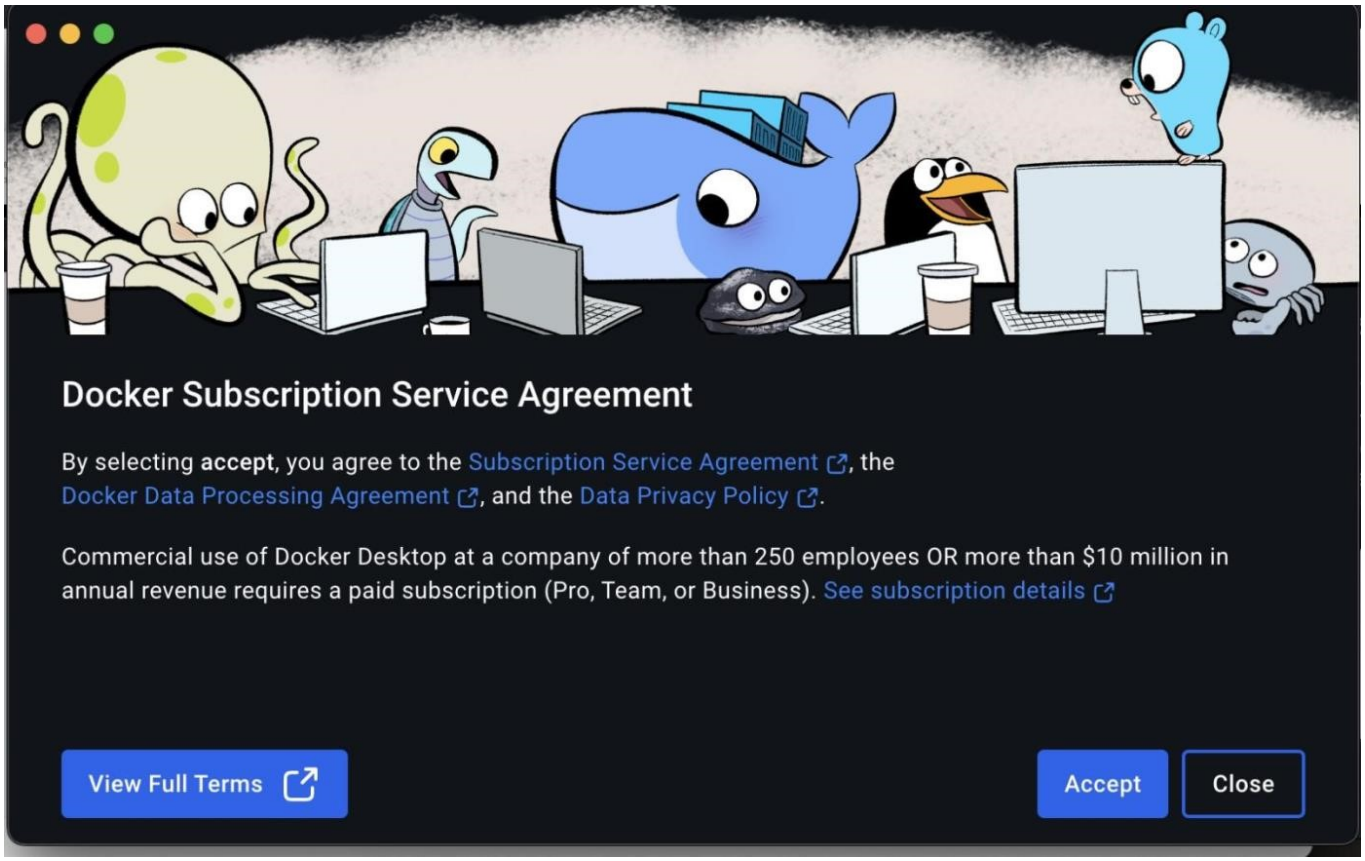
**Dead**: Container failed and cannot restart.

Visit docker.com and download Docker Desktop



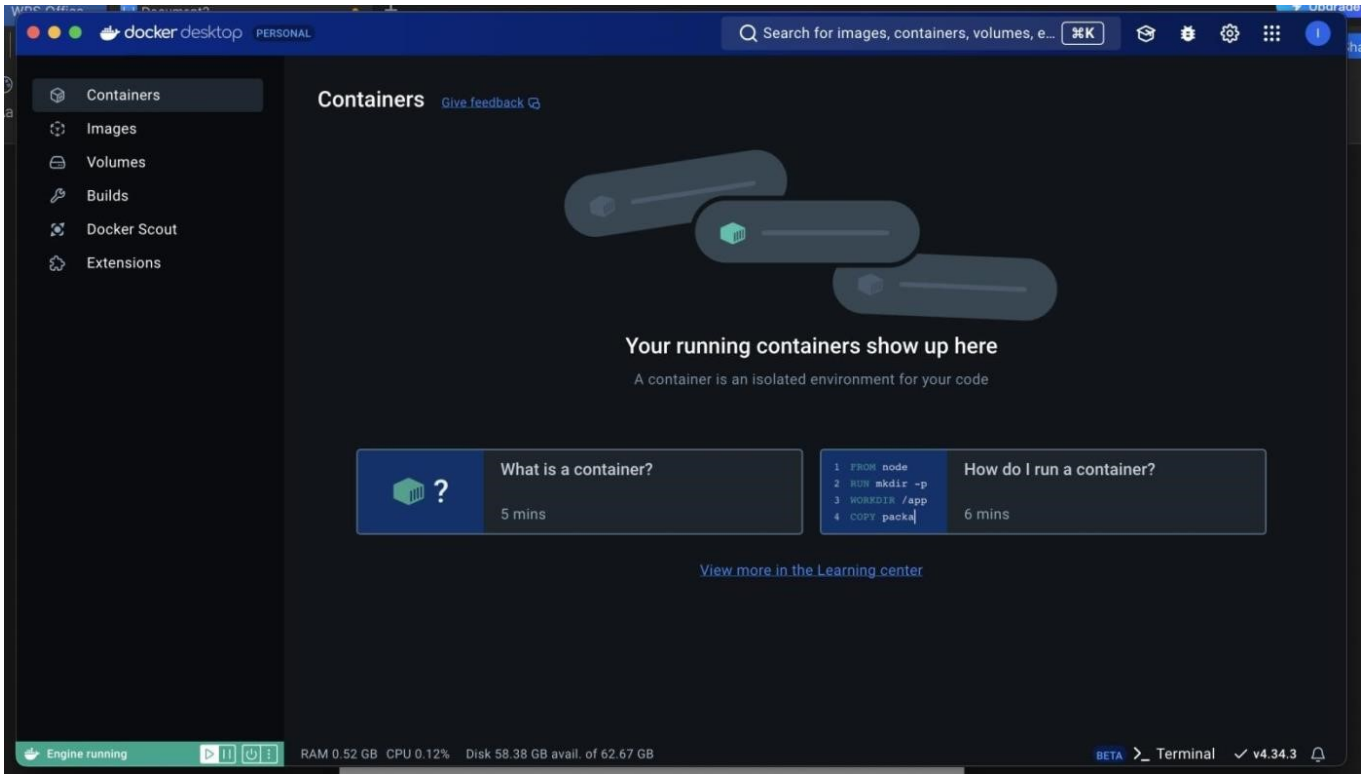


Complete the installation the steps

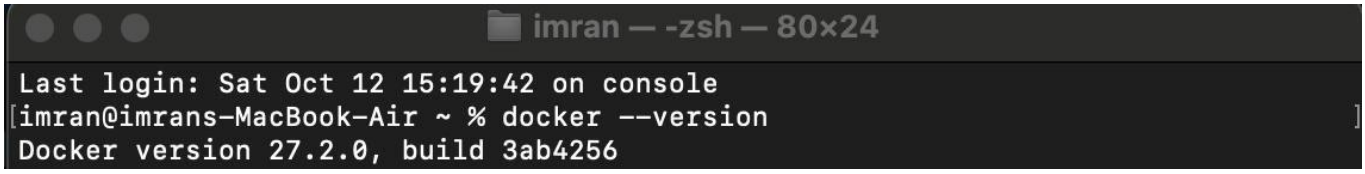




Docker Desktop dashboard

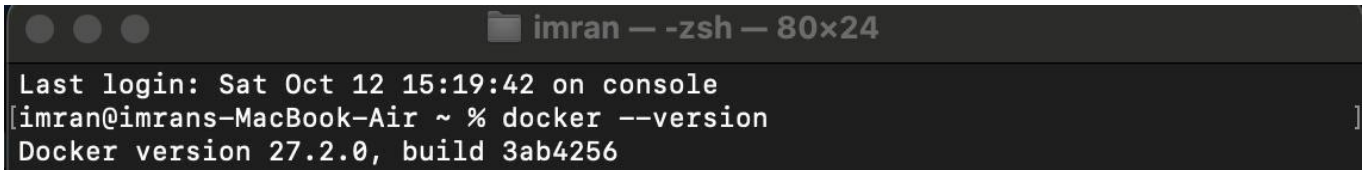



Verify Docker installation




**Docker commands:**

- **docker --version**
- Displays the current version of Docker installed.




- **docker pull <image_name>**
- Pulls a Docker image from Docker Hub (e.g., docker pull ubuntu).

```
imran@imrans-MacBook-Air ~ % docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
478afc919002: Download complete
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout q
uickview hello-world
```

- **docker images**
- Lists all Docker images stored locally on your system.

```
imran@imrans-MacBook-Air ~ % docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    d211f485f2dd   17 months ago   21kB
```

- **docker run <image_name>**
- Runs a Docker container from the specified image.

```
imran@imrans-MacBook-Air ~ % docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

- **docker run -it <image_name>**
- Runs a container interactively, allowing you to interact with the container (e.g., docker run -it ubuntu).

```
imran@imrans-MacBook-Air ~ % docker run -it python
Unable to find image 'python:latest' locally
latest: Pulling from library/python
56cf5ce41fb3: Download complete
07620df08508: Download complete
6d11c181ebb3: Download complete
d71ace0e8bbd: Download complete
2b238499ec52: Download complete
41b754d079e8: Download complete
a66dd39158c0: Download complete
Digest: sha256:45803c375b95ea33f482e53a461eca8f247617667d703660a06ccf5eb3d05326
Status: Downloaded newer image for python:latest
Python 3.13.0 (main, Oct  8 2024, 00:05:26) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("print this is imran")
print this is imran
>>> 2**8
256
```

- **docker ps**
- Lists currently running containers.

```
imran — -zsh — 80×24
Last login: Sat Oct 12 15:22:22 on ttys000
imran@imrans-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS
NAMES
160a7f90d86a   python    "python3"   11 seconds ago   Up 11 seconds
gallant_roentgen
```

- **docker ps -a**
- Lists all containers, including stopped ones.

```
imran@imrans-MacBook-Air ~ % docker ps -a
CONTAINER ID   IMAGE         COMMAND     CREATED              STATUS
          PORTS     NAMES
160a7f90d86a   python        "python3"   35 seconds ago       Up 34 seconds
                 gallant_roentgen
8a247cd1b316   python        "python3"   About a minute ago   Exited (0) 52 seco
nds ago             crazy_hamilton
e346895cee8d   hello-world   "/hello"    3 minutes ago        Exited (0) 3 minut
es ago              boring_hamilton
```

- **docker stop <container_id>** • Stops a running container.

```
imran@imrans-MacBook-Air ~ % docker stop 160
160
```

- **docker start <container_id>** • Starts a stopped container.

```
imran@imrans-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
imran@imrans-MacBook-Air ~ % docker start 160
160
```

- **docker restart <container_id>** • Restarts a running or stopped container.

```
imran@imrans-MacBook-Air ~ % docker restart 160
160
imran@imrans-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS          PORTS     N
AMES
160a7f90d86a   python    "python3"   2 minutes ago   Up 16 seconds             g
allant_roentgen
```

- **docker rm <container_id>**
- Removes a stopped container.

```
imran@imrans-MacBook-Air ~ % docker rm hello-world
Error response from daemon: No such container: hello-world
imran@imrans-MacBook-Air ~ % docker rm e34
e34
imran@imrans-MacBook-Air ~ % docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS
PORTS     NAMES
160a7f90d86a   python    "python3"   3 minutes ago   Up 56 seconds
          gallant_roentgen
8a247cd1b316   python    "python3"   4 minutes ago   Exited (0) 3 minutes ago
          crazy_hamilton
```

- **docker rmi <image_name>**
- Removes a Docker image from your local system.

```
imran@imrans-MacBook-Air ~ % docker rmi hello-world
Untagged: hello-world:latest
Deleted: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
```

- **docker logs <container_id>**
- Displays the logs of a running or stopped container.

```
imran@imrans-MacBook-Air ~ % docker logs 160
Python 3.13.0 (main, Oct  8 2024, 00:05:26) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> Python 3.13.0 (main, Oct  8 2024, 00:05:26) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> Python 3.13.0 (main, Oct  8 2024, 00:05:26) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

- **docker exec <container_id> <command>**
- Runs a command inside a running container (e.g., docker exec <container_id> ls).

```
imran@imrans-MacBook-Air ~ % docker exec 160 ls
bin
boot
dev
etc
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
```

- **docker search <image_name>** • Searches docker hub for an image

```
imran@imrans-MacBook-Air ~ % docker search hello-world
NAME                                DESCRIPTION
    STARS     OFFICIAL
hello-world                         Hello World! (an example of minimal Dockeriz
…   2330      [OK]
rancher/hello-world                 This container image is no longer maintained
…   6
okteto/hello-world
    0
atlassian/hello-world
    0
tutum/hello-world                   Image to test docker deployments. Has Apache
…   90
dockercloud/hello-world             Hello World!
    20
crccheck/hello-world                Hello World web server in under 2.5 MB
    24
koudaiii/hello-world
    0
ppc64le/hello-world                 Hello World! (an example of minimal Dockeriz
…   2
tsepotesting123/hello-world
```

**Conclusion:** Docker Desktop was installed and various Docker commands were successfully executed

**DOP Lab Experiment no 09**

**Roll no: 221436**

**Aim:** Lear Docker file instructions; build an image for a sample web application using Docker file.

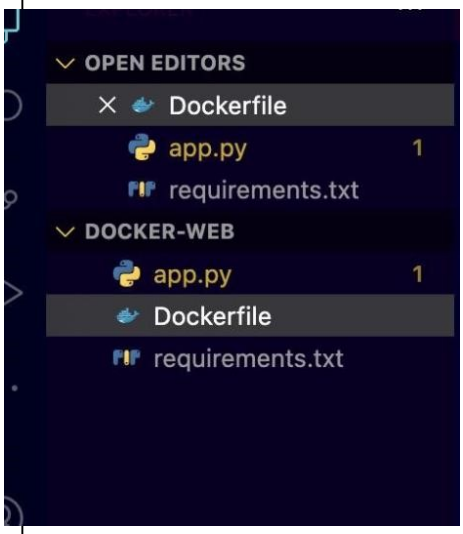
A Docker file is a script that contains a series of commands to define how to build a Docker image. It provides instructions to Docker on how to set up and configure the environment within the image that will later be used to run a container. The commands in a Docker file are executed step-by-step to build an image layer by layer.

Common Docker file Commands:

- FROM: Specifies the base image, such as ubuntu or python. This is the starting point for building your image.
- WORKDIR: Sets the working directory inside the container.
- COPY: Copies files or directories from the host (your machine) to the container.
- RUN: Executes a command in the shell, such as installing dependencies.
- EXPOSE: Documents the port on which the container will listen for incoming connections.
- CMD: Defines the default command to run when the container starts.

Implementation:

File Structure:



Dockerfile app.py 1 requirements.t

app.py > ...

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Demo Flask & Docker application is

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5010)
```

Dockerfile app.py 1 requirement

Dockerfile

```
FROM python
WORKDIR /docker-web
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ['python','app.py']
```

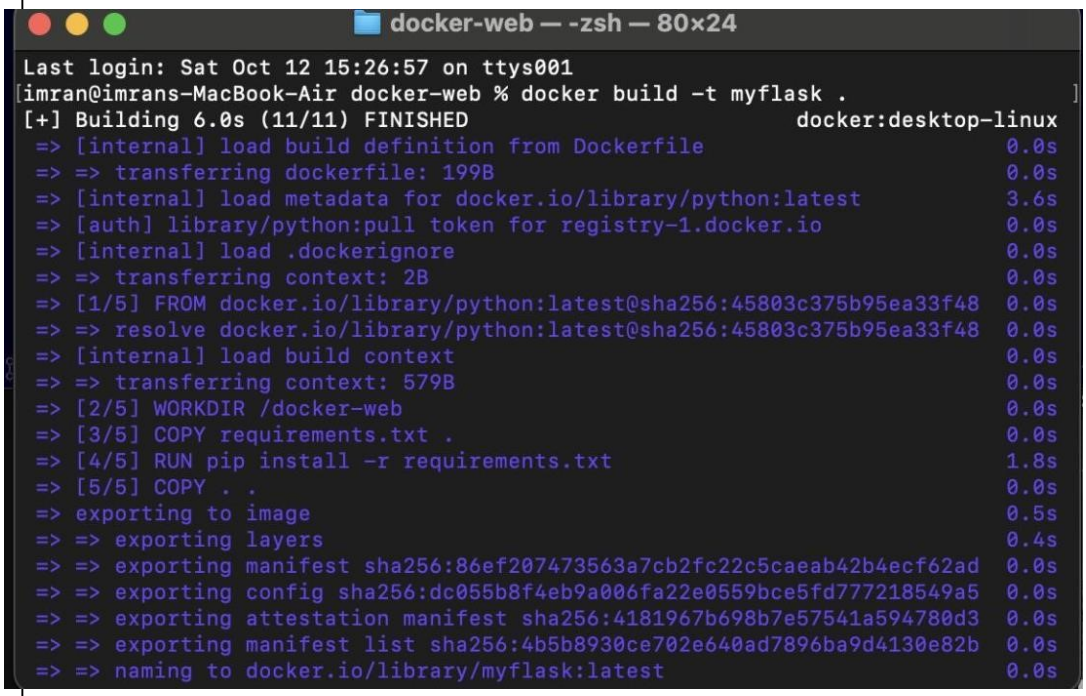
docker-web

Dockerfile app.py 1 requirements.txt

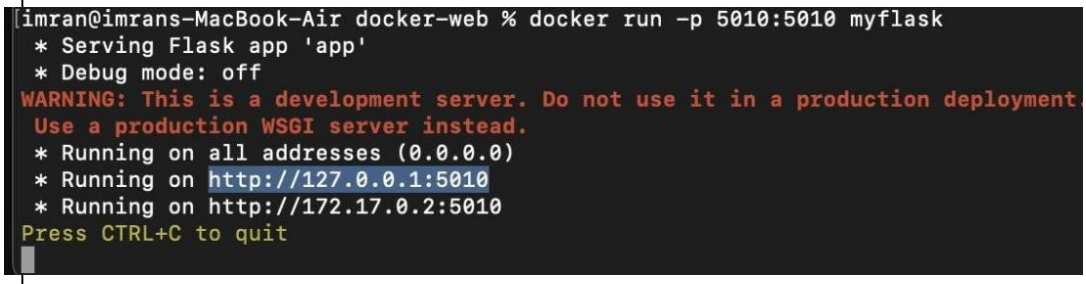
requirements.txt

```
Flask
```

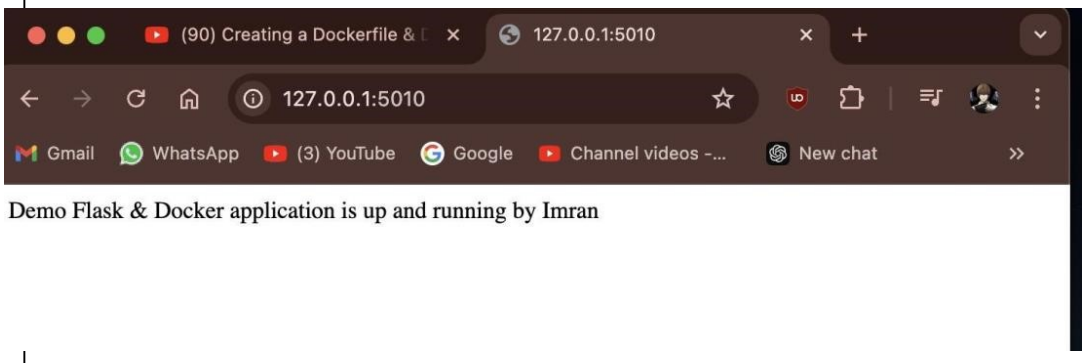
Build the application



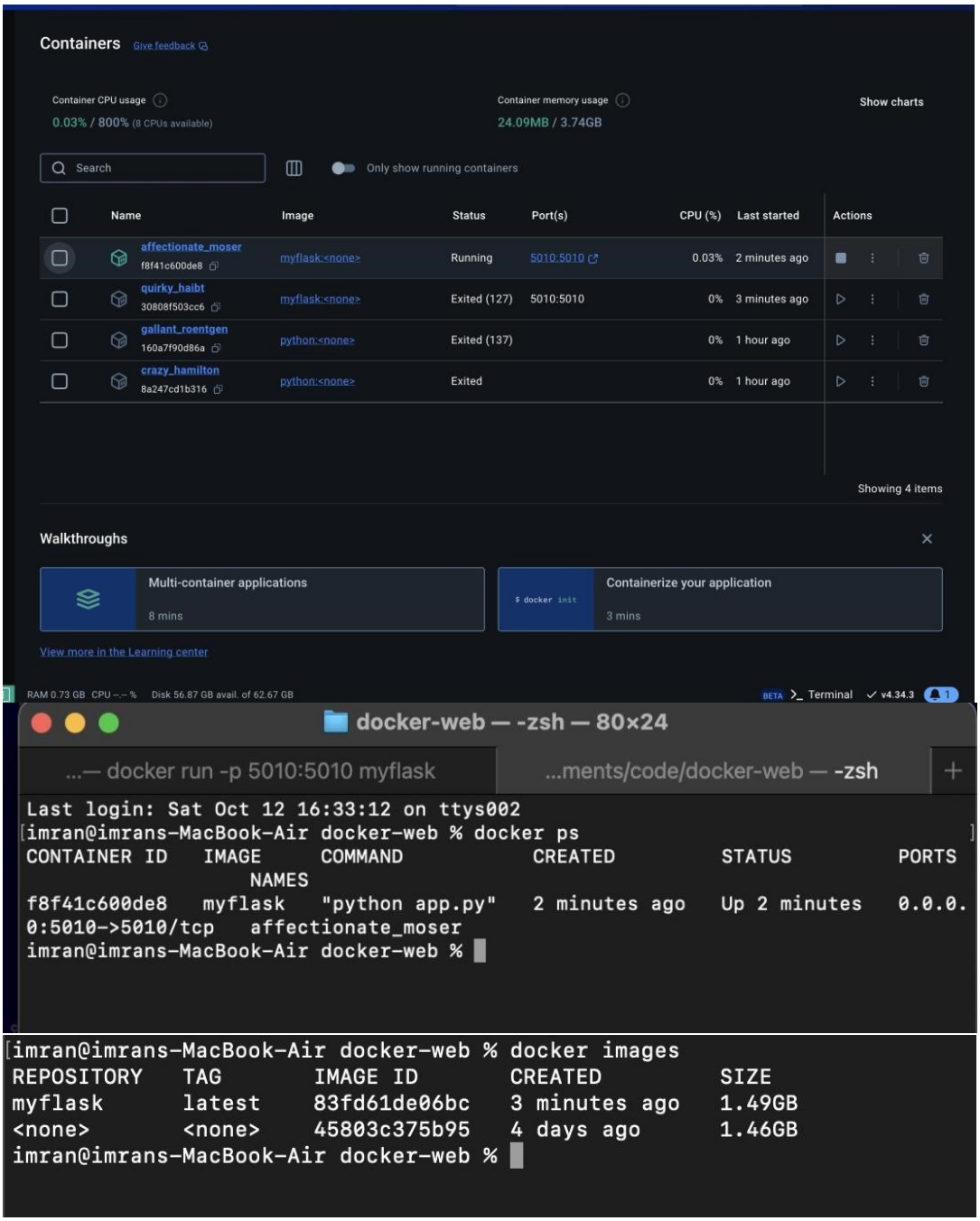

Run the application at port 5010




Application




New ‘myflask’ image





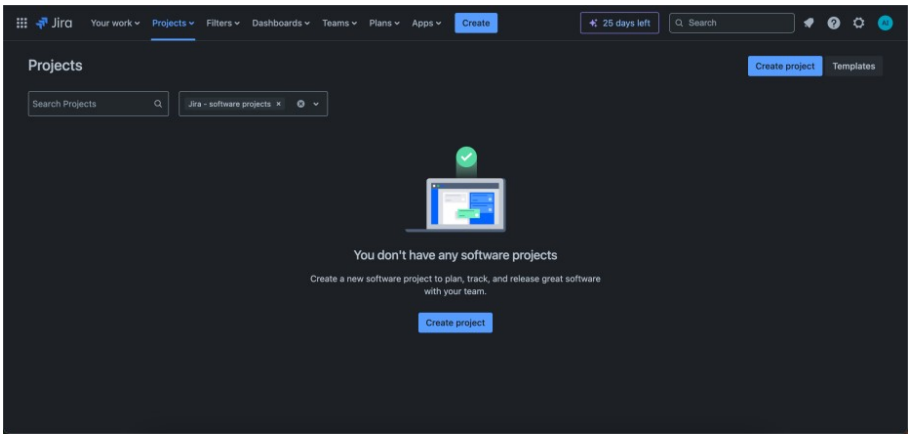


**Conclusion:** Docker file instructions were learnt, an image for a sample web application was built using a Docker file.

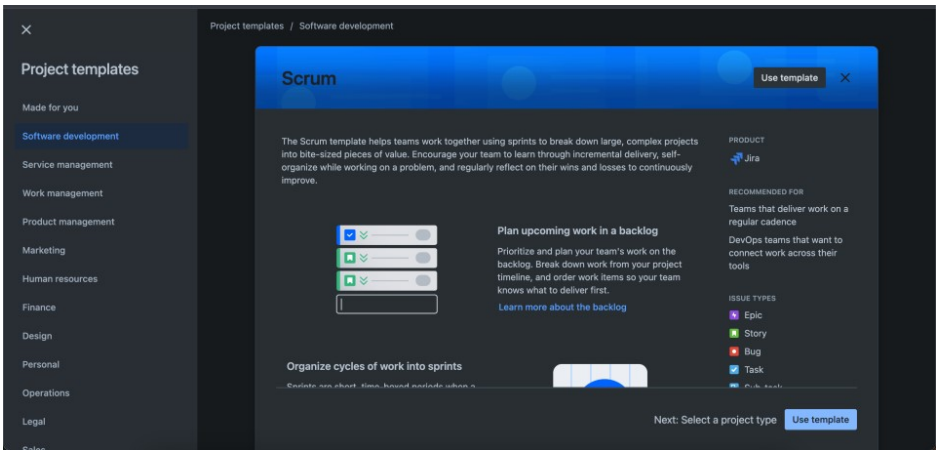
# EXPERIMENT NO 10

Steps for organisation and management of a project using Jira tool.
Create a free trial Jira account.

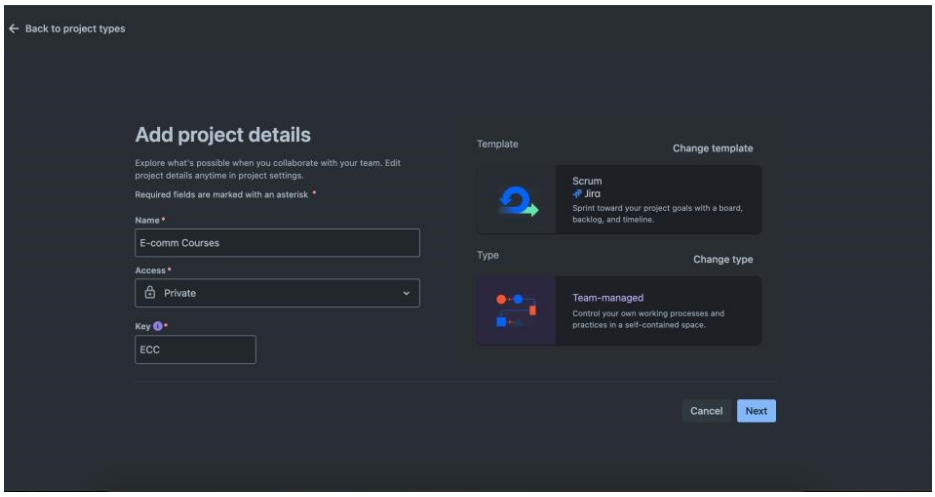
1. Start a project on Jira



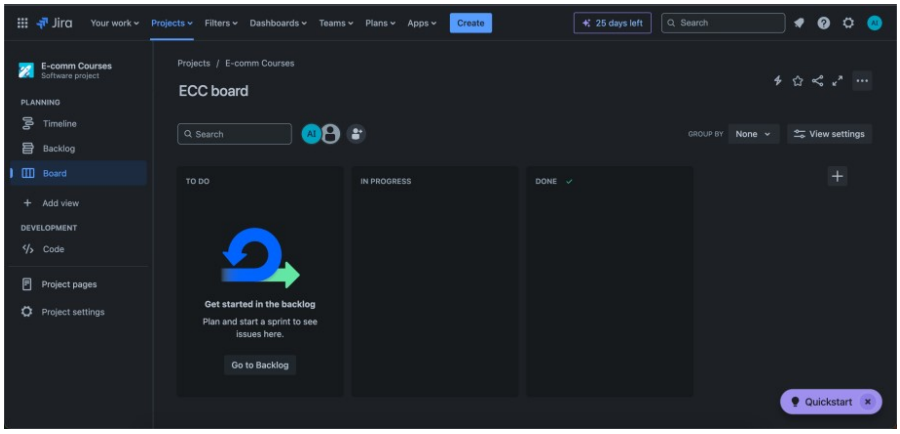

2. Choose a project template (Scrum is used here)



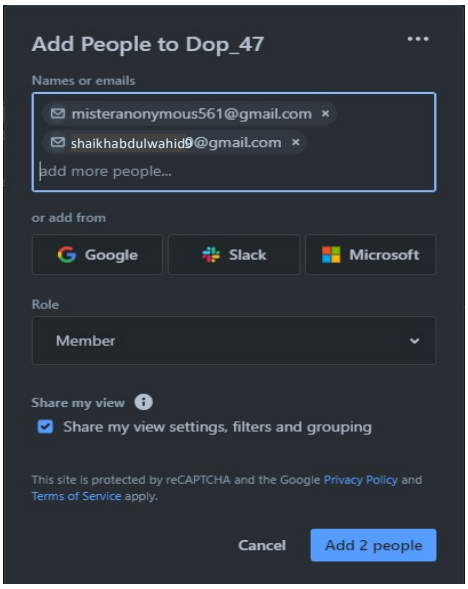

3. After naming it finish the cration process



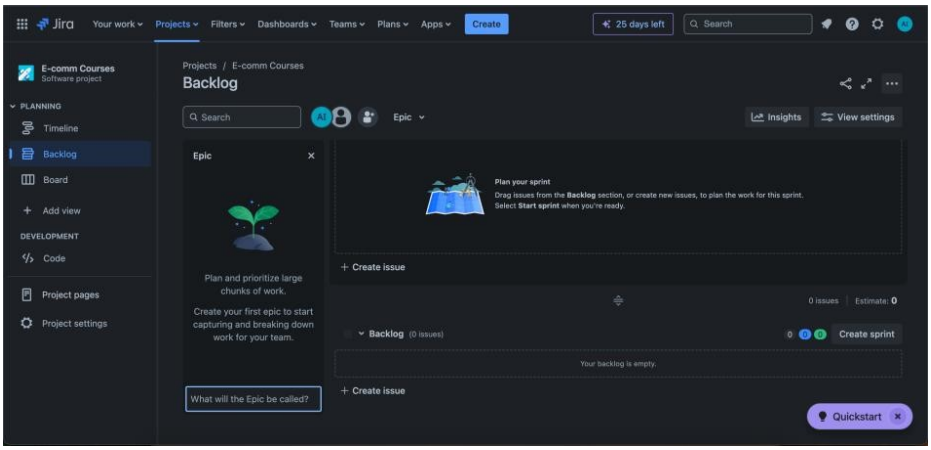

Project Dashboard



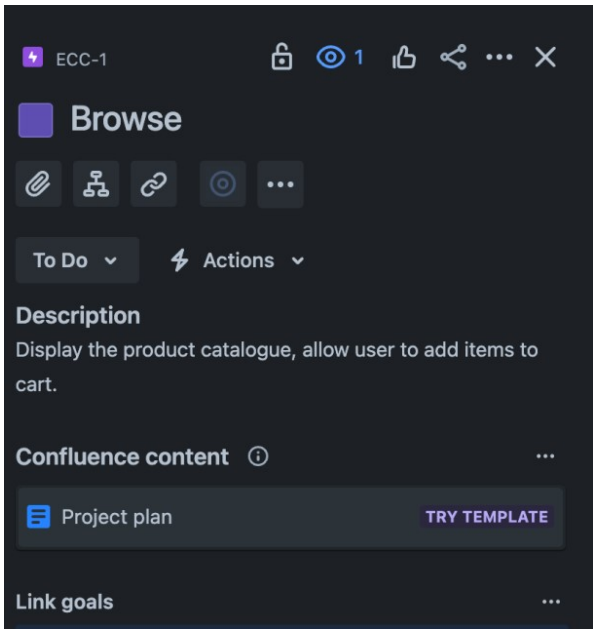

Invite team members to the project

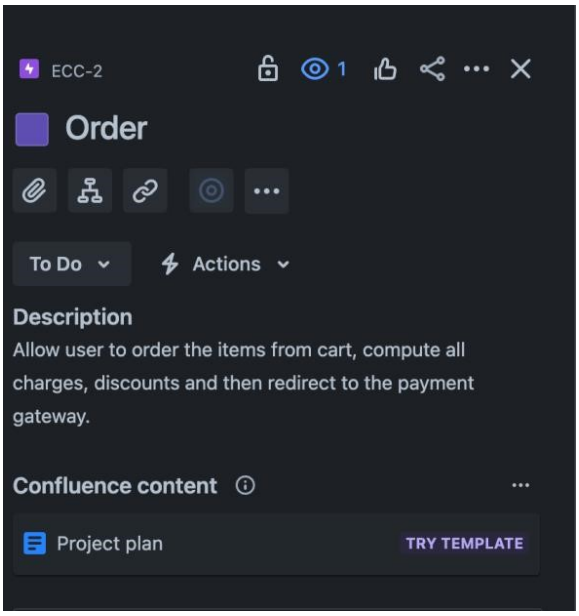


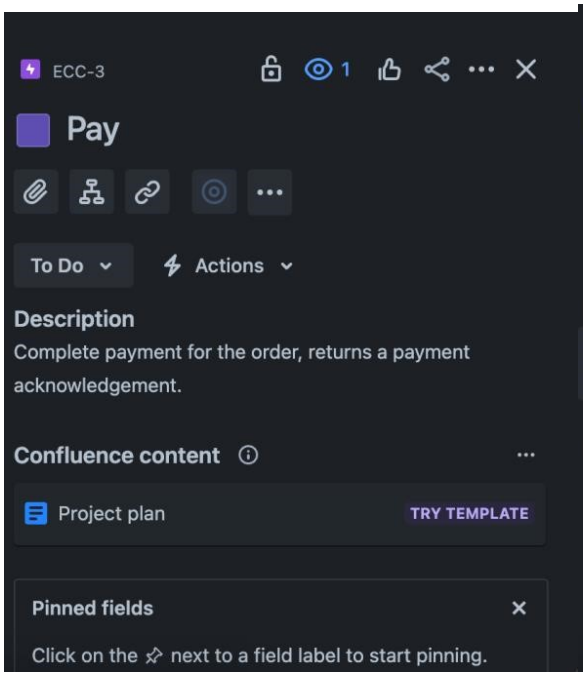

**4.** In Backlog option, create as many Epics as per project requirements.

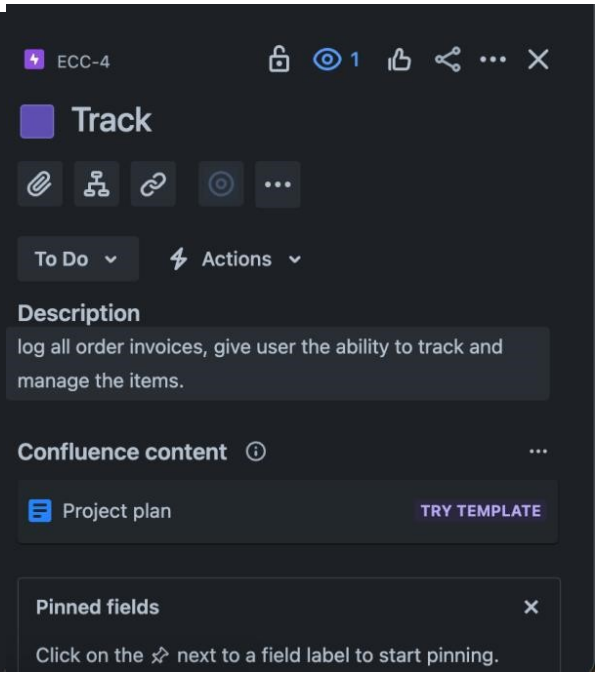



**5.** Give description to the created Epics

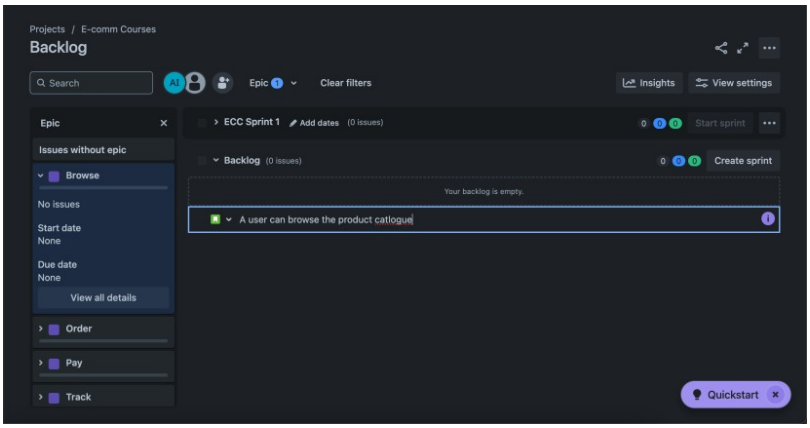


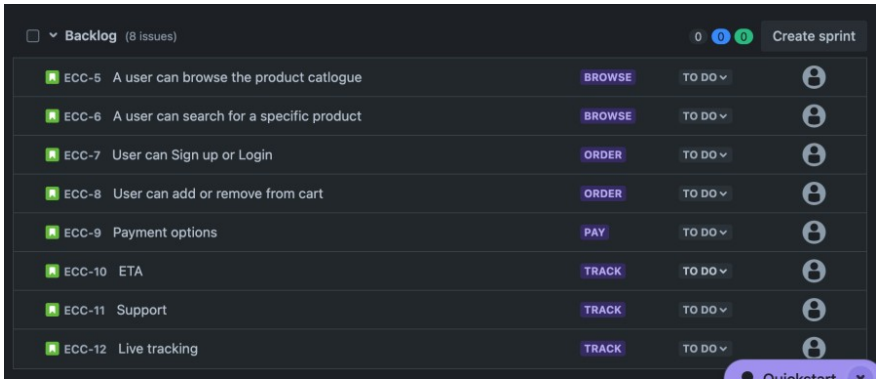



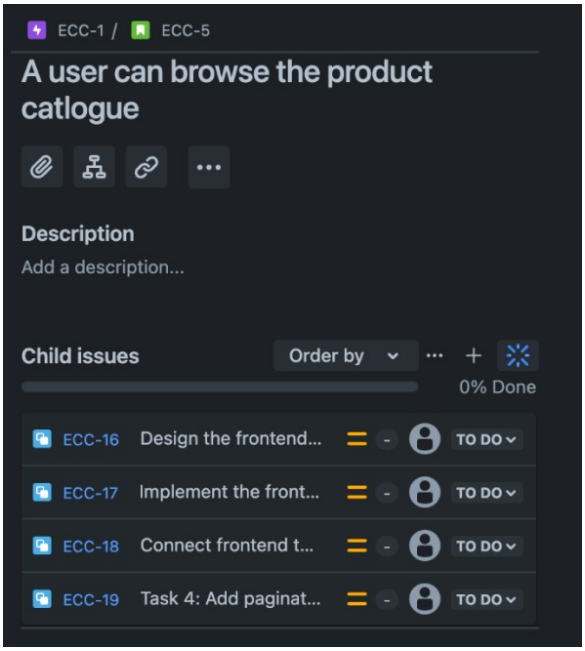



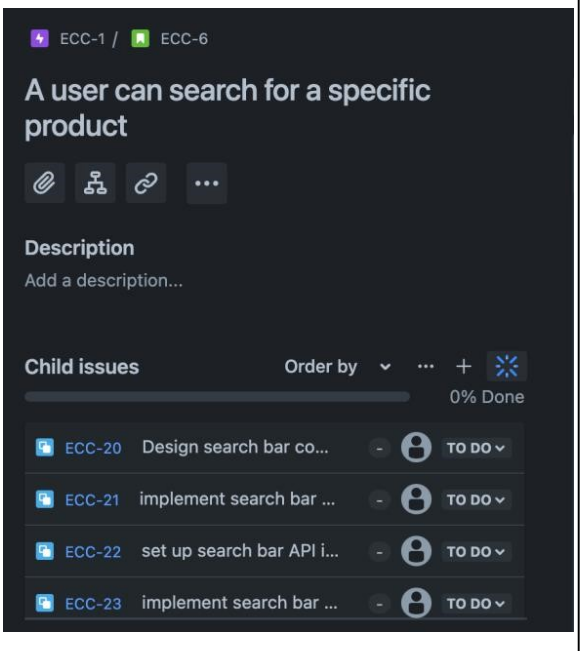

**6.** User stories can be created for the respective Epics



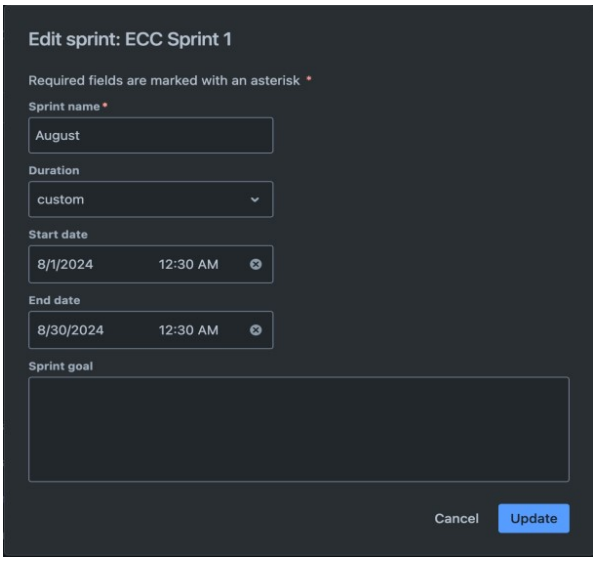




**7.** Break down User Stories into Tasks

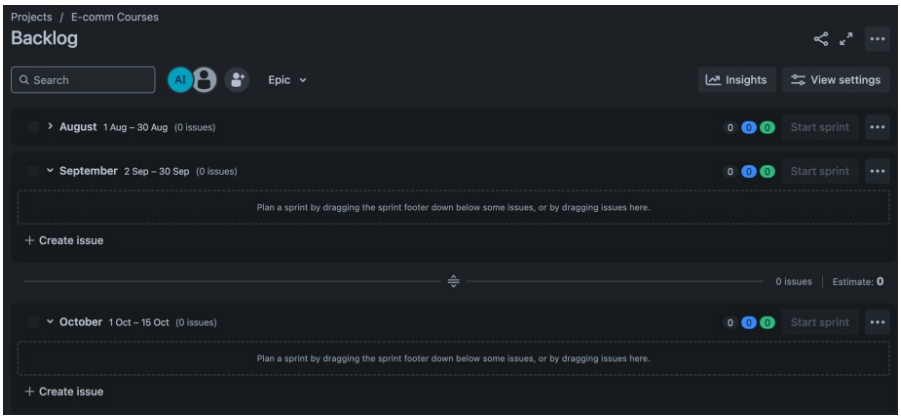


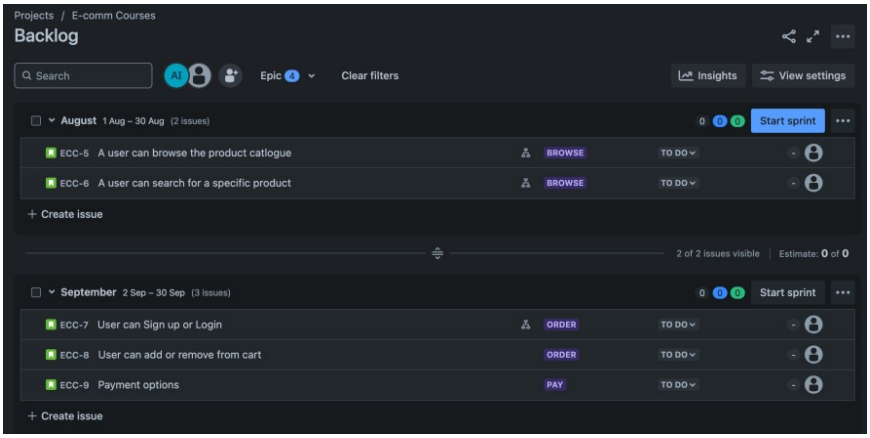

**8.** Create Sprints.

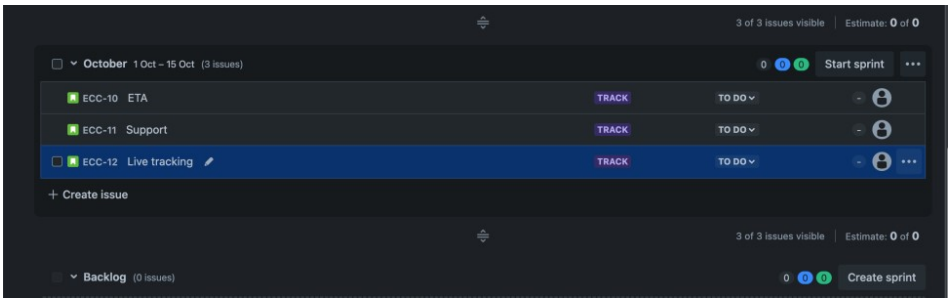



Sprints created:

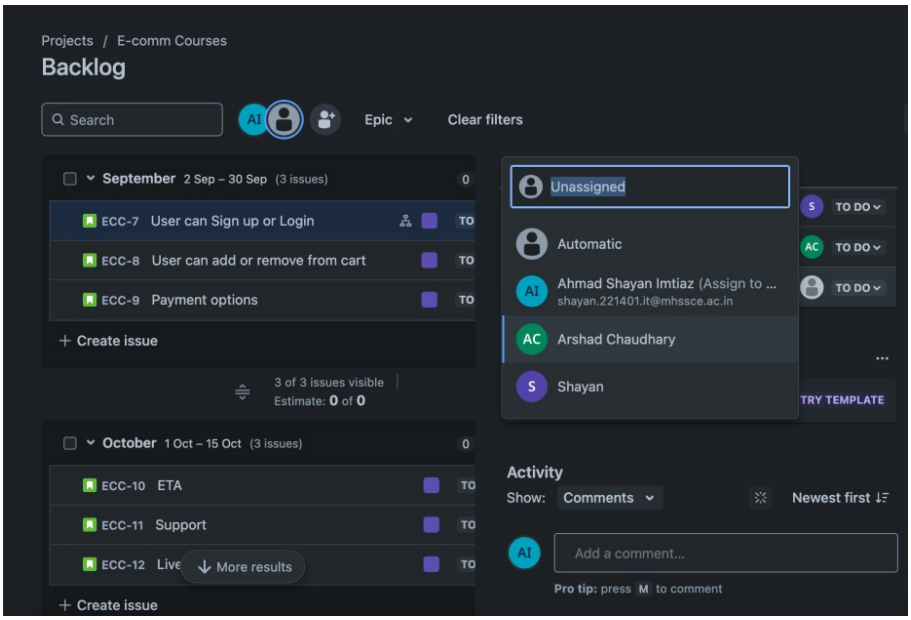



**9.** Assign tasks to sprints

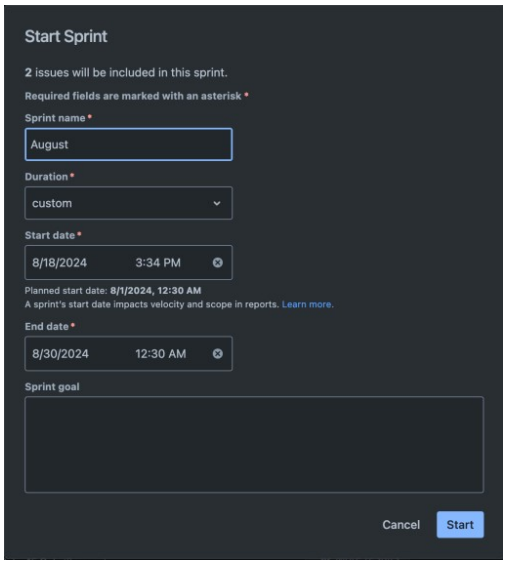




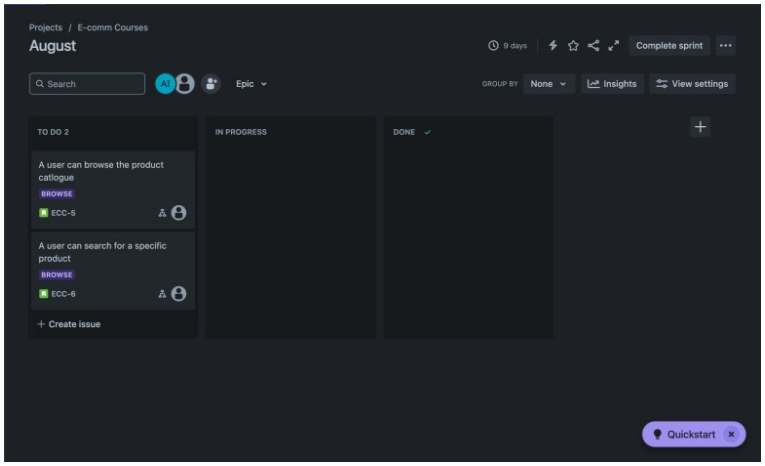

**10.** Assign Tasks to team members



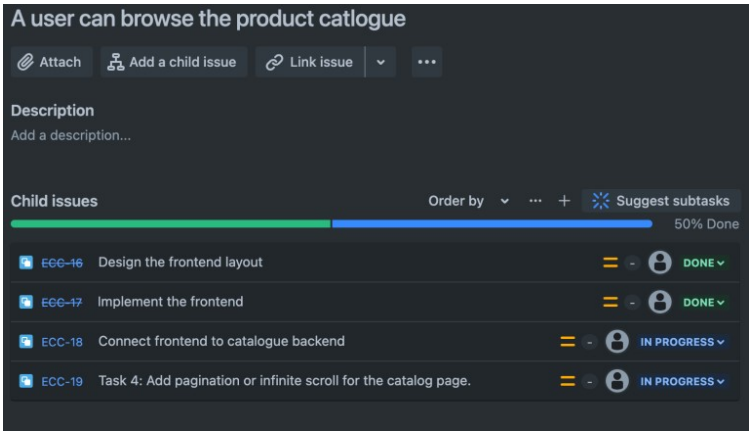

**11.** After assignment, start a sprint

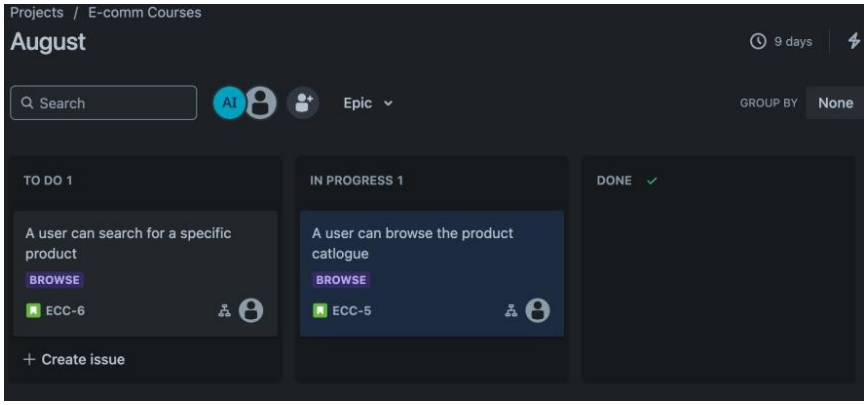


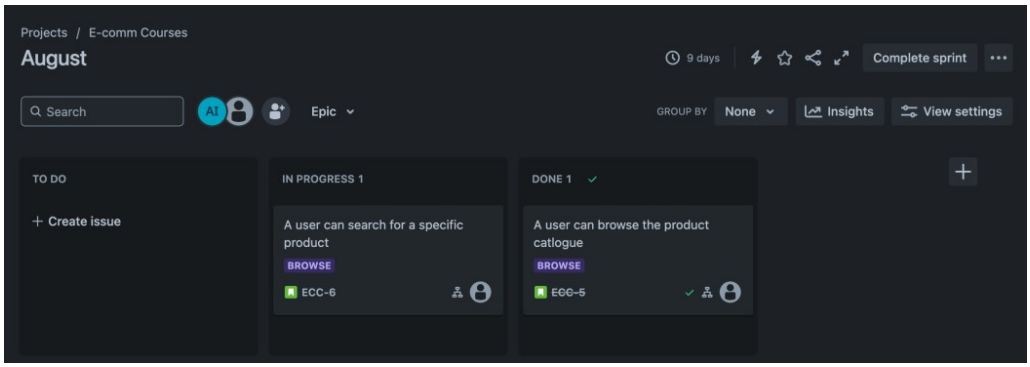
Tasks will be in TO DO section by default



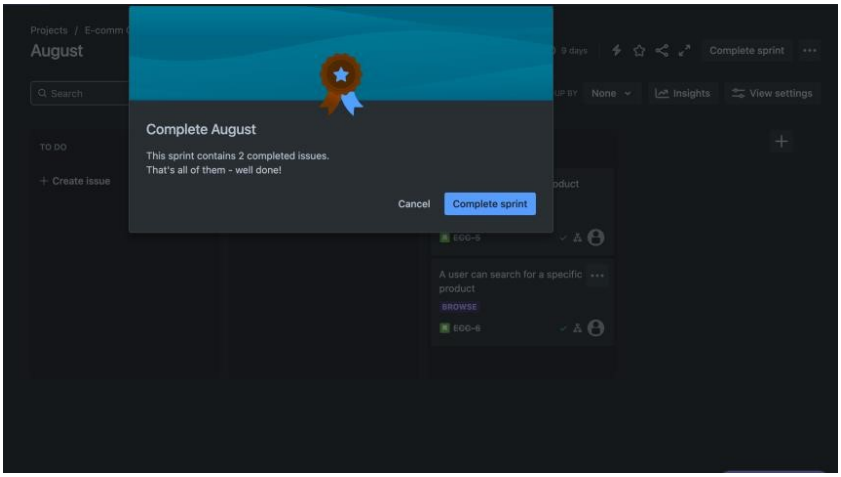

Move tasks through the workflow



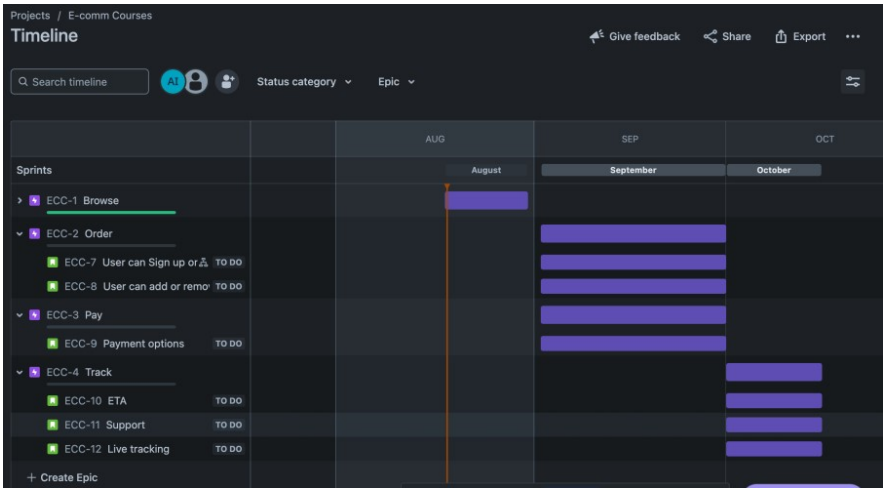






**12.** Once all the tasks are done complete the sprint




Timeline chart can be used to track the progress of the project




Conclusion: A project was successfully organised using Jira tool.