UNIT III – DEVOPS AND CI/CD CONCEPTS [9 hours]

What is DevOps? Why is it used in industry?,CI/CD – Continuous Integration and Deployment, Introduction to GitHub Actions / Jenkins, Introduction to Docker and Dockerfile, Automating build and test for an app

---

INTRODUCTION TO JENKINS

Jenkins is an open source automation tool which allows continuous integration (CI). It is written in Java. Jenkins builds and tests our software projects continuously which makes it easy for developers to integrate the changes in the project and reduces the cost of manual testing and increases quality. Jenkins integrates all types of development life cycle processes including build, document, test, package, phase, deploy and so on. Jenkins supports a large number of plugins which makes it easy to configure and customize any project.

Features of Jenkins:

Following are the features of Jenkins:

Open-Source and Free:

Jenkins is freely available and supported by a large, active community, ensuring continuous development and extensive resources.

Easy Installation and Configuration:

It is a self-contained Java-based program with straightforward installation processes across various operating systems like Windows, Linux, and macOS. Configuration is managed through a user-friendly web interface.

Extensive Plugin Ecosystem:

Jenkins boasts a vast library of plugins (over 1,500) that extend its functionality and enable integration with a wide array of tools and technologies in the software development lifecycle, including version control systems (Git, SVN), build tools (Maven, Gradle), testing frameworks, and deployment platforms (Docker, Kubernetes, AWS).

Continuous Integration and Delivery (CI/CD) Capabilities:

Jenkins automates the entire CI/CD pipeline, including building, testing, and deploying software projects whenever code changes are committed.

Distributed Builds:

Jenkins can distribute build jobs across multiple agent machines (nodes), optimizing resource utilization and accelerating build and test times.

Monitoring and Reporting:

Jenkins provides built-in capabilities for monitoring build status, viewing logs, tracking performance trends, and generating reports to provide insights into the development process.

Notifications:

It can send notifications about build and deployment results through various channels like email or chat platforms.

ADVANTAGES AND DISADVANTAGES OF USING JENKINS

Advantages

- Highly extensible with a huge variety of existing plugins. Plugins contribute to Jenkins' flexibility and rich scripting and declarative language which supports advanced, custom pipelines.
- Robust and reliable at almost any scale.
- Mature and battle-tested.
- Supports hybrid and multi-cloud environments.
- Offers an extensive knowledge base, documentation, and community resources.
- Based on Java, an enterprise development language with a broad ecosystem, making it suitable for legacy enterprise environments.

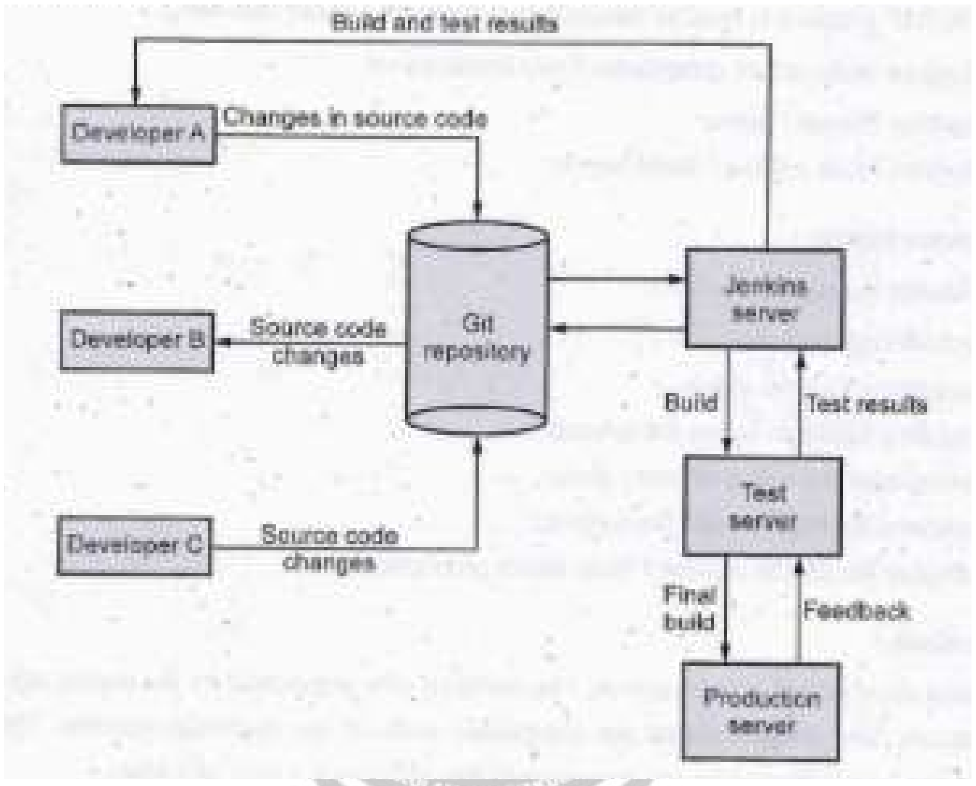
Disadvantages:

- Single server architecture—uses a single server architecture, which limits resources to resources on a single computer, virtual machine, or container. Jenkins doesn't allow server-to-server federation, which can cause performance issues in large-scale environments.

- Jenkins sprawl—this is a common problem which also stems from lack of federation. Multiple teams using Jenkins can create a large number of standalone Jenkins servers that are difficult to manage.
- Relies on dated Java architectures and technologies—specifically Servlet and Maven. In general, Jenkins uses a monolithic architecture and is not designed for newer Java technologies such as Spring Boot or GraalVM.
- Not container native—Jenkins was designed in an era before containers and Kubernetes gained popularity, and while it supports container technology, it does not have nuanced support for container and orchestration mechanisms.
- Difficult to implement in production environments—developing continuous delivery pipelines with Jenkinsfiles requires coding in a declarative or scripting language, and complex pipelines can be difficult to code, debug, and maintain.
- Offers no functionality for real production deployments—"deploying with Jenkins" means running a fully customized set of scripts to handle the deployment.
- Jenkins itself requires deployment—this can be difficult to automate. Organizations that need to combine Jenkins with a continuous delivery solution have traditionally used configuration management to do this, but this adds another layer of complexity and is error-prone.
- Complicated plugin management—Jenkins has nearly 2,000 plugins, which can be overwhelming to sort through until you find a useful plugin. Many plugins also have dependencies that increase the management burden, while some plugins may conflict with each other. There is no guarantee a plugin you use will continue to be maintained.
- Groovy expertise requirements—Jenkins has programmatic pipelines implemented in Groovy, a language that is currently not in wide use and can make scripts difficult to work with. Jenkins supports scripted and declarative Groovy modes.

JENKINS ARCHITECTURE OVERVIEW:

Jenkins Workflow:



- The developers commit the code to a source code repository such as GitHub. Jenkins checks the repository at regular intervals for noticing the changes committed in the repository.
- When Jenkins server finds that the changes are committed in the repository, it starts preparing a new build.
- If the build fails, then the developers are notified about it.
- If the build is successful, then the Jenkins server deploys the build on the test server.
- The test server tests the build and generates the feedback. The Jenkins server gets this feedback and notifies about build and test results to the developers.
- If everything is perfect then the build is deployed on the production server
- In all the above activities, Jenkins server continuously verifies the code repository for changes made in the source code. If so, the above activities are repeated continuously.

Jenkins Architecture:

Jenkins works on Master-Slave Architecture also called Controller-Agent. It manages distributed builds using this Master-Slave architecture. The TCP/IP protocol is used to communicate between Master-Slave architecture. Jenkins architecture comprises of two components

1. Jenkins Master / Server
2. Jenkins Node / Slave / Build Server

1. Jenkins Controller(Formerly Master)

The Jenkins Controller serves as the central system for managing a Jenkins instance, often referred to as its "heart." It oversees agents and their connections, determining the tasks they should perform. Additionally, the Jenkins Controller loads plugins and ensures that jobs run in the correct sequence. The Jenkins Controller gives instructions to the agents, which follow the directions to complete their work efficiently.

Master is responsible for:

- Scheduling the jobs
- Assigning them to slaves
- Sending builds to slaves for execution
- Monitoring the status of every slave
- Retrieve the build results from the slaves
- Display the results received from the slave to the console.

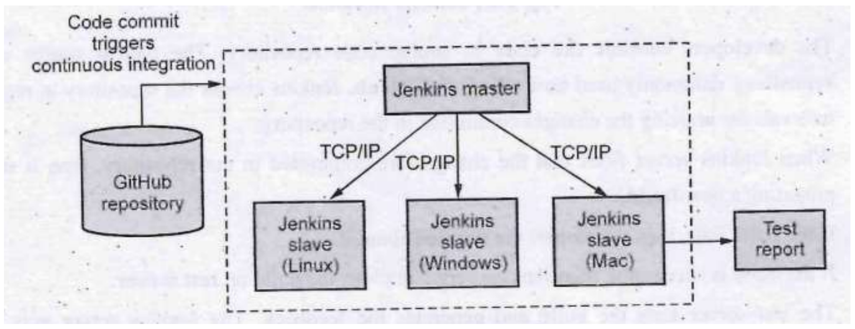
2. Jenkins Agent(Formerly Slave)

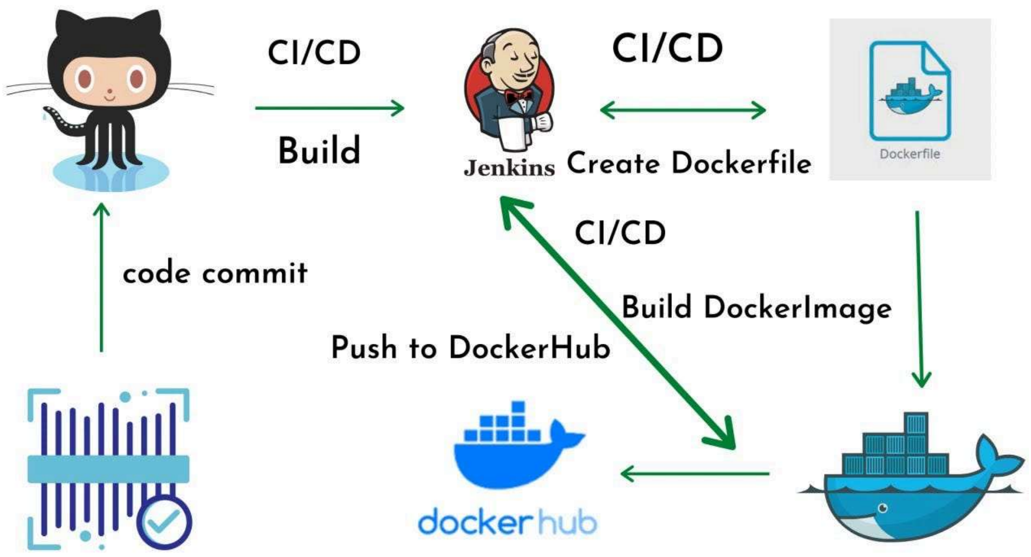
Jenkins Agent is a machine that performs tasks like running scripts, executing tests, or building components, etc. These tasks are assigned by the Jenkins Controller. Each agent can have its setup, like different operating systems, software, or hardware. This helps Jenkins handle many types of tasks and work faster by spreading the load. The jenkins slave can be configured on any server including Windows, Linux and mac.

There are two main types of agents:

Permanent Agents: These are always ready and connected to Jenkins. They're like dedicated workers who are always on standby.

Ephemeral Agents: These are temporary. Jenkins starts them only when needed, usually in the cloud or using tools like Docker. When the job is done, they're shut down.





CREATING AND CONFIGURING A JENKINS JOB

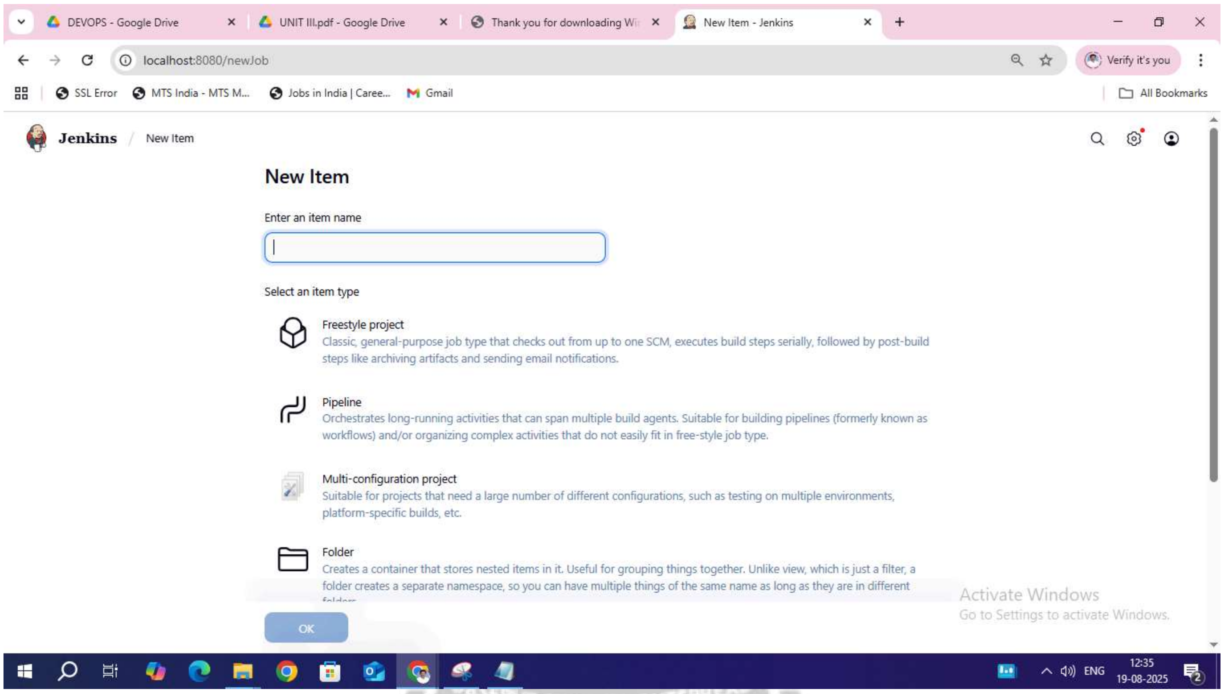
In Jenkins, Jobs (also called Projects) represent tasks or automated processes. These tasks often involve building software, running tests, deploying applications, or performing any other repetitive task that can be automated.

Jobs are at the core of Jenkins' functionality, enabling continuous integration and continuous delivery (CI/CD) pipelines.

Jobs include triggers, build steps, and post-build actions, enabling streamlined automation in development workflows. Jenkins job is created by using the following steps.

Step 1: Open Jenkins dashboard in browser by typing localhost:8080

Step 2: Create a free style project by clicking New Item and provide the name and click OK



New Item

Enter an item name

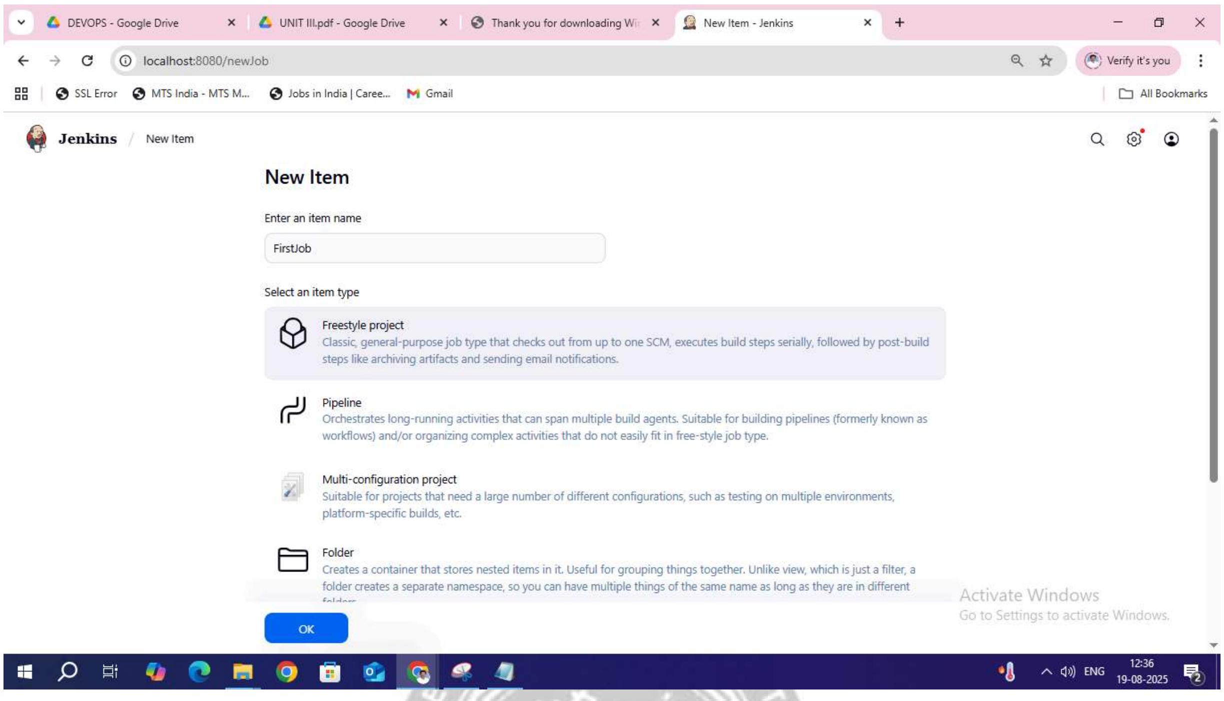
Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different


Jenkins / New Item

New Item

Enter an item name

FirstJob

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

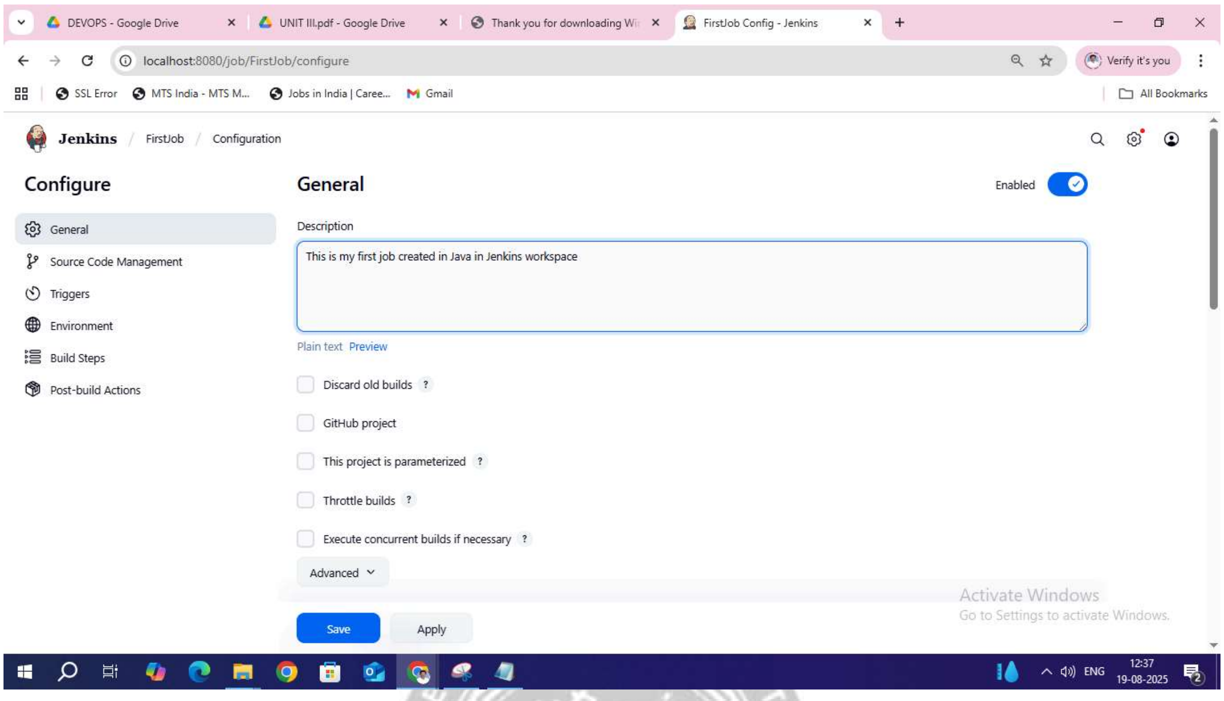
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Step 3: Configure the project as follows:

Provide the description



Jenkins / FirstJob / Configuration

Configure

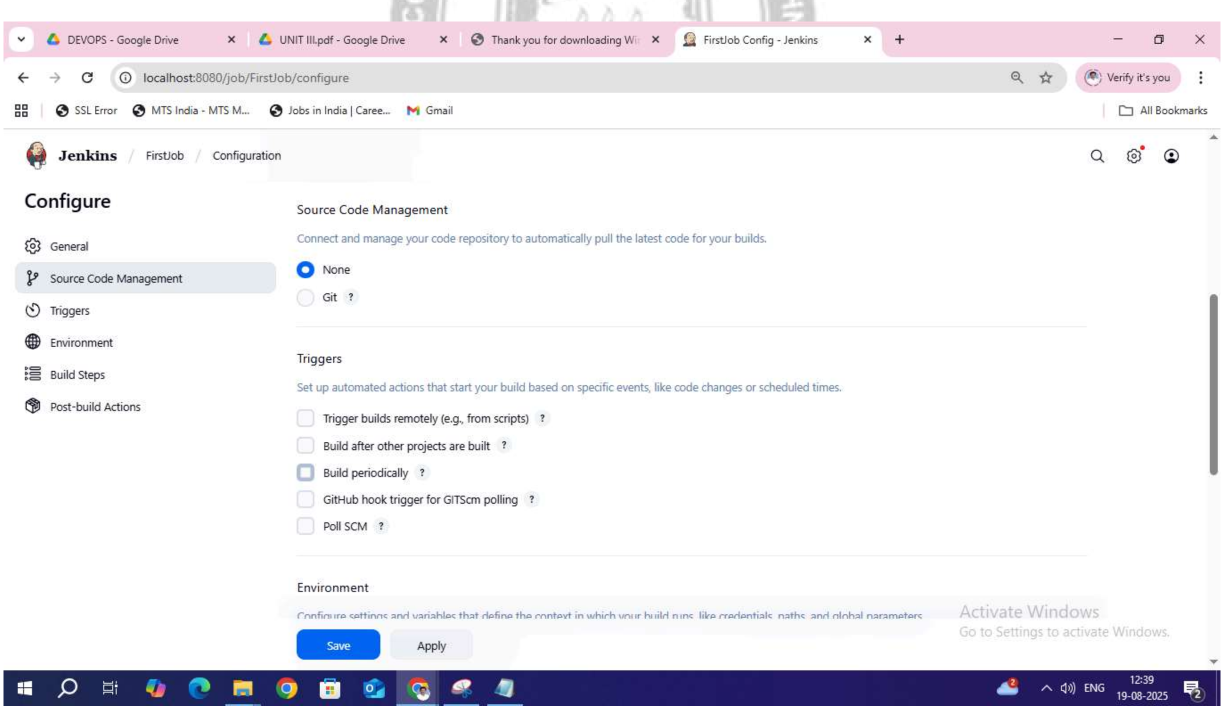
General

Description: This is my first job created in Java in Jenkins workspace

Enabled

- Discard old builds
- GitHub project
- This project is parameterized
- Throttle builds
- Execute concurrent builds if necessary

Step 4: In the source code management, specify the location of the source code.



Jenkins / FirstJob / Configuration

Configure

Source Code Management

Connect and manage your code repository to automatically pull the latest code for your builds.

- [✓ None]
- Git

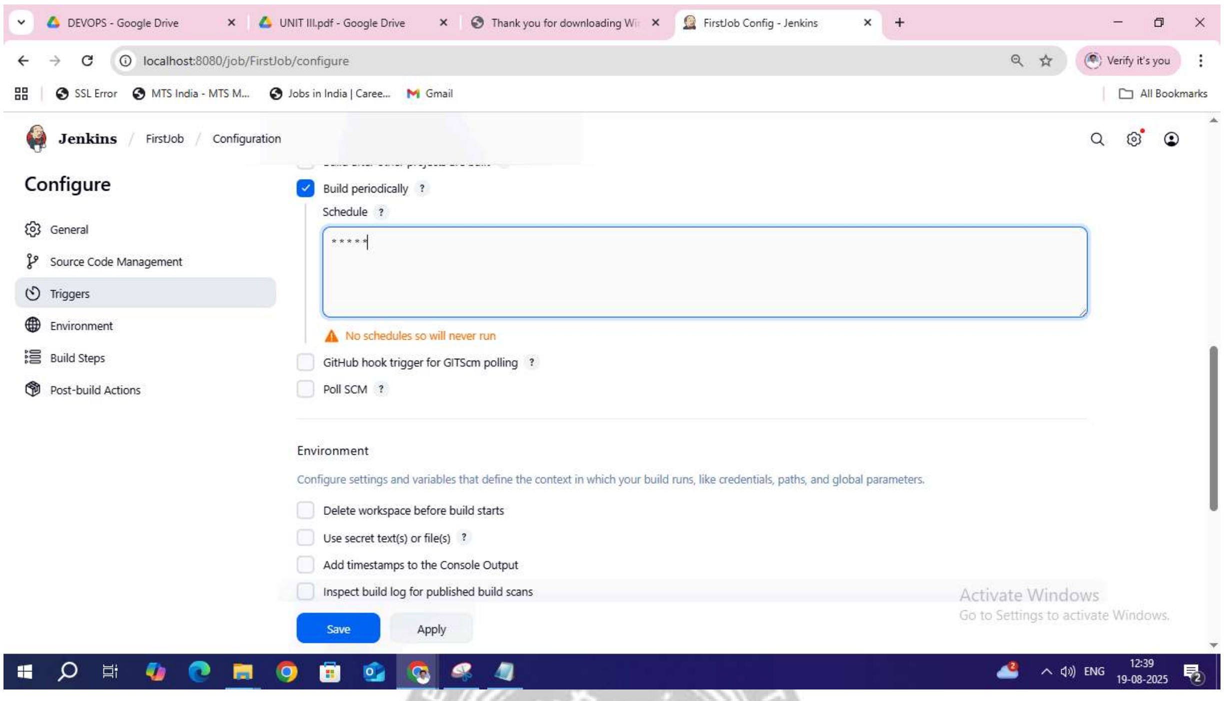
Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- Trigger builds remotely (e.g., from scripts)
- Build after other projects are built
- Build periodically
- GitHub hook trigger for GITScm polling
- Poll SCM

Environment

Step 5: In the triggers sections, specify the expression to do periodic build action.


localhost:8080/job/FirstJob/configure

Jenkins / FirstJob / Configuration

Configure

Build periodically ✓

Schedule

```
* * * * *
```

No schedules so will never run

GitHub hook trigger for GITScm polling

Poll SCM

Environment

Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.

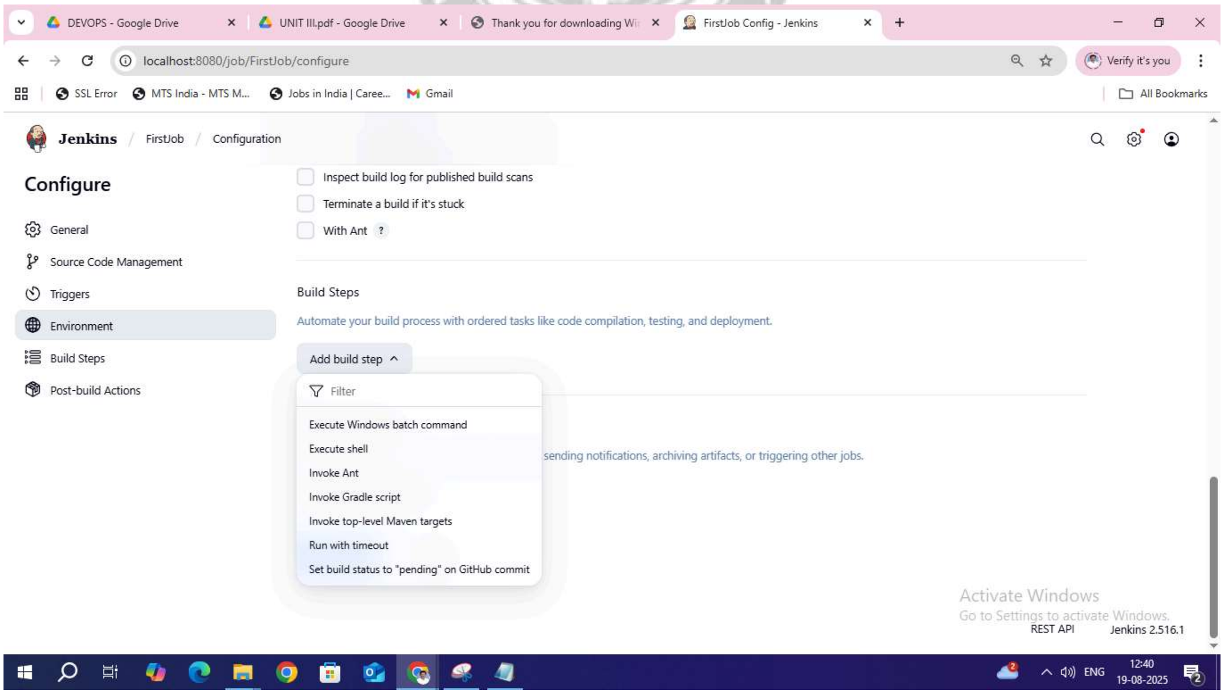
Delete workspace before build starts

Use secret text(s) or file(s)

Add timestamps to the Console Output

Inspect build log for published build scans

Step 6: In the build sections, specify the steps that you want Jenkins to perform when the job is run. Type the java commands to execute the java program


localhost:8080/job/FirstJob/configure

Jenkins / FirstJob / Configuration

Configure

Inspect build log for published build scans

Terminate a build if it's stuck

With Ant

Build Steps

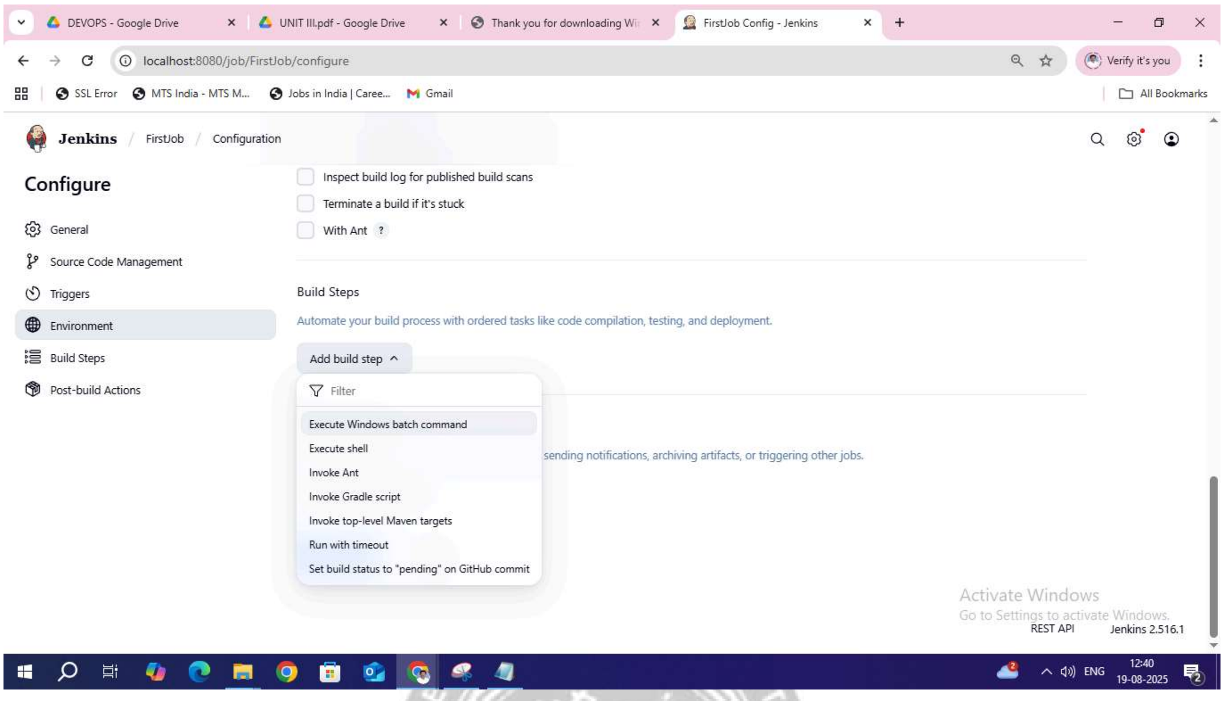
Automate your build process with ordered tasks like code compilation, testing, and deployment.

Add build step

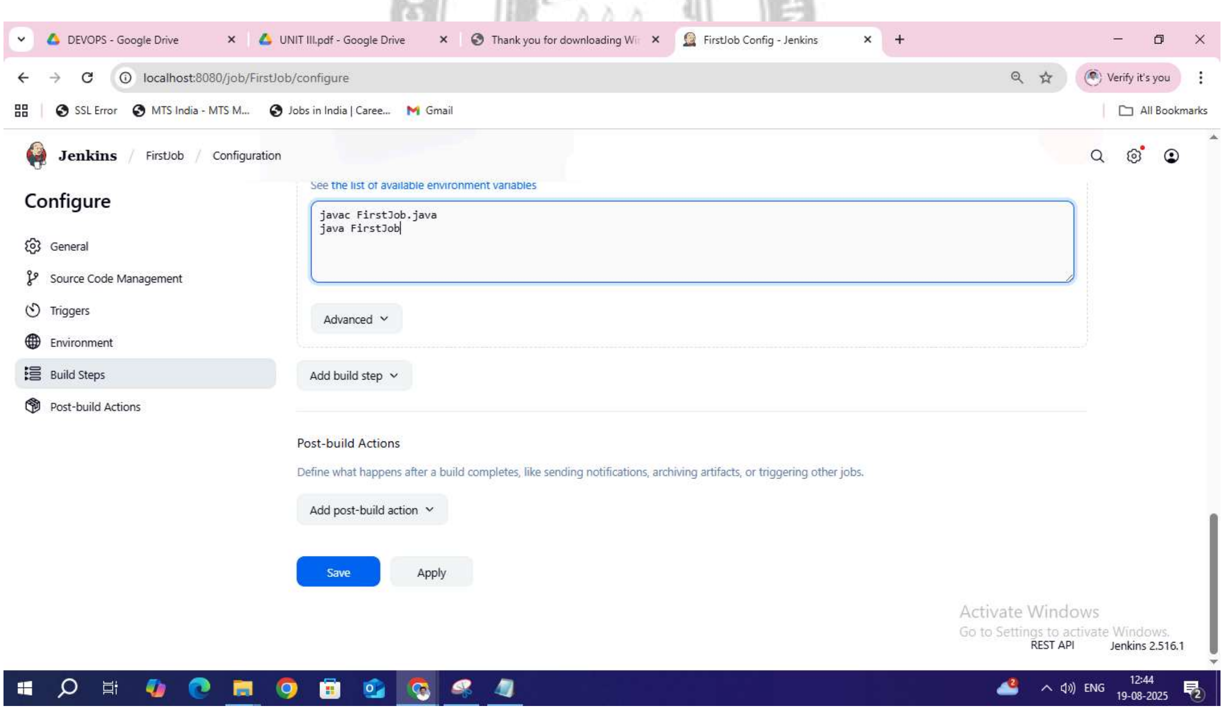
- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Set build status to "pending" on GitHub commit

sending notifications, archiving artifacts, or triggering other jobs.

REST API Jenkins 2.516.1



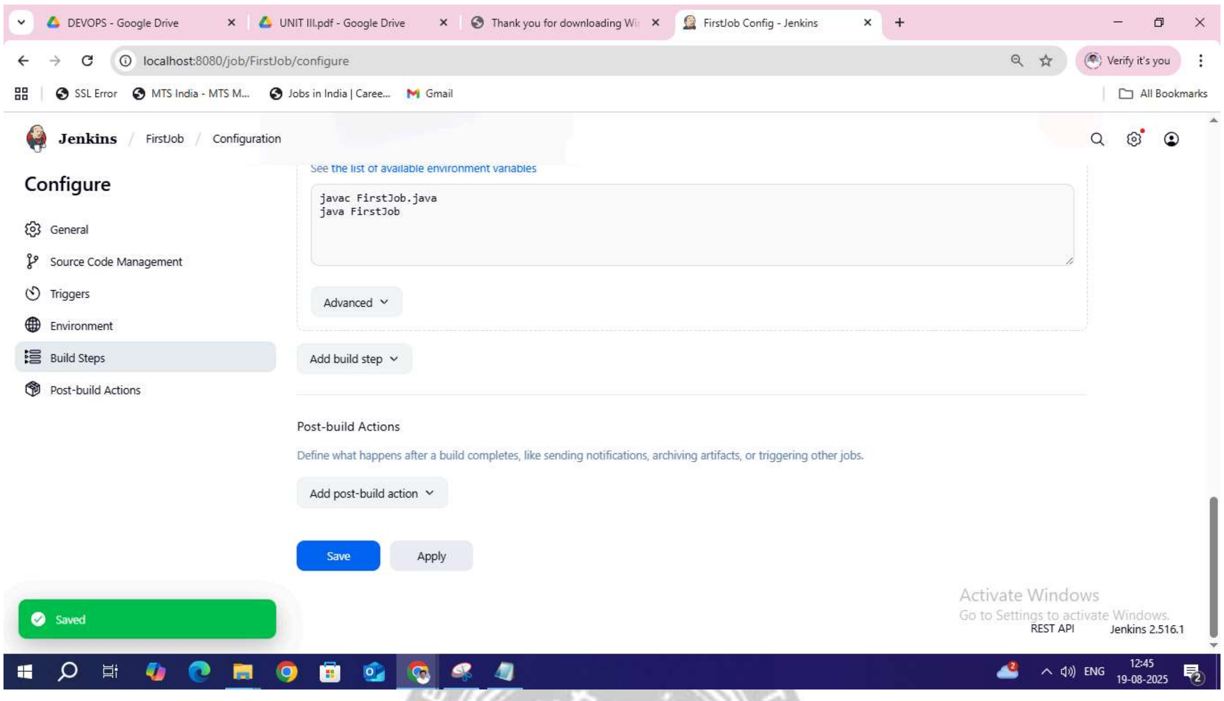
localhost:8080/job/FirstJob/configure
Jenkins / FirstJob / Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Inspect build log for published build scans
Terminate a build if it's stuck
With Ant
Build Steps
Automate your build process with ordered tasks like code compilation, testing, and deployment.
Add build step
Filter
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
REST API
Jenkins 2.516.1



localhost:8080/job/FirstJob/configure
Jenkins / FirstJob / Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
See the list of available environment variables
Advanced
Add build step
Post-build Actions
Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.
Add post-build action
Save
Apply
REST API
Jenkins 2.516.1

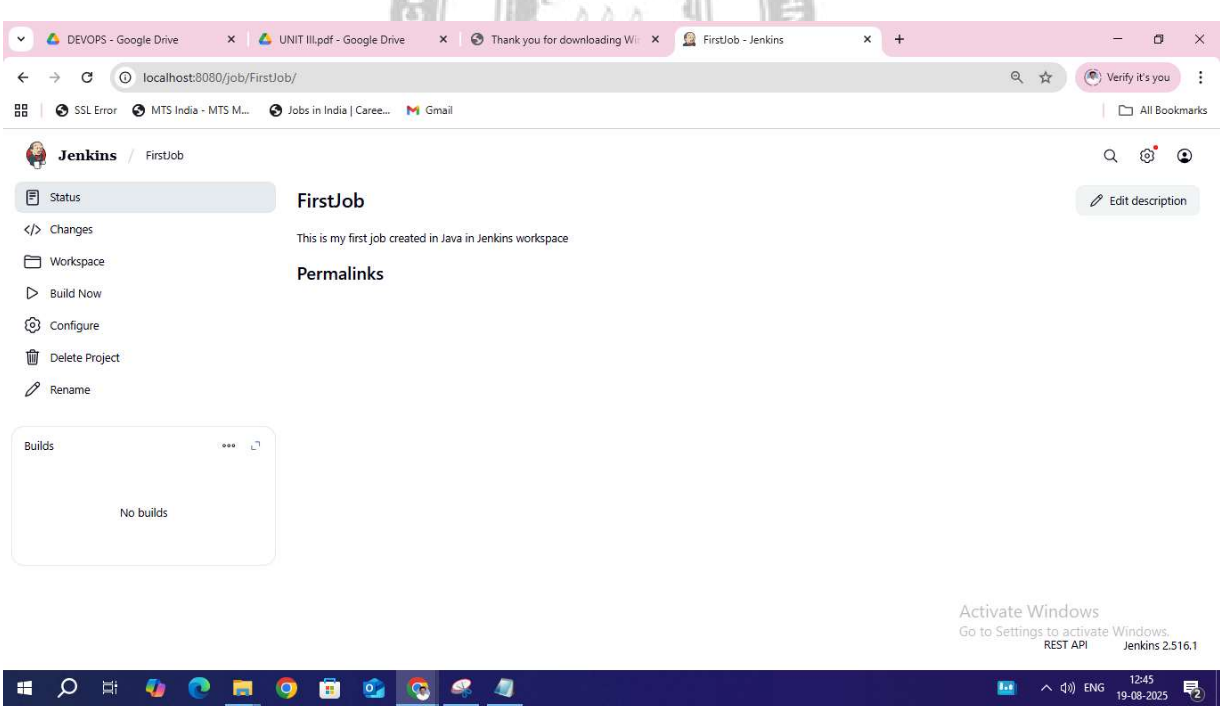
```
javac FirstJob.java
java FirstJob
```

Step 7: Click Apply and Save


Jenkins / FirstJob / Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
See the list of available environment variables

```
javac FirstJob.java
java FirstJob
```

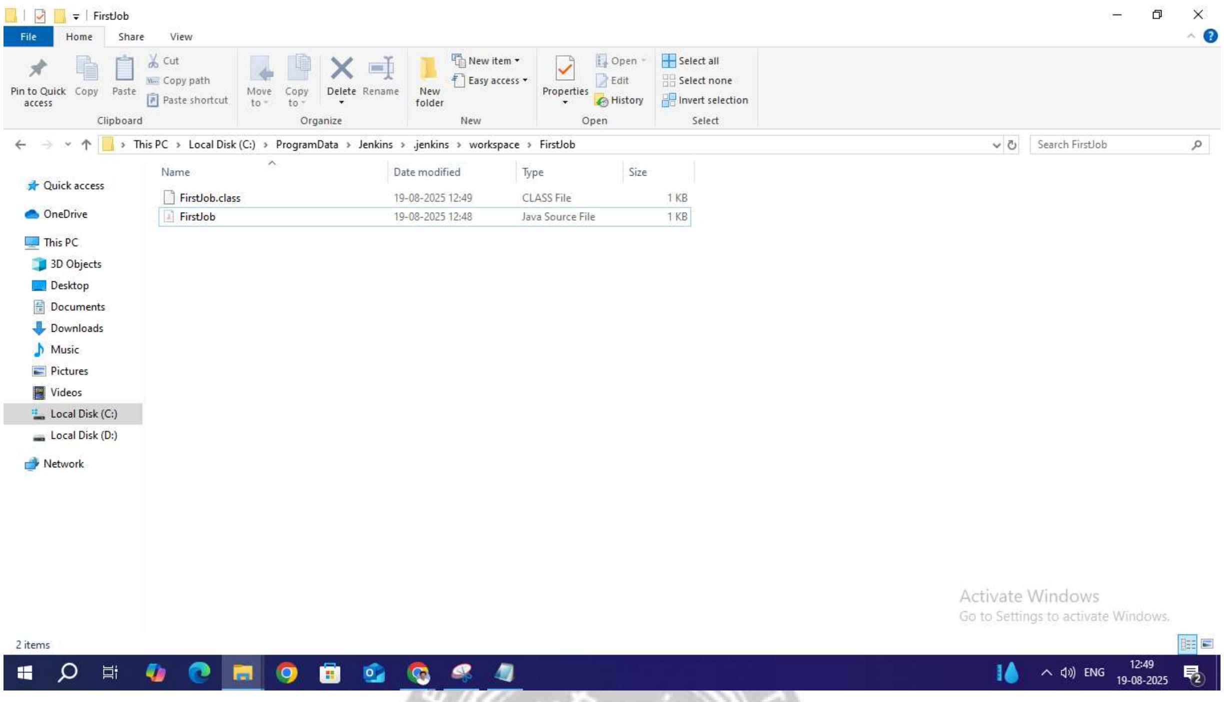
Advanced
Add build step
Post-build Actions
Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.
Add post-build action
Save
Apply
Saved



FirstJob

This is my first job created in Java in Jenkins workspace

Permalinks

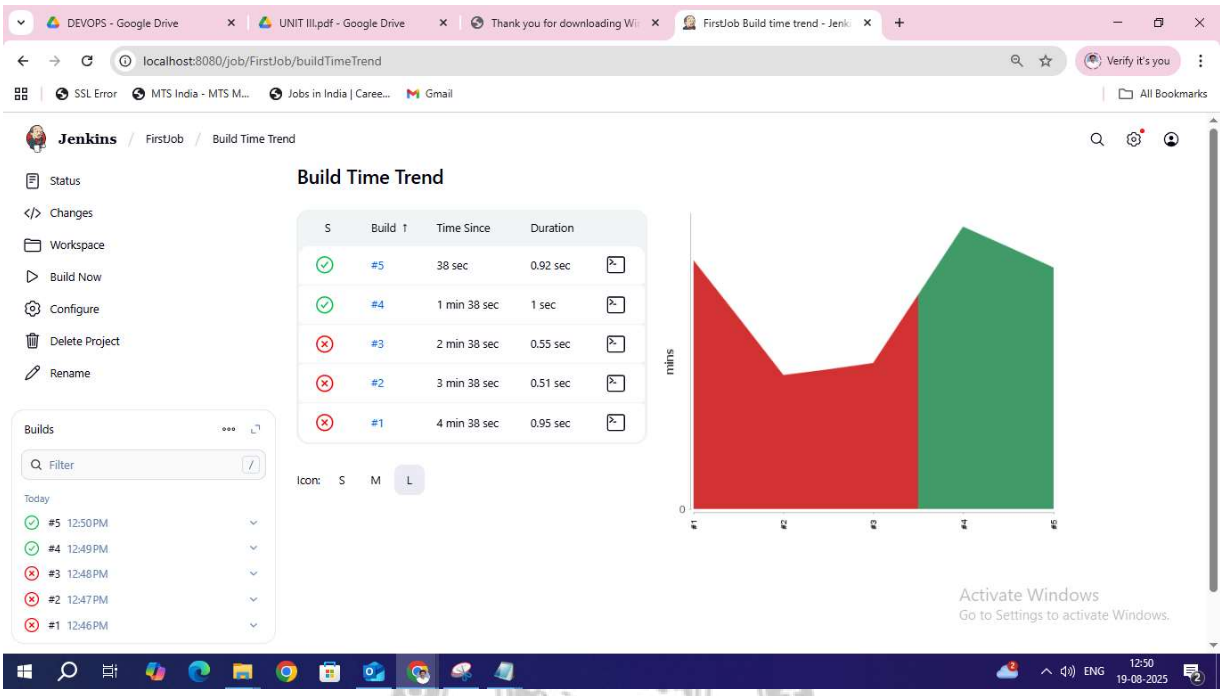


This PC > Local Disk (C:) > ProgramData > Jenkins > .jenkins > workspace > FirstJob

Name	Date modified	Type	Size
FirstJob.class	19-08-2025 12:49	CLASS File	1 KB
FirstJob	19-08-2025 12:48	Java Source File	1 KB

```
public class FirstJob
{
        public static void main(String args[])
        {
                System.out.println("First Java Program in Jenkins");
        }
}
```

Step 8: Check the status of the build by clicking on the build number



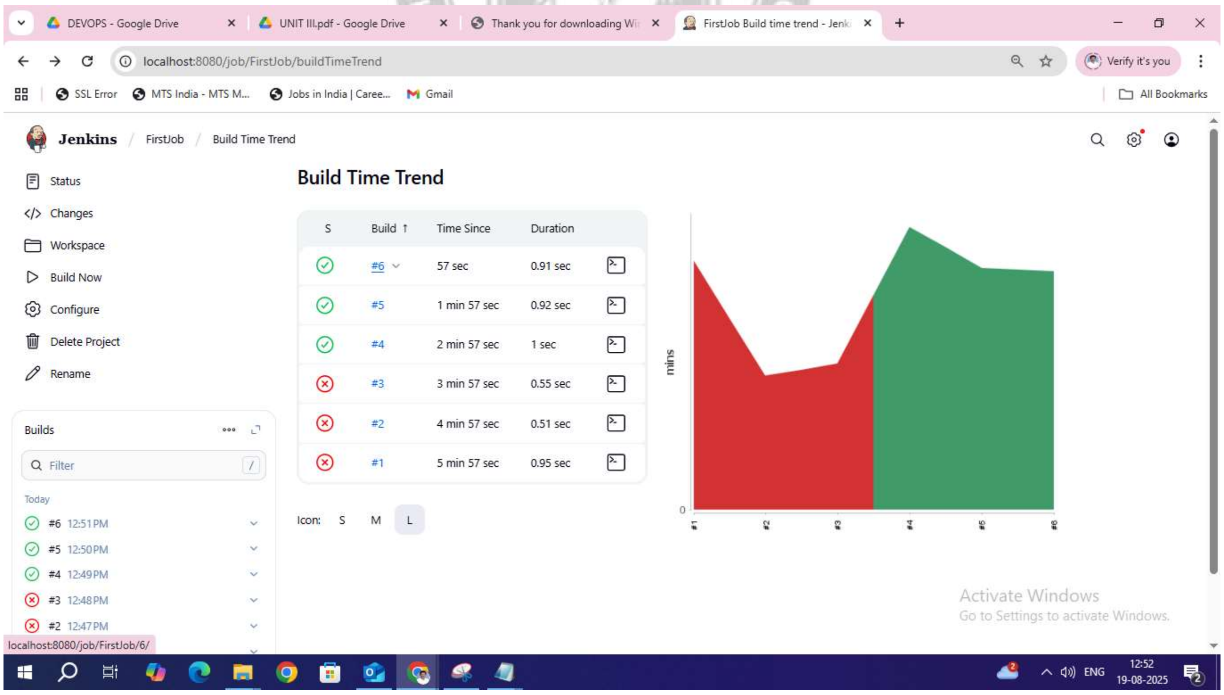
Jenkins / FirstJob / Build Time Trend

Build Time Trend

S	Build ↑	Time Since	Duration
✓	#5	38 sec	0.92 sec
✓	#4	1 min 38 sec	1 sec
✗	#3	2 min 38 sec	0.55 sec
✗	#2	3 min 38 sec	0.51 sec
✗	#1	4 min 38 sec	0.95 sec

Builds — Today

✓	#5	12:50 PM
✓	#4	12:49 PM
✗	#3	12:48 PM
✗	#2	12:47 PM
✗	#1	12:46 PM



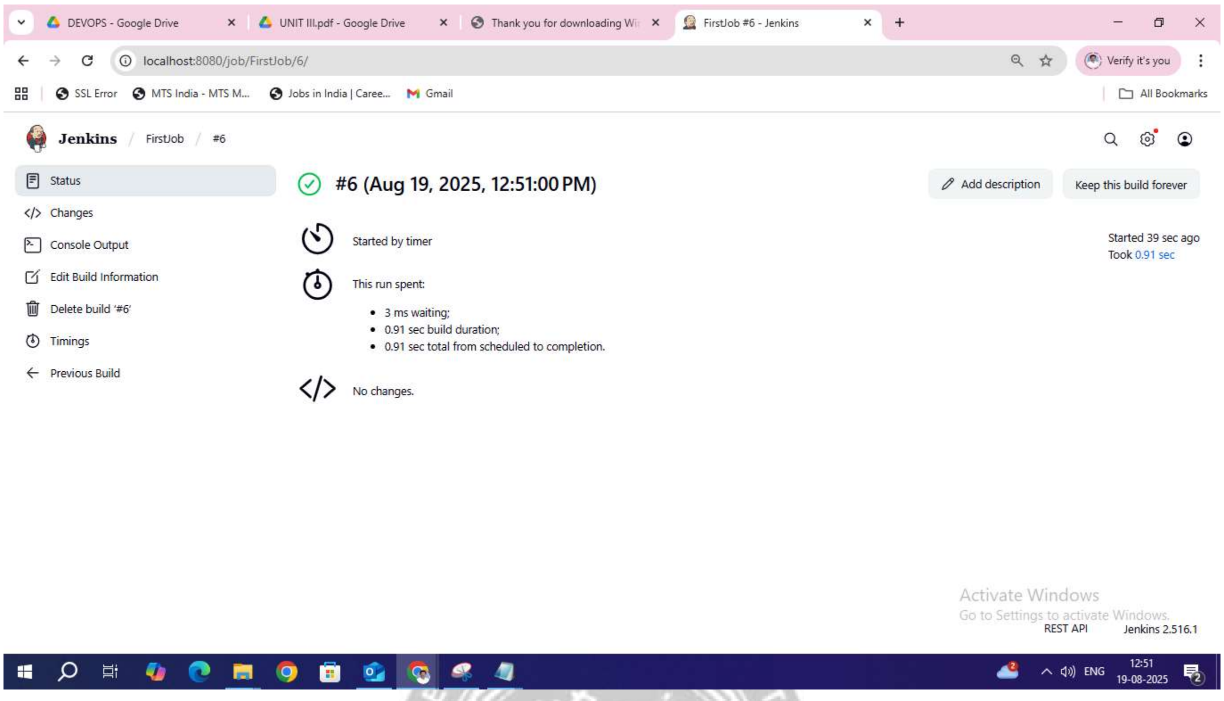
Jenkins / FirstJob / Build Time Trend

Build Time Trend

S	Build ↑	Time Since	Duration
✓	#6	57 sec	0.91 sec
✓	#5	1 min 57 sec	0.92 sec
✓	#4	2 min 57 sec	1 sec
✗	#3	3 min 57 sec	0.55 sec
✗	#2	4 min 57 sec	0.51 sec
✗	#1	5 min 57 sec	0.95 sec

Builds — Today

✓	#6	12:51 PM
✓	#5	12:50 PM
✓	#4	12:49 PM
✗	#3	12:48 PM
✗	#2	12:47 PM



#6 (Aug 19, 2025, 12:51:00 PM)

Started by timer

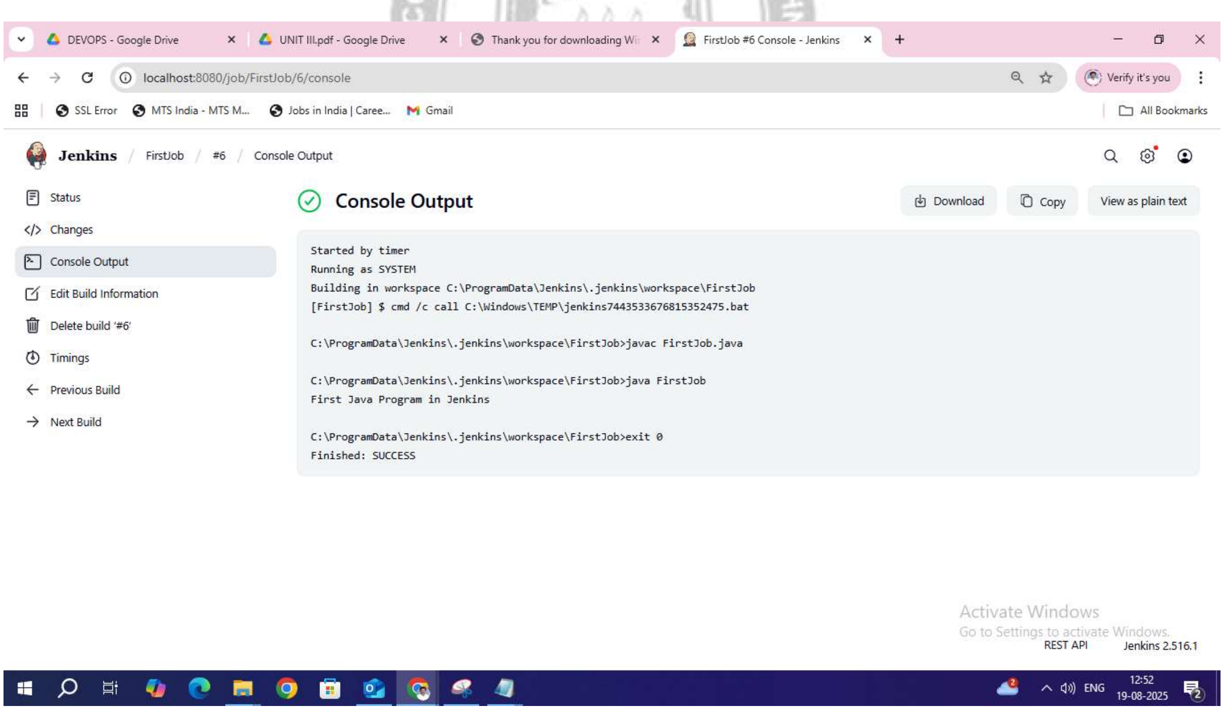
Started 39 sec ago
Took 0.91 sec

This run spent:

- 3 ms waiting;
- 0.91 sec build duration;
- 0.91 sec total from scheduled to completion.

No changes.

Step 8: Check the output in Console Output



Console Output

```
Started by timer
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\FirstJob
[FirstJob] $ cmd /c call C:\Windows\TEMP\jenkins7443533676815352475.bat

C:\ProgramData\Jenkins\.jenkins\workspace\FirstJob>javac FirstJob.java

C:\ProgramData\Jenkins\.jenkins\workspace\FirstJob>java FirstJob
First Java Program in Jenkins

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