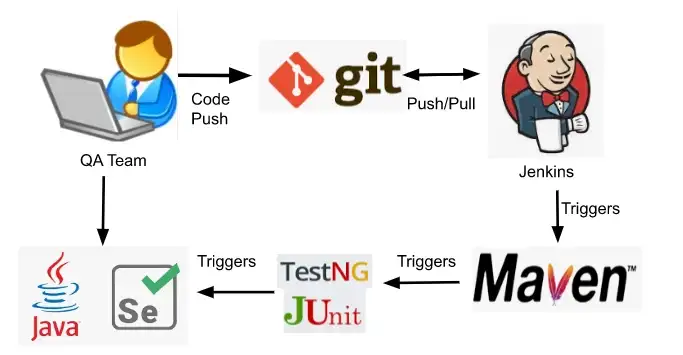
**Maven Integration with Jenkins**



**Introduction**

Inmodern software development, automation plays a crucial role in enhancing productivity and ensuring the quality of deliverables. Maven and Jenkins stand out as indispensable tools in this landscape. Maven simplifies build processes and dependency management for Java projects, while Jenkins automates continuous integration and delivery workflows. Integrating Maven with Jenkins offers a powerful solution for automating builds, testing, and deployment, streamlining the software development lifecycle.

**Overview**

Maven, with its Project Object Model (POM) and convention over configuration principles, provides a standardized approach to building Java projects. It manages dependencies, compiles source code, and packages applications efficiently.

On the other hand, Jenkins acts as an automation server, orchestrating continuous integration and delivery pipelines. It integrates seamlessly with version control systems and offers a wide range of plugins for customizing workflows.

**Maven**

Maven is a powerful build automation tool primarily used for Java projects. It simplifies the build process by managing project dependencies, compiling source code, and packaging applications.

Maven utilizes a Project Object Model (POM) file to define project configuration and dependencies. It follows convention over configuration principles, providing a standardized directory structure and build lifecycle.

Maven integrates with repositories to automatically download dependencies, reducing manual effort and ensuring project consistency. With its extensive plugin ecosystem, Maven supports various tasks such as testing, documentation generation, and deployment, making it an essential tool for Java developers to streamline their development workflow.

**POM**

The pom.xml file serves as the cornerstone of Maven projects, encapsulating project metadata, configuration, and dependencies.

It defines the project’s structure, including its group ID, artifact ID, version, and packaging type. Additionally, the pom.xml file specifies project dependencies, facilitating automatic dependency resolution during builds.

Maven relies on the information contained within the pom.xml file to orchestrate the build process, ensuring consistency and reproducibility across development environments. Through its declarative nature, the pom.xml file streamlines project management and enhances collaboration by providing a centralized source of project configuration.

**Jenkins**

Jenkins is an open-source automation server used for continuous integration and continuous delivery (CI/CD). It provides a web-based interface for automating various stages of the software development lifecycle, including building, testing, and deploying applications.

Jenkins supports integration with version control systems like Git, enabling automatic triggering of builds upon code commits. It offers a wide range of plugins for integrating with different build tools, testing frameworks, and deployment platforms.

Jenkins facilitates collaboration among development teams by providing visibility into build results and enabling automated testing. Its scalability and flexibility make it suitable for projects of all sizes, from small teams to large enterprises, enhancing productivity and reducing time to market.

**Integration**

Maven and Jenkins integration enables automated builds and continuous integration workflows for Java projects. Jenkins can be configured to trigger Maven builds based on code changes, ensuring that applications are built consistently and deployed promptly.

Jenkins plugins facilitate seamless integration with Maven, allowing users to configure build pipelines, schedule builds, and monitor build statuses. By integrating Maven and Jenkins, development teams can automate repetitive tasks, improve code quality through automated testing, and accelerate the software delivery process.

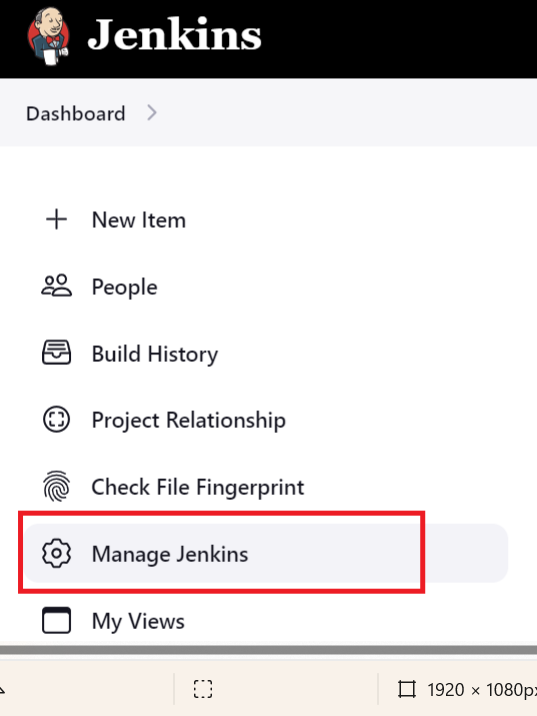
**How To Integrate Maven & Jenkins ?**

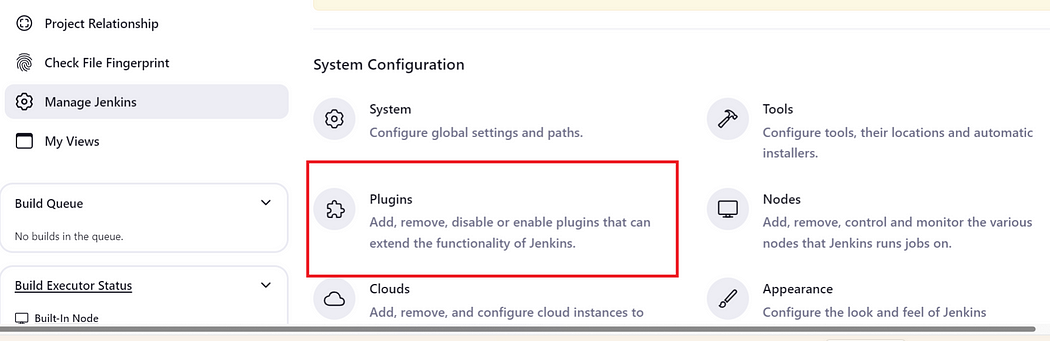
**Practical Part:**

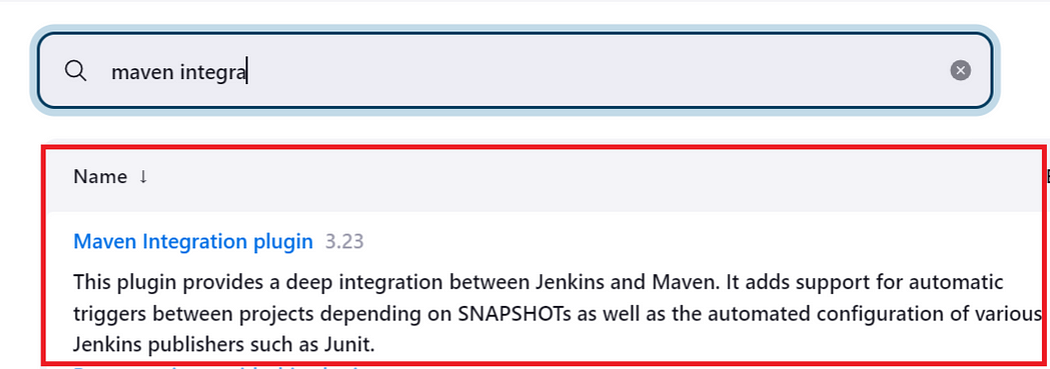
To integrate Maven with Jenkins, start by installing the Jenkins server and the necessary plugins, including the Maven Integration plugin.

**1. Creating Jenkins Server and installing required plugin.**

→ Then install ‘Maven Integration’ plugin in jenkins.

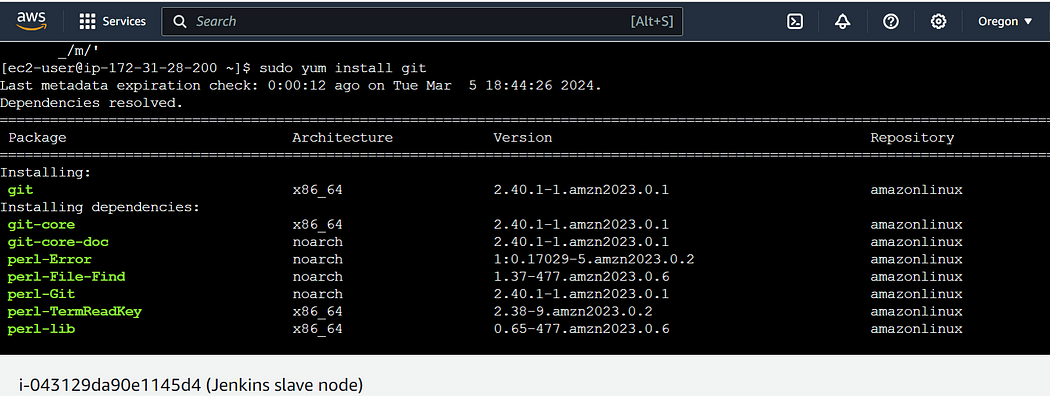


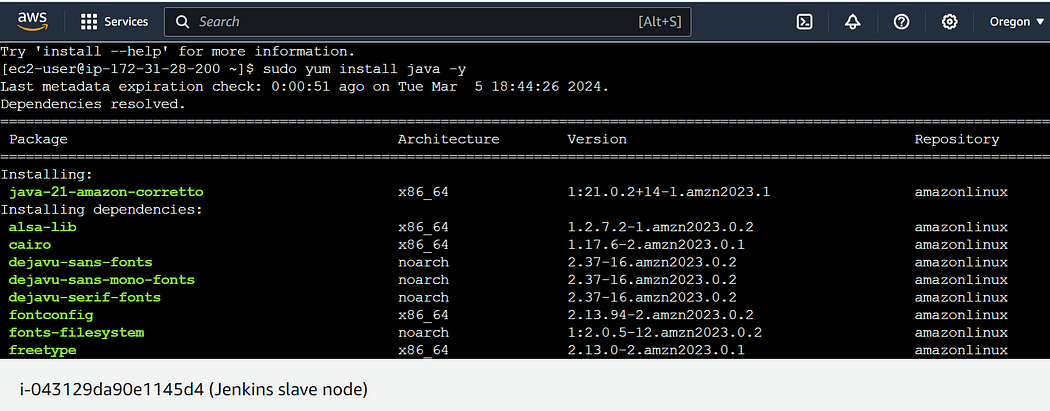


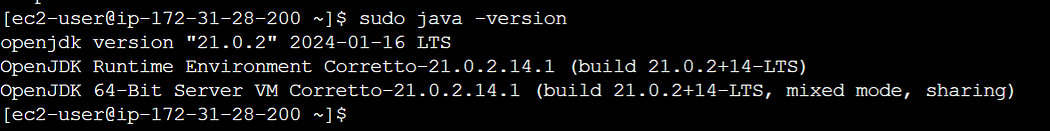


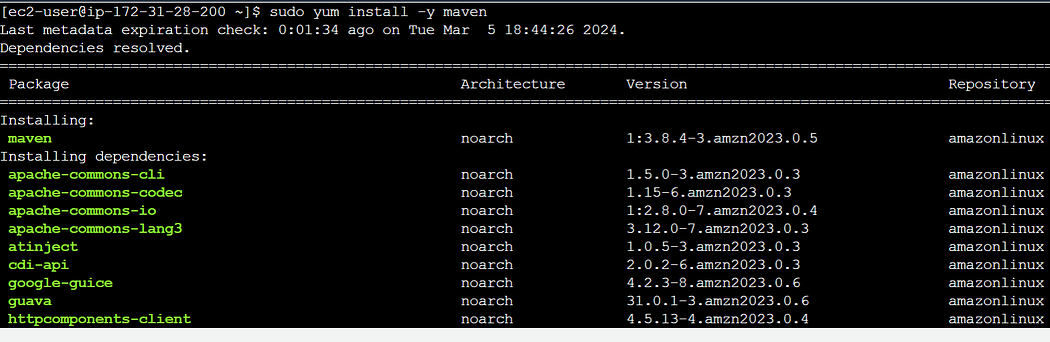
→ Install this plugin and restart the jenkins.

**2. Setting up Jenkins-Agent for executing maven.**

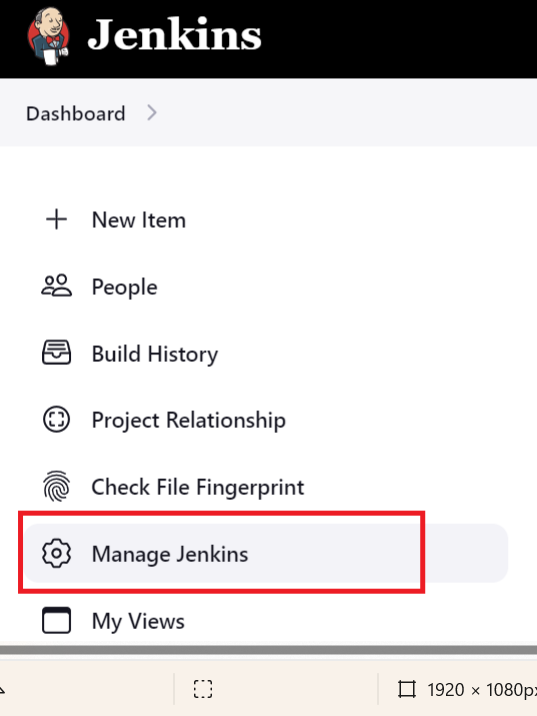


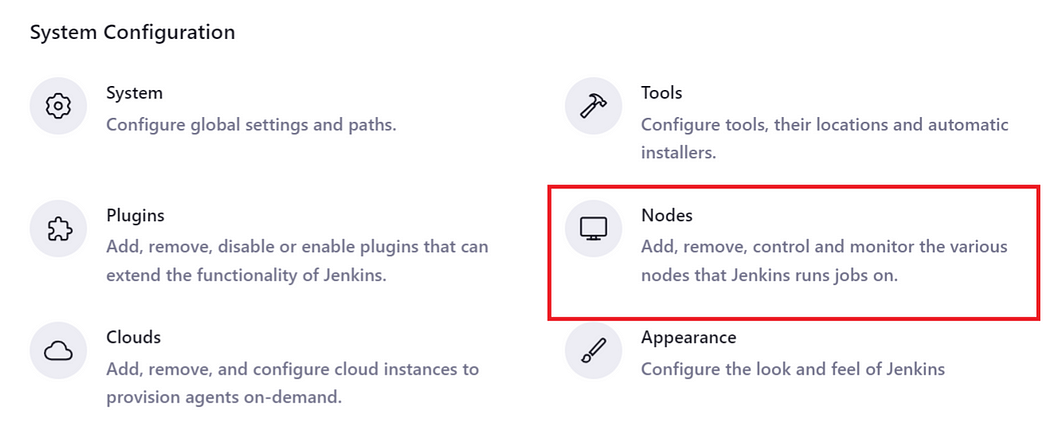




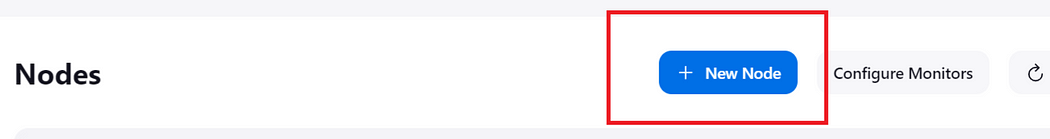


→ After installing these required softwares, we will add and launch this agent in jenkins.

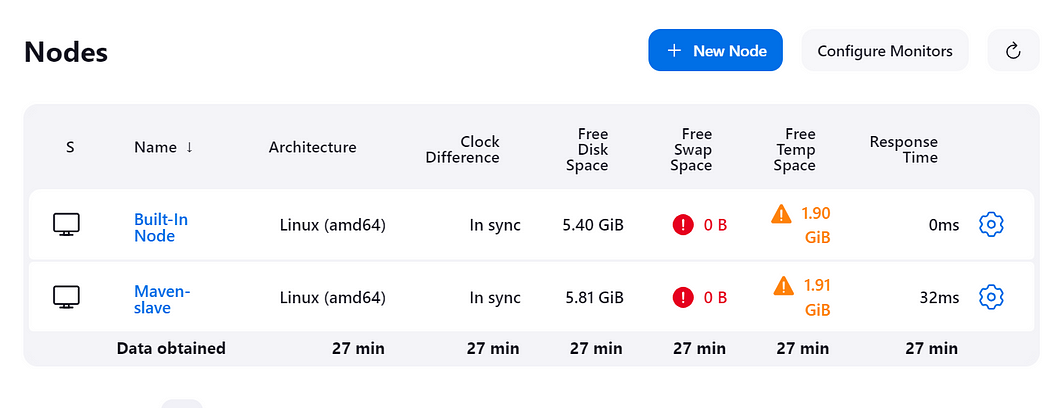




Then click on “new node” and setup the node.

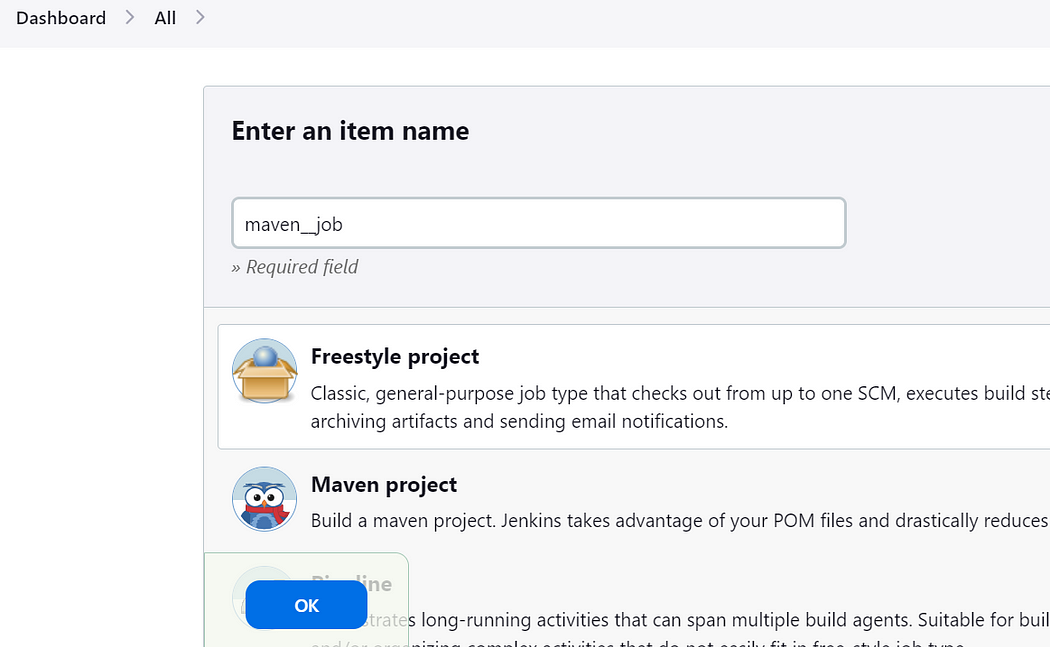


**Your set-up is going to look like this :**



**3. Creating a Job for maven**

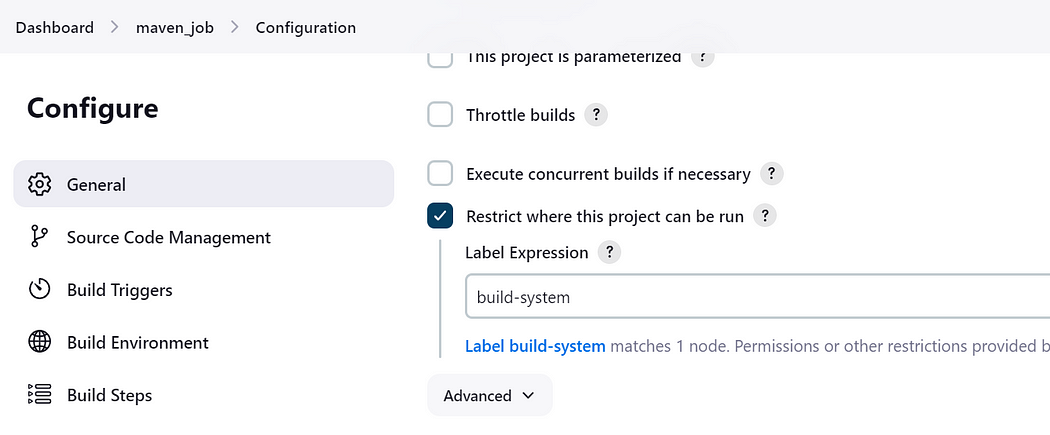
→ Click on the new-item and write the name of this job & choose Freestyle project.



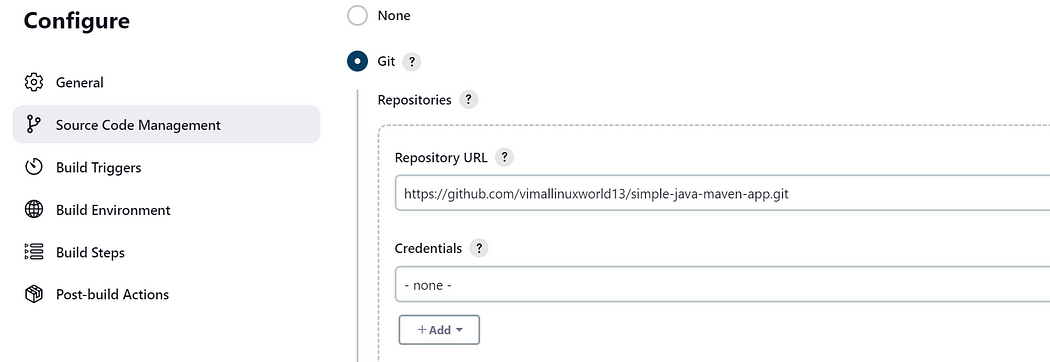
→ Add the project means github link where you have pom.xml file, your code and everything.

Note : Use this link for project and github url : <https://github.com/vimallinuxworld13/simple-java-maven-app.git>

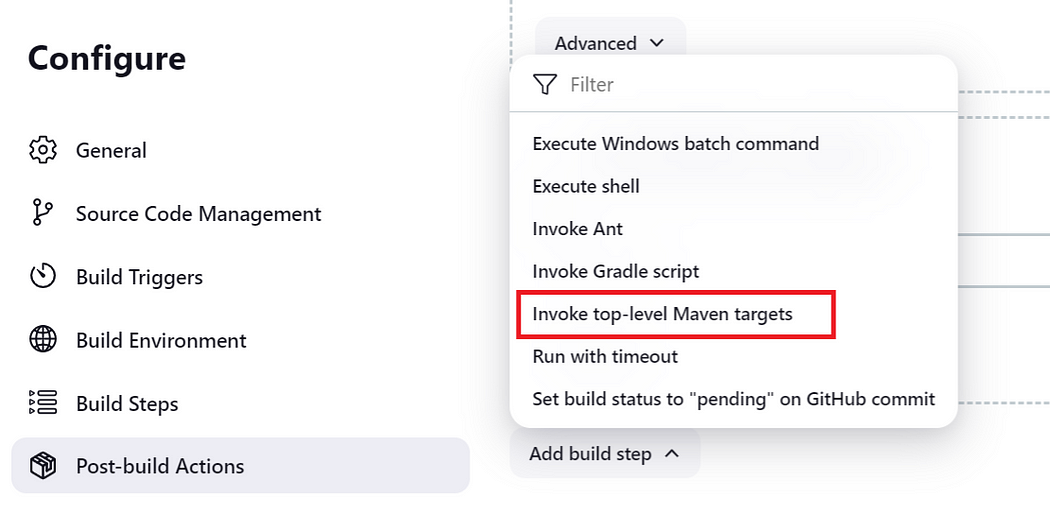
→ Restrict this job to your agent only.



→ Then add your github url in the source code management section :

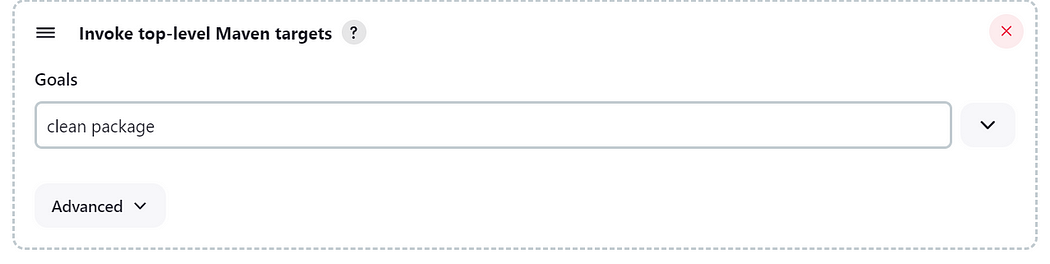


→ Click on Build Steps option and choose **Invoke top-level Maven targets**



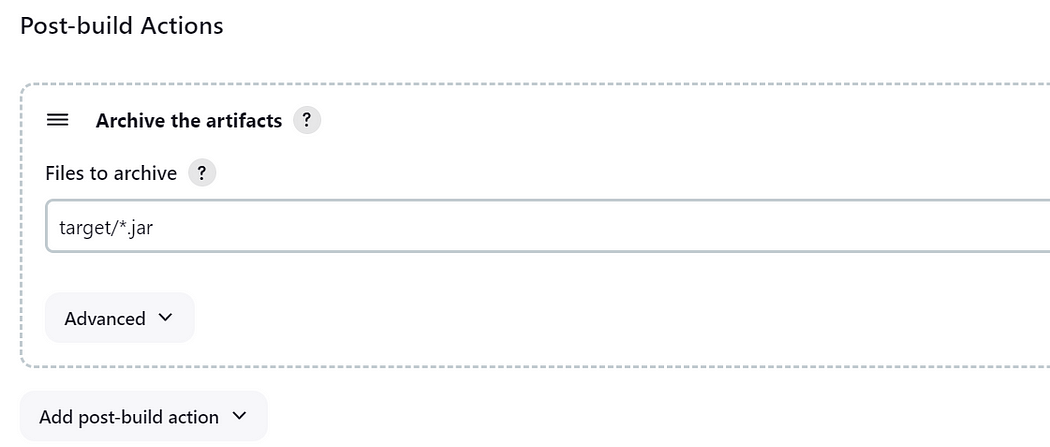
→ Below Goal will first clean the previous packages or builds and create the new one on every building of this job.

→ “mvn package” is a command that build, test and packaged the code in one go. That’s why we used this here.

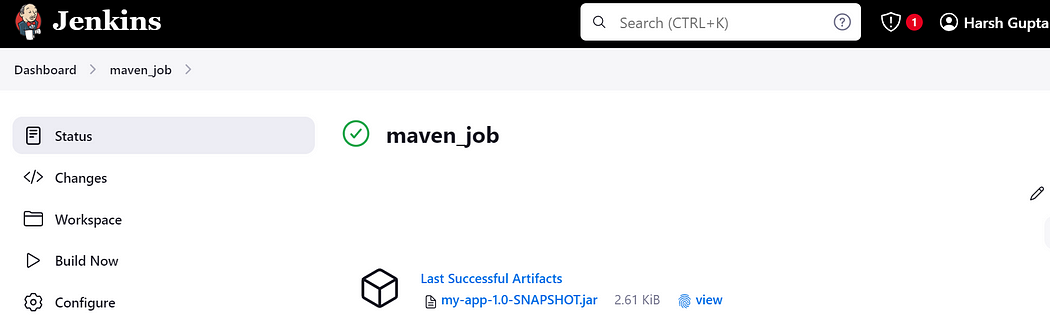


→ Now after creating the package, we have to store it on master node as its not a good practice to store the package locally on agent.

→ For this we have a option “**Archive the artifacts”**



After creating the job, now start building it:



Click on this artifact, and you will get the download link.

**And now we can create further job to deploy it on real production environment on any server like tomcat, and further making use of docker or Kubernetes to scale it. But our main goal is to show that how to integrate maven with Jenkins and congratulation we have successfully integrate both the resources.**

**Conclusion**

The integration of Maven with Jenkins empowers development teams to automate build processes, streamline workflows, and accelerate software delivery. By leveraging the strengths of these two tools, organizations can achieve greater efficiency, consistency, and reliability in their software development practices. Embracing this integration fosters a culture of automation and continuous improvement, driving success in today’s fast-paced development environments.