Jenkins for DevOps

Jenkins is a crucial tool for DevOps engineers, allowing them to automate repetitive tasks, maintain code quality, facilitate CI/CD, and manage the deployment process efficiently. DevOps engineers use Jenkins as part of their daily routine to ensure smooth software development and delivery workflows.

Top 5 Jenkins Use cases for DevOps Engineer

1. **Continuous Integration (CI)**:
Jenkins automatically builds and tests code changes as soon as they are committed to the version control repository. It helps catch integration issues early, maintain code quality, and foster collaboration among developers.

2. **Continuous Delivery (CD):**
Jenkins automates the process of deploying applications to different environments, such as staging and production. This ensures that code changes are reliably and consistently delivered to users.

3. **Automated Testing:**
Jenkins runs various automated tests, including unit tests, integration tests, and end-to-end tests. This guarantees the stability and reliability of the software before it is deployed to production.

4. **Scheduled Jobs and Cron Jobs:**
Jenkins can execute scheduled tasks or jobs at specific times or on a recurring basis using cron expressions. These tasks can include maintenance activities, backups, and other routine operations.

5. **Infrastructure as Code (IaC) and Deployment Orchestration:** Jenkins can integrate with infrastructure automation tools like Terraform or Ansible to automate the provisioning and management of infrastructure. This ensures that the environment is consistent and reproducible.

Day to Day activities performed using Jenkins by DevOps engineer

Pipeline Maintenance:
DevOps engineers manage Jenkins pipelines, ensuring they are up-to-date, efficient, and reliable. They might modify the pipeline stages, steps, and conditions based on changing requirements.

Troubleshooting and Debugging:
DevOps engineers monitor Jenkins jobs for any failures or errors. When an issue arises, they investigate the root cause, resolve the problem, and optimize the pipeline for better performance.

Plugin Management:
Jenkins offers a wide range of plugins to extend its functionality. DevOps engineers review and manage these plugins to incorporate new features or address security vulnerabilities.

Security and Access Control:
DevOps engineers configure security settings in Jenkins, defining user access levels, ensuring sensitive information is protected, and implementing best practices for secure CI/CD workflows.

Performance Optimization:
DevOps engineers continuously monitor Jenkins server performance, identify bottlenecks, and optimize resource utilization to maintain a smooth and responsive Jenkins environment.

Integrations:
Jenkins often needs to integrate with various tools and services used in the development and deployment process. DevOps engineers set up these integrations, which can include version control systems, issue trackers, testing frameworks, and deployment platforms.

Scalability and High Availability:
For larger projects and organizations, DevOps engineers work on scaling Jenkins and setting up a highly available infrastructure to ensure continuous operation and avoid single points of failure.

Continuous Integration

CI in Jenkins can be thought of as a set of stages that automatically handle the process of building, testing, and verifying code changes as they are added to the shared repository. Each stage serves a specific purpose and contributes to ensuring the code's health and stability.

Common stages of Continuous Integration in Jenkins:

Checkout Stage: This stage pulls the latest code changes from the version control repository (e.g., Git) to Jenkins workspace. It ensures that Jenkins has the latest code to work with.

Build Stage: In
this stage, Jenkins compiles the code and creates executable files, libraries, or other artifacts necessary for the application.The build process ensures that the code can be turned into a working application.

Test Stage: Once
the build is complete, Jenkins runs various tests, such as unit tests, integration tests, or functional tests, to verify that the code changes didn't introduce any bugs or errors. The tests help ensure the code's quality and functionality.

Code Analysis Stage: This
stage involves running static code analysis tools (e.g., SonarQube) to identify code quality issues, potential vulnerabilities, or areas for improvement. It ensures that the code adheres to best practices and coding standards.

Artifact Generation Stage: After a successful build and tests, Jenkins generates and stores artifacts (e.g., deploy-able packages, Docker images) that can be used for deployment to different environments.

Deployment Stage: If required, Jenkins can deploy the tested and validated code to various environments, such as development, staging, or production. Deployment automation ensures consistency across environments.

Notification Stage: At the end of the CI process, Jenkins can send notifications to relevant stakeholders, such as developers, testers, or project managers, informing them about the build and test results.

These stages can be customized and extended based on the specific requirements of the project. Jenkins provides a flexible and easy-to-use interface for configuring and orchestrating these stages in the CI pipeline.

Jenkins Master Node Configuration

Step 1: Install Java

#sudo apt update
#sudo apt install openjdk-11-jdk
#java --version

Step 2: Add Jenkins repository

```
#curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key |sudo tee
/usr/share/keyrings/jenkins-keyring.asc > /dev/null

#echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian-stable binary/ |
sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

#sudo apt update
```

Step 3: Install Jenkins

#sudo apt install Jenkins

Step 4: Start Jenkins

```
#sudo systemctl start Jenkins
#sudo systemctl status jenkins
```

Step 5: Configure firewall

Allow port 8080 in the azure portal under networking

Once this steps are executed in server, then get server public ip address and login to jenkins dashboard weburl .

Also one user with name jenkins will be added by default with installation , we must add this user to sudoers file.

Step 6: Set up Jenkins using public IP address of jenkins master server

To set up Jenkins, go to your browser and open http://publicIP:8080 where Jenkins server is running.

The Unlock Jenkins screen will appear as shown below, asking for the administrator password. read the file "sudo cat /var/lib/jenkins/secrets/initialAdminPassword" this is the inital password to login to jenkins

Jenkins Terms

Work Space: Jenkins workspace is a folder where jenkins stores all of its configuration. Default workspace is /var/lob/jenkins. If you want to change the default , we need work on environmental variable workspace.

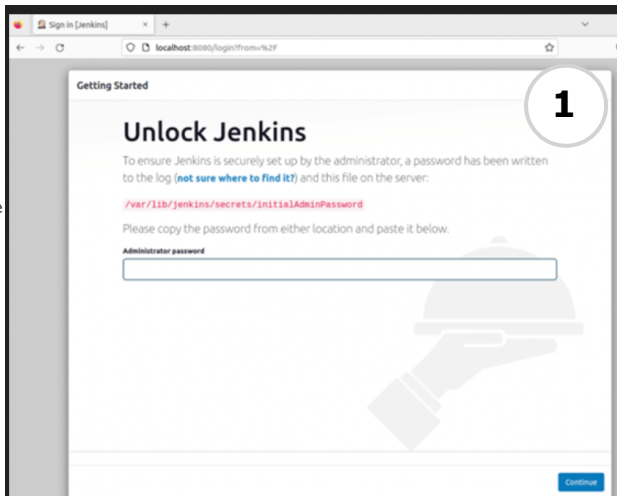
BackUp of jenkins is backup of workspace.

Project: This contains the activity that needs to be performed on triggers and we have different types of projects like Freestyle

Node: This represents the machine on which build can be executed and we can multiple nodes, Also we can configure each node to handle multiple build by executors.

Build: This represents the execution of project and every build for a project has a running number called as build id.

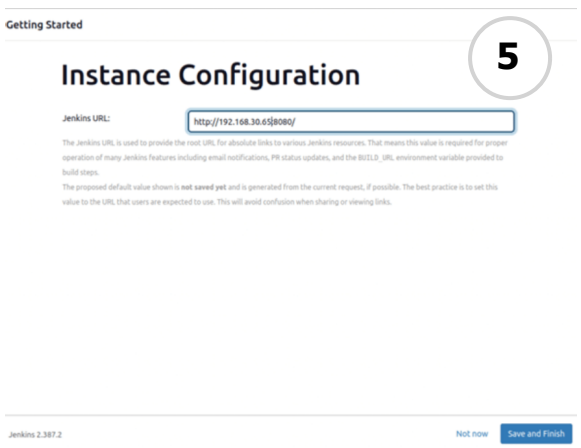
Jenkins Installation

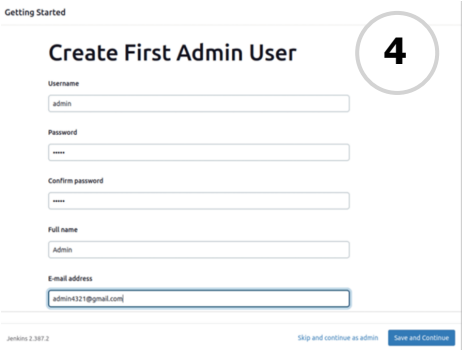


Once default initial password is provided . you will be presented with customize jenkins page

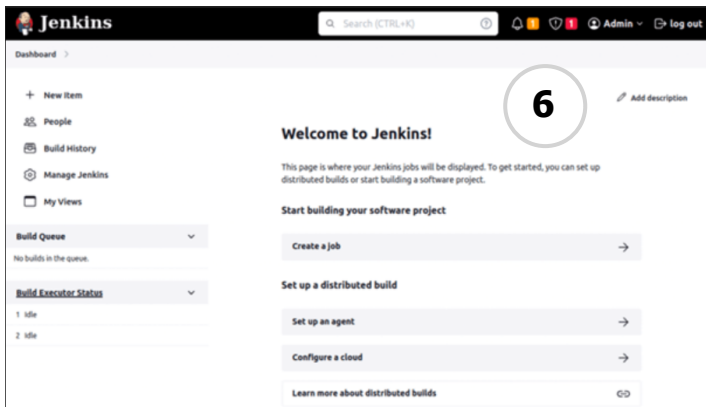
You need to customize Jenkins and install plugins

In the next step, you need to mention the Jenkins url where the Jenkins instance will get configured to run. Enter a domain name or ip address with port 8080 and click Save and Finish.



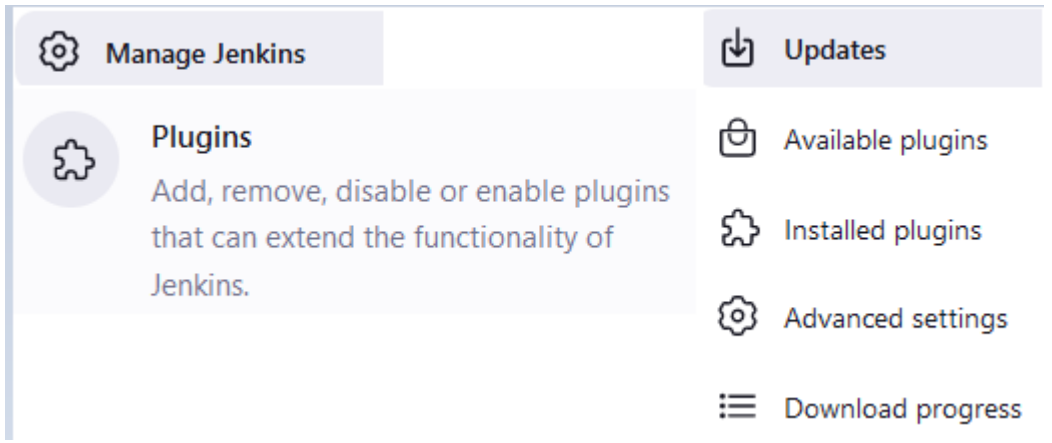


You will be prompted to create your first admin user after installing the plugins. After completing the required fields, click "Save and continue." You also have the option to skip this step and continue as Admin.



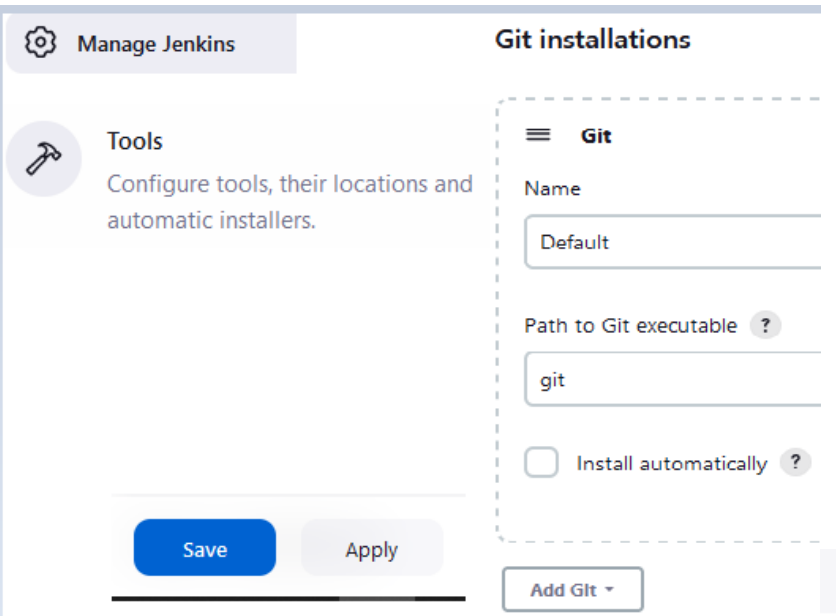
Finally, Jenkins dashboard will open where you can start creating your pipeline jobs.

Integrate Github to jenkins using plugins

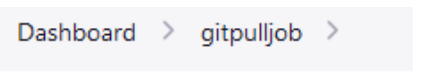


1. Jenkins Dashboard search for Manage Jenkins
2. Under Manage Jenkins , search for Plugins
3. Under Plugins tab go to Available Plugins and search github ans install it .
4. Once installed you can see this under installed plugins tab

Configure github to jerkins

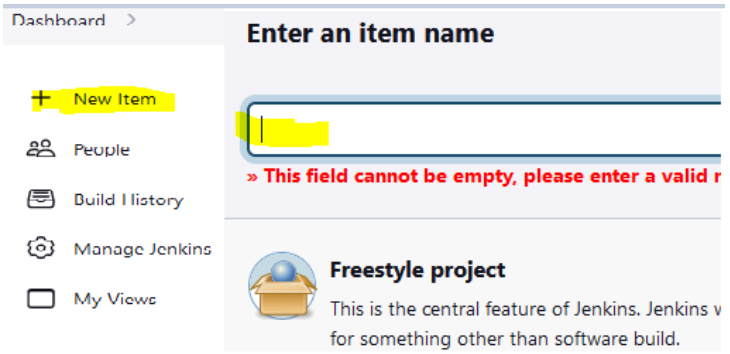


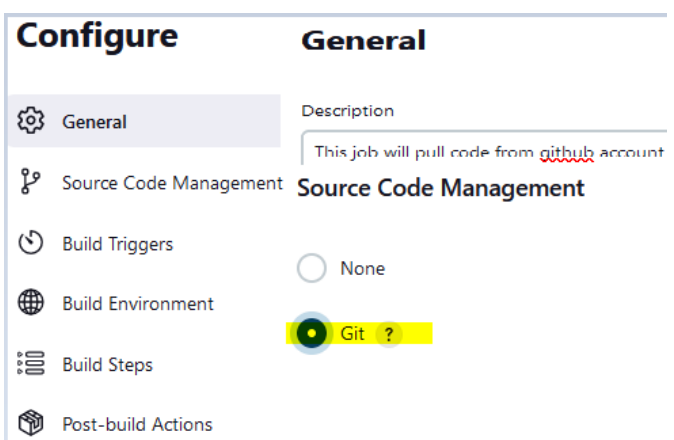
1. Jenkins Dashboard search for Manage Jenkins
2. Under Manage Jenkins , search for Tools .
3. Once you select tools tab, you will have page where you can see git installation section
4. Under this section you need to specify Name, Path to git executable
5. Finally Apply and save the configuration



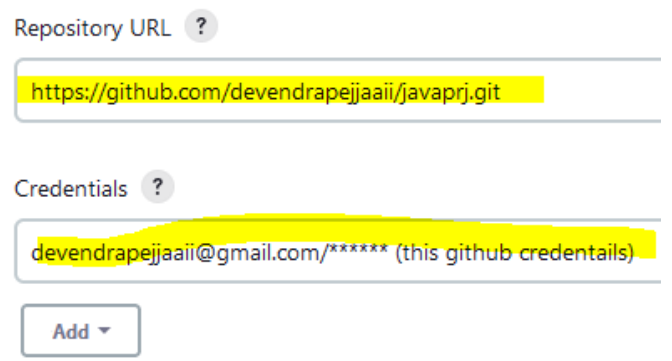
Run a Sample Jenkins job to pull code from Github

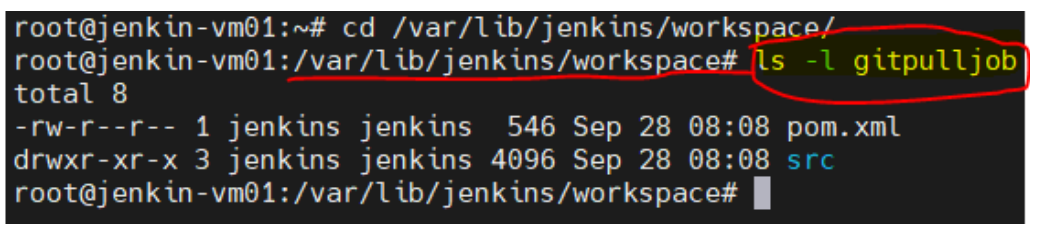
1. To run a simple job we will select New Item under dashboard
2. Now select Freestyle project and give name of job .
3. Next you will have the options to Configure you job as required





4.Next you have to provide the credentials for github account

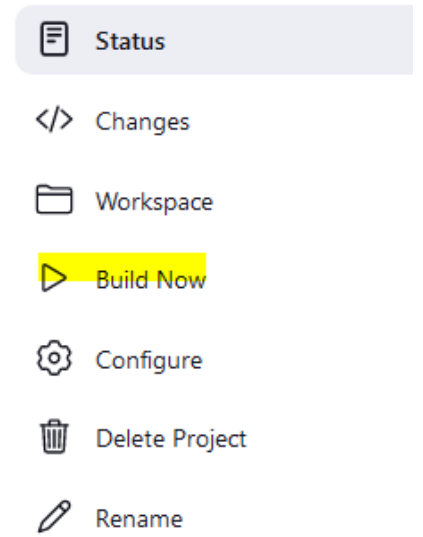




5. Once Every thing is done , now you will have option call Build Now , click on it and your job will run.

6. The purpose of this job is to pull code from you github account to jenkins master node , and the location where it store the code is "/var/lib /jenkins/workspace" under this directory you will find your code with name mentioned in job name.

7. Same you can verify in server as show in below screenshot



Integrate Maven to Jenkins

1.Setup Maven on Jenkins Server

1. download Apache maven from its official website or use following command to download Apache Maven 3.8.6 on your system.

2. Download the tar file using wget command to jenkisn master server

```
wget https://dlcdn.apache.org/maven/maven-3/3.8.6/
binaries/apache-maven-3.8.6-bin.tar.gz
```

3. Now extract the downloaded Maven archive file using the following command.

```
sudo tar xzf apache-maven-3.8.6-bin.tar.gz -C /opt
sudo ln -s /opt/apache-maven-3.8.6 /opt/maven
```

The above commands will extract Maven under the **/opt** directory and create a symbolic link **/opt/maven** to that directory.

4. set the environment variables by creating a new file **/etc/profile.d /maven.sh**.

```
export JAVA_HOME=/usr/lib/jvm/default-
java
export M2_HOME=/opt/maven
export MAVEN_HOME=/opt/maven
export PATH=${M2_HOME}/bin:${PATH}
```

5.Next load the environment variables in the current shell using the following command.

```
source /etc/profile.d/maven.sh
```

During the next system reboot, the environment will automatically load.

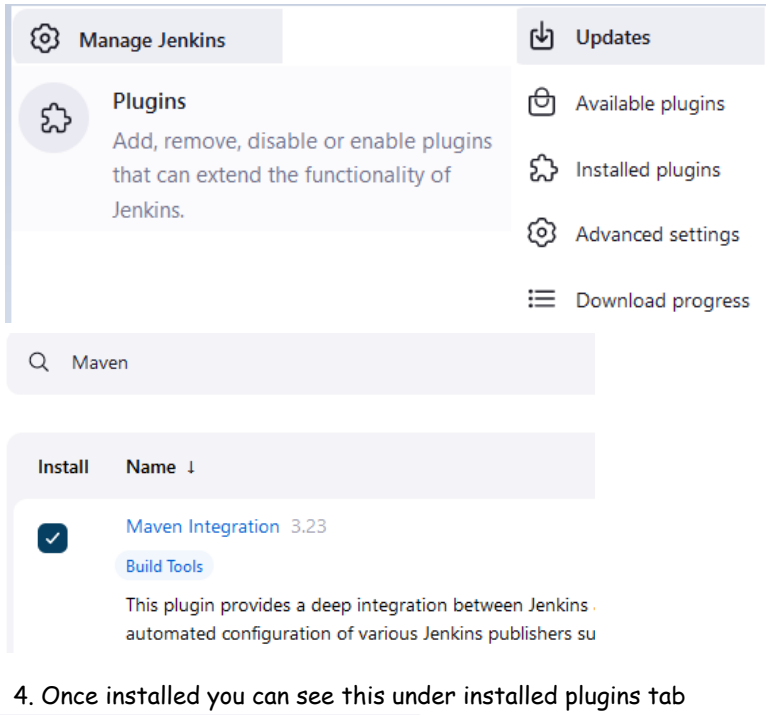
6.check the installed Maven version with the following command:

```
mvn -version
```

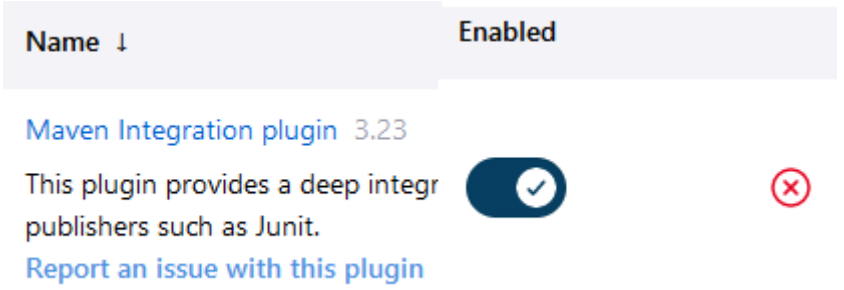
```
root@jenkin-vm01:~# mvn -v
Apache Maven 3.6.3
Maven home: /usr/share/maven
Java version: 11.0.20.1, vendor:
Default locale: en, platform enc
OS name: "linux", version: "6.2.
root@jenkin-vm01:~#
```

2.Install Maven Plugin

1. Jenkins Dashboard search for Manage Jenkins
2. Under Manage Jenkins , search for Plugins
3. Under Plugins tab go to Available Plugins and search maven and install it .

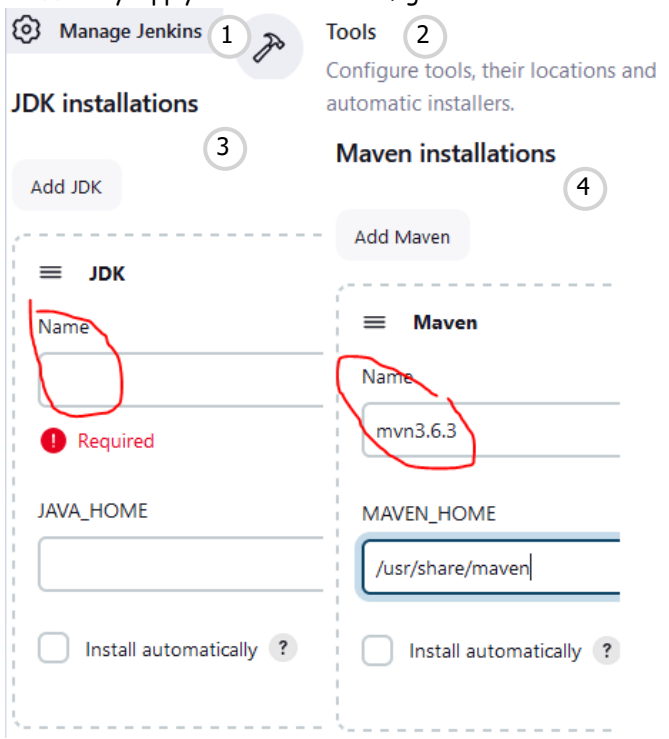


4. Once installed you can see this under installed plugins tab

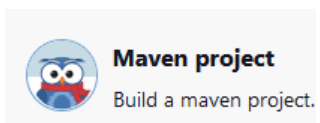


3.Configure Maven & Java

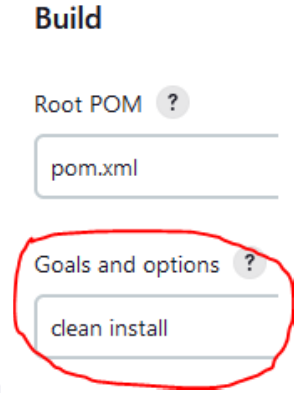
1. In Jenkins Dashboard search for Manage Jenkins
2. Under Manage Jenkins , search for Tools .
3. Once you select tools tab, you will have page where you can see JDK & maven installation section
4. Under this section you need to specify Name, Path to JDK & maven executable
5. Finally Apply and save the configuration



5. Now Create a build job using java & maven, For this again you have to go to dashboard and select new Item . Since we have installed maven , now you will be able to select the maven project.



6. Main part in maven based projects is to set the Root POM & Goals and options , for this visit offical maven page to see the build life cycle



7.Once done, now you can execute the build job for simple java project using maven and produce artifacts