



# CI/CD Pipeline Using Jenkins Unleashed

Solutions While Setting Up  
CI/CD Processes

---

Pranoday Pramod Dingare

Apress®

# **CI/CD Pipeline Using Jenkins Unleashed**

Solutions While Setting  
Up CI/CD Processes

Pranoday Pramod Dingare

Apress®

## **CI/CD Pipeline Using Jenkins Unleashed**

Pranoday Pramod Dingare  
Pune, Maharashtra, India

ISBN-13 (pbk): 978-1-4842-7507-8  
<https://doi.org/10.1007/978-1-4842-7508-5>

ISBN-13 (electronic): 978-1-4842-7508-5

### **Copyright © 2022 by Pranoday Pramod Dingare**

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Spandana Chatterjee  
Development Editor: Laura Berendson  
Coordinating Editor: Divya Modi  
Copy Editor: Kezia Endsley

Cover designed by eStudioCalamar

Cover image designed by Freepik ([www.freepik.com](http://www.freepik.com))

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [booktranslations@springernature.com](mailto:booktranslations@springernature.com); for reprint, paperback, or audio rights, please e-mail [bookpermissions@springernature.com](mailto:bookpermissions@springernature.com).

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at [www.apress.com/https://github.com/Apress/CI-CD-Pipeline-Using-Jenkins](http://www.apress.com/https://github.com/Apress/CI-CD-Pipeline-Using-Jenkins). For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*I would like to dedicate this book to my parents, to my lovely son, to my wife, and to my role models from the Dingare family, for being my eternal sources of inspiration*

# Table of Contents

<b>About the Author .....</b>	<b>xix</b>
<b>About the Technical Reviewer .....</b>	<b>xi</b>
<b>Acknowledgments .....</b>	<b>xxiii</b>
<b>Introduction .....</b>	<b>XXV</b>
<b>Chapter 1: Understanding CI/CD .....</b>	<b>1</b>
The Development Workflow .....	2
Running Unit Tests Locally.....	2
Pushing and Merging Code to the Central Repository.....	3
Compiling Code after the Merge.....	3
Running Tests on the Merged Code .....	3
Deploying an Artifact .....	3
Continuous Delivery/Continuous Deployment.....	3
CI/CD Workflow Case Study .....	4
Pulling the Application's Latest Code .....	4
Developing and Running Unit Test Cases .....	4
Code Development .....	5
Rerunning the Unit Test Case .....	6
Pushing and Merging the Code to the Central Repository.....	6
Compiling the Code after the Merge.....	6

## TABLE OF CONTENTS

Running Tests on the Merged Code.....	6
Deploying an Artifact .....	6
Run e-e Tests on a Deployed Application .....	7
Summary.....	7
<b>Chapter 2: Introducing Jenkins .....</b>	<b>9</b>
What Is Jenkins?.....	10
A History of Jenkins.....	10
Implementing CI/CD Using Jenkins .....	11
The Jenkins Architecture.....	12
Summary.....	14
<b>Chapter 3: Installing Jenkins.....</b>	<b>15</b>
Installing Jenkins on Windows.....	15
Hardware/Software Requirements .....	15
Different Ways to Install Jenkins on Windows.....	16
Understanding the Configuration Files and Directory Structure of Jenkins .....	17
Understanding Important Settings in Jenkins.xml .....	18
Summary.....	20
<b>Chapter 4: Configuring Jenkins .....</b>	<b>21</b>
Configuring Global Settings and Paths.....	21
Signing into Jenkins .....	21
Understanding the Configure System Options.....	24
Resetting the Username and Password in Jenkins .....	27
Adding a New User .....	31
Summary.....	32

## TABLE OF CONTENTS

<b>Chapter 5: Managing Plugins in Jenkins .....</b>	<b>33</b>
What Are Plugins .....	33
Commonly Used Jenkins Plugins .....	34
Installing the Plugins in Jenkins.....	35
Understanding the Plugin Manager .....	36
Troubleshooting Installation Problems .....	40
Problem 1 .....	40
Problem 2 .....	42
Summary.....	44
<b>Chapter 6: Understanding the Global Tool Configuration Page .....</b>	<b>45</b>
Global Tool Configuration Settings .....	45
Understanding the Global Tool Configuration Settings .....	47
Summary.....	54
<b>Chapter 7: Managing Security with Jenkins .....</b>	<b>55</b>
Configuring Global Security in Jenkins .....	55
Configuring LDAP with Jenkins .....	64
Why We Need to Configure LDAP with Jenkins .....	64
How to Configure LDAP with Jenkins .....	66
Summary.....	67
<b>Chapter 8: Managing Credentials .....</b>	<b>69</b>
Understanding Credentials in Jenkins .....	69
Creating Credential Entries in Jenkins.....	70
Understanding Scope and Domains .....	71
Creating Credential Entries in Jenkins .....	72
Creating a Credentials Entry in a Global Domain (Default Domain) and a Global Scope.....	72

## TABLE OF CONTENTS

Updating a Credentials Entry .....	76
Creating a Credentials Entry in a Particular Domain .....	79
Configuring a Credentials Provider .....	84
Summary.....	89
<b>Chapter 9: Managing Users .....</b>	<b>91</b>
Creating Users in Jenkins .....	91
Assigning Roles to Users in Jenkins .....	93
Installing the Role-Based Authorization Strategy Plugin .....	93
Enabling Role-Based Strategy in Jenkins.....	96
Creating User Roles in Jenkins.....	97
Assigning Roles to Users in Jenkins.....	99
Checking the Assignment of a Role to a User.....	101
Creating Project-Based Roles in Jenkins.....	101
Assigning Project-Based Roles to Users .....	102
Verifying the Assignment of the Project-Based Role to the User.....	104
Understanding Matrix-Based Security in Jenkins.....	106
Understanding the Project-Based Matrix Authorization Strategy in Jenkins ....	109
Summary.....	113
<b>Chapter 10: Understanding Jobs in Jenkins.....</b>	<b>115</b>
What Is a Job in Jenkins? .....	115
What Is a Build in Jenkins?.....	116
What Is a Free-Style Job in Jenkins?.....	116
How to Create a Job in Jenkins .....	117
How to Configure a Job in Jenkins .....	118
This Project Is Parameterized.....	120
Disabling a Project.....	122
Executing Concurrent Builds .....	122

## TABLE OF CONTENTS

<b>Quiet Period .....</b>	<b>123</b>
<b>Block Build When Upstream Project Is Building .....</b>	<b>124</b>
<b>Block Build When Downstream Project Is Building .....</b>	<b>125</b>
<b>Use Custom Workspace .....</b>	<b>125</b>
<b>Display Name .....</b>	<b>125</b>
<b>Keep the Build Logs of Dependencies .....</b>	<b>126</b>
<b>Source Code Management .....</b>	<b>127</b>
<b>Branches to Build .....</b>	<b>133</b>
<b>Build Triggers.....</b>	<b>133</b>
<b>How to Run a Job in Jenkins and Check Its Output .....</b>	<b>138</b>
<b>How to Edit a Job in Jenkins.....</b>	<b>141</b>
<b>How to View a Job's Workspace .....</b>	<b>143</b>
<b>How to Clear a Job's Workspace.....</b>	<b>144</b>
<b>How to Delete a Job.....</b>	<b>145</b>
<b>Summary.....</b>	<b>145</b>
<b>Chapter 11: Preparing a Java API Project Using Maven.....</b>	<b>147</b>
<b>Understanding the Maven Build Tool.....</b>	<b>148</b>
<b>    Typical Development Flow of a Java API Project .....</b>	<b>148</b>
<b>    How the Build Tool Helps Developers .....</b>	<b>150</b>
<b>    How to Use the Maven Build Tool to Build a Java API Project.....</b>	<b>150</b>
<b>    Understanding the Maven Project Directory Structure.....</b>	<b>159</b>
<b>    Understanding Java API Project Code Files .....</b>	<b>160</b>
<b>    Understanding the pom.xml File in the Java API Project.....</b>	<b>161</b>
<b>    How to Use Maven from the CLI .....</b>	<b>166</b>
<b>    Understanding Maven's settings.xml File.....</b>	<b>174</b>
<b>Summary.....</b>	<b>175</b>

## TABLE OF CONTENTS

<b>Chapter 12: Integrating Maven with the Nexus Repository and Creating Free-Style Jobs to Release the Java API on the Nexus Repository.....</b>	<b>177</b>
Understanding Git .....	178
Installing Git .....	180
Understanding GitHub/GitLab.....	180
Understanding End-End Use of Git for the API Project .....	180
Step 1: Creating a Local Repository .....	181
Step 2: Creating a Central Repository on GitLab.....	183
Step 3: Committing Code to the Local Repository .....	185
Step 4: Pushing the Code from the Local Repository to the Central Repository on GitLab .....	188
Step 5: Creating a Master Branch in the Central Repository on GitLab .....	189
Understanding the Nexus Repository.....	191
What is an Artifact? .....	191
What is the Nexus Repository?.....	191
Installing the Nexus Repository .....	191
How to Start the Nexus Repository System .....	192
Installing Nexus as a Service .....	193
Accessing the Nexus Repository Manager.....	194
Creating a Hosted Repository to Release the Artifacts .....	195
Integrating Maven and the Nexus Repository .....	195
Releasing the CalculatorAPI.jar File in the Nexus Repository.....	197
Creating a Jenkins Free-style Job to Release the CalculatorAPI.jar in the Nexus Repository .....	198
Step 1: Setting Up Maven in Jenkins.....	198
Step 2: Adding Git Repository Credentials to Jenkins .....	199
Step 3: Creating a Free-Style Job from the Jenkins Dashboard.....	199

## TABLE OF CONTENTS

Step 4: Add a Subtraction Function and Unit Test Cases to the API Project .....	200
Step 5: Commit Changes in the Local Repository and Push them to the Central GitLab Repository.....	202
Step 6: Merge the SubtractionFunction Branch with the Master Branch on the Central GitLab Repository.....	203
Running a Jenkins Free-Style Job to Perform a Release of the CalculatorAPI.jar in the Nexus Repository.....	204
Summary.....	206
<b>Chapter 13: Creating an Auto-Trigger Free-Style Job to Manage Java API Releases.....</b>	<b>207</b>
How to Add a New Code Contributor to a Private GitLab Repository .....	208
How to Invite a Team Member to the Code Repository.....	208
Understanding SSH Authentication.....	211
Why You Need SSH Authentication .....	211
How SSH Authentication Works with GitLab .....	212
Applying SSH Authentication to the Java API Code Repository.....	212
Step 1: Generating the Public and Private Keys .....	212
Step 2: Adding the Public Key to the JenkinsBookCalculatorAPI GitLab Repository .....	215
Accessing the JenkinsBookCalculatorAPI Repository Using SSH URL and Adding new Arithmetic Functions to the CalculatorAPI.....	216
Step 1: Cloning the JenkinsBookCalculatorAPI Repository.....	217
Step 2: Adding a Multiplication Function to the Calculator Class .....	221
Step 3: Adding Unit Test Cases for the Multiplication Function .....	221
Step 4: Changing the Version Number to 3.0 in pom.xml .....	224
Step 5: Unit Testing the Recent Multiplication Function and Regression Testing for the New Functionalities .....	224
Step 6: Committing New Functionality Changes in a Branch Named Multiplication Function in the Local Repository .....	225

## TABLE OF CONTENTS

Step 7: Pushing the MultiplicationFunction Branch to the Remote Repository .....	225
Step 8: Creating a Merge Request for this New Branch .....	226
Creating an Auto-Trigger Jenkins Job with Email Notification.....	227
Step 1: Creating a Free-Style Job from Jenkins Dashboard: Click the New Item link on Jenkins Dashboard .....	227
Setting Up Jenkins to Send Email Notifications.....	229
Triggering the New Jenkins Job .....	231
Understanding the Execution of Jobs with SCM Set.....	233
Failing the Build to See the Email Notification.....	234
Summary.....	235
<b>Chapter 14: Understanding the Jenkins Pipeline .....</b>	<b>237</b>
What Is a Jenkins Pipeline? .....	237
Why Use a Jenkins Pipeline? .....	238
Understanding Different Pipeline Concepts .....	239
Pipeline Syntax Overview.....	239
Fundamentals of Declarative Pipelines .....	240
Fundamentals of Scripted Pipelines.....	241
Defining a Simple Pipeline in Jenkins UI .....	243
Pipeline Speed/Durability Override.....	244
Resolving <code>java.lang.IllegalArgumentException: Unsupported Class File Major Version Error</code> .....	248
Understanding String Interpolation in the in Jenkins Pipeline.....	248
String Interpolation Example .....	249
Creating a Pipeline Job to Release the Java API.....	250
Running a Pipeline Job and Release the Calculator API .....	257
Summary.....	259

<b>Chapter 15: Creating Jenkins Jobs to Manage a Web Application Project .....</b>	<b>261</b>
Understanding the Calculator Web Application Source Code .....	262
Building the Calculator Web Application .....	266
Deploying the Calculator Web Application .....	266
Accessing the Calculator Web Application.....	267
Understanding E-E Testing the Calculator Web Application Using the Selenium WebDriver.....	268
What Is UI Automation Testing .....	268
How UI Automation Tools Work.....	268
What Is a Selenium WebDriver .....	269
Understanding the Selenium Tests Written in Python.....	269
Software Setup to Run these Test Cases.....	272
Running Selenium Python Tests .....	274
Pushing the WebApplication and Automation Project to the GitHub Repository .....	276
Step 1: Sign up on Github.com .....	277
Step 2: Creating a New Repository.....	277
Step 3: Creating a New SSH Key Pair for the GitHub Repository.....	278
Step 4: Include the SSH Private Key File Path in the \$(user.name)\.ssh\config File .....	278
Step 5: Adding an SSH Public Key to the GitHub Repository .....	279
Step 6: Committing the Calculator Web Application to the Master Branch in the Local Repository .....	282
Step 7: Pushing the Master Branch to the Remote Repository.....	282
Pushing the Selenium Python Automation Project to GitHub .....	283

## TABLE OF CONTENTS

<b>Creating Parameterized Auto-Trigger Free-Style Jenkins Jobs .....</b>	<b>283</b>
<b>Step 1: Installing the Parameterized Trigger Plugin.....</b>	<b>284</b>
<b>Step 2: Creating a BuildAndDeployCalculator WebApplication Job .....</b>	<b>285</b>
<b>Step 3: Creating the TestCalculatorWeb Application Job .....</b>	<b>288</b>
<b>Configuring an Extended Email Notification.....</b>	<b>291</b>
<b>Step 1: Installing the Email Extension Plugin .....</b>	<b>292</b>
<b>Step 2: Configuring Extended Email Extension settings.....</b>	<b>292</b>
<b>Step 3: Adding an Email Notification Step in the Post-Build Section of the TestCalculatorWeb Application Job .....</b>	<b>293</b>
<b>Running a Parameterized Free-Style Job Manually.....</b>	<b>295</b>
<b>Auto-Triggering a Parameterized Free-Style Job.....</b>	<b>296</b>
<b>Creating a Parameterized Pipeline Job.....</b>	<b>298</b>
<b>Creating a Build and Deploying the Calculator Web Application Parameterized Pipeline Job .....</b>	<b>299</b>
<b>Creating a TestCalculatorWebApplicationPipeline Job .....</b>	<b>301</b>
<b>Running the Parameterized Pipeline Job Manually .....</b>	<b>305</b>
<b>Automatically Triggering the Parameterized Pipeline Job .....</b>	<b>305</b>
<b>Summary.....</b>	<b>306</b>
<b>Chapter 16: Understanding Pipeline as Code .....</b>	<b>307</b>
<b>What Is API Authentication .....</b>	<b>307</b>
<b>How to Apply API Authentication to GitHub Repositories .....</b>	<b>308</b>
<b>How to Use an API Token to Access a Private GitHub Repository.....</b>	<b>310</b>
<b>Creating an API Access Token in GitLab .....</b>	<b>313</b>
<b>How to Use the API Token to Access a Private GitLab Repository .....</b>	<b>313</b>
<b>How to Access a GitHub/GitLab Repository Using an API Token in Jenkins .....</b>	<b>315</b>
<b>Step 1: Setting Up Maven in Jenkins.....</b>	<b>316</b>
<b>Step 2: Creating a Free-Style Job from the Jenkins Dashboard.....</b>	<b>316</b>

## TABLE OF CONTENTS

Step 3: Change the Version in pom.xml of the Cloned API Project.....	318
Step 4: Commit Changes in the Local Repository and Push them to the Central GitLab Repository.....	318
Running Free-Style Jobs Accessing the GitLab Repository with the API Token.....	319
Understanding Pipeline as Code (Jenkinsfile) .....	320
Writing a Pipeline in a Jenkinsfile .....	321
Triggering a Jenkins Job Using a GitLab Webhook.....	331
Creating a Pipeline Job to Trigger Using GitHub Webhooks .....	335
Triggering Pipeline Jobs Using GitHub Webhook .....	337
Summary.....	338
<b>Chapter 17: Jenkins Distributed Builds.....</b>	<b>339</b>
Jenkins Distributed Architecture.....	340
Ways to Connect the Master and Slaves.....	341
Understanding the Configuration to Connect the Master to the Agent Using SSH .....	342
Step 1: Install the SSH Build Agent's Plugin .....	342
Step 2: Install Java on the Slave Node .....	342
Step 3: Create an SSH Public-Private Key Pair .....	343
Step 4: Add a Credentials Entry with a Private Key to the Master Machine.....	343
Step 5: Add a Node Entry to the Jenkins Master .....	344
Step 6: Add a Public Key to the authorized_keys File on the Slave Machine .....	344
Step 7: Change the Permissions of the authorized_keys File on the Slave Machine .....	351
Step 8: Restart the sshd Service .....	353
Step 9: Check the Connection to the Agent Machine from the Master.....	353
Step 10: Launch the New Node from the Master Machine .....	353

## TABLE OF CONTENTS

Creating a Free-Style Job to Run on the Node1 Agent.....	354
Running the New Free-Style Job on the Node1 Agent.....	354
Understanding the Configuration to Connect the Agent to the Master Using JNLP .....	354
Step 1: Configure the Jenkins Master to Receive JNLP Agent Connection Requests .....	354
Step 2: Set the Jenkins URL .....	355
Step 3: Create a New Node Entry from the Agent (Slave) Machine .....	355
Creating a Job to Run on the JNLP Node .....	357
Summary.....	357
<b>Chapter 18: Integrating Jenkins with AWS.....</b>	<b>359</b>
Understanding an EC2 Instance on AWS.....	359
Creating an EC2 Instance on AWS.....	360
Step 1: Sign Up on AWS.....	360
Step 2: Sign in to AWS .....	360
Step 3: Create an EC2 Instance .....	361
Step 4: Start an EC2 Instance .....	362
Configuring the EC2 Instance to Deploy the Calculator Web Application .....	363
Step 1: Install the IIS Web Server .....	363
Step 2: Configure the IIS Web Server.....	364
Step 3: Configure the SSH Connection Between the Local Computer and the EC2 Instance .....	365
Copying the Calculator.html File to the EC2 Instance.....	370
Accessing Calculator.html Copied to the EC2 Instance from a Browser on a Local Computer.....	371
Creating a Jenkins Job to Deploy the Calculator Web Application on an EC2 instance .....	372
Changing the Calculator Application URL in Selenium Framework.....	372

## TABLE OF CONTENTS

Running the BuildAndDeployCalculator WebApplication OnEC2Instance Job .....	373
Summary.....	373
<b>Chapter 19: Miscellaneous Topics Part 1 .....</b>	<b>375</b>
Understanding the Jenkins CLI .....	375
How to Interact with Jenkins Using its CLI .....	376
How to Create a Job Using the Jenkins CLI .....	377
Authenticating Users Using Basic Authentication (Username-Password/API Token) .....	378
Authenticating Users Through SSH While Using CLI Commands .....	380
How to Build Jobs with the Jenkins CLI Using SSH Authentication.....	382
How to Export All Jobs .....	387
How to Import All Jobs.....	388
Summary.....	389
<b>Chapter 20: Miscellaneous Topics Part 2 .....</b>	<b>391</b>
Understanding Jenkins Remote Access API.....	392
Using Jenkins Remote Access API .....	393
Getting the Configuration of Existing Jenkins Jobs Using Jenkins Remote API.....	394
Creating New Jenkins Jobs Using Jenkins Remote API .....	395
Triggering Parameterized Jenkins Job Using Jenkins Remote API .....	396
Triggering Normal (Non-Parameterized) Jenkins Job Using Jenkins Remote API.....	398
Working with the Jenkins Server Using Python-Jenkins .....	400
Using Python-Jenkins Package Libraries .....	401

## TABLE OF CONTENTS

How to Use Shared Libraries in the Jenkins Pipeline .....	405
Step 1: Creating a Shared Library in the .Groovy File.....	405
Step 2: Push the Created Shared Library File to the GitLab Repository .....	407
Step 3: Configure the Shared Library in Jenkins .....	407
Step 4: Create a Pipeline Job to Use the Shared Library.....	408
Step 5: Running the Pipeline Job.....	409
Summary.....	409
<b>Index.....</b>	<b>411</b>

# About the Author



**Pranoday Pramod Dingare** is a certified software testing professional with more than 15 years of experience in software testing, including more than 10 years in automation testing. Pranoday has been leading test automation of mobile applications for the last eight years and has been involved in test automation tools evangelism, R&D, proof of concept, and pilot projects. He has worked as a freelance test automation consultant for various startups and mid-sized IT companies from India and abroad. Pranoday's open-source test automation tools have successfully replaced licensed automation tools, leading to major savings. He is responsible for incorporating DevOps practices into test automation processes of organizations by implementing DevOps tools such as Jenkins, Gitlab, Nexus, Docker, etc. He has recently shifted into full-time DevOps profile and has been working as a Lead DevOps professional since last 1 year. He has implemented various DevOps tools like Dockers, Maven, Kubernehese, Git, Nexus, Azure DevOps, AWS, SonarQube, Jenkins etc. and has been instrumental in automating various applications' build and deployment processes.

Pranoday is a tutor who has been involved in software testing and DevOps training for more than nine years, including conducting more than 200 retail and corporate trainings on the latest test automation and DevOps tools. He is a blogger on the latest test automation tools and technologies. Pranoday is passionate about working as a test automation architect, teaching and sharing knowledge about the latest tools and technologies, and helping professionals achieve their dreams.

# About the Technical Reviewer



**Prasanth Sahoo** is a thought leader, an adjunct professor, a technical speaker, and a full-time practitioner in Blockchain, DevOps, Cloud, and Agile working for PDI Software. He was awarded the “Blockchain and Cloud Expert of the Year” award in 2019 from the TCS Global Community for his knowledge share within the academic services to the community. He is passionate about driving digital technology initiatives and handling various community initiatives through coaching, mentoring, and grooming techniques.

Prasanth has a patent under his name and to date, he has interacted with over 50,000 professionals, mostly within the technical domain. He is a working group member of the Blockchain Council, CryptoCurrency Certification Consortium, Scrum Alliance, Scrum Organization, and International Institute of Business Analysis.

# Acknowledgments

It has been one of my oldest dreams to write a book of my own. First of all, I would like to thank Apress in helping me find the author inside and helping me achieve this dream. I would like to extend a sincere thanks to my acquisition editor, Spandana Chatterjee, for such a nice proposal and for being the first person to trust my writing skills. We had countless discussions about the content to be presented in the book and Spandana made this an excellent book. I am sure their suggestions and feedback will help me become a much better author. Thank you also to Laura Berendson for being such a nice mentor and giving me the confidence when I needed it. I would like to thank Mark Powers for tolerating my writing mistakes and helping me make this book better with his expertise and suggestions. Thanks to Divya Modi for guiding me through this process and making me comfortable.

I am thankful to the HR department, the executive management, the team leaders, and all my colleagues from Magic Software Enterprises for encouraging me throughout this journey.

A special thanks to my father Pramod Dingare and mother Rajani Dingare for always encouraging me to take different paths in life and for supporting me when I was trying to make my way as an actor, software professional, trainer, and now as an author. Thanks to my wife Aruna Dingare for tolerating my busy writing schedule and helping with her MS Word skills when necessary. Thanks to my lovely son Rugved Dingare for being my source of energy, which I needed dearly throughout this writing journey.

## ACKNOWLEDGMENTS

I would like to thank the entire Dingare family for leading by example and inspiring me throughout my life since childhood.

I would like to thank countless authors who unknowingly inspired me to pursue this dream of being author one day!

Last but definitely not the least thanks to all my students for helping me grow as a constant learner.

# Introduction

This book will help readers understand continuous integration (CI), continuous delivery, and continuous deployment (CD) with Jenkins. These processes allow users as well as administrators to catch problems as soon as they are injected into software systems.

This book starts with an introduction to Jenkins and covers its architecture and role in CI/CD. The basics are covered, including installing and configuring Jenkins. Tool configuration and plugins are discussed as well as available security measures such as credentials. Readers will learn what is meant by *jobs* in Jenkins, including their types, sections, and much more. The book explains Java API: projects, jobs, and configuration. The chapters take you through creating pipelines, their role in managing web apps, and distributed pipelines. There are lot of examples and scenarios included to explain the workings of the distributed version control system called Git and working with it using different authentication techniques like SSH. The book also covers unit testing with TestNG as well as end-to-end testing using Selenium Python as part of building a lifecycle and setting up Jenkins on different physical and Docker environments. It also explains Jenkins integration with cloud environments such as AWS. This book covers how to create reusable libraries for use in Jenkins pipelines and how to control Jenkins servers using Jenkins CLI and REST APIs.

The book helps you understand CI/CD implementation using Jenkins from scratch in your projects and prepares you for end-to-end DevOps practices.

## INTRODUCTION

### What You Will Learn

- Apply Jenkins to create end-to-end pipelines
- Integrate Jenkins with AWS, Docker, Git, and many more tools
- Use Selenium automation for end-to-end testing
- Create distributed pipelines

### Who Is This Book For

This book is best suited for developers and test automation professionals who are involved in creating CI/CD pipelines as well as prospective DevOps aspirants who want to make their way as professionals.

## CHAPTER 1

# Understanding CI/CD

This chapter explains what CI/CD is in practice and how it has influenced software development in general. In modern-day application development processes, where the Agile development lifecycle is used, it's common to have frequent changes to the application code.

---

**Note** Agile is a development lifecycle model in which each member of the development team (i.e., the developers, testers, business analysts, etc.) works on the same set of requirements at the same time. In other development lifecycle models, such as Rapid Application Development (RAD), different people work on different requirements. For example, the software testers work on a piece of functionality that the developers have finished working on, and the developers might be developing requirements that the business analysts have completed.

---

In the Agile development lifecycle model, application requirements are divided into multiple sets of prioritized requirements, which are taken up for implementation as part of the application development process. A *sprint* is a timeframe within which a single set of requirements is worked on, from the phase of requirements review until their final implementation in an application.

In Agile, the application evolves with every sprint. A sprint, which usually spans one to three weeks, adds a considerable amount of functionality in a short period. In a single day, multiple developers working on an application will complete development of assigned functionalities and will commit their changes to the main branch. (The code from this branch is used to create a build; it's usually a master branch in the case of GitLab and a main branch in the case of GitHub.) This practice of merging all the changes at once can introduce problems, like regression defects, integration defects, and merge conflicts, and these problems can take hours to find and resolve.

To avoid these problems, *Continuous Integration (CI)* is practiced in Agile methodology. In CI, work completed by the developers is merged frequently into the main branch, rather than waiting until the end of the day or until the end of the sprint. This practice of merging fewer changes at a time allows developers to resolve merge conflicts earlier and address regression defects more efficiently and quickly.

## The Development Workflow

### Running Unit Tests Locally

The developer obtains their own copy of the latest code from the central repository and implements the required changes. These changes are usually implemented using test-driven development practices. The development and changes to the code continue until all written unit test cases pass.

---

**Note** Test-driven development (TDD) is a practice in which test cases are written before developing the code. Written test cases are executed, which will fail as there is no code written as per the expectations prescribed in the test case. Then the code is developed to fulfill the test cases' expectations.

---

## Pushing and Merging Code to the Central Repository

Once the developers finish the development work in the local copy, they push this code to the central repository to be merged into the main branch.

## Compiling Code after the Merge

Once the developer's code is merged with the main branch, this merged code must be compiled. This reveals any new compilation errors that were introduced due to merging the newly developed code with the existing code.

## Running Tests on the Merged Code

After a successful merger, the unit and integration tests are executed to reveal any regression defects. In addition to these, a few CI processes encourage static analysis to be executed on the code to check for adherence to the coding standards, existence of dead code, and so on.

## Deploying an Artifact

Once the merged code is checked against the various quality attributes, an artifact is built and deployed to the deployment server. All stable code is packed and deployed to the server for the end users to use. These artifacts take the form of .WAR or .JAR files, for example.

# Continuous Delivery/Continuous Deployment

In Continuous Integration, incremental changes to the application are tested in a development environment, which reveals any failures. Once the changes are tested and confirmed in the development environment, including these changes in periodic builds is very important.

It may happen that the application is working fine in the development environment, but having problems in the production environment. These problems could be because a new change is not compatible with the hardware/software in the production environment. If the application is not frequently deployed in the production environment, then debugging and resolving such problems could turn out to be a nightmare.

## CI/CD Workflow Case Study

For this case study, you'll develop a Calculator Web application that includes addition, subtraction, multiplication, and division functionalities.

### Pulling the Application's Latest Code

You'll pull the latest application code on the local machine from a central code repository to start developing the addition functionality. It will add two numbers and return the result.

### Developing and Running Unit Test Cases

You will develop unit test cases before you start writing the code. You will execute the written unit test cases, which will fail, as there is no code written to satisfy the expectations mentioned in the test cases.

For example, calling `Addition(10, 20)` and comparing the return value with 30 to confirm the addition functionality works fine. See Listing 1-1.

### ***Listing 1-1.*** Unit Test Case for the Addition Functionality

```
{  
    Result=Addition(10,20);  
    Assert.assertEquals(Result,30,"Addition functionality does not  
    work fine with positive numbers");  
}
```

This test code calls an `Addition` function with two positive numbers, 10 and 20. It receives the result in the `Result` variable and compares the value in the `Result` variable with 30. If the `Result` variable contains the value 30, the test case will be marked as Passed. Otherwise, it is marked as Failed.

This test case fails with the error `Addition(10, 20)` function does not exist (we do not have a function called `Addition()` yet). In test-driven development, the test cases should fail first until the developer develops the functionality with the intention of making the failed tests pass.

---

**Note** We don't assume the use of any particular language or unit testing tool here. For now, concentrate on understanding the concept and flow of CI/CD instead of getting into the details of language and specific tools

---

## **Code Development**

To make the failed test case pass, you must implement the `Addition` function, as shown in Listing 1-2.

***Listing 1-2.*** Code for the Addition Functionality

```
Addition(a,b)
{
    Result=a+b;
    return Result;
}
```

Along with this backend code, component changes would also be made to the frontend code. For example, an Addition pushbutton would be added to the UI to trigger this function.

## Rerunning the Unit Test Case

If you rerun the unit test case, it will now pass.

## Pushing and Merging the Code to the Central Repository

You can now push the Addition function code to the central repository to be merged with rest of the application's code.

## Compiling the Code after the Merge

After the `Addition()` function code is merged, the whole application will be compiled.

## Running Tests on the Merged Code

A few integration tests and tests related to other functionalities like subtraction, multiplication, and so on, will be executed to confirm that this newly added Addition functionality has not broken any previous functionalities.

## Deploying an Artifact

Now you build the web application and deploy it in the production environment on an application server (e.g., tomcat).

## Run e-e Tests on a Deployed Application

Finally, you need to run e-e tests using UI automation tools like Selenium and confirm that the application's end-end flows are working fine.

## Summary

This chapter explained the problems of following traditional development processes in a fast-paced development model like Agile. This chapter also explained how you can benefit from adopting CI/CD processes that are quite in line with the requirements of the Agile methodology. The next chapter introduces Jenkins, a widely used automation server that helps in establishing CI/CD processes.

## CHAPTER 2

# Introducing Jenkins

In the previous chapter, you learned about the importance of using Continuous Integration, Continuous Delivery, and Continuous Deployment as a practice. This chapter introduces the widely used automation server called Jenkins, which helps to automate CI/CD processes and leverages their benefits. In CI/CD practice, you continuously integrate a smaller number of changes frequently, test them, and provide feedback on each increment you introduce into the build. Managing this process manually would be labor intensive, tedious, cumbersome, and prone to errors.

The Agile methodology also supports releasing a workable product after introducing incremental changes to the software. Teams should not only see the features working, but they should also see the additional value to the software due to the introduction of the new feature. Due to their philosophical similarities, CI/CD works very well with the Agile methodologies.

The CI/CD process executes the workflow on any application, beginning by taking the latest copy of the source code from a source code management system such as Git or SVN. It then compiles the code, which requires the execution of a compiler, and executes the unit tests, again interacting with a different set of tools. Executing a workflow that spans across multiple tools/technologies multiple times a day would put pressure on developers, QAs, support engineers, right?

Using Jenkins, you can build an application on different platforms. It also allows you to automate pull-request integration processes along with artifact publishing on artifact repositories such as Nexus.

Different unit and integration testing frameworks and testing containers are not sufficient to release a quality build if you do not complement them by using Jenkins. These tools need to be orchestrated to produce a quality product and Jenkins is the tool to do this.

Jenkins interacts with different sets of tools and executes an end-end workflow on the application at a specified frequency, without putting any unnecessary pressure on the teams building the application.

## What Is Jenkins?

Jenkins helps automate different phases of the software development process, including pulling the latest copy of the source code from the centralized source code management repository, compiling the source code, running any unit tests, packaging the implementation in the form of different types of artifacts, and then deploying these artifacts on different kinds of environments. It is a free and open-source server.

Jenkins is a server system that runs inside the servlet containers such as Apache Tomcat. It is written in Java and supports a different family of tools that take part in software development.

## A History of Jenkins

Jenkins' history begins in 2004. Kohsuke Kawaguchi, the developer of Jenkins, was working at Sun Microsystems as a Java developer. At that time, Kawaguchi was involved in several development projects. He didn't like breaking the builds due to code failure. This made him look for something that would help him know whether the code would work before it was committed to the repository.

This curiosity led the way to the development of an automation server named Hudson. In 2011, there was an infamous dispute between the independent Hudson open-source community and Oracle, which now has Sun Microsystems under its umbrella.

This dispute led to a fork, which was named Jenkins. Both Jenkins and Hudson continued to exist for a long time; however, Jenkins was the preferred choice. The Hudson project was shut down in January 2020. Jenkins is still active.

## Implementing CI/CD Using Jenkins

CI/CD is a process that tests every change (even a smallest one) to application code at the code level, through an end-end build lifecycle until the change is built in the form of an executable or library and deployed on the production environment.

CI/CD consists of different dependent lifecycle phases which, if followed in a specified order, will convert the bare source code into a workable application by taking it through all the required sub-processes.

Jenkins as an automation server supports implementing these build lifecycle phases with the help of DSL (the Domain Specific Language). While taking any application through different build phases, we need to use different categories of tools like build tools, static analysis tools, different source code management tools, etc.

Jenkins provides a huge collection of plugins so we can implement end-end build lifecycle phases for any application.

We can write a Jenkins script, called a *pipeline*, which defines a sequence of tasks and subtasks that Jenkins will perform as part of each build phase. These build phases will be implemented sequentially and in such a way that each phase takes the output of its previous phase, does the required processing that it is supposed to do as part of the end-end build process, and passes the output to the next phase.

When these sequential and dependent phases are executed from start to end, a workable build is created that end users can use.

If any phase fails during the build process, the subsequent phase (which depends on the output of the failed phase) is not executed and the entire build process fails.

For example, say after pulling the latest merged code from an SCM (a source code management tool like Git), the compilation fails. There is no point in going ahead with the next phase of unit testing, as it requires successfully compiled code in order to run the unit tests.

## The Jenkins Architecture

This section discusses how Jenkins can be used by developers as well as testers. The following is the flow of a typical Jenkins CI/CD process:

1. Multiple developers push their individual branches containing their changes to the central repository.  
After the code is reviewed, these branches are merged into another branch (usually a developer branch).
2. Jenkins gets a notification about the change to the branch.
3. As a result of the notification, Jenkins triggers a job.

A *job* is set of tasks/subtasks implemented as a sequential process to perform different phases of the build release lifecycle. It will typically include the following steps:

1. Pulling a change from the repository using a plugin, which will integrate with the given source code management system (e.g., a Git plugin).

2. Compiling a change by using a build tool like Maven and the Jenkins plugin (e.g., a Maven plugin).
3. Running unit/integration testing on the compiled code using the build tool again.
4. Running static analysis to check the code against coding standards and for dead code. This is done using a tool like SonarQube, which again is triggered using the Jenkins plugin.
5. Bundling the compiled and tested files in the form of library files, like .JAR or .WAR files. This also can be done by using a build tool that's triggered using the Jenkins plugin.
6. Deploying this built library file on the production/test environment.
7. Running end-end tests on this deployed application using end-end test automation tools (e.g., UI automation tools like Selenium, Protractor, etc.).
8. Sending an email notification to the concerned team members regarding the status of this newly created application build. Includes the report of the end-end tests.

Before Jenkins came into an existence, integration testing was done when all the developers' changes were accepted and merged into the branch and the build was created. Also, integration and end-end testing was done on the whole application code, which was to be considered for creating an artifact. When bugs were found during regression testing, finding their root causes was not easy. Even locating and fixing build errors was a time-consuming tasks and used to delay the delivery of the software application.

With Jenkins, all changes implemented in the application are unit tested and merged into the branch as and when they are implemented and unit tested successfully. Once the changes are merged into the branch, they are tested for any integration defects. If errors are found during the deployment phase, only the files added by all developers since the last successful build are checked. Similarly, if the integration tests find bugs in the new build, the changes implemented and merged in the current build are looked at as the culprit. Due to Jenkins' frequent execution of build lifecycle phases like testing, compilation, and deployment, these incremental changes have been formalized.

## Summary

This chapter explained the steps required in order to implement successful CI/CD processes. It also explained the widely used automation server, Jenkins. Prior to the advent of Jenkins, performing frequent testing and deploying an application was very effort intensive. The next chapter dives deep into Jenkins' features, which allow us to implement successful CI/CD processes.

## CHAPTER 3

# Installing Jenkins

Now that you understand the importance of the CI/CD process and the role Jenkins plays in implementing CI/CD, let's get started using Jenkins. This chapter covers the Jenkins installation process on different environments like Windows and Dockers and discusses the hardware/software requirements. It also discusses the different ways Jenkins can be installed, such as through .WAR (Web Application Archive) files, using the .MSI file, and installing Jenkins as a Windows service. We cover different problems/errors we might encounter during the installation process and see how to troubleshoot them.

## Installing Jenkins on Windows

This section covers the hardware and software requirements for installing Jenkins. Make sure you have the required version of Java and the other software installed as specified in this section, so you can follow along with rest of the chapters in this book.

## Hardware/Software Requirements

Let's first cover the hardware requirements and then the software requirements we need to install Jenkins on a machine.

The minimum hardware requirements are as follows:

- 256MB of RAM
- 1GB hard disk space (If you are running Jenkins as a Docker container, 10GB is recommended.)

Software requirements:

- Java: Jenkins supports only JDK 8 and JDK 11. JDK 11 is supported since Jenkins version 2.164 and 2.164.1. Older versions of Java are not supported.

## Different Ways to Install Jenkins on Windows

We can install Jenkins on Windows in three different ways:

- **Installing using the MSI Installer:**

Download the Jenkins MSI installer file from  
[www.jenkins.io/download/thank-you-downloading-windows-installer](http://www.jenkins.io/download/thank-you-downloading-windows-installer)

Run the MSI installer file to start installing Jenkins and follow the steps mentioned in this link:

[www.jenkins.io/doc/book/installing/windows/](http://www.jenkins.io/doc/book/installing/windows/)

- **Installing using a .WAR file:**

Jenkins is written in the Java language. Web applications developed using Java are bundled in .WAR files. We can install and run the Jenkins server through its .WAR file bundle.

Download the Jenkins .WAR file from <https://get.jenkins.io/war/2.290/jenkins.war>

Follow the steps mentioned from the following link to install Jenkins through its .WAR file.

[www.jenkins.io/doc/book/installing/war-file/](http://www.jenkins.io/doc/book/installing/war-file/)

- **Installing Jenkins as a Docker Image:**

Understanding the entire Docker concept is outside the scope of this book. In layman terms, Docker is an application build and deployment tool. It is based on the idea that we can package our code with dependencies into a deployable unit called a *container*. We usually replace physical machines with virtual systems that emulate operating systems with the help of software. Dockers are very lightweight concepts. We can even code configurations for our application (Infrastructure as a Code).

To install Jenkins server using Docker, follow the steps from this link:

[www.jenkins.io/doc/book/installing/docker/](http://www.jenkins.io/doc/book/installing/docker/)

## Understanding the Configuration Files and Directory Structure of Jenkins

Jenkins by default is installed in the `$user.home` directory (i.e., the current user directory), inside the `.jenkins` folder. If you installed Jenkins through `.msifile`, it will be installed inside the folder you chose as the installation directory.

The Jenkins installation directory is also called `JENKINS_HOME`. Its directory structure is shown in Listing 3-1.

***Listing 3-1.*** Diagrammatic Representation of the JENKINS\_HOME Directory Structure

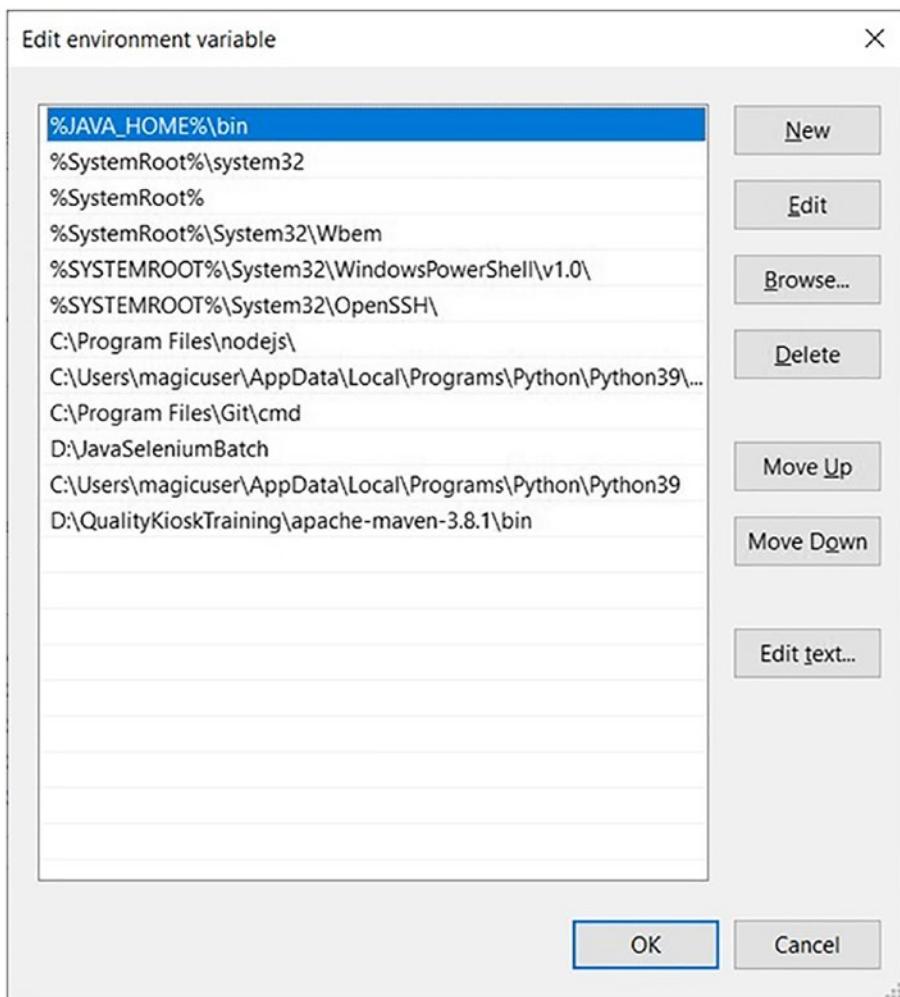
```

JENKINS_HOME
+- config.xml      (jenkins root configuration)
+- *.xml           (other site-wide configuration files)
+- userContent     (files in this directory will be served under
                   your http://server/userContent/)
+- fingerprints   (stores fingerprint records)
+- nodes           (slave configurations)
+- plugins         (stores plugins)
+- secrets         (secrets needed when migrating credentials
                   to other servers)
+- workspace       (working directory for the version
control system)
    +- [JOBNAME]  (sub-directory for each job)
+- jobs
    +- [JOBNAME]   (sub-directory for each job)
        +- config.xml    (job configuration file)
        +- latest        (symbolic link to the last
                           successful build)
        +- builds
            +- [BUILD_ID]   (for each build)
                +- build.xml    (build result summary)
                +- log          (log file)
                +- changelog.xml (change log)

```

## Understanding Important Settings in Jenkins.xml

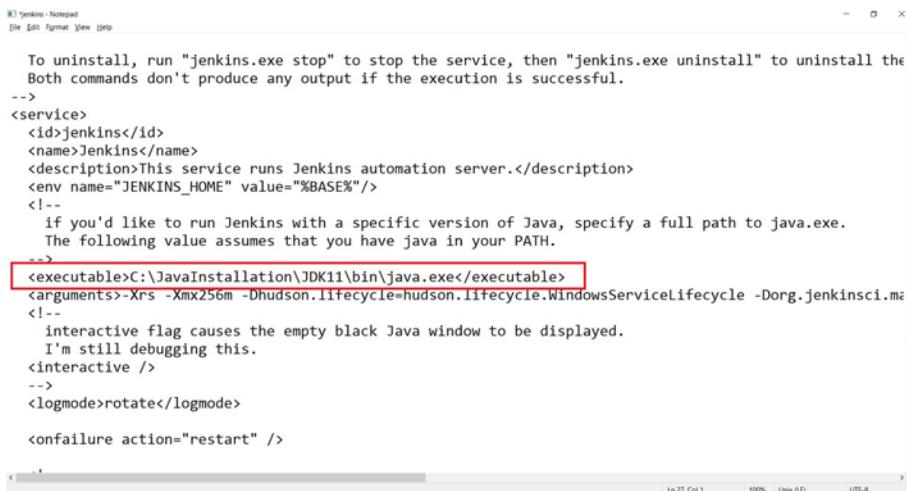
The Jenkins.xml file exists inside the JENKINS\_HOME directory. Find the <Service> tag in this file. This tag contains important settings, like the Java.exe location, which is used by Jenkins. By default, Jenkins takes the Java mentioned in the PATH environment variable, as shown in Figure 3-1.



**Figure 3-1.** The Edit Environment Variable window showing Java added to the PATH environment variable

If we want to change the version of Java that Jenkins should use, we can mention the full path of the Java.exe from the desired Java installation directory in an `<executable>` tag, which is inside a `<service>` tag, as shown in Figure 3-2.

## CHAPTER 3 INSTALLING JENKINS



```
Jenkins - Notepad
File Edit Format View Help

To uninstall, run "jenkins.exe stop" to stop the service, then "jenkins.exe uninstall" to uninstall the
Both commands don't produce any output if the execution is successful.
-->
<service>
  <id>jenkins</id>
  <name>Jenkins</name>
  <description>This service runs Jenkins automation server.</description>
  <env name="JENKINS_HOME" value="%BASE%" />
<!--
  if you'd like to run Jenkins with a specific version of Java, specify a full path to java.exe.
  The following value assumes that you have java in your PATH.
-->
<executable>C:\JavaInstallation\JDK11\bin\java.exe</executable>
<arguments>-Xrs -Xmx256m -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -Dorg.jenkinsci.m
<!--
  interactive flag causes the empty black Java window to be displayed.
  I'm still debugging this.
<interactive />
-->
<logmode>rotate</logmode>

<onfailure action="restart" />
-->
```

The screenshot shows a Windows Notepad window with Jenkins configuration XML. The `<executable>` tag, which specifies the path to the Java executable, is highlighted with a red rectangle. The XML code also includes details about stopping and uninstalling the service, environment variables, and log mode.

**Figure 3-2.** The `<executable>` tag used to indicate the path of Java we want Jenkins to use

## Summary

This chapter explained how to install Jenkins using its installation wizard by running the .MSI file. You also saw how to start Jenkins as a Windows service as well as by using Jenkins.war. After starting the Jenkins server, we can configure a Jenkins instance using the Jenkins setup wizard. At the end of the chapter, you saw the directory structure of the Jenkins installation folder, along with an important configuration file called `Jenkins.xml`. In the next chapter, we discuss how to set different settings to configure a Jenkins instance.

## CHAPTER 4

# Configuring Jenkins

The last chapter covered installing Jenkins on different platforms and discussed important configuration files and directory structure, so now it's time to configure Jenkins for actual use.

This chapter discusses starting Jenkins using the .WAR file as well as through Windows services (if Jenkins is installed as a service). You learn how to sign into Jenkins and configure the global settings and paths of different tools you'll use with Jenkins as a part of Jenkins's jobs and pipelines.

## Configuring Global Settings and Paths

This section explains how to configure the different tools/software you'll use with Jenkins, such as Maven and the JDK.

## Signing into Jenkins

Before you start configuring Jenkins, we explain how to start the Jenkins server and sign into Jenkins.

## Starting the Jenkins Server

You can start the Jenkins server by using the .WAR file or, if you installed Jenkins as a service, you have to start the Jenkins service. Let's look at both the ways of starting a Jenkins server on a Windows machine.

- To start the Jenkins server using the .WAR file (version 2.289 is used throughout this book), run the following command in the command prompt:

```
Java -jar <Path of .war file> -- httpListenAddress=<Ip address of machine> --httpPort=<port not used by any other running process on your machine>
```

After running this command, you must wait until the Jenkins server is fully up and running.

- To start the Jenkins server by starting a service, go to start menu and type **Services**.

Select the Services option, which will open Services window.

Right-click the Jenkins service entry and click Start. (If the Start options are disabled, that means the Jenkins service is already running on your machine.)

After clicking Start, the Jenkins service will start.

Once you click the Start menu, the Jenkins service will start the server on localhost:8080. If you want to start the server on an IP address and a different port other than 8080, open \$JENKINS\_HOME\jenkins.xml.

JENKINS\_HOME refers to the Jenkins installation directory, which by default is created inside the CurrentUser directory present on SystemDrive. On my machine, the Jenkins installation path is C:\magicuser\.jenkins.Add - httpListenAddress=<IP address of your machine> and -httpPort=<Desired port number> in <arguments> tag, as shown in Figure 4-1.

```
--httpPort=8081 --httpListenAddress=192.168.43.10
```

**Figure 4-1.** *httpPort and httpListenAddress changed in Jenkins.xml*

Restart the service by right-clicking the Jenkins service in the Services list and clicking the Restart menu.

- To start the Jenkins server as a Docker container, run the following command from your Windows command prompt. It will start the Jenkins server using the Jenkins Docker image available on the Docker hub.

```
docker run -p 8080:8080 -name=Jenkins-server Jenkins/jenkins
```

In this command, `-p 8080:8080` is *port forwarding*. Port forwarding means that all requests targeted to the 8080 port (the left side of :) on the Docker container will be forwarded to the 8080 port (the right side of :) on a machine running a Jenkins server as a Docker container.

## Starting the Jenkins Service on Linux

You need to install Jenkins as a service before you can start it with the following command:

```
sudosystemctl start Jenkins
```

---

**Note** If the Jenkins server fails to start due to port 8080 being in use, edit the `/etc/default/Jenkins` file and replace the `-HTTP_PORT=8080` line with `-HTTP_PORT=<your desired port number>`. For example, if you want to start the Jenkins server on port number 8081, use `HTTP_PORT=8081` in this file, save the changes in the file, and restart Jenkins.

---

You can check the status of the Jenkins service using the following command:

```
sudosystemctl status Jenkins
```

---

**Note** Starting Jenkins using the .WAR file on Linux is no different than starting it using the .WAR file on a Windows machine, so it is not explained explicitly here.

---

## Opening the Browser and Signing In

Open a browser and enter the URL containing the IP address and port you used to start the Jenkins server. Enter the username and password you entered while performing the initial setup process and then click the Sign In button.

---

**Note** If you started the Jenkins server as a Docker container, then use the port mentioned to the right of : in the port forwarding option when starting the container. For example, if your command to start the Jenkins Docker container is `docker run -p 8080:8081 -name=Jenkins-server Jenkins/Jenkins`, then you should access Jenkins using port 8081 in the Jenkins URL.

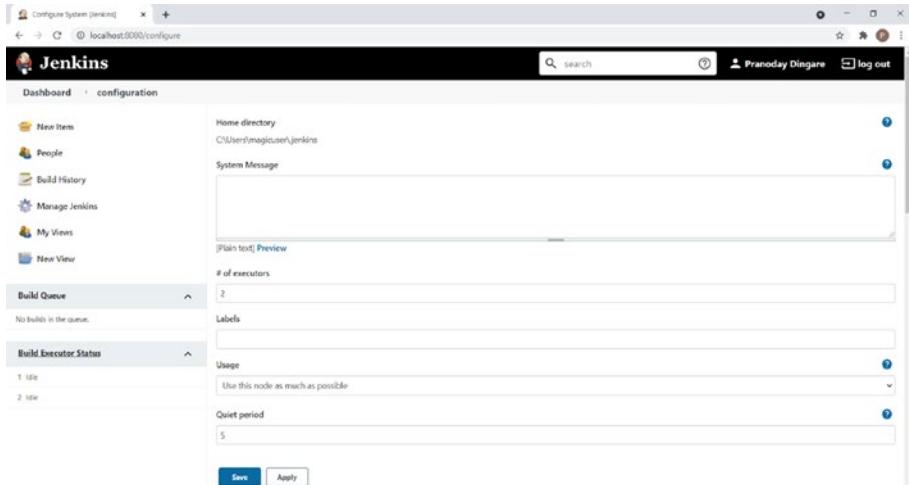
---

## Understanding the Configure System Options

This section explains the Jenkins system settings. Let's start with the settings page. After you log into the system, you will see the Jenkins dashboard.

Click the Manage Jenkins link, which will take you to the Manage Jenkins page. Then click the Configure System link.

This will take you to the System Configuration page, as shown in Figure 4-2.



**Figure 4-2.** The System Configuration page

Let's look at a few important and widely used settings on this page one by one:

- Home directory:

This setting is not editable from the Jenkins UI. This is where you find all the Jenkins related files and folders, like Jenkins jobs, configuration files, etc. You can change the location of `JENKINS_HOME`. You may need to do this if you don't have enough space or the required directory access, for example. If you change the location, make sure to clean up all the files left behind in the old location.

There are a few ways to change the Jenkins home directory:

- Edit the `JENKINS_HOME` variable in your Jenkins configuration file (e.g., `/etc/sysconfig/jenkins` on Red Hat Linux).
- Use your web container's admin tool to set the `JENKINS_HOME` environment variable.
- Set the `JENKINS_HOME` environment variable in the operating system environment variables (in the System Environment variable) before launching Jenkins directly from the .WAR file.
- Set the `JENKINS_HOME` Java system property when launching your web container, or when launching Jenkins directly from the .WAR file.
- Modify `web.xml` in `jenkins.war` (or its expanded image in your web container). This is not recommended.
- Jenkins URL:

This is the URL on which the Jenkins server is accessible. This contains the IP address and port used to start the Jenkins server. If you start the Jenkins server on localhost and the default port, this field will be `http://localhost:8080`. You can change this URL to your machine IP address if you want to connect to this Jenkins instance from different machines.

- System admin email address:

This is where you configure an email address in the from header of an email that's sent from Jenkins as notification of the execution of Jenkins jobs. (We cover configuring email notifications in upcoming chapters.)

## Resetting the Username and Password in Jenkins

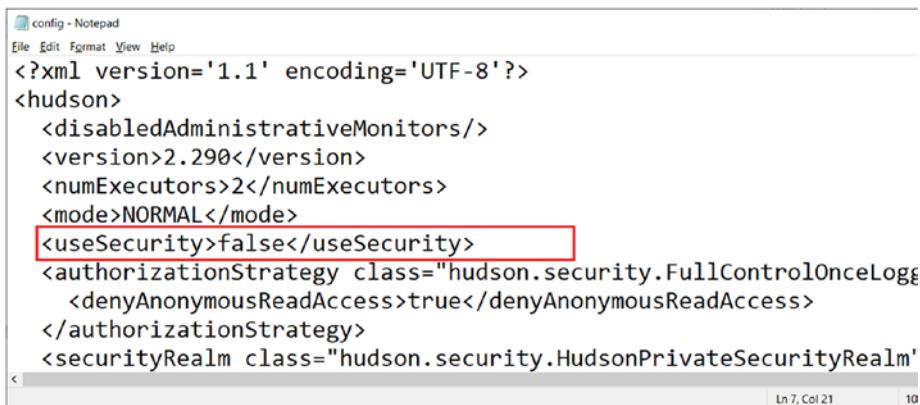
If you forget the username and password, you have to restore them before you can log into Jenkins. Follow these steps to reset the username and password when Jenkins server is running on a Windows machine:

1. *Stop the Jenkins server if it is already running.*

To stop the Jenkins server, close the command prompt you are using to run the Jenkins server using the Jenkins .WAR file. If you are using Jenkins as a service, go to Services, right-click the Jenkins service, and choose Stop menu option to stop the service.

2. *Edit the CONFIG.XML file from \$(JENKINS\_HOME).*

Go to the \$(JENKINS\_HOME) directory (i.e., the Jenkins installation directory) and open the config.xml file. Find the </useSecurity> tag in the file. The value of this tag is set by default to true. Change it to false and save the change, as shown in Figure 4-3.



```
<?xml version='1.1' encoding='UTF-8'?>
<hudson>
  <disabledAdministrativeMonitors/>
  <version>2.290</version>
  <numExecutors>2</numExecutors>
  <mode>NORMAL</mode>
  <useSecurity>false</useSecurity> <useSecurity>false</useSecurity>
  <authorizationStrategy class="hudson.security.FullControlOnceLoggedInUserAuthorizationStrategy">
    <denyAnonymousReadAccess>true</denyAnonymousReadAccess>
  </authorizationStrategy>
  <securityRealm class="hudson.security.HudsonPrivateSecurityRealm">
    <realmName>Jenkins</realmName>
  </securityRealm>
</hudson>
```

**Figure 4-3.** The `<useSecurity>` tag value changed to `false` in `config.xml`

---

**Note** If you installed Jenkins using the .MSI file, then the `$(JENKINS_HOME)` directory is in `C:\System32\Config\SystemProfile\AppData\Local\Jenkins\.jenkins`.

---

3. *Restart the Jenkins server.*

Once changes in `config.xml` are saved, restart the Jenkins server, either by running the .WAR file from the command prompt or by starting the Jenkins service if Jenkins is installed as a Windows service.

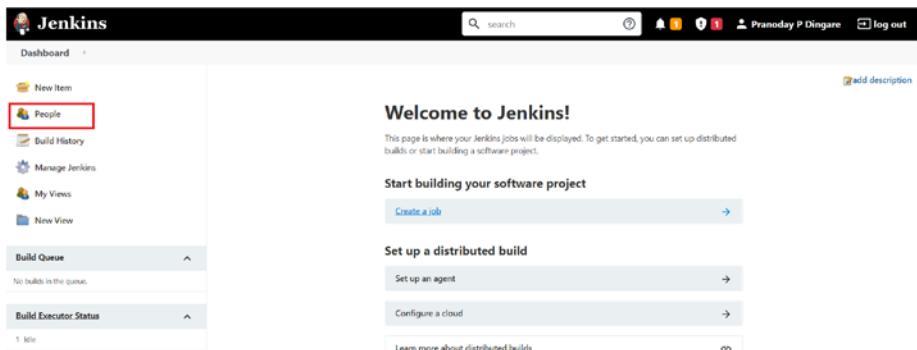
4. *Open the Jenkins server URL in a browser.*

Open your browser and enter the Jenkins server's URL. Jenkins will not ask you for a username and password and will take you directly inside the Jenkins dashboard.

If you are not opening the URL the first time, you may see a few Jenkins jobs in the dashboard.

5. Click Manage Jenkins to open the Manage Jenkins page.
6. Click the Configure Global Security link. This will open the Configure Global Security page.
7. Select the Anyone Can Do Anything radio button, which is present in the Authorization section.
8. Click the Save button.
9. *Delete the user whose username and password you forgot.*

*Click the People link on the left side (highlighted in Figure 4-4).*



**Figure 4-4.** The people link highlighted with box

10. Click the user entry in the table. The user ID is pranodayd in this case, as highlighted in Figure 4-5.

## CHAPTER 4 CONFIGURING JENKINS

The screenshot shows the Jenkins 'People' page. On the left, there's a sidebar with links for 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'New View'. The main area has a title 'People' with a subtitle 'Includes all known "users", including login identities which the current security realm can enumerate, as well as people mentioned in commit messages in recorded changelogs.' Below this is a table with columns 'User ID', 'Name', and 'Last Commit Activity'. A single row is shown for 'pranodday' with 'Pranoday Dingare' in the Name column and 'N/A' in the Last Commit Activity column. At the bottom of the table, it says '1 item'. The status bar at the bottom right shows 'REST API Jenkins 2.290'.

**Figure 4-5.** User entry to be clicked

11. Click the Delete link on the left, as shown in Figure 4-6.

The screenshot shows the Jenkins 'User' page for 'Pranoday Dingare'. The left sidebar has links for 'People', 'Status', 'Builds', 'Configure', 'My Views', and 'Delete'. The 'Delete' link is highlighted with a red box. The main area shows the user's details: 'Pranoday Dingare' and 'Jenkins User ID: pranodday'. There's also a 'Delete' button. The status bar at the bottom right shows 'REST API Jenkins 2.290'.

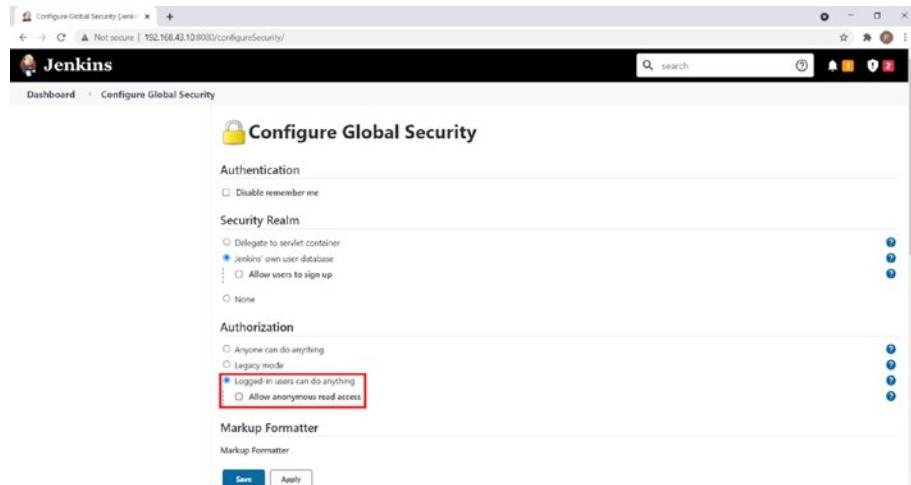
**Figure 4-6.** The Delete link

12. Click the Yes button to confirm the delete operation.

## Adding a New User

To add a new user, click the Manage Jenkins link. Then click the Configure Global Security link. Select the Jenkins' Own User Database option under the Security Realm section. Then follow these steps:

1. Select the Logged-in Users Can Do Anything option in the Authorization section and uncheck the Allow Anonymous Read Access checkbox, as shown in Figure 4-7.



**Figure 4-7.** The options selected from the Authorization section

2. Click the Save button.
3. Fill in the details of the admin user on the Create First Admin User screen and then click the Create First Admin User button.

You have successfully created a new Admin user.

## Summary

This chapter explained how to start Jenkins as a server as well as use it as a service. You also learned how to restore a forgotten username and password, which is a very common problem Jenkins users need to handle. You also learned about the different Jenkins configuration options available. In next chapter we are going to learn about Jenkins Plugin Manager to install/uninstall/update different Jenkins plugins.

## CHAPTER 5

# Managing Plugins in Jenkins

The software build lifecycle usually includes phases like pulling the source code from a source code management tool, compiling the source code, unit and integration testing, and then building the library and releasing it. To perform those phases, you need to use different categories of tools, including source code management (SCM) tools, unit/integration testing tools, build tools, etc. Jenkins needs to interface with those tools as part of executing the end-end build lifecycle. Jenkins uses *plugins* to interface with these tools.

This chapter explains what a plugin is, covers the plugins commonly used with Jenkins, and explains how to install plugins using the Jenkins Plugin Manager. During the installation process, you may face issues, so the last section of this chapter discusses how to resolve some of these common issues.

## What Are Plugins

A *plugin* is a software component that adds a specific feature to an existing computer program. There are various plugins developed by the Jenkins team to customize the usage of Jenkins.

## Commonly Used Jenkins Plugins

The following list includes some commonly used Jenkins plugins along with the tools they integrate into:

- **Git:**

The Git plugin is used to integrate with the Git version controlling system.

Git is a distributed version controlling system that allows developers to work collaboratively with software application code. We talk about Git in more detail in Chapters [10](#) and [12](#).

- **Maven Integration:**

The Maven Integration plugin integrates with the Maven build tool. Maven is a build tool that helps you automate important build phases like compilation, packaging, testing, etc.

We cover Maven in more detail in upcoming chapters.

- **Email Extension:**

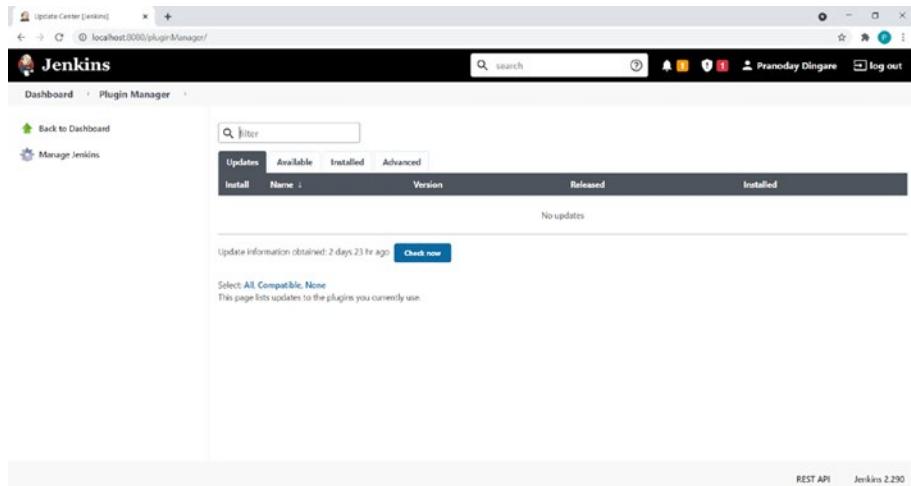
In Jenkins you can configure email notifications that will notify the team about the status of the current build. Using the Email Extension plugin, you can customize the email notifications and add more details to the email sent as a notification. You'll see this in action in upcoming chapters.

# Installing the Plugins in Jenkins

Now that you know what a plugin is and know how they are going to help you, this section explains the process of installing plugins in Jenkins.

Use the following steps to go to the Manage Plugins page and begin installing the required plugins.

1. Log into Jenkins. Once you are logged in, you will see the Jenkins dashboard.
2. Go to the Manage Jenkins page. Click the Manage Jenkins link available on the Jenkins dashboard to open the Manage Jenkins page.
3. Go to the Manage Plugins page.
4. Click the Manage Plugins link. Clicking this link will open the Plugin Manager page, as shown in Figure 5-1.



**Figure 5-1.** The Jenkins Plugin Manager

## Understanding the Plugin Manager

The Plugin Manager page updates and installs the different Jenkins plugins. There are four tabs on the Plugin Manager configuration page. They are Updates, Available, Installed, and Advanced. We are going to look at each one.

### The Updates Tab

This tab lists the installed plugins when updated versions of them are available to download and install. You can install updated versions of plugins by checking the checkboxes of desired plugin updates and clicking the Download Now and Install After Restart button, as shown in Figure 5-2.

The screenshot shows the Jenkins Plugin Manager interface. At the top, there is a search bar labeled 'Filter:' with the text 'Blue Ocean'. Below the search bar, there are four tabs: 'Updates' (which is selected), 'Available', 'Installed', and 'Advanced'. The main content area displays a table of plugin entries. The table has columns for 'Install' (checkbox), 'Name' (link), 'Version', and 'Installed'. One entry is visible: 'Blue Ocean beta' (checkbox checked, indicating it's up-to-date), version 1.0.0-b14, installed version 1.0.0-b13. Below the table, there are two buttons: 'Download now and install after restart' and 'Check now'. A status message says 'Update information obtained: 1 day 19 hr ago'.

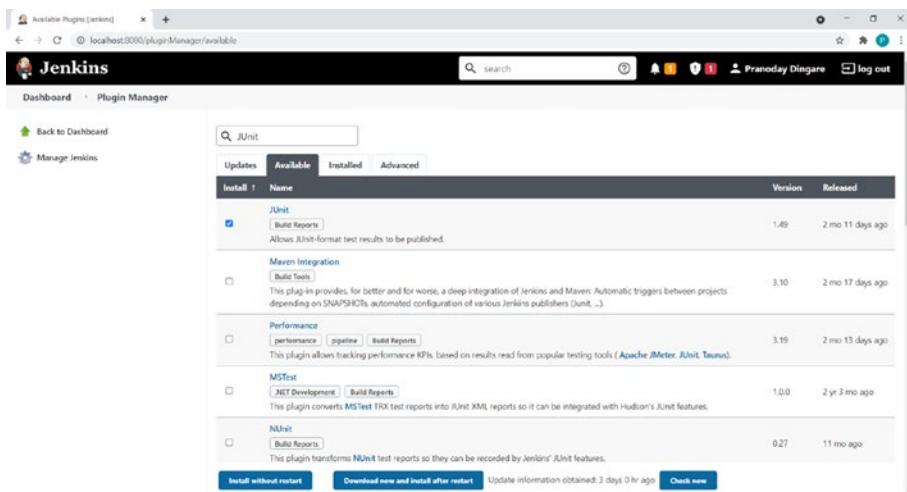
Install	Name	Version	Installed
<input checked="" type="checkbox"/>	<a href="#">Blue Ocean beta</a> A new user experience for Jenkins	1.0.0-b14	1.0.0-b13

**Figure 5-2.** Plugin entry to be updated

### The Available Tab

This tab shows a list of Jenkins plugins available to download and install.

To install a new plugin, go to the Available tab and type the name of the plugin in the Search field. For example, if you want to install the JUnit plugin then you would enter **Junit** into the search field. You will see the JUnit plugin entry at the top of the plugins list. Then you simply check the Junit plugin entry's checkbox. See Figure 5-3.



**Figure 5-3.** The Junit plugin entry is selected to start the installation

After checking the checkbox, you can click either the Install Without Restart or the Download Now and Install After Restart button.

- If you click the Install Without Restart button, the plugin download will start. Once it's downloaded, the plugin will be installed immediately.

You have to wait until the JUnit plugin installation status is shown as *Success* in green. It will first download the plugin and then immediately install it.

This may take a while depending on the speed of your Internet connection. Once you see *Success* as the JUnit plugin installation status, the plugin is ready to be used.

- If you click the Download Now and Install After Restart button, the plugin will be downloaded but not installed. The plugin will be installed after you restart the Jenkins server.

## The Installed Tab

This tab shows the list of all installed Jenkins plugins in your Jenkins server along with each plugin's installed version.

To uninstall an existing plugin, from the Installed tab, search for the plugin to be uninstalled by typing its name in the Search field. Check the checkbox available for the plugin and then click the Uninstall button.

Plugins can also be uninstalled by removing the corresponding .HPI file from the `JENKINS_HOME/plugins` directory.

*Disabling* a plugin is a way to retire a plugin. Jenkins will consider the plugin as installed, but it will not start it and any extensions contributed by plugin will not be visible.

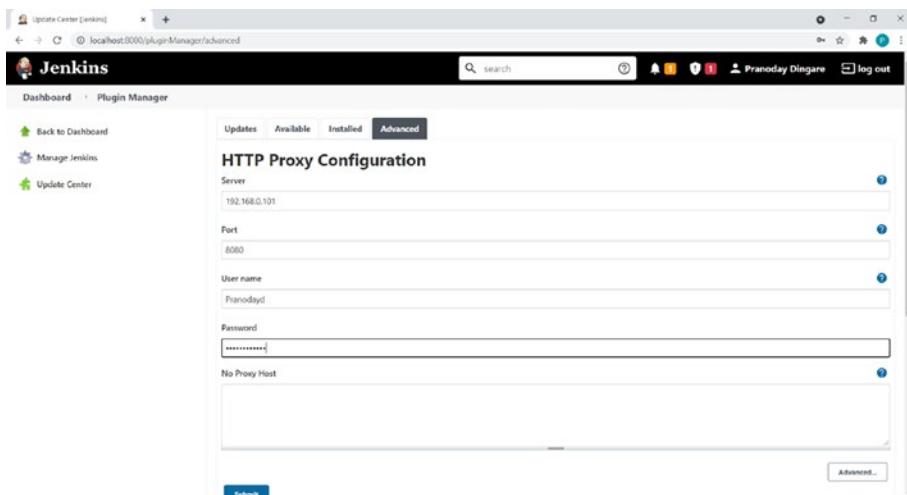
To disable a plugin, uncheck the checkbox on the Installed tab of Manage Jenkins page.

## The Advanced Tab

Internet access typically goes through and is controlled by proxy servers. Machines cannot make any direct Internet requests when a proxy server is configured. Internet requests should go to the proxy server and then reach the Internet.

If your organization has such proxy server, Jenkins cannot access the Internet while running any job. For example, accessing `gitlab.com` to pull the source code will not be allowed to the Jenkins instance. You will learn about Jenkins jobs in upcoming chapters. Jenkins cannot access the Update Center to download Jenkins plugins either. You need to configure proxy settings in Jenkins in order to work with Jenkins running behind the proxy.

You have to ask for details of the proxy server from your IT department and then configure those details in the HTTP Proxy Configuration section (see Figure 5-4).



**Figure 5-4.** HTTP Proxy Configuration page

Let's look at the different fields from this section one by one:

- **Server:** Enter the IP address of your proxy server in the Server field.
- **Port:** Enter the port of the proxy server machine.
- **Username:** If the proxy's server has authentication set, enter the username of those credentials in the Username field. If authentication is not set, keep this field blank.
- **Password:** If the proxy's server has authentication set, enter the password of those credentials in the Password field. If authentication is not set, keep this field blank.
- **No Proxy Host:** Specify the IP address or hostname patterns that should not go through the proxy. This setting is used to mention servers that can be accessed directly (without going through a proxy server).

After specifying all the required details, click the Submit button.

The Update Center only allows you to install the latest released version of a plugin. If you need a specific version of the plugin, you can download the .HPI archive of the plugin and install it manually by using the Upload Plugin section. Follow these steps:

1. Download the .HPI file of the required plugin version.
2. Browse the downloaded .HPI file by clicking the Choose File button.
3. Click the Upload button.

Jenkins plugins are downloaded from the Update Center URL, which is shown in the URL section.

## Troubleshooting Installation Problems

There are some common problems encountered during the Jenkins plugin installation. This section discusses such common problems encountered, their reasons, and their solutions.

### Problem 1

Jenkins Plugin Installation Failed: Downloaded file /var/lib/jenkins/plugins/ does not match expected SHA-

Error: While installing Jenkins plugins, it fails with the following SHA -1 error:

```
java.io.IOException: Downloaded file /var/lib/jenkins/plugins/*.tmp does not match expected SHA-1,  
expected 'f2ncNlydUUSPrk6SoG255v+2kQU=', actual  
'1ZRJco40uv1j0AG4Aet7HadHg/Q='
```

Reason: The Jenkins Update Center is not synced with Jenkins. You are running Jenkins on a machine which operates behind the proxy.

Solution for “Jenkins Update Center is not synced with Jenkins”:

1. Choose Manage Jenkins ➤ Manage Plugins and click the Advanced tab. Scroll down the page to find the Update Site section. Click the Check Now button. This will sync your Jenkins server with the Update Center.
2. Restart the Jenkins server and try to install the plugin again.

Solution for “You are running Jenkins on a machine which operates behind the proxy”:

1. If you are running Jenkins on a machine that operates behind the proxy, you have to configure the proxy machine settings by choosing Manage Jenkins ➤ Manage Plugins. On the Plugin Manager page, go to the Advanced tab.  
If the proxy machine does not have authentication set, leave the username and password fields blank.
2. Click the Advanced button, enter the Jenkins Update Center’s URL in the Test URL field, and then click the Validate Proxy button.
3. After a successful proxy validation, click the Submit button.
4. Try to install the plugin again. If it fails, then restart the Jenkins server and try again.

## Problem 2

```
javax.net.ssl.SSLHandshakeException: PKIX path building failed:  
sun.security.provider.certpath.SunCertPathBuilderException:  
unable to find valid certification path to requested target
```

While installing plugins, you get this error:

```
sun.security.provider.certpath.SunCertPathBuilderException:  
unable to find valid certification path to requested target
```

Reason: The Jenkins Update Center URL in Manage Jenkins ➤ Manage Plugins is set to an HTTPS URL by default, which must be accessed with additional security measures, like security certificates for example.

JDK, which we are using to run the Jenkins server, is bundled with lots of trusted Certificate Authority (CA) certificates. These certificates are in a file called *cacerts*, which is present in the lib\security folder inside the Java installation directory. This cacerts file does not have a certificate, which will trust the Jenkins Update Center website URL due to nonavailability of latest security certificates. This could be due to old version of Java being used to start the Jenkins server.

Solutions: There are various solutions to this problem.

Solution 1: Change the Update Center URL to use http:// instead of https://:

1. Choose Manage Jenkins ➤ Manage Plugins.
2. Go to the Advanced tab.
3. Scroll down the page to the end, where you will see the Update Site section.
4. Change the URL from <https://updates.jenkins.io/update-center.json> to <http://updates.jenkins.io/update-center.json> and click the Submit button.
5. Try to install the plugin again.

Solution 2: Update Java to the latest version:

1. Download and install the latest Java version available.
  2. Add a path to the `bin` folder from the latest installed Java in the `PATH` environment variable.
  3. Save this environment change and restart the Jenkins server.
  4. Try to install the plugin again.
- 

**Note** If you are updating Java to versions other than Java 1.8 or Java 1.11 then, when starting the Jenkins server using the `.WAR` file, you need to mention the `--enable-future-java` flag. For example, the command to start `Jenkins.war` is `java -jar D:\Jenkins\Jenkins.war --enable-future-java`.

---

Solution 3: Start the Jenkins server using the `.WAR` file from the `lib\security` folder in the latest Java installation folder.

1. Install the latest version of Java.
2. Start the command prompt and change the working directory to the Java Installation `dir\lib\security` folder.

If you have the latest JDK 15 installed on your machine at `D:\JDK15\jdk-15.0.2`, open the command prompt and change the working directory to `D:\JDK15\jdk-15.0.2\lib\security`.

3. Run the Jenkins.war file from D:\JDK15\jdk-15.0.2\lib\security.
4. Try to install the plugin again.

Use Solution 3 if other applications on your machine need a specific version of Java other than the latest one and hence you cannot update the PATH environment variable to point to the latest version of Java.

---

**Note** If you are using the latest version of Java to start the Jenkins server in order to resolve Problem 2, make sure you revert to the Jenkins supported Java versions (JDK 1.8 or JDK 1.11) after the plugin installation is done. If you continue using unsupported versions, you will encounter errors when running Jenkins pipeline jobs. (We cover pipeline jobs in upcoming chapters.)

---

## Summary

This chapter discussed how the Jenkins Plugin Manager helps you install/update/uninstall different Jenkins plugins. You also learned about errors that you might come across when installing Jenkins plugins, as well as the different solutions to resolve them. The next chapter discusses how to configure some commonly used tools like Maven, Git, and Java from the Global Tools Configuration page.

## CHAPTER 6

# Understanding the Global Tool Configuration Page

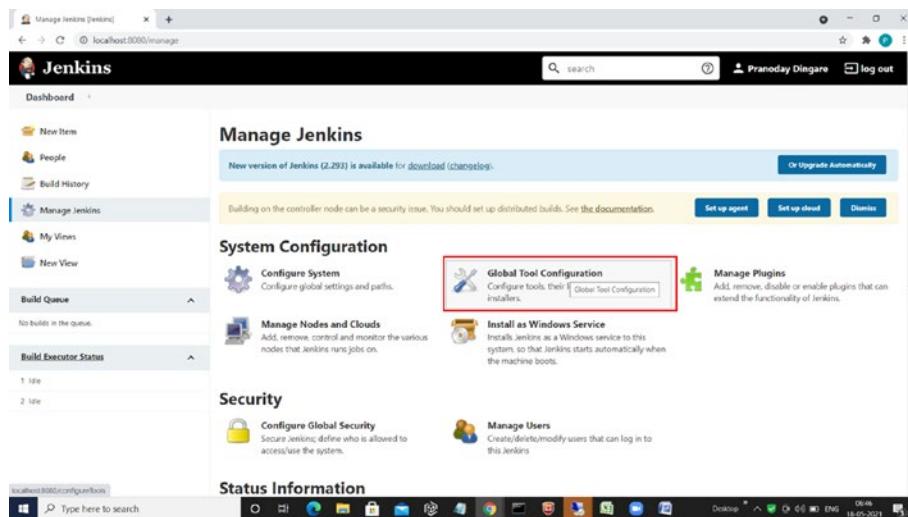
When working with Jenkins, you may need to use different tools/technologies like Java, Maven, etc. This chapter explains how to configure JDK and Maven from the Global Tool Configuration page.

## Global Tool Configuration Settings

Follow these steps to go to the Global Tool Configuration Settings page.

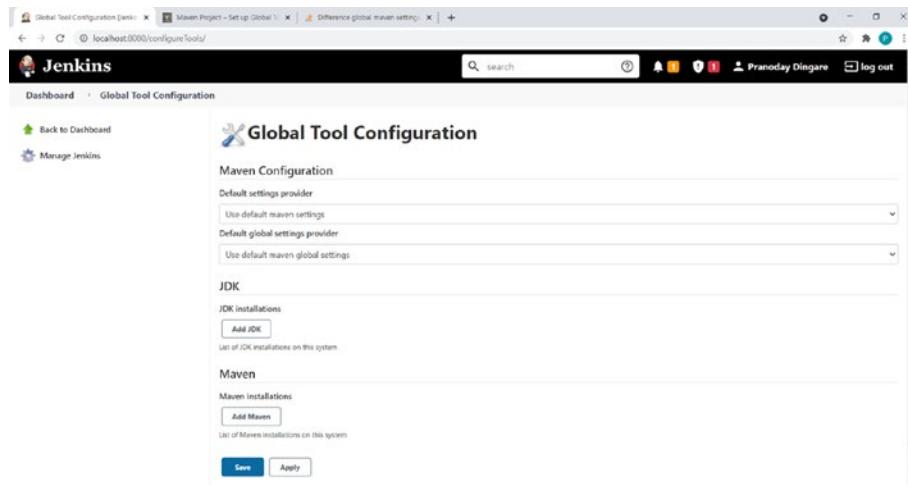
1. First, log in to Jenkins. Once you are logged in, you will see the Jenkins dashboard.
2. Click the Manage Jenkins link available on the Jenkins dashboard to open the Manage Jenkins page.
3. Go to the Global Tool Configuration page by clicking the Global Tool Configuration link highlighted in Figure 6-1.

## CHAPTER 6 UNDERSTANDING THE GLOBAL TOOL CONFIGURATION PAGE



**Figure 6-1.** The Manage Jenkins page

Clicking the Global Tool Configuration link will open the Global Tool Configuration page shown in Figure 6-2.



**Figure 6-2.** The Global Tool Configuration page

If you have plugins related to other third-party tools such as Gradle or Git installed in Jenkins, you will see settings related to these tools on this page.

## Understanding the Global Tool Configuration Settings

Let's look at the settings related to a build tool called Maven. Maven helps you automate different build phases of a Java application like compilation, packaging, and deploying. Maven is a command-line interface build tool that performs its tasks with the help of different plugins. Maven can be configured using the following two configuration files:

- **The Maven install (called the global settings):** The default location of this file is \${maven.home}\conf and the file is called `settings.xml`. `maven.home` refers to the Maven installation directory.
- **A user's install (called the user settings):** The default location of this file is \${user.home}/.m2 and this file is also called `settings.xml`. The `{user.home}` part refers to the current user directory. We cover Maven in detail in Chapter 11.

These files contain settings required to execute Maven in order to build different Java projects. If different users working on the same machine want to keep their specific Maven settings, they keep these settings in the user settings file, whereas all common Maven settings that will be shared by all user profiles go in the global settings file.

You don't have to have the Maven `settings.xml` in the user's install, i.e., \${user.home}/.m2/settings.xml. If this file is not present, the required settings are taken from the Maven install's `settings.xml` file. If the same setting is present in both `settings.xml` files, the setting in the \${user.home}/.m2/settings.xml gets preference.

## CHAPTER 6 UNDERSTANDING THE GLOBAL TOOL CONFIGURATION PAGE

While configuring Maven inside Jenkins, Jenkins needs to know the locations of the user and global `settings.xml` files. If these files are present at their default locations, keep the Use Default Maven Settings option selected in the Default Settings Provider field and the Use Default Maven Global Settings option selected in the Default Global Setting Provider field, as shown in Figure 6-3.



**Figure 6-3.** The Maven Configuration section

## Maven Configuration

If these files are present in different locations, select the Settings File in Filesystem option from the Default Settings Provider dropdown and the Global Settings File on Filesystem option from the Default Global Settings Provider dropdown. Then specify the path of these files, as shown in Figure 6-4.

## Maven Configuration

Default settings provider

Settings file in filesystem

File path

D:\UserSettings.xml

Default global settings provider

Global settings file on filesystem

File path

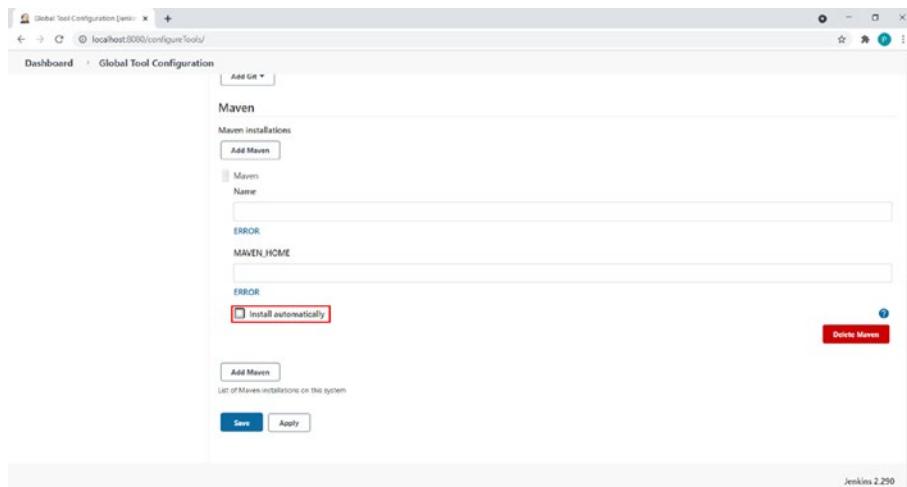
E:\CommonSettings.xml

**Figure 6-4.** Maven Configuration section with *settings.xml* paths

The second part that you need to understand related to Maven is its installation. Let's look at how you configure the Maven installations.

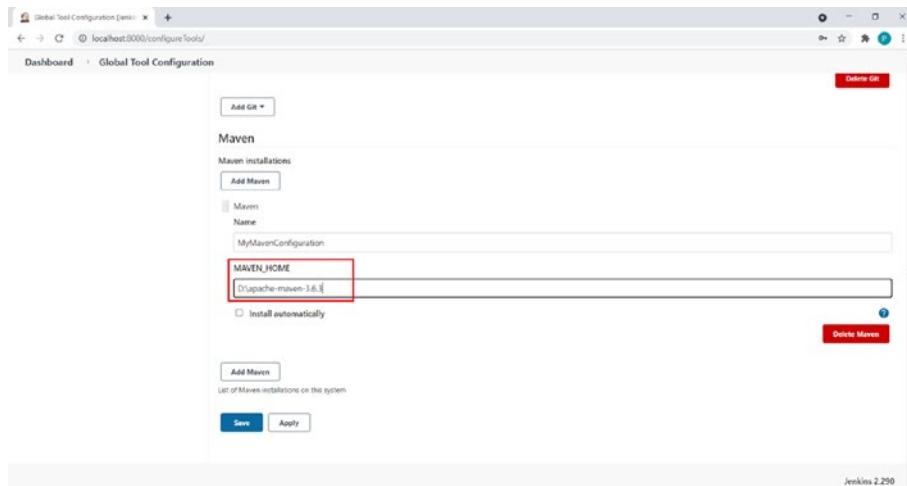
Click the Add Maven button. This will open the Maven Installations section. In the Name field, give the Maven configuration a name. Do not use numbers, whitespace, or special characters in the name, as Jenkins Pipelines wanting to use Maven refer this Maven configuration. (More on Jenkins Pipelines in Chapter 14.) If a machine running a Jenkins job already has Maven installed, then uncheck the Install Automatically checkbox, as highlighted in Figure 6-5.

## CHAPTER 6 UNDERSTANDING THE GLOBAL TOOL CONFIGURATION PAGE



**Figure 6-5.** Install Automatically checkbox from the advanced settings related to the Maven Installation section

In the MAVEN\_HOME field, provide the location of the Maven installation directory, as shown in Figure 6-6.



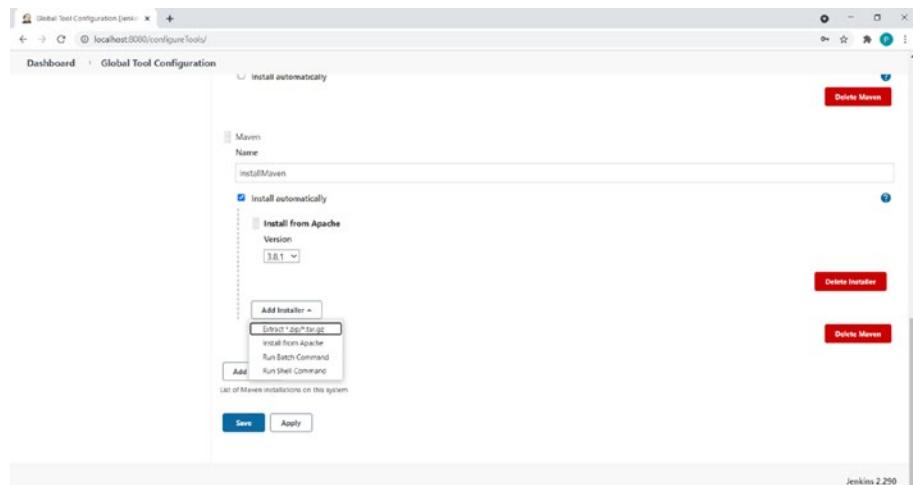
**Figure 6-6.** The Required fields filled in inside the Maven Installation section

Click the Save button to save the configuration.

If the machine running the Jenkins job does not have Maven installed and you want Jenkins to install it automatically while running a job, you have to check the Install Automatically checkbox and configure the installer. Jenkins will do this only the first time it runs a job on a machine where Maven isn't installed.

Let's see how to configure a Maven installation whereby a Maven .ZIP/.TAR file is extracted.

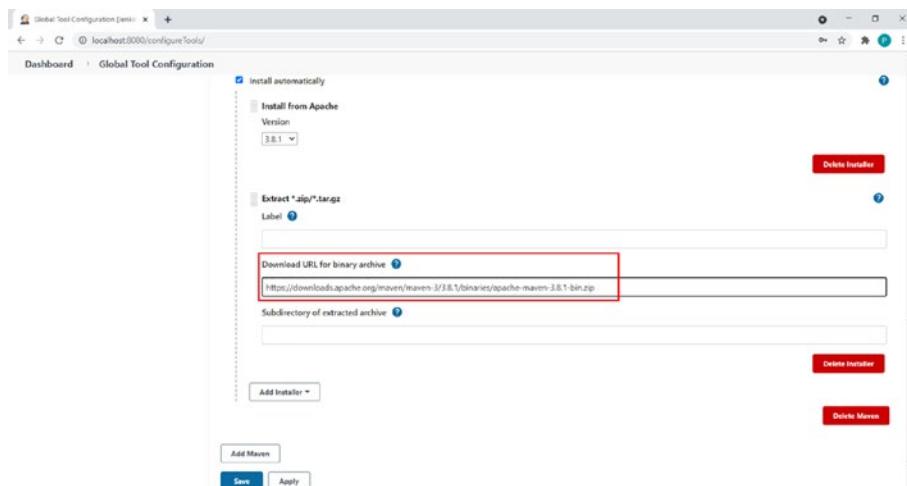
Click the Add Installer dropdown and select the Extract \*.zip/\*tar.gz option, as shown in Figure 6-7.



**Figure 6-7.** Configuring the Maven Installer

In the Download URL for Binary Archive field, enter the apache-maven-3.8.1-bin.zip URL from the Maven website, as shown in Figure 6-8. For example: <https://dlcdn.apache.org/maven/maven-3/3.8.2/binaries/apache-maven-3.8.2-bin.zip>.

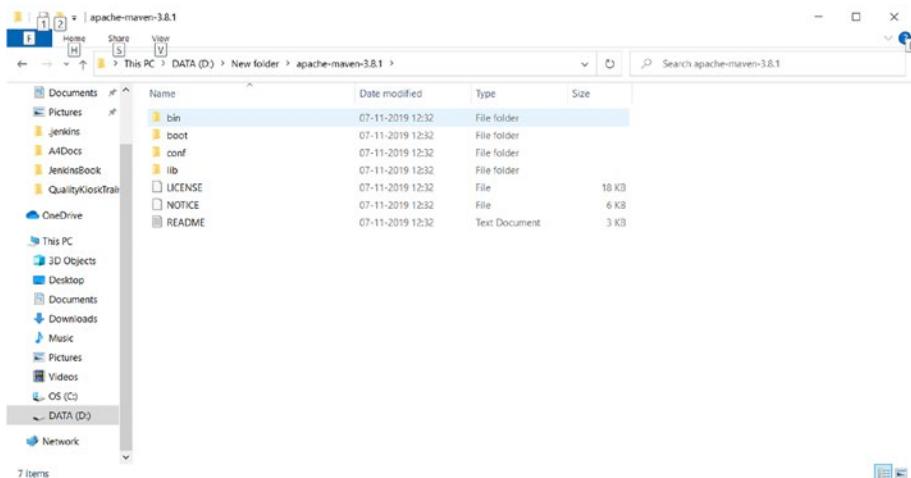
## CHAPTER 6 UNDERSTANDING THE GLOBAL TOOL CONFIGURATION PAGE



**Figure 6-8.** Maven download URL configured in Maven Installer

In the Subdirectory of Extracted Archive field, provide the name of the directory that will contain the Maven installation after unzipping the folder.

When Jenkins performs the Maven installation while running a Jenkins job that needs it, it will unzip the folder downloaded from the URL mentioned in the Download URL for Binary Archive field. Then Jenkins will unzip this .ZIP to provide the Maven build tool to the directory structure shown in Figure 6-9.



**Figure 6-9.** The directory structure of Maven

As shown in Figure 6-9, the Maven tool is inside a directory named apache-maven-3.8.1. So apache-maven-3.8.1 should be the value in the Subdirectory of Extracted Archive field. That way, Jenkins will consider this directory the Maven installation location and can access Maven using the mvn command present in the bin folder.

Click the Save button on the page to save the configuration.

## Java Configuration

Click the Add JDK button

If you already have Java on your machine, uncheck the Installed Automatically checkbox, provide a suitable name in the Name field, and include the absolute path of the Java installation directory in the JAVA\_HOME field, as shown in Figure 6-10.



**Figure 6-10.** The Name and JAVA\_HOME fields configured in the JDK Configuration section

If you do not have JDK installed and want Jenkins to install it on demand, then check the Installed Automatically checkbox and configure the installer as explained in the previous Maven configuration section.

## Summary

This chapter explained how to set up important tools like Maven and JDK. If any Jenkins job needs these tools and job execution machine does not have them, those jobs will fail. Jenkins installs these tools using configured installers, which is considered a very important feature of Jenkins. The next chapter talks about how to manage security in Jenkins using its different security related features to implement authentication and authorization in Jenkins.

## CHAPTER 7

# Managing Security with Jenkins

In the previous chapter, you learned how to set up Maven and JDK and integrate them with Jenkins.

This chapter talks about different security-related settings that help you configure the authentication and authorization features in Jenkins.

---

**Note** In simple terms, *authentication* means valid users can log into the Jenkins system and invalid users cannot. *Authorization* defines different kinds of accesses granted to the different types of users, such as admin/non-admin.

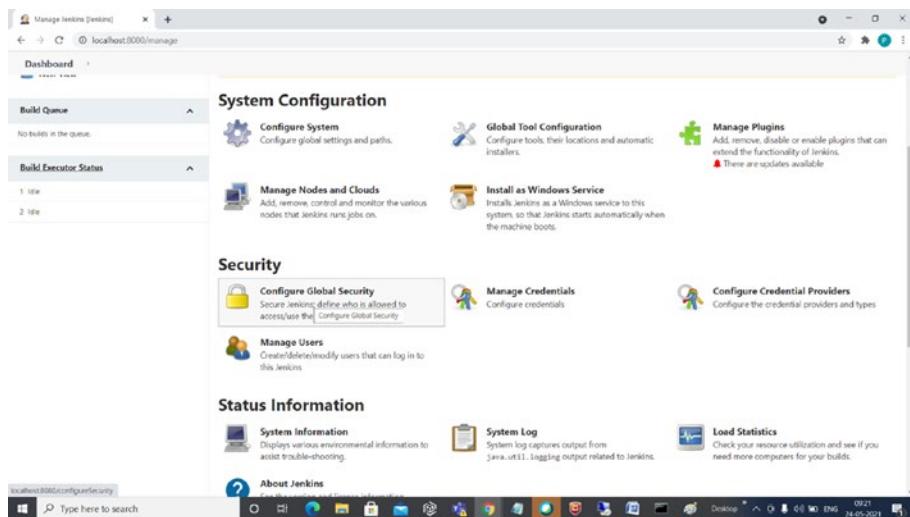
---

## Configuring Global Security in Jenkins

In this section, you work with the different settings on the Configure Global Security page. Log into Jenkins; you will see the Jenkins dashboard. Then use the following steps to go to the Configure Global Security page.

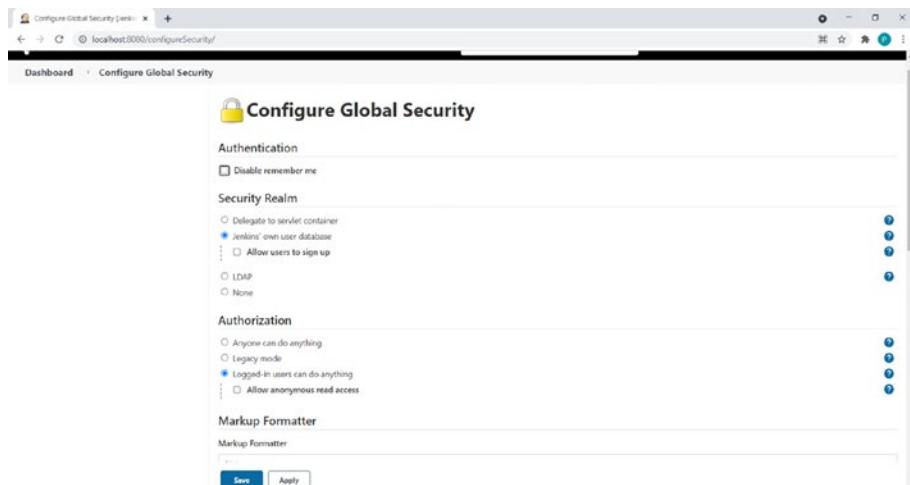
Click the Configure Global Security link available inside the Security section on the Jenkins dashboard to open the Configure Global Security page shown in Figure 7-1.

## CHAPTER 7 MANAGING SECURITY WITH JENKINS



**Figure 7-1.** The System Configuration page

Figure 7-2 shows the Configure Global Security page.



**Figure 7-2.** The Configure Global Security Page

Let's look at each of the settings from this page one by one:

- **Disable remember me:** This checkbox will be unchecked by default.

When you open the Login screen, it shows the Keep Me Signed In checkbox. If you keep this checked, Jenkins will not ask you to log in again when you exit without logging out.

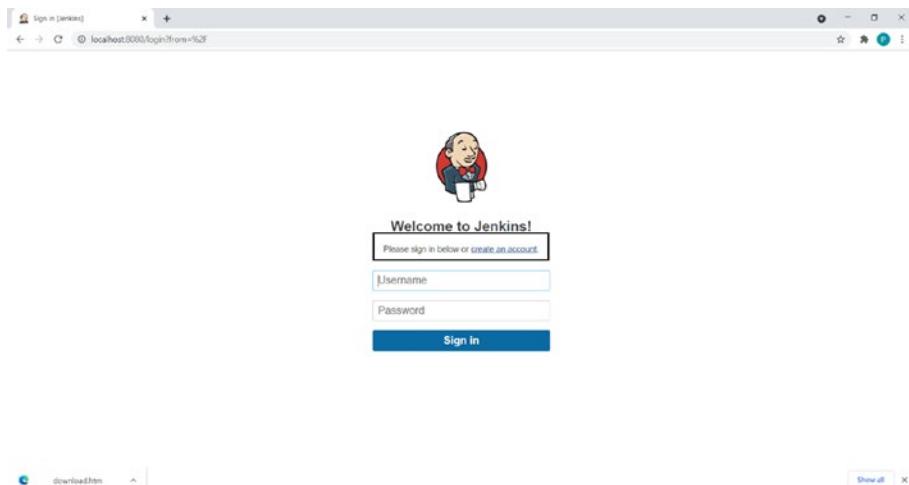
You can check the Disable Remember Me checkbox. If this checkbox is checked, the Login page will not show the Keep Me Signed In checkbox.

- **Delegate to servlet container:** Jenkins is a automation server that runs inside a Java servlet container like Jetty (which is the default servlet container used to run Jenkins server), tomcat, etc. If you want Jenkins to use users configured in these containers, then select this option.
- **Jenkins' own user database:** Jenkins allows you to create users and then maintains them in its own database. So if you don't want to rely on a third-party entity to give the list of users allowed to use Jenkins, you can create users and ask Jenkins to use it.

To create a new user, follow these steps:

1. Go to the Dashboard.
2. Choose Manage Jenkins ➤ Manage Users
3. Click the Create User link.
4. Enter all the details like username, password, confirm password, full name, and email address on the Create User page and click the Create User button. Now this user can be used to log in to Jenkins.

- **Allow users to sign up:** On the Configure Global Security page (see Figure 7-2), this checkbox is available under the Jenkin's Own User Database option. This checkbox is unchecked by default. If you check this checkbox, then the Create an Account link will be available on the Jenkins welcome page, as shown in Figure 7-3.



**Figure 7-3.** Jenkins login screen with the *Create an Account* link

Click on this link and fill in the required details on the Create an Account page to create an authenticated user.

It's not recommended that you keep the Allow Users to Sign Up checkbox checked, as anyone from your Jenkins server domain could then create a user account and become an authenticated user.

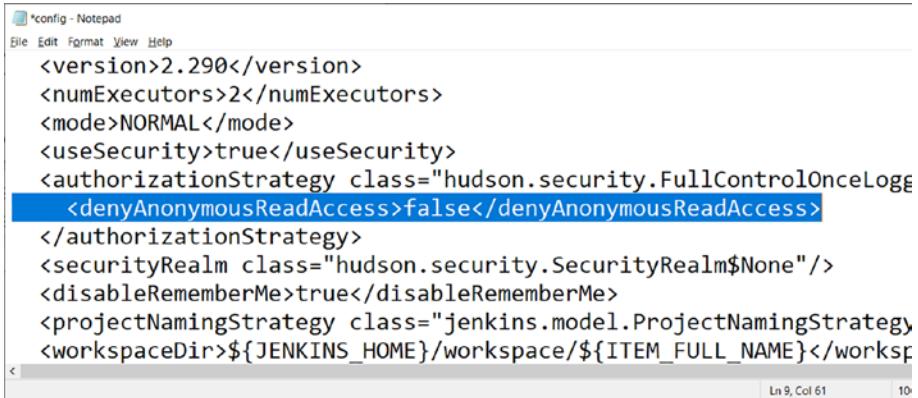
- **None:** On the Configure Global Security page (see Figure 7-2), if you select the None radio button from the Security Realms section, it will not ask for any authentication and the user will be treated as anonymous.

If you set this option and try to access Jenkins, you will get the following error in the Jenkins server logs:

```
anonymous is missing the Overall/Read permission
```

To allow anonymous user access, follow these steps:

1. Go to the \$Jenkins\_Home\config.xml file.
2. Change the value of the <denyAnonymousReadAccess> tag from true to false, as shown in Figure 7-4.



```
*config - Notepad
File Edit Format View Help
<version>2.290</version>
<numExecutors>2</numExecutors>
<mode>NORMAL</mode>
<useSecurity>true</useSecurity>
<authorizationStrategy class="hudson.security.FullControlOnceLoggedInAuthorizationStrategy">
    <denyAnonymousReadAccess>false</denyAnonymousReadAccess>
</authorizationStrategy>
<securityRealm class="hudson.security.SecurityRealm$None"/>
<disableRememberMe>true</disableRememberMe>
<projectNamingStrategy class="jenkins.model.ProjectNamingStrategy$StandardProjectNamingStrategy"/>
<workspaceDir>${JENKINS_HOME}/workspace/${ITEM_FULL_NAME}</workspaceDir>

```

**Figure 7-4.** The denyAnonymousReadAccess setting in the config file

3. Restart the server and access the Jenkins URL.

Now you will not be asked for authentication and will be taken to the Jenkins dashboard directly. But your access will be read only.

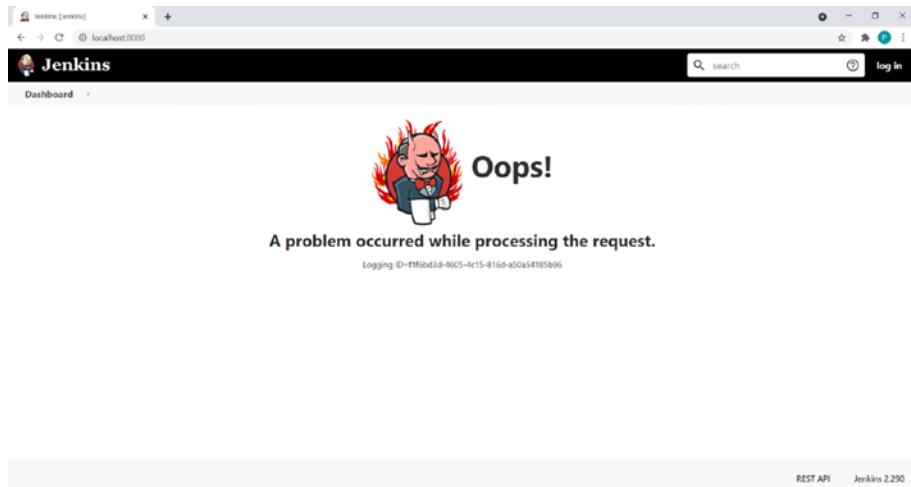
- **Anyone can do anything:** On the Configure Global Security page (see Figure 7-2), this option is present in the Authorization section.

If this option is selected, then everyone, including anonymous users who have not signed in, will get full control of Jenkins.

You may use this setting when you are using Jenkins through your company's intranet, which is a trusted environment. That way, users don't have to sign into the Jenkins system every time and this will allow for quick changes. Using this setting is generally not recommended.

- **Legacy mode:** In this mode, if the user has an Admin role, they will be granted full control over the system. Other users (not having the admin role), including anonymous users, will have read only access. Jenkins behaves by default before release 1.164. This option is not recommended.
- **Logged in user can do anything:** If this option is selected, every user must log into Jenkins. Logged in users will have full control whereas anonymous users will get read only access
- **Allow anonymous read access:** This checkbox is available under the Logged In Users Can Do Anything option. If this checkbox is checked, users who are not logged in (called anonymous users) will have read only access.

If this checkbox is unchecked, then anonymous users will not be able to access Jenkins. If they try to access Jenkins, they will see the screen shown in Figure 7-5.



**Figure 7-5.** The anonymous user is not able to access Jenkins

- **TCP port for inbound agents:** On the Configure Global Security page (see Figure 7-2), this option is available in the Agents section.

In the case of distributed builds (we are going to talk about distributed build in Chapter 17), you need to configure different Jenkins agent machines. Jenkins uses the port number mentioned in this setting to communicate with the Jenkins node (agent) machines.

- If you select the Random option, a random port number not being used by any other process is chosen dynamically.
- If you select the Fixed option, you need to check the available port number on your machine and provide it in the text box.

Usually people prefer the Fixed option over the Random option because a machine's firewall settings need to include the port number in the Inbound Rules, in order to allow the incoming communication requests through the specific port.

You are going to see the steps for creating inbound rules in the Windows firewall in the Jenkins Distributed Builds chapter (Chapter 17).

It's easy to create inbound rules if you use a Fixed port. Creating inbound rules for random ports is not possible, as Jenkins keeps on changing the port it will use to communicate with Jenkins agents. If you do not have any firewall restrictions, you can use the Random option, as you will not have to create a firewall rule in that case.

If you are not using distributed builds, select the Disable radio button.

- **CSRF protection:** CSRF (Cross-Site Request Forgery) protection uses a token that Jenkins creates based on user information and sends it to the user. If any form submission or any action results in any kind of modifications like changing build configurations, that token (called a crumb in Jenkins) must be provided. This token contains information identifying a particular user it was created for. So submissions done with another user's token would be rejected.

A *crumb* is a unique hash that gets created based on the following user-specific information:

- Username
- Web session ID
- IP address of the user's machine
- A salt unique to this Jenkins instance

Once the crumb is created, the user can use it to authenticate.

- **API Token:** This section of the configuration is used to configure the access to Jenkins when access is done through REST APIs exposed by Jenkins, CLI commands of Jenkins, or different applications. (We talk about this in more detail in Chapters [19](#) and [20](#).) Jenkins exposes various REST API endpoints as well as provides different CLI commands to perform things like triggering new builds, creating/copying existing jobs, etc.

If you check the Generate a Legacy API Token for Each Newly Created User checkbox, then Jenkins will create an API token for every user, which would be created in Jenkins database or using LDAP. When the Jenkins server is accessed through REST APIs, this token is used to authenticate the user.

If you check the Allow Users to Manually Create a Legacy API Token checkbox, the users can create their own legacy API token manually.

Both options are deprecated in Jenkins versions 2.129 and above. Jenkins recommends users create their own new API tokens which we discuss in Chapters [19](#) and [20](#).

- **Agent ➤ Controller security:** We will discuss this in Chapter [17](#).
- **SSH Server:** Jenkins has built-in commands that you can use to access Jenkins through shell programs. You can create batch files in the Windows environment and Shell scripts in the Linux environment to perform tasks like creating jobs, triggering builds, etc. using Jenkins CLI commands. We discuss Jenkins CLI in Chapter [20](#). During this kind of access, Jenkins can act as an SSH server, which will allow CLI access to the program wanting to access Jenkins server using its commands.

If you select the Random option, a random port number not being used by any other process is chosen dynamically. The server will use this port to listen to the incoming connections from the SSH client.

If you select the Fixed option, you need to check the available port number on your machine and provide it in the text box. Jenkins will use that port number to listen to incoming connection requests from SSH clients.

People typically prefer the Fixed option over the Random one, as configuring inbound rules on a Windows firewall for specific fixed port number is easy. Random numbers require you to open all the ports in the firewall, which is quite difficult. You learn the steps for creating inbound rules in a Windows firewall in Chapter [17](#).

If you are not going to access Jenkins server using Jenkins CLI commands, then it's better to select the Disable radio button.

## Configuring LDAP with Jenkins

LDAP (Lightweight Directory Access Protocol) is a software protocol that allows anyone to locate data about organizations, individuals, and other resources such as files, whether on the Internet or an intranet.

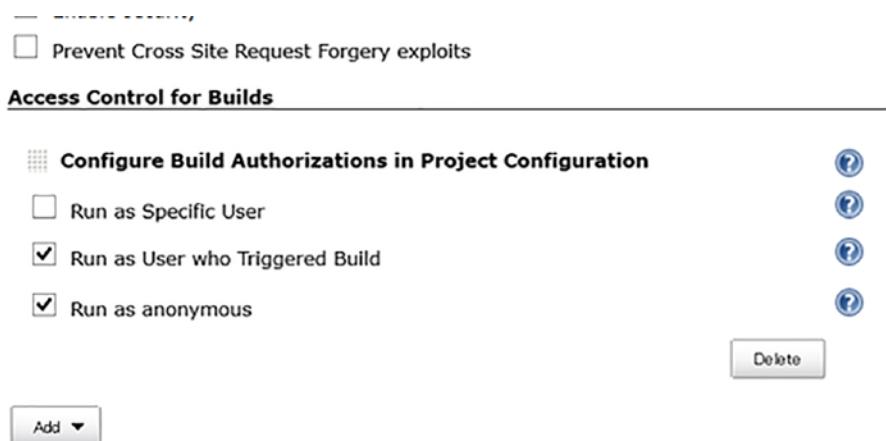
Common uses of LDAP include providing a central place of authentication, i.e., it stores usernames and passwords. LDAP can be used to validate usernames and passwords with Docker, Jenkins, etc.

## Why We Need to Configure LDAP with Jenkins

The Jenkins and LDAP integration is explained in this section. Usually in organizations, read/write access to directories/files or access to particular servers is given to a few domain users. Consider one practical scenario here. I am an employee of company named ABC and my company domain username is Pranodayd. The Pranodayd user has access to a server where builds are developed. Now say Jenkins is installed on a machine with a

user called `adminuser`. This user does not have access to that build server. A Jenkins job (a task that you ask Jenkins to do, discussed in Chapter 10) wanting to deploy an application on the build server will not be able to do so.

*Note that Jenkins runs the jobs using the operating system's logged-in user and our logged-in user does not have access to the build server.* We need to run the Jenkins job as a different user, `Pranodayd`. If we configure LDAP with the information of all domain users' usernames and passwords, we can log into Jenkins using our own domain username and password. Once we log into Jenkins using the domain's username and password, we can ask Jenkins to access a user who has triggered a build. We do this using an authorized project plugin. We would then be able to handle this case. Figure 7-6 shows the required setting in the Authorize Project plugin.



**Figure 7-6.** The settings for the Authorize Project plugin

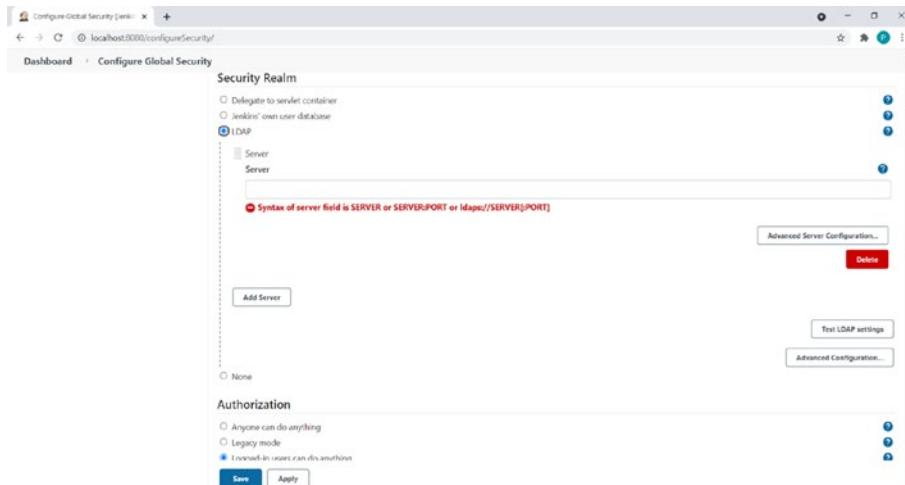
We don't discuss using this plugin in this book. I just wanted to make a point here that sometimes you may need Jenkins to access users from the Active Directory (LDAP) rather than from Jenkins's own user database.

## How to Configure LDAP with Jenkins

To configure LDAP with Jenkins, you need to install the LDAP plugin in Jenkins first. Choose Dashboard > Manage Jenkins > Manage Plugins. Then go to the Available tab and type **LDAP** in the Search field.

Select the checkbox for the LDAP plugin and click the Install Without Restart button. The LDAP plugin installation will start. Wait until the installation status shows success. Go back to the Dashboard by clicking the Back to Dashboard link.

To configure the LDAP plugin, choose Manage Jenkins > Configure Global Security. You will see LDAP radio button under the Security Realm section. Select the LDAP radio control, as shown in Figure 7-7.



**Figure 7-7.** The LDAP configuration setting on the Configure Global Security page

Contact your IT department to complete the LDAP setup in Jenkins.

## Summary

This chapter explained the different settings found on the Configure Global Security page. You also learned about the need to integrate Jenkins with LDAP. With knowledge of the topics discussed in this chapter, you can configure the required security settings in Jenkins.

## CHAPTER 8

# Managing Credentials

Jenkins, as a CI/CD automation server, needs to access different third-party tools—like Nexus artifact repository, Git code repository, etc.—to perform its CI/CD tasks. These artifact and code repositories implement different kinds of authentications, including basic authentication, where users are authenticated using their usernames and passwords, SSH authentication, where users are authenticated by matching private and public keys, and API token-based authentication, where users are authenticated based on an API token sent along with the access request. Jenkins needs to provide the required authentication information when accessing these tools. If a tool that Jenkins is trying to access has basic authentication, Jenkins needs to provide a username and password. If the tool's authentication is set to SSH, then Jenkins needs to provide a private key, and so on.

Information required for authentication is called the *credentials* in Jenkins. You can save this information in a Jenkins instance by creating credential entries. This chapter explains how to create different kinds of credentials in Jenkins.

## Understanding Credentials in Jenkins

Credentials are composed of authentication information that's stored in Jenkins. You can use this information to connect to different third-party tools through Jenkins.

By creating different credential entries in Jenkins, you can store authentication information in a more secure way. Information stored in Jenkins in the form of credentials can be shared among different Jenkins jobs as well. You do not need to specify username-passwords or private keys in the pipeline code in the human readable text. Each credential entry must have a unique Credentials ID. With this Credentials ID, you can use the stored authentication information in your Jenkins jobs.

## Creating Credential Entries in Jenkins

There are different kinds of authentication techniques used by tools, such as Git, Nexus etc., to authenticate the user. The following authentication techniques are mainly used:

- Basic authentication
- SSH authentication
- API token
- Certificate

The Jenkins Credentials plugin allows you to create credentials of different types in order to store the required authentication information. Once the credential entry has been created, you can refer to it by using the Credentials ID in the Jenkins job/pipeline with the help of the Credentials Binding plugin. (You learn about the Credentials Binding plugin in Chapter 14.) In this section, you see how to create different types of credential entries to store different kinds of authentication information.

## Understanding Scope and Domains

Before we look at the steps needed to create credential entries in Jenkins, we have to understand the following two important concepts which play an important role in credentials management in Jenkins.

- **Scope:** When you create a credentials entry, you have to mention the scope. Scope defines where a particular credentials entry is available for usage. There are two types of scope:
  - **Global:** A credential defined with Global scope is available for use in all Jenkins jobs created, as well as to the Jenkins server that works as a system.
  - **System:** A credential defined with System scope is available only to the Jenkins instance to perform system administration functions like email authentication, agent connections, etc. This credential is not available for use in Jenkins jobs. Jenkins jobs are series of steps implemented to automate a build lifecycle. You learn about Jenkins jobs in Chapter [10](#).
- **Domain:** Domain is a way to group credentials used to access similar systems. By default, credential entries are created in the default domain. Say that you have ten credential entries, three of which are used to access GitLab code repositories. The other seven credential entries are to access other systems like AWS. When you are configuring access to your GitLab code repositories in your Jenkins job, all ten credential entries are shown in the credentials list. It could be difficult to choose the correct credential entry from the available ones.

To solve this problem, you can create a domain called “GitLab Credentials” and place the three credential entries under that domain. Now when you will configure access to your code repository from GitLab, you will only see the three credential entries instead of all ten. Choosing the right one would be easy in that case.

## Creating Credential Entries in Jenkins

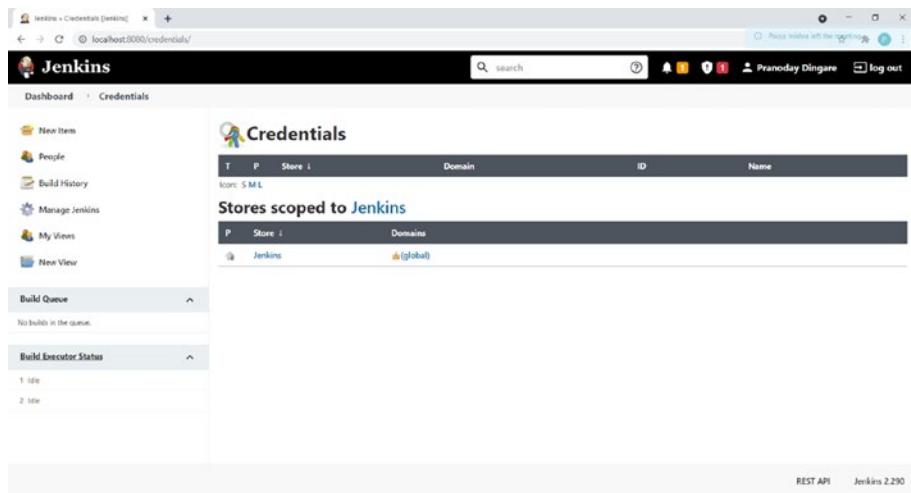
This section shows you how to create different types of credential entries in the default domain, in a different domain, and in two different scopes: global and system.

We are also going to discuss the implications of the availability of the credential entries after creating them in different domains and different scopes, by looking into the configuration of a Jenkins Job. Jenkins jobs and the steps to create them are discussed in the next few chapters.

### Creating a Credentials Entry in a Global Domain (Default Domain) and a Global Scope

1. Go to the Manage Jenkins page by clicking the Manage Jenkins link from the Jenkins dashboard.
2. Click the Manage Credentials link.

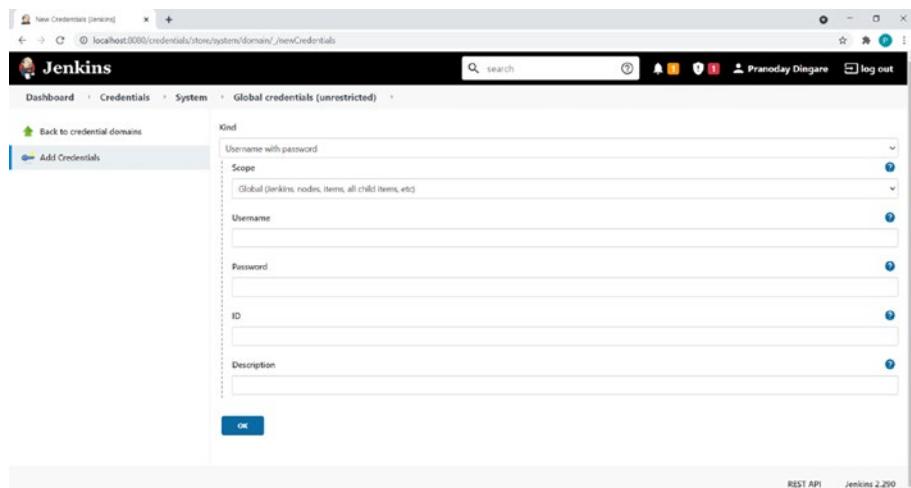
This will open the Credentials page shown in Figure 8-1.



**Figure 8-1.** The Credentials page

3. Click the (global) link in the Domain column inside the Stores Scoped to Jenkins section. This will open the Global Credentials (Unrestricted) page.
4. Click the Add Credentials link shown on the left side to open the page shown in Figure 8-2.

## CHAPTER 8 MANAGING CREDENTIALS



**Figure 8-2.** The Credentials Creation page

5. In the Kind field dropdown, select a type of a credential entry you want to create, such as Username with Password, which is used for Basic Authentication, SSH Username with private key for SSH Authentication, and so on.

Let's create a credential entry for Basic Authentication by keeping the username with the password option selected.

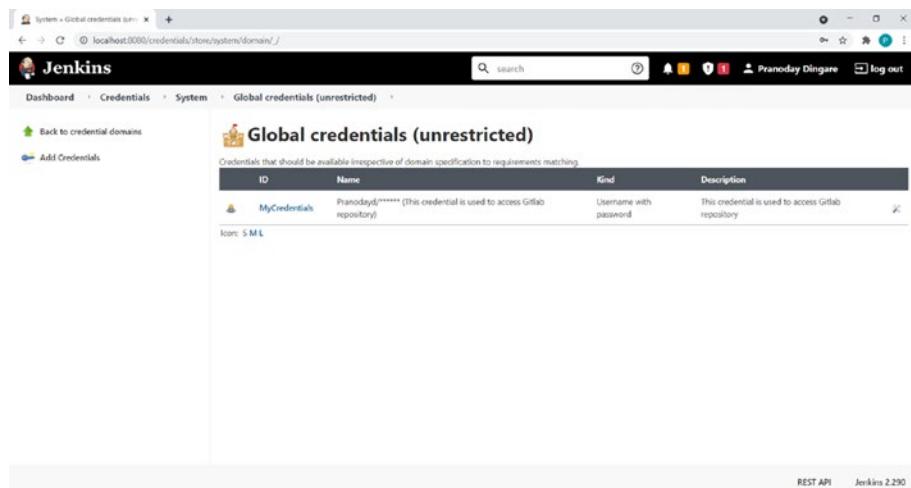
6. In the Scope field, there are two options available—Global (Jenkins, nodes, items, all child items, etc.) and System (Jenkins and nodes only).

Keep the Global option (Jenkins, nodes, items, all child items, etc.) selected

7. In the Username field, enter the username used to authenticate the user on a tool/platform you want to connect to. For example, if you are creating a credentials entry to connect to a Gitlab.com from Jenkins, you would enter the username of your GitLab account.
8. In the Password field, enter the password used to authenticate the user on a tool/platform you want to connect to. For example, if you are creating a credentials entry to connect to Gitlab.com from Jenkins, then you would enter the password of your GitLab account.
9. Enter any string in the ID field. This is how the credential is referenced in the Jenkins jobs. You may keep it blank in which case a random unique ID will be generated by Jenkins.
10. Enter any string in the Description field. You may keep this field blank as well.
11. Click the OK button to save the credentials entry.

Once the credentials entry is saved, it is shown on the Global Credentials (Unrestricted) page, as shown in Figure 8-3.

## CHAPTER 8 MANAGING CREDENTIALS



**Figure 8-3.** The Global Credentials (Unrestricted) page

This credentials entry you just created in the global scope and global domain will be shown in the Credentials list available page, which you can use to configure a Jenkins job. Chapter 10 explains how to get to this page to configure jobs.

---

**Note** You are going to learn how to create different types of jobs in upcoming chapters.

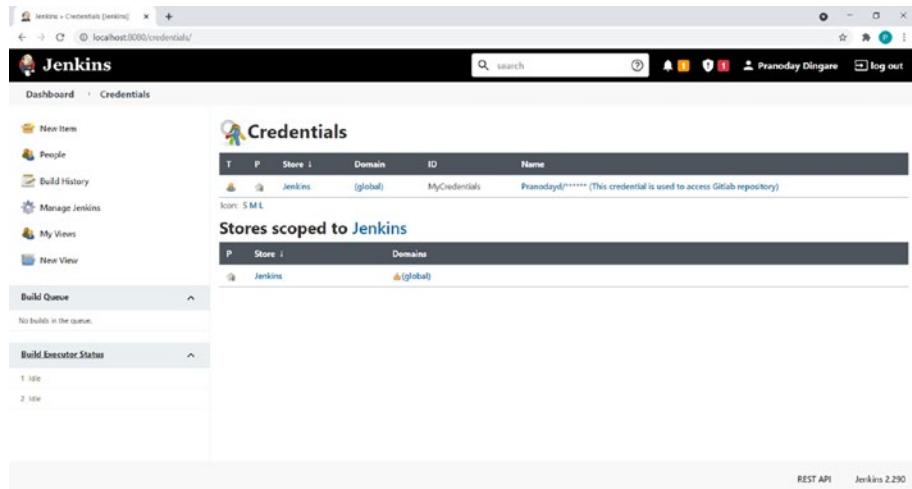
---

## Updating a Credentials Entry

This section explains how to update a credential entry. As an example, you will see how to update the scope of the entry you just created from global to system.

1. Go to the Manage Jenkins page by clicking the Manage Jenkins link on the Jenkins dashboard.
2. Click the Manage Credentials link.

This will open the Credentials page showing the entries (see Figure 8-4).



The screenshot shows the Jenkins interface with the title bar "Jenkins > Credentials [jenkins]". The left sidebar includes "Dashboard", "New Item", "People", "Build History", "Manage Jenkins", "My Views", and "New View". Under "Build Queue" and "Build Executor Status", there are no builds listed. The main content area is titled "Credentials" and displays a table with one row:

T	P	Store	ID	Name	
		Jenkins	(global)	MyCredentials	Pranodayd/********* (This credential is used to access GitLab repository)

Below this table, a section titled "Stores scoped to Jenkins" lists:

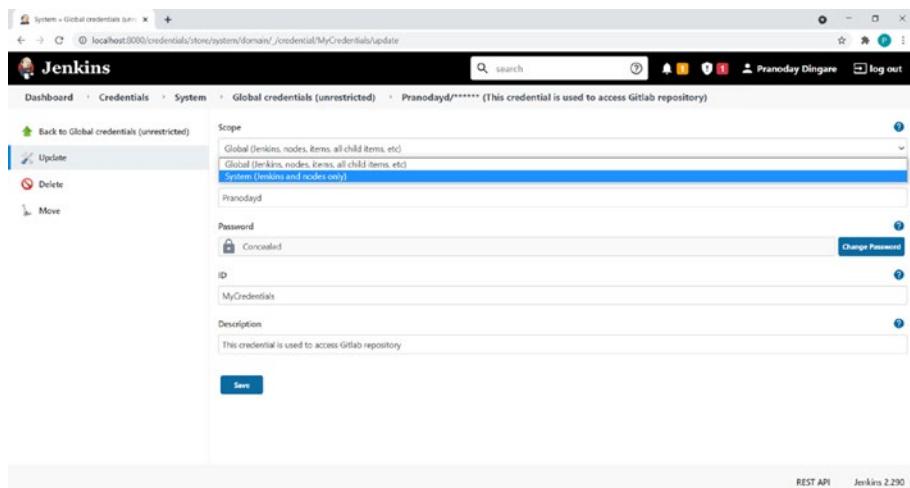
P	Store	ID	Domain
	Jenkins		(global)

At the bottom right of the page, it says "REST API Jenkins 2.290".

**Figure 8-4.** The created credentials entry on the Credentials page

3. Click the name of the credentials entry shown in the Name column. In this case, the name of the credentials entry is Pranodayd/\*\*\*\*\*\*\*\*\* (This credential is used to access the GitLab repository).
4. Click the Update link shown on the left.
5. Select the System (Jenkins and nodes only) option in the Scope field (Figure 8-5).

## CHAPTER 8 MANAGING CREDENTIALS

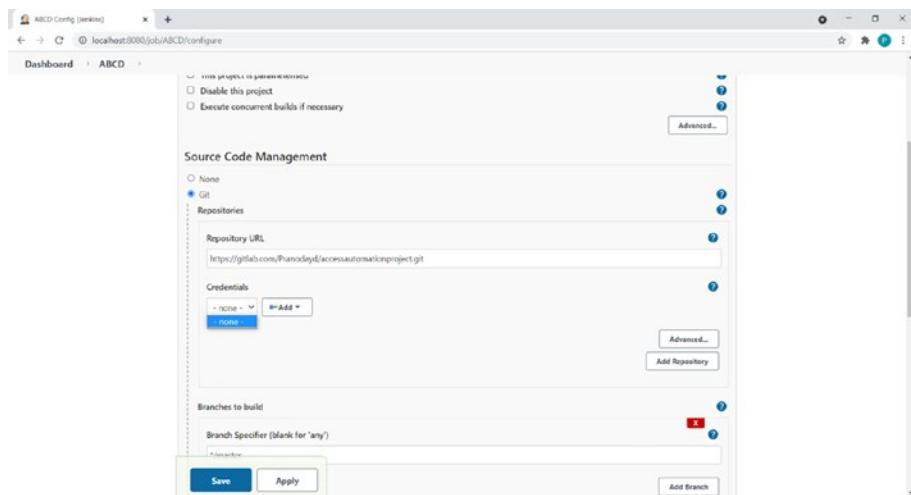


**Figure 8-5.** The Scope field selection

6. Click the Save button.

This entry's scope has been changed from global to system.

Figure 8-6 shows the effect of this change on the credential list in the Jenkins job.



**Figure 8-6.** The credential entry is not shown inside the Jenkins job any longer

The reason that this credential entry is not shown in the dropdown is that the credential scope has been set to system, so it now will be available for use only by the Jenkins system and its nodes.

## Creating a Credentials Entry in a Particular Domain

Use the following steps to create a credentials entry in a different domain (that is, a non-default domain).

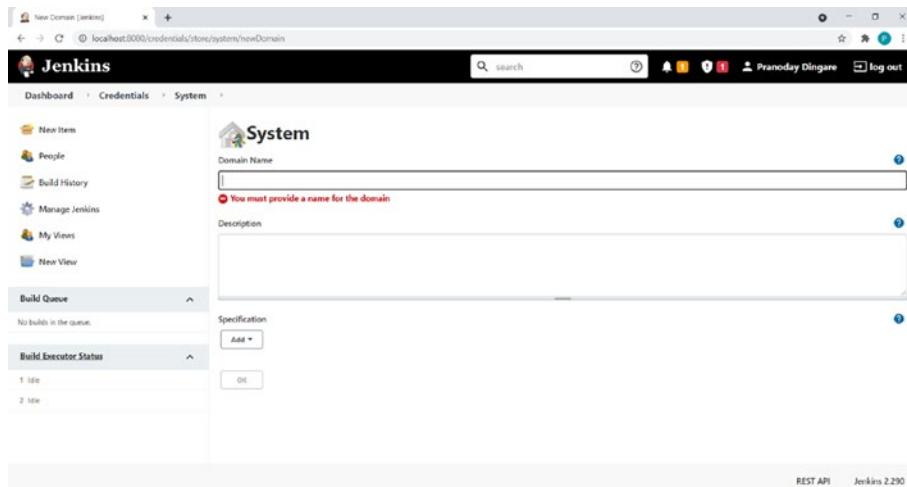
1. Go to the Manage Jenkins page by clicking the Manage Jenkins link on the Jenkins Dashboard.
2. Click the Manage Credentials link.

This will open Credentials page.

Note that if you are creating credentials entry for the first time, you will not see any entries on this page.

## CHAPTER 8 MANAGING CREDENTIALS

3. Click Add Domain from the Jenkins dropdown in the Store column shown under the Stores Scoped to Jenkins section. This will open the Domain page (see Figure 8-7).

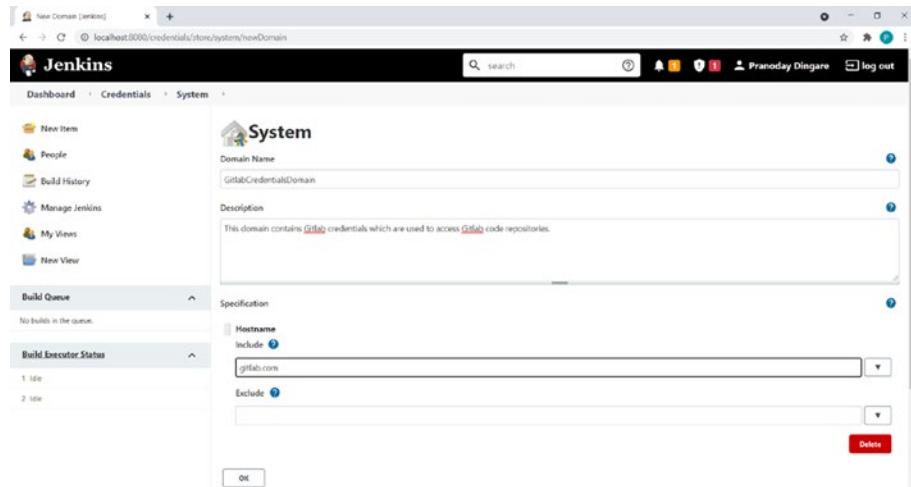


**Figure 8-7.** The Domain page

4. In the Domain Name field, enter a short name to identify the domain of the credentials you are creating.
5. In the Description field, enter a short description on what this domain is going to contain. This field is optional.
6. The Specification field allows you to configure the parameters based on which Jenkins would filter the Credential entries.

Let's look at this with an example. Say you want to create a domain to hold credential entries that would only be available to Jenkins jobs that want to connect to Gitlab.com.

To configure this, select the Hostname option in the Specification dropdown. Then enter **gitlab.com** in the Include field (see Figure 8-8).



**Figure 8-8.** The hostname is configured for Gitlab.com

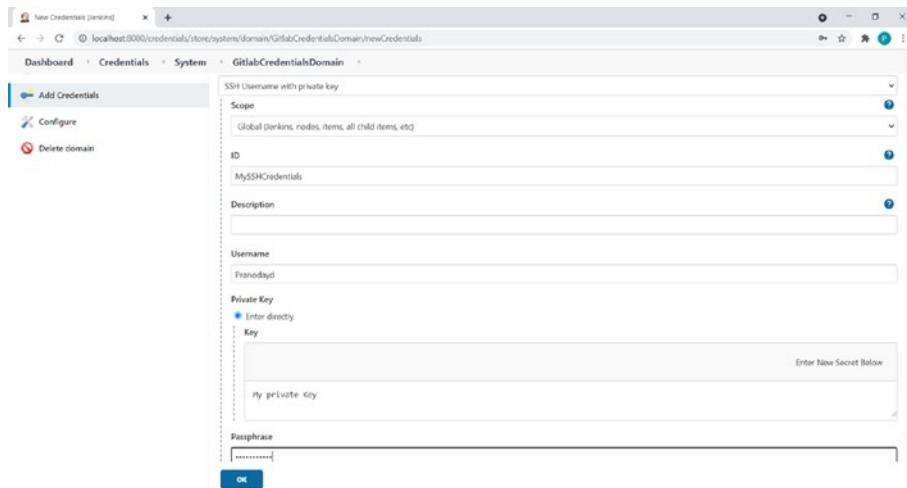
Click the OK button. The new domain will be shown when the page opens after clicking the OK button.

Let's now add the credentials to the new domain:

1. Click the Add Credentials link shown on the left.
2. We'll create an SSH credentials entry this time.  
Select the SSH Username with private key option in the Kind field.
3. Keep the Global (Jenkins, nodes, items, all child items etc.) option selected.
4. Enter any string in the ID field. There is no need to enter anything in the Description field.
5. In the Username field, enter the username, which is set as part of the authentication in the system you want to connect to—for example, your GitLab username.

## CHAPTER 8 MANAGING CREDENTIALS

6. Select the Enter Directly radio button and paste the private key that was created.
7. Enter a passphrase if any in the Passphrase field (see Figure 8-9).



**Figure 8-9.** Creating a credential entry of type SSH Username with Private Key

8. Click the OK button.

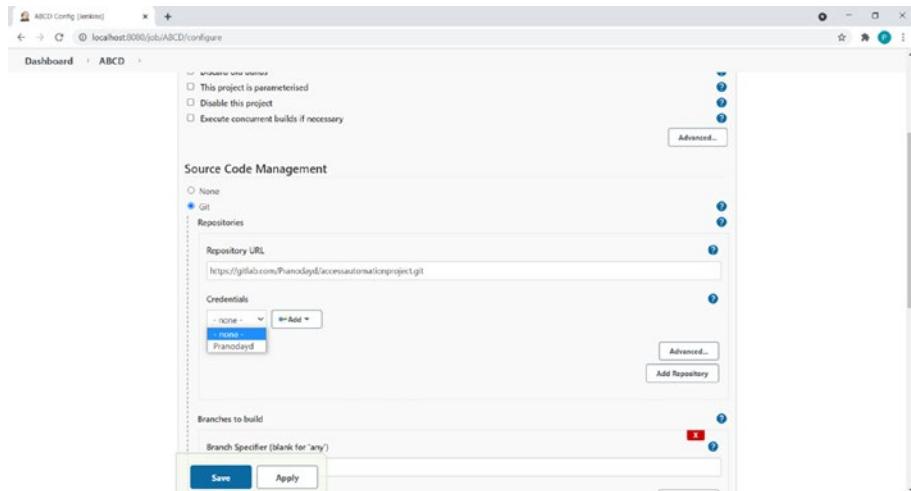
---

**Note** Creating SSH authentication credentials in Jenkins first requires you to create public and private keys using the `ssh-keygen` command. We cover creating public and private keys in upcoming chapters. The focus here is mainly on understanding how to create a key in a specific domain.

---

After you click the OK button, the credentials entry is created in the GitlabCredentialsDomain.

Figure 8-10 shows how to use this new credentials entry from GitlabCredentialsDomain in a Jenkins Job.

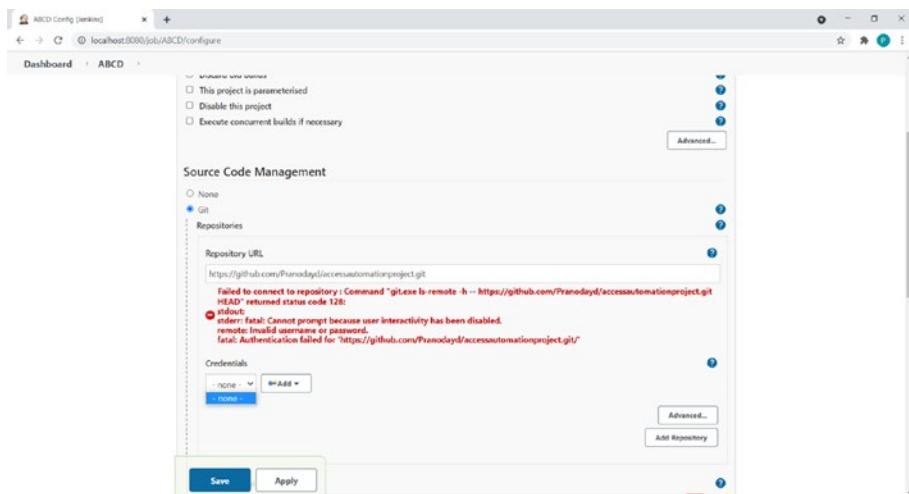


**Figure 8-10.** Using the credential entry in a Jenkins job

In my Jenkins job, I mentioned the URL from Gitlab.com and the Credentials dropdown is showing the credentials entry we just created in the GitLab Credentials domain.

Recall that when you created the GitLab Credentials domain, you specified `gitlab.com` as the hostname for the Specification field. Now consider this interesting point. If you change the URL in the Repository URL field to have `Github.com` instead of `Gitlab.com`, something amazing happens (see Figure 8-11).

## CHAPTER 8 MANAGING CREDENTIALS



**Figure 8-11.** The credential entry is not listed in the Credentials dropdown

Ignore the error for now, as we are going to talk about it when we discuss integrating Jenkins and Git in upcoming chapters. Note that the new credentials entry is not listed in the Credentials dropdown. That's because it is part of a domain created for the `Gitlab.com` hostname and not `Github.com`.

Superb feature, isn't it?

## Configuring a Credentials Provider

In Jenkins you can configure different types of credentials.

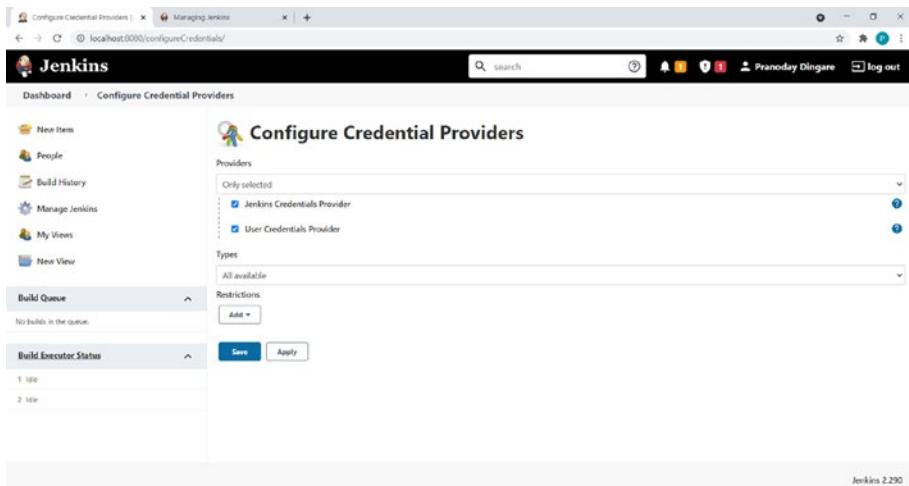
1. Click the Manage Jenkins ➤ Configure Credential Providers link.

This will open the Configure Credentials Provider page.

2. It will show two dropdowns—Providers and Types—with the default selected value as All available (see Figure 8-12).

Let's look closer at these two settings.

- **Provider:** Select the Only Selected option in the Providers dropdown, which will show two options—Jenkins Credentials Provider and User Credentials Provider. See Figure 8-12.



**Figure 8-12.** The Configure Credential Providers page

- **Jenkins Credentials Provider:** This provider will provide credentials from the Jenkins root and will allow you to create Global and System types of credentials in Jenkins (default domain), as well as specific domains (non-default).

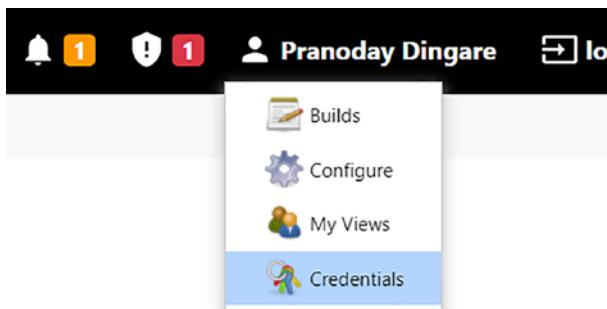
Global types of credentials are provided to all jobs from all users.

## CHAPTER 8 MANAGING CREDENTIALS

- **User Credentials Provider:** Credentials created in the User Credentials Provider are only available to the jobs created and triggered by the user who created the credential.

Follow these steps to create credentials in the User Credentials Provider:

1. Click the <UserName> ► Credentials link (see Figure 8-13).



**Figure 8-13.** The Credentials link in the menu opens after clicking the username

In my case, I click the PranodayDingare ► Credentials link.

2. Click the link provided in the Store column in the Stores Scoped to <UserName> table (see Figure 8-14).

Stores scoped to User: Pranoday Dingare		
P	Store 1	Domains
User: Pranoday.Dingare	<a href="#">(global)</a>	

**Figure 8-14.** The credential entry in stores scoped to User: Pranoday Dingare

I will click the User:PranodayDingare link and this will open the user page shown in Figure 8-14.

3. Click the Global Credentials (Unrestricted) link to open the Global Credentials (Unrestricted) page shown in Figure 8-15.

The screenshot shows a Jenkins web interface. At the top, there's a navigation bar with links for 'Dashboard', 'Pranoday Dingare', 'Credentials', 'User', and 'Global credentials (unrestricted)'. Below the navigation is a search bar and a 'log out' button. The main content area has a title 'Global credentials (unrestricted)' with a brief description: 'Credentials that should be available irrespective of domain specification to requirements matching...'. A table header is visible with columns for 'ID', 'Name', 'Kind', and 'Description'. A note at the bottom says 'This credential domain is empty. How about adding some credentials?' with icons for 'S' (Small), 'M' (Medium), and 'L' (Large). At the bottom right, there are 'REST API' and 'Jenkins 2.290' links.

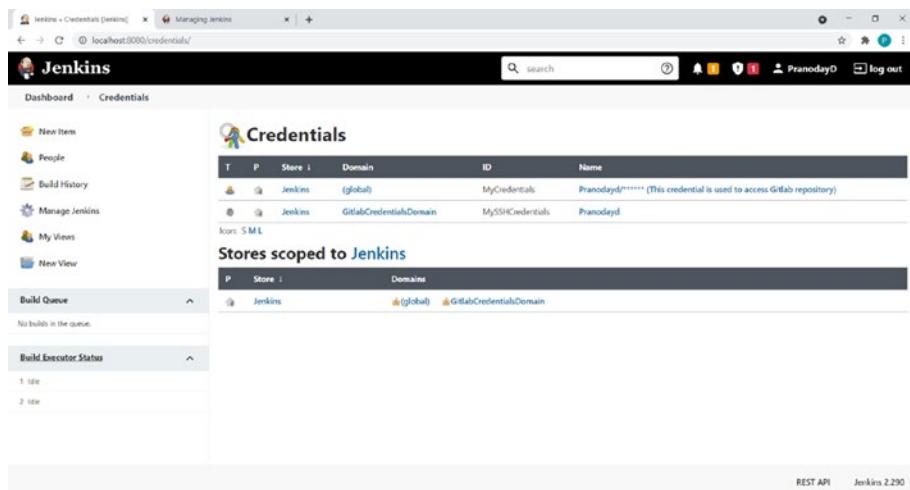
**Figure 8-15.** The Global Credentials (Unrestricted) page

4. Click the Add Credentials link.

After this, the rest of the steps are the same as in the previous sections. I created a new credentials entry by following the steps already discussed.

If I log out and back in as a different user, I will not see the credentials I just created, but I can see the credentials we created in the previous sections (which were from the Jenkins Credentials Provider). See Figure 8-16.

## CHAPTER 8 MANAGING CREDENTIALS



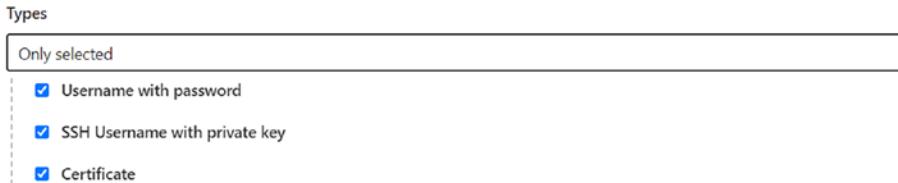
The screenshot shows the Jenkins web interface with the title "Jenkins > Credentials [jenkins]". The main content area is titled "Credentials" and displays two entries in a table:

T	P	Store	I	Domain	ID	Name
J	J	Jenkins	(global)		MyCredentials	Pranodayd/***** (This credential is used to access Gitlab repository)
J	J	Jenkins	GitlabCredentialsDomain		MySSHCredentials	Pranodayd

Below the table, there is a section titled "Stores scoped to Jenkins" which lists the Jenkins store and its global and domain domains.

**Figure 8-16.** The credential entries created in the Jenkins Credentials Provider

- **Types:** Select the Only Selected option in the Types dropdown, which will show the options in Figure 8-17.



The screenshot shows a configuration page for "Types". A dropdown menu is open, showing the option "Only selected". Below the dropdown, three items are listed, each with a checked checkbox:

- Username with password
- SSH Username with private key
- Certificate

**Figure 8-17.** The only options that are selected

Here you can configure which types of credential entries you want to be able to create. If you uncheck the Username with Password checkbox and click the Save button, this option is not shown in the Kind dropdown. The Username with Password option is not available either.

## Summary

This chapter covered how to create credentials with different scopes and different domains. You also saw how to configure the Credentials Provider to create different types of credential entries. Understanding the concepts in this chapter will help you create Jenkins jobs that need access to third-party tools. The next chapter explains how to manage users and their access rights in Jenkins.

## CHAPTER 9

# Managing Users

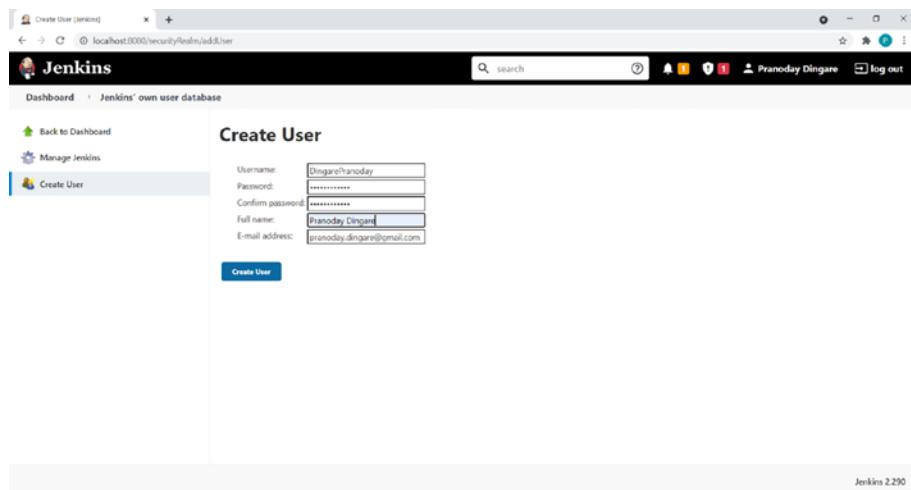
Jenkins is used by the different members of a team. A few team members will work as system administrators so they will have all rights and privileges required to manage the whole system, whereas others will have the least possible access to the system and can only view Jenkins jobs and execute them. This chapter explains how to create different users in Jenkins and how to assign them different rights based on their roles.

## Creating Users in Jenkins

Follow these steps to create users in Jenkins.

1. Go to the Manage Jenkins page on the Jenkins dashboard and then click the Manage Users link  
I already have two users created.
2. Click the Create User link shown on the left side of page.
3. Enter the details in the Username, Password, Confirm Password, Full Name, and E-mail Address fields, as shown in Figure 9-1.

## CHAPTER 9 MANAGING USERS



**Figure 9-1.** The Create User screen with the required fields filled in

4. Click the Create User button.

The new user will be seen in the list of users (see Figure 9-2).

User ID	Name	Action
DingarePranoday	DingarePranoday	
pd	pd	
pranodayd	pranodayd	

**Figure 9-2.** All users available in Jenkins

# Assigning Roles to Users in Jenkins

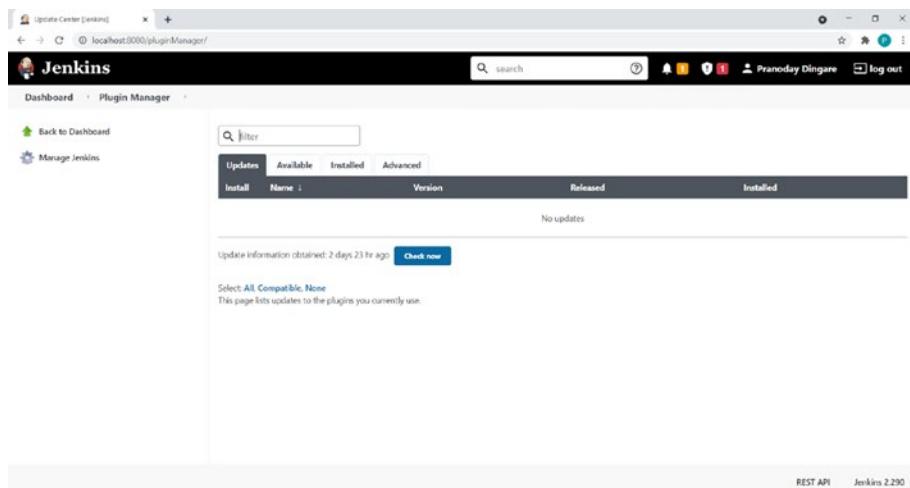
Before you can assign roles to your users, you have to create roles and then assign different rights (i.e., accesses) to these roles. To create different roles, you need to install the Role-Based Authorization Strategy plugin.

## Installing the Role-Based Authorization Strategy Plugin

Let's first install Role-Based Authorization Strategy Plugin in the Jenkins instance. Follow these steps to install this plugin.

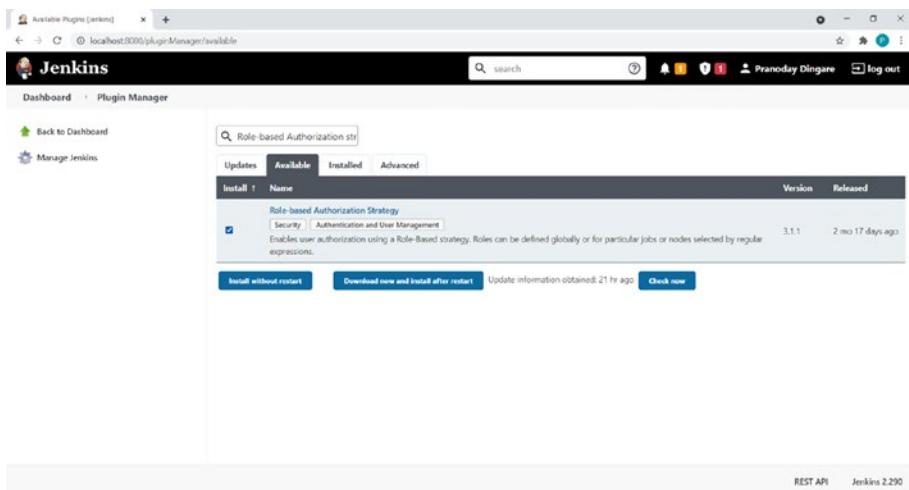
1. Log into Jenkins: Log in with the credentials of your Jenkins administrator. Once you are logged in, you will see the Jenkins dashboard.
2. Go to the Plugin Installation Manager: Click the Manage Jenkins link on the Jenkins dashboard and then click the Manage Plugins link to go to the Plugin Installation Manager (Figure 9-3).

## CHAPTER 9 MANAGING USERS



**Figure 9-3.** The Plugin Installation Manager

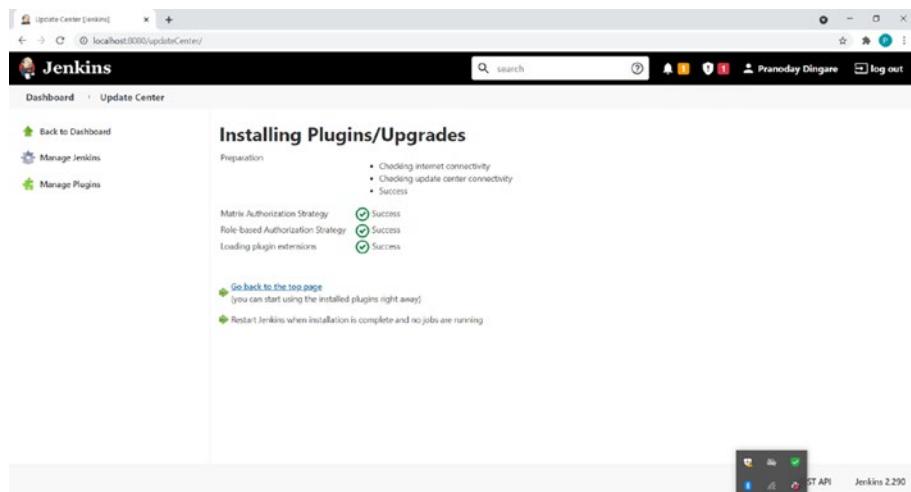
3. Install the plugin: Go to the Available tab and type **Role-based Authorization Strategy plugin** into the Search field. This will filter out other options from the list of plugins and will show the Role-Based Authorization Strategy plugin at the top of the plugins list.
4. Click the checkbox to select the plugin, as shown in Figure 9-4.



**Figure 9-4.** The Role-based Authentication Strategy plugin entry is selected

5. Select the plugin and click the Install without Restart button, which will start the plugin installation.
6. Wait until Jenkins finishes installing the plugin and shows the Success status, as shown in Figure 9-5.

## CHAPTER 9 MANAGING USERS



**Figure 9-5.** The plugin has been successfully installed

## Enabling Role-Based Strategy in Jenkins

After installing the plugin, go to the Manage Jenkins ➤ Configure Global Security link. This will open the Configure Global Security page. You will see the Role-Based Strategy option under the Authorization section (see Figure 9-6).

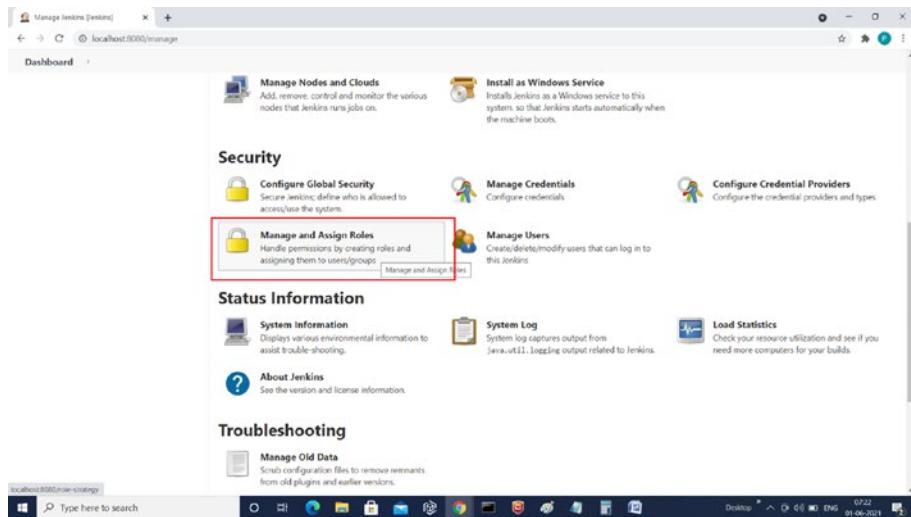
A screenshot of the Jenkins "Configure Global Security" page. The "Authorization" section is highlighted. It contains several options: "Anyone can do anything" (radio button), "Legacy mode" (radio button), "Logged-in users can do anything" (radio button, selected), "Allow anonymous read access" (checkbox checked), "Matrix-based security" (radio button), "Project-based Matrix Authorization Strategy" (radio button), and "Role-Based Strategy" (radio button, highlighted with a red border). The "Role-Based Strategy" option is the one being referred to in the figure caption.

**Figure 9-6.** The Role-Based Strategy option

Click the Role-Based Strategy option. Then click the Save button.

# Creating User Roles in Jenkins

1. Go to the Manage and Assign Roles page.
2. Click the Manage Jenkins link. You will see the new Manage and Assign Roles link under the Security section.
3. Click the Manage and Assign Roles link highlighted in Figure 9-7.

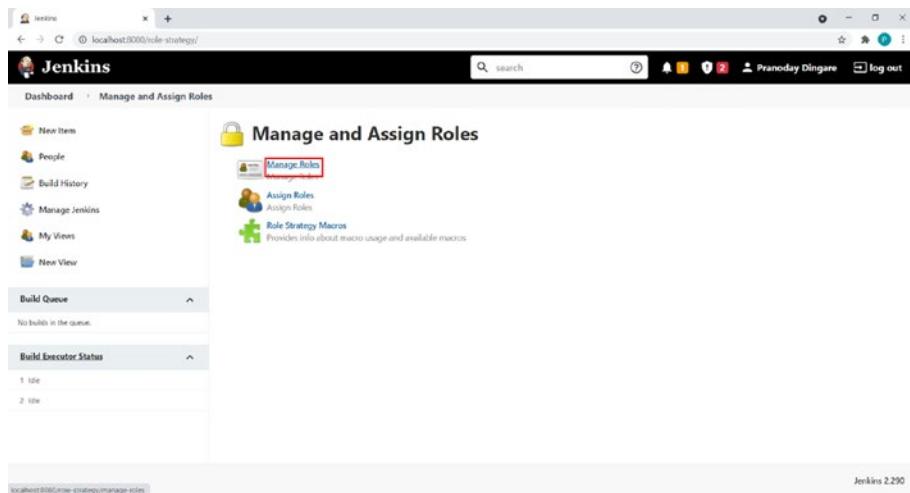


**Figure 9-7.** The Manage and Assign Roles link on the Jenkins dashboard

This will open the Manage and Assign Roles page.

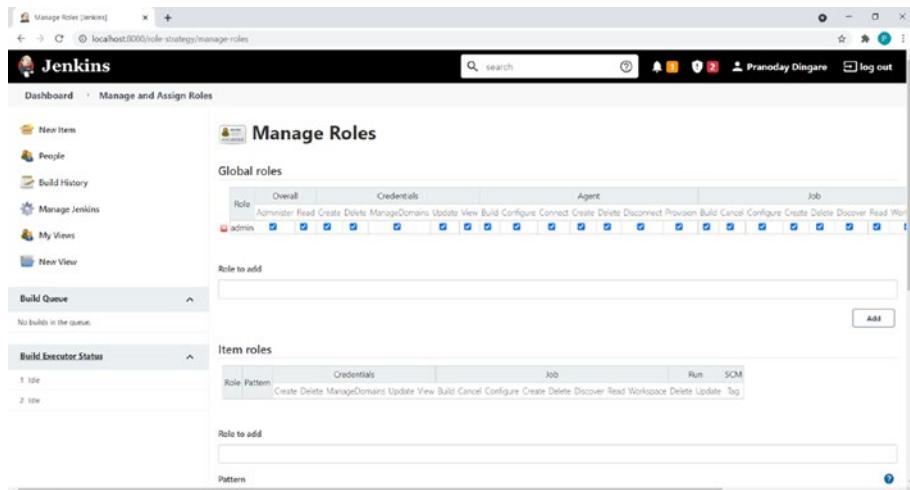
4. To create the role, click the Manage Roles link on the Manage and Assign Roles screen, as highlighted in Figure 9-8.

## CHAPTER 9 MANAGING USERS



**Figure 9-8.** The Manage and Assign Roles screen

This will open the Manage Roles page, as shown in Figure 9-9.



**Figure 9-9.** The Manage Roles screen

5. Enter a name for the role in the Role to Add field and click the Add button.

This will create a new role entry with the name specified in the Global Roles table. I used the name *View* for the role name.

6. Click the appropriate checkboxes under each section to assign the required rights to the role.

I select all the checkboxes under the View section (see Figure 9-10), as I want to create a view-only user.



The screenshot shows a table titled 'Global Roles'. The columns are 'Role', 'Overall', 'Credentials', 'Agent', 'Job', 'Run', 'View', and 'SCM'. Under the 'View' column, the 'View' role has all checkboxes checked, while the 'admin' role has only the 'Read' checkbox checked. Other columns like Overall, Credentials, Agent, Job, Run, and SCM have checkboxes for both roles.

Role	Overall	Credentials	Agent	Job	Run	View	SCM
admin	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Delete <input checked="" type="checkbox"/> Manage Domains <input checked="" type="checkbox"/> Update <input checked="" type="checkbox"/> View <input checked="" type="checkbox"/> Build <input checked="" type="checkbox"/> Configure <input checked="" type="checkbox"/> Connect <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Delete <input checked="" type="checkbox"/> Disconnect <input checked="" type="checkbox"/> Provision <input checked="" type="checkbox"/> Build <input checked="" type="checkbox"/> Cancel <input checked="" type="checkbox"/> Configure <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Delete <input checked="" type="checkbox"/> Discover <input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Workspace <input checked="" type="checkbox"/> Delete <input checked="" type="checkbox"/> Update <input checked="" type="checkbox"/> Configure <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Delete <input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Tag						
View	<input type="checkbox"/> Read <input type="checkbox"/> Create <input type="checkbox"/> Delete <input type="checkbox"/> Manage Domains <input type="checkbox"/> Update <input type="checkbox"/> View <input type="checkbox"/> Build <input type="checkbox"/> Configure <input type="checkbox"/> Connect <input type="checkbox"/> Create <input type="checkbox"/> Delete <input type="checkbox"/> Disconnect <input type="checkbox"/> Provision <input type="checkbox"/> Build <input type="checkbox"/> Cancel <input type="checkbox"/> Configure <input type="checkbox"/> Create <input type="checkbox"/> Delete <input type="checkbox"/> Discover <input type="checkbox"/> Read <input type="checkbox"/> Workspace <input type="checkbox"/> Delete <input type="checkbox"/> Update <input type="checkbox"/> Configure <input type="checkbox"/> Create <input type="checkbox"/> Delete <input type="checkbox"/> Read <input type="checkbox"/> Tag						

**Figure 9-10.** All checkboxes are checked under the view section

7. Scroll down the page to find the Save button and click it.

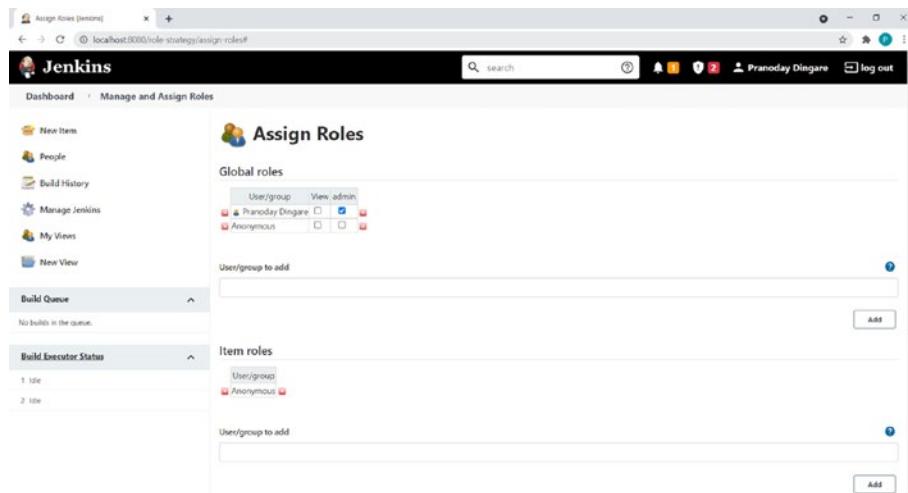
## Assigning Roles to Users in Jenkins

Once the role is created, you need to assign the role to the respective user. Follow these steps to assign a role to a user.

1. Go to the Assign Roles page.
2. Click the Assign Roles link on the Manage and Assign Roles page.

This will open the Assign Roles page (see Figure 9-11).

## CHAPTER 9 MANAGING USERS



**Figure 9-11.** The Assign Roles screen

3. Enter the user's ID into the User/Group to Add field and click the Add button.

My users in my Jenkins system are shown in Figure 9-12.

A screenshot of the Jenkins 'Users' screen. The left sidebar includes 'Back to Dashboard', 'Manage Jenkins', and 'Create User'. The main content area is titled 'Users' and contains a table with three rows. The table has columns for 'User ID' and 'Name'. The rows are:

User ID	Name
dingarepranoday	dingarepranoday
pd	pd
pranodayd	pranodayd

Each row has a small circular icon with a red slash next to it. At the bottom right of the page, it says 'Jenkins 2.290'.

**Figure 9-12.** Three users currently available in my Jenkins system

Out of these three, I want to assign a newly created role to the *pd* user.

I will enter pd into the User/Group to Add field from and click the Add button. This will create a user entry in the Global Roles table.

4. To assign a View role, click the checkbox in the View column, from the pd user row.
5. Page down to find to the Save button and click it.

## Checking the Assignment of a Role to a User

To see if the role was successfully assigned to the user, you can log in with the credentials of the user to whom the View role was assigned.

I tried to log in with the credentials of the *pd* user, which shows an Access Denied message.

## Creating Project-Based Roles in Jenkins

In Jenkins, you can create different jobs to perform different CI/CD operations in your application. If you want to restrict the access of a particular user to only few jobs, you can create a project-based role and assign it to that user.

Follow these steps to a create project-based role:

1. Go to the Manage Roles page. Click the Manage Jenkins ➤ Manage and Assign Roles ➤ Manage Roles link to open the Manage Roles page.
2. Create a new role to access specific jobs. Say you want to create a role that will allow access only to testing jobs that run unit testing and e-e testing.

3. Under the Item Roles section, enter a name for the role in the Role to Add field and .\*Testing in the Pattern field (see Figure 9-13). This will allow access to the jobs that have the word *testing* in them, such as unittesting, e-e testing, etc.

The screenshot shows the Jenkins Item Roles configuration page. At the top, there's a navigation bar with tabs: Role, Pattern, Credentials, Job, Run, and SCM. Below the tabs are buttons for Create, Delete, ManageDomains, Update, View, Build, Cancel, Configure, Discover, Read, Workspace, Delete, Update, and Tag. A sub-navigation bar below the main one has buttons for Create, Delete, ManageDomains, Update, View, Build, Cancel, Configure, Create, Delete, Discover, Read, Workspace, Delete, Update, and Tag. The main content area is titled 'Item roles'. It has two input fields: 'Role to add' containing 'TestingOnlyRole' and 'Pattern' containing '.\*Testing'. There's also an 'Add' button at the bottom right.

**Figure 9-13.** The value entered in the Pattern field

4. Click the Add button.

This will add an entry to the Item Roles table.

5. Give all rights to the job by checking all the checkboxes in the Job column for newly created role.
6. Scroll down the page to the Save button and click it to save the changes.

## Assigning Project-Based Roles to Users

Let's assign this project-based role to the user now:

1. Go to Assign Roles page. Click the Manage Jenkins ➤ Manage and Assign Roles ➤ Assign Roles link, which will open the Assign Roles page.

2. Under the Item Roles section, In the User/Group to Add field, enter the user ID of the user. Click the Add button.
3. Let's assign the TestingOnlyRole to the dingarepranoday user. I have entered dingarepranoday in the User/Group to Add field and clicked the Add button.

This will create a dingarepranoday user entry in the Item Roles table.

4. Click the TestingOnlyRole checkbox in the dingarepranoday user's row.
5. Under the Global Roles section, in the User/Group to Add field, enter **dingarepranoday** and click the Add button.
6. Assign the view role to this user, as this role contains an Overall type of access in its definition.
7. The dingarepranoday user now has two roles assigned—the View role from Global roles and the TestingOnlyRole from the Item Roles section (see Figure 9-14). Note that if users do not have overall access, then they will not be able to see anything on the dashboard.

## CHAPTER 9 MANAGING USERS

Global roles

User/group	View	admin	
Pranoday Dingare	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Anonymous	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
dingarepranoday	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

User/group to add

Item roles

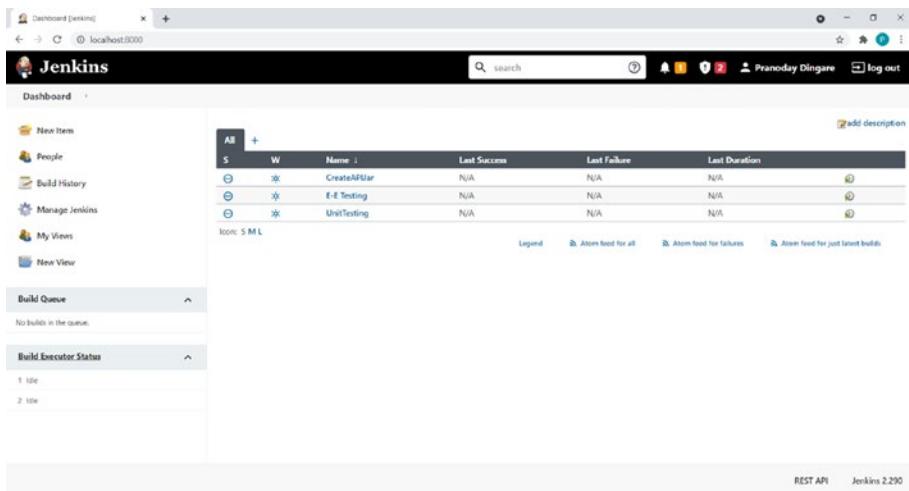
User/group	TestingOnlyRole	
Anonymous	<input type="checkbox"/>	
dingarepranoday	<input checked="" type="checkbox"/>	

**Figure 9-14.** The dingarepranoday user is assigned two roles

8. Scroll down the page and click the the Save button.

## Verifying the Assignment of the Project-Based Role to the User

In this section, you see how this project-based role works. I have created a few Jenkins jobs (see Figure 9-15).



**Figure 9-15.** Jenkins showing the list of jobs created

There are three jobs—called CreateAPIJar, E-E Testing, and UnitTesting and I am currently logged in as user PranodayDingare. Let's log in with the credentials of the dingarepranoday user.

If you log in as this user, you can see that dingarepranoday is able to see only the jobs that end with the word *testing* (see Figure 9-16).

## CHAPTER 9 MANAGING USERS

S	W	Name	Last Success	Last Failure	Last Duration
	*	E-E Testing	N/A	N/A	N/A
		UnitTesting	N/A	N/A	N/A

**Figure 9-16.** Showing only jobs ending with the word “testing”

## Understanding Matrix-Based Security in Jenkins

The previous section discussed how to create different roles and manage access at the job level, node level, etc.

This section discusses the matrix-based security option, which is available under the Authorization section on the Configure Global Security page (see Figure 9-17).

**Authorization**

Anyone can do anything  
 Legacy mode  
 Logged-in users can do anything  
 Matrix-based security

User/group	Overall	Credentials	Agent	Job	Run	View	SCM
Administrator	Manage Domains Delete Read Read	Manage Domains Delete Getset Update	Connect Configure Build Build Delete	Disconnect Delete Create Configure Cancel	Build Build Discover Delete Create	Revert Delete Discover Delete Create	Tag Read Delete Update Create
Anonymous Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Add user or group...

Project-based Matrix Authorization Strategy  
 Role-Based Strategy

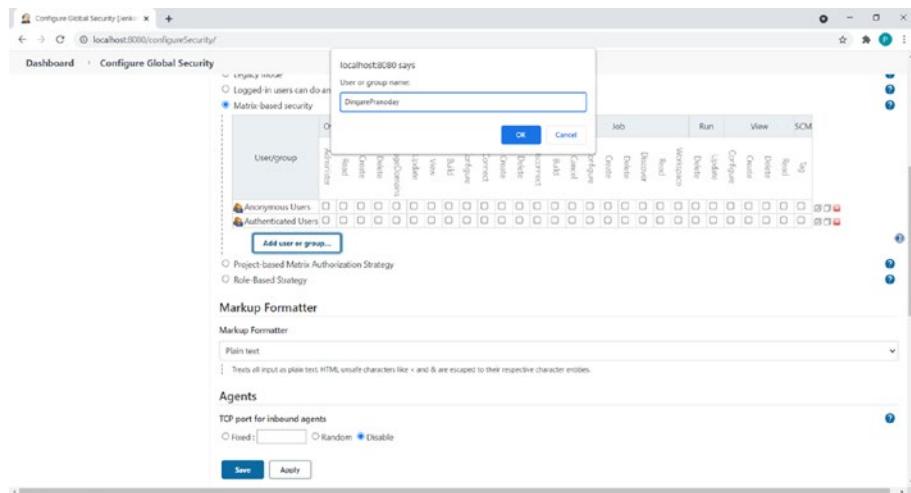
**Figure 9-17.** The matrix-based security option in the Authorization section on the Configure Global Security page

There are two main groups in this section—Anonymous Users and Authenticated Users.

- **Anonymous users:** A special type of user who is not authenticated.
- **Authenticated users:** All authenticated users from the Jenkins system.

To assign permissions, you have to add a user or group by clicking the Add User or Group button, which will open a window in the browser (see Figure 9-18). Enter the user ID.

## CHAPTER 9 MANAGING USERS



**Figure 9-18.** The browser prompt after clicking the Add User or Group button

Click the OK button from the prompt. Select the checkboxes for the user from the available sections to set the access rights.

I want to give DingarePranoday user rights to the Jobs section. So I checked all the checkboxes in the Job section and the Read checkbox from the Overall section (see Figure 9-19).

Matrix-based security		Overall	Credentials	Agent	Job	Run	View	SCM
User/group								
Anonymous Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Dingare Pranoday	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Pranoday Dingare	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Administrator	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		Manage Domains						
		Update						
		Delete						
		Create						
		Risk						

**Figure 9-19.** The DingarePranoday user has been assigned all rights from the Job section

Scroll down the page and click the Save button.

Now log in with DingarePranoday's credentials. Once I log in with this user's credentials, I can see only a few menu options on the left side.

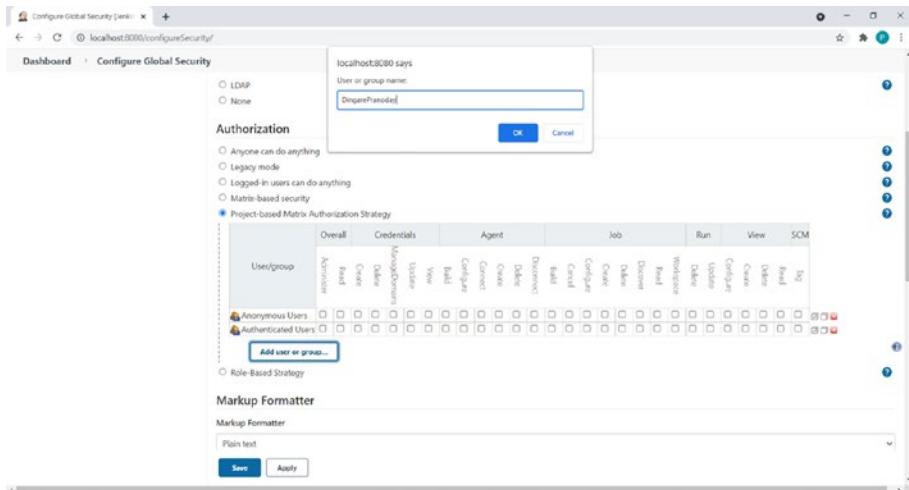
Other menu options, such as Manage Jenkins, are not available to this user.

## Understanding the Project-Based Matrix Authorization Strategy in Jenkins

This mode is an extension to matrix-based security, which allows additional matrixes to be defined for each project.

You can see the Project-based Matrix Authorization Strategy option in the Authorization section of the Configure Global Security page.

To add a user to this setting, click the Add User or Group button. It will open the browser prompt dialog box. Enter the user ID (see Figure 9-20). I added the DingarePranoday user to this setting by listing its name in the prompt dialog.



**Figure 9-20.** The browser prompt opens after clicking the Add User or Group button

## CHAPTER 9 MANAGING USERS

Clicking the OK button will add the user entry. You can see that the DingarePranoday user entry was added.

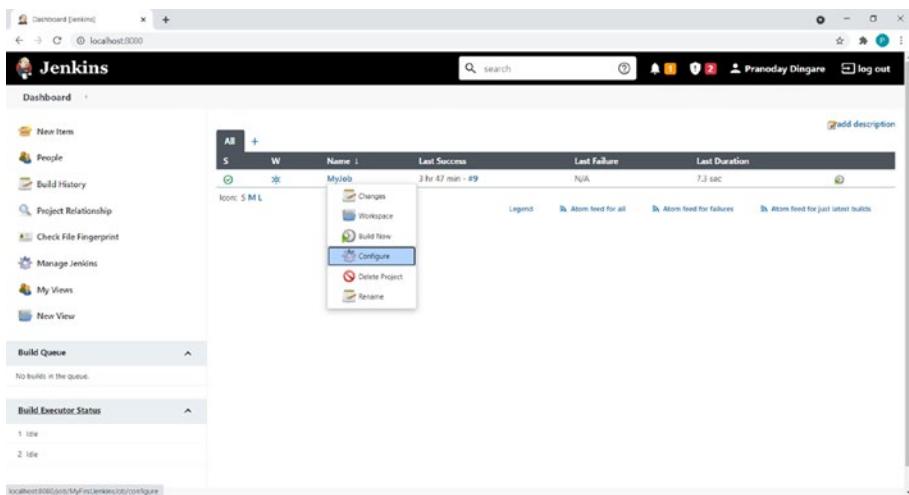
Select the checkboxes from the available sections to set the access rights.

I want to give the DingarePranoday user rights to Build jobs, so I checked the Build checkbox from the Job section and the Read checkbox from the Overall section (see Figure 9-21).

**Figure 9-21.** The DingarePranoday user now has Build and Read rights

Click the Save button on the page.

Let's now configure the security settings for a particular job. Go to the Jenkins dashboard to see the list of jobs. Click the Configure option in dropdown shown in Figure 9-22.



**Figure 9-22.** The Configure menu option for a job

We will enable project-based security settings.

This setting is shown to all the jobs only if the Project-based Matrix Authorization Strategy option in the Authorization section of the Configure Global Security page is turned on.

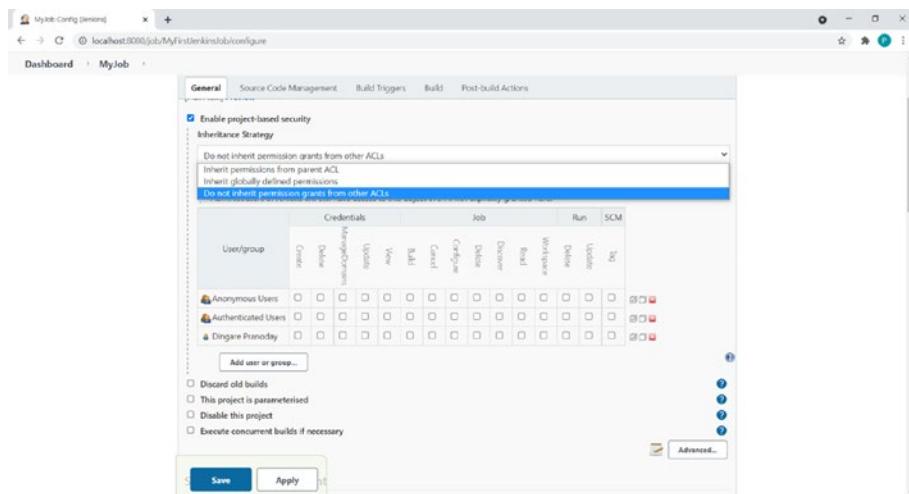
Enable this setting by clicking the checkbox.

To add a user to this setting, click the Add User or Group button. It will open the browser prompt dialog box. Enter the user ID.

I added the DingarePranoday user to this setting by listing this name in the prompt dialog and clicking the OK button.

In the Inheritance Strategy dropdown, select the Do Not Inherit Permission Grants from Other ACLs option. Permissions set here will be granted to the user for this job (see Figure 9-23).

## CHAPTER 9 MANAGING USERS

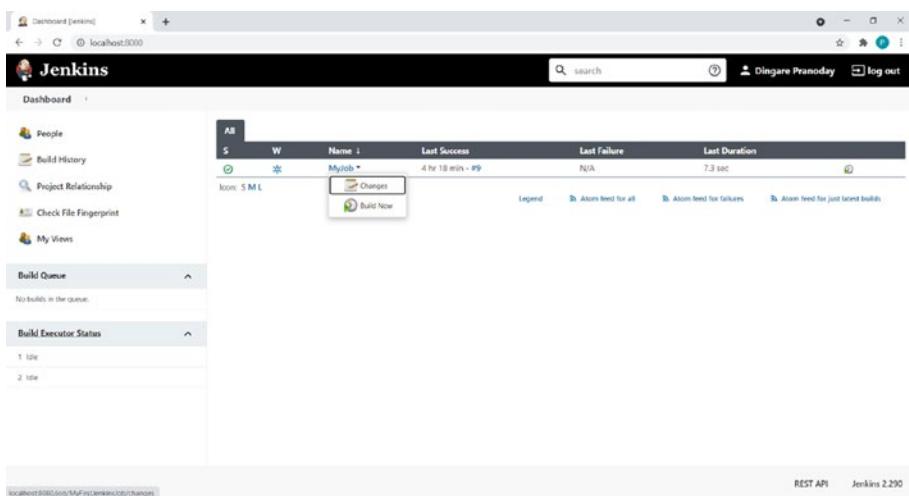


**Figure 9-23.** The *Do Not Inherit Permission Grants from Other ACLs* option in the *Inheritance Strategy* dropdown

I want to give only Build and Read permissions to the DingarePranoday user, so I selected these checkboxes.

Let's log in with the DingarePranoday user credentials now.

The DingarePranoday user has only Read and Build access. They can build the job by clicking the clock sign but cannot edit/view the job configuration and cannot delete it. The Configure and Delete Project options are not available in the dropdown shown in Figure 9-24.



**Figure 9-24.** The Configure and Delete Project menu options are not available

## Summary

This chapter explained how to create multiple users and assign them different rights according to the role they play on a team. You also learned about the project-based matrix authorization and matrix-based strategies, which allow Jenkins to establish good access control over a Jenkins system. The next chapter introduces the Jenkins job. Stay tuned!!

## CHAPTER 10

# Understanding Jobs in Jenkins

In previous chapters, we learned about all the concepts necessary to start leveraging the benefits of this CI/CD tool in real-time projects. In subsequent chapters, we are going to focus on using Jenkins in real-time projects. This chapter is your first step towards that new journey.

Recall that Jenkins, as an automation server, can perform different tasks to automate end-end build lifecycles. These tasks are configured in the form of *jobs*. This chapter introduces this very important and integral concept. There are different kinds of jobs you can configure in Jenkins. This chapter talks about jobs in general. Specific types of jobs are discussed in detail in subsequent chapters.

## What Is a Job in Jenkins?

A Jenkins job is a set of instructions that tell Jenkins what to do and when to do it. A job is also called a Jenkins Project. While configuring any type of a job, you can configure the following three types of instructions:

1. **When to do certain task:**

You can tell Jenkins when you want it to start doing the task mentioned in the job. In Jenkins' terminology, this set of instructions is called a *trigger*.

**2. What to do as part of the task:**

You can configure the steps to be performed as part of a task to achieve a particular objective. In Jenkins' terminology, this set of instructions is called a *build step*. For example, a build step could be running a simple batch command.

**3. What to do once the task is finished:**

You can configure what you want Jenkins to do once it is done with a given task. In Jenkins' terminology, this set of instructions is called the *post build actions*. For example, it could notify users about the success or failure of the task. Or if a Jenkins task is to compile Java code, then the post build action could be to copy the generated class files to a desired location.

## What Is a Build in Jenkins?

A *build* in Jenkins is a particular execution of a Jenkins job. You can run a Jenkins jobs multiple times, and each execution gets a unique build number. All the details pertaining to a particular execution, like artifacts created, console logs, and so on, are stored with that build number.

## What Is a Free-Style Job in Jenkins?

Jenkins allows you to create different types of jobs, like pipeline jobs and free-style jobs, according to your needs. Free-style jobs are typical build jobs or tasks. They can be as simple as running tests, building or packaging an application, or sending a report. Free-style jobs are suitable for simple build tasks. I introduced the free-style job concept in this chapter because you are going to look at different job configurations by creating one. More information on free-style jobs and their use are mentioned in subsequent chapters.

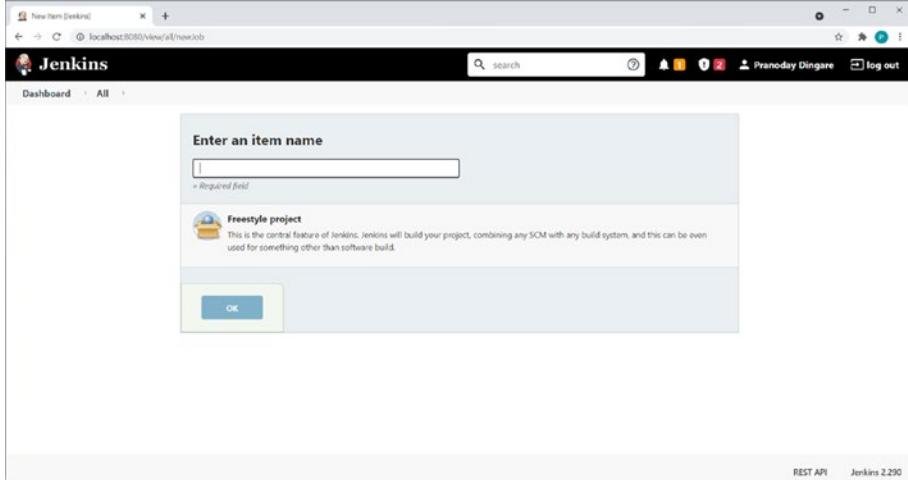
# How to Create a Job in Jenkins

Now that you have a basic understanding of the concept of a job in Jenkins, you're ready to create one. Follow these steps:

1. Log into the Jenkins system and go to the dashboard.
2. Create a job by clicking the Create a Job or New Item link.
3. If you do not have any current jobs, you will get the Create a Job link on a blank Jenkins dashboard.

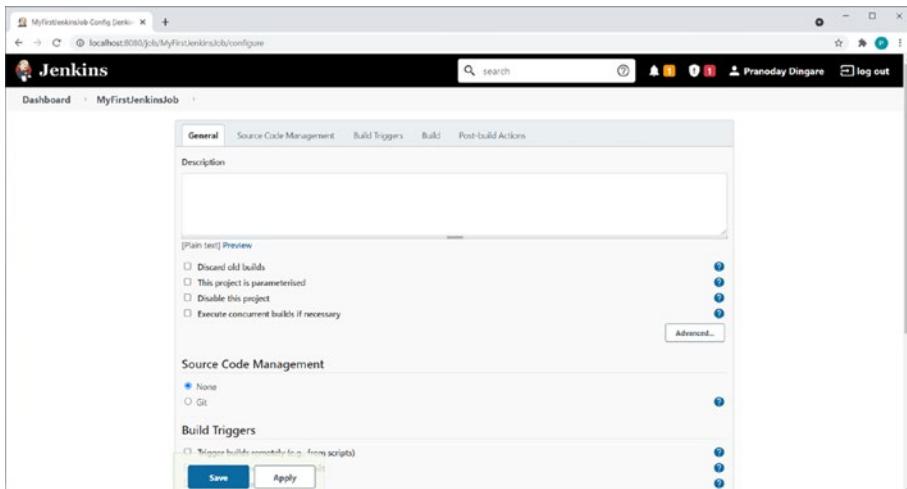
If you have one or more Jenkins jobs, you will not see this link on your dashboard. In this case, you can click the New Item link provided on the left side of the dashboard.

4. Clicking either of these links will take you to the page shown in Figure 10-1.



**Figure 10-1.** The field to enter the name of a Jenkins job

5. Enter a suitable name for your job and select the Freestyle Project option. Click the OK button, which will take you to the Job configuration page shown in Figure 10-2.



**Figure 10-2.** The Job Configuration page

## How to Configure a Job in Jenkins

On the Job Configuration page, you have different sections to configure settings. Let's look at each section and its settings one by one.

The General section has the following options:

- **Description:** In this field you can describe the purpose of this Jenkins job (such as compiling a Java library project).
- **Discard old builds:** Build represents data related to the specific execution of a job, like console output, artifacts created after the execution of a job, etc.

There are two options to decide when to delete build history data:

- **Build Age:** Discard the builds that are older than seven days for example.
- **Build Count:** Discard the oldest build when a certain number of builds exist.

By default, you can keep a build for a maximum of 14 days, but only up to a limit of 50 builds. If either of these limits are exceeded, the oldest build will be deleted. Selecting this option and the configuring options is very important to save on disk space.

Once you select this checkbox, you will see the settings to configure these two options, as shown in Figure 10-3.



**Figure 10-3.** The settings related to Discard Old Builds

You can configure how many days builds of this job are to be kept by adding a number to the Days to Keep Builds field. If you add 5 here, then this job's builds will be kept for a maximum of 5 days.

You can specify a maximum number of builds to be kept as well, by using the Max# of Builds to Keep field. If you specify 10 here, once the build number reaches 11, the oldest builds will be deleted.

Clicking the Advanced button will reveal two more fields—Days to Keep Artifacts and Max # of Builds to Keep with Artifacts—as shown in Figure 10-4.

Days to keep artifacts  
if not empty, artifacts from builds older than this number of days will be deleted, but the logs, history, reports, etc for the build will be kept

Max # of builds to keep with artifacts  
if not empty, only up to this number of builds have their artifacts retained

**Figure 10-4.** The advanced settings related to the Discard Old Build option

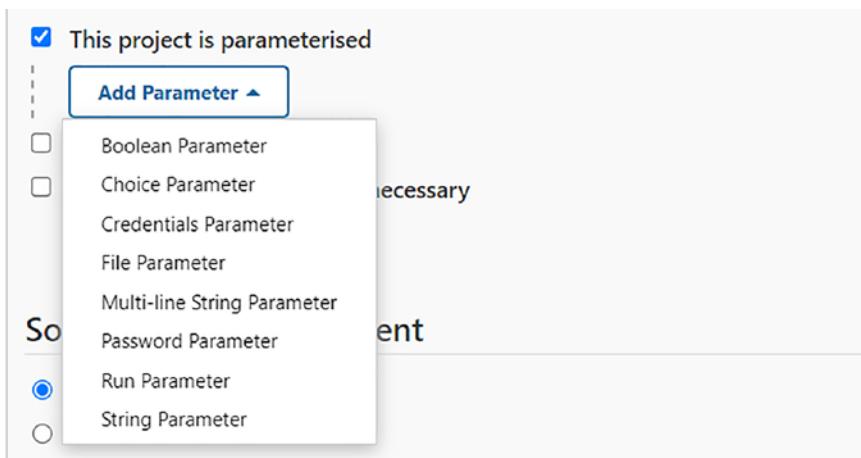
An *artifact* of a build is any output that is generated after running that job. For example, if the job creates a .JAR file after compiling a Java application, then this .JAR file is an artifact of that job.

You can configure how many days artifacts from a build are kept by using the Days to Keep Artifacts field. If you specify 3 in this field, artifacts of that build will be deleted after three days, but the build's other details, like the console log, won't be deleted until this build becomes older than the number of days in the Days to Keep Build field.

If you specify 5 in the Max # of Builds to Keep with Artifacts field, then artifacts of the five latest builds will be kept.

## This Project Is Parameterized

If this job needs any external inputs, you have to check this checkbox and select them from the available dropdown options. You'll need to indicate which type of external input you need, such as Boolean, string, etc. See Figure 10-5.

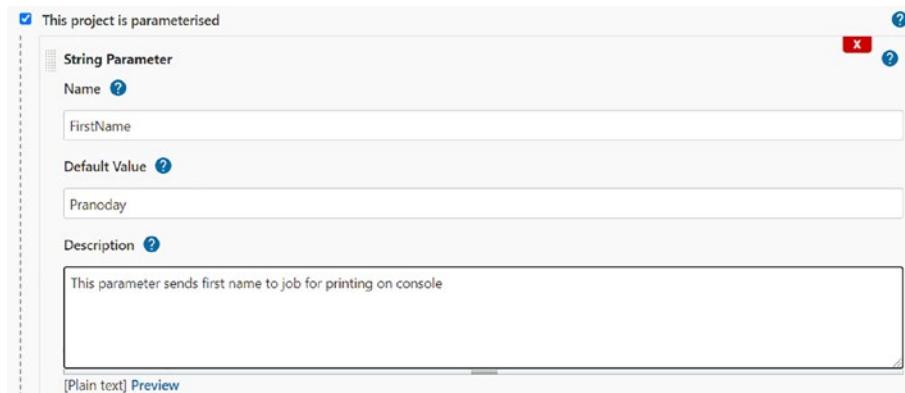


**Figure 10-5.** The list of job parameters

For example, if you want to send a string value as the input, then select the String Parameter option. Once you select this option, you will see the String Parameter section.

You can enter the name of a parameter in the Parameter Name field, the default value of the parameter in the Default Value field, and a description about the purpose of the parameter in the Description field.

Say I want to send my firstname to a job as a parameter and get it printed on the console. I add those details to these fields, as shown in Figure 10-6.



**Figure 10-6.** The String Parameter with values in the Name, Default Value, and Description fields

I can access this parameter value in the build step of a job and print it on the console. Subsequent chapters cover how to access values of a parameter in the build step of a job and the real-time use of parameters.

## Disabling a Project

If this option is selected, this job will not be executed and no new builds will be created. This is a useful settings if you do not want to use a particular job for a temporary period, maybe because the required infrastructure is not available.

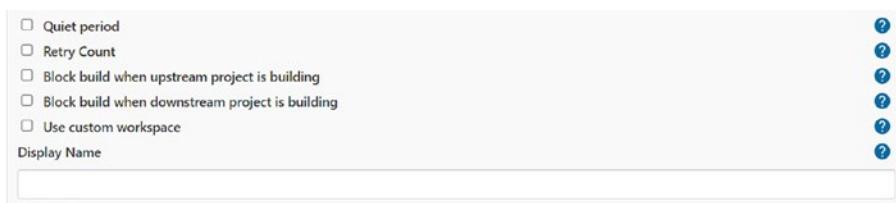
## Executing Concurrent Builds

By default, you can execute only a single build of a job at any one time. If you try to execute a job multiple times, then subsequent build executions will be kept in the queue until the previous execution completes.

This default option is important if your job needs exclusive permissions of directories/files to fulfill its task.

But if you want to start parallel executions of a build, you have to check the concurrent builds option. This option is useful when you have a lengthy build process to execute that's divided into a number of phases and each phase is not dependent on the other. The time of lengthy job can be substantially reduced if different stages of a build process are started in parallel. You learn about real-time use in Chapter 17.

Click the Advanced button to see the advanced options. The options shown in Figure 10-7 are then available.



**Figure 10-7.** The additional options shown after clicking the Advanced button

## Quiet Period

When this option is selected, a new build will not be triggered immediately. It will be added to the build queue and will wait for a specified time before it starts.

This option is useful when you want your build to take multiple code commits at approximately the same time. For example, if this option is not checked and multiple developers commit code at approximately the same time in the code repository, your build will be triggered immediately on arrival of the first commit and the rest of the commits will not be taken for this build. When you check this option, your build waits in the build queue for a specific time and takes all the commits happening at approximately the same time.

After checking this setting, you will get a field to set the time in seconds for the build to wait in the queue until it starts—this is called the *quiet period*.

If the Quiet Period field is set to 5 seconds, the build will wait in the build queue for 5 seconds before it starts its execution.

When the Retry Count option is not selected, and your job is configured to use a Source Code Management System (SCM) like a Git repository, Jenkins marks the job as failed if the first attempt of checking out SCM fails.

Using this option, you can set the number of times Jenkins should try to check out SCM before the job is marked as failed.

If you specify a 3 in the SCM Checkout Retry Count field, then Jenkins will try to check out SCM three times by waiting for ten seconds between each retry.

In Jenkins, a tasks can be configured in two different jobs such that an artifact generated by Job A would be used by Job B to complete its tasks. We can configure Job B to be dependent on Job A. For example, Job A is compiling a Java application and generating .CLASS files, and Job B is using these class files to create a .JAR file.

Here, Job A is called an *upstream job* of Job B. Job B is called a *downstream job* of Job A.

## Block Build When Upstream Project Is Building

When this option is selected, Jenkins will not execute this project/job when the dependency project (a project on which this job depends) is in the build queue.

Jenkins will not start Job B if Job A is in the build queue.

## Block Build When Downstream Project Is Building

When this option is selected, Jenkins will not execute this project/job when its children or dependent job is in the build queue.

Jenkins will not start Job A if Job B is in the build queue.

## Use Custom Workspace

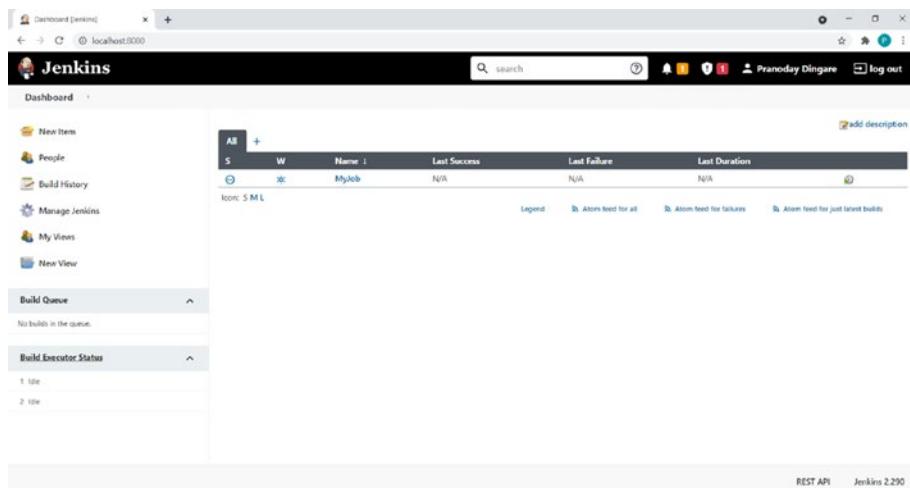
Before we talk about this setting, you need to understand what a workspace is in Jenkins. A *workspace* is a directory in which builds are executed. If the Jenkins job is checking out a Source Code Repository then it checks out in this directory. When any build starts executing, a workspace directory with the name of the job being executed is created by default in the  `${JENKINS_HOME}\workspace` folder. You will find  `${JENKINS_HOME}` usually at  `${CurrentUser}\.Jenkins`. You can change the workspace directory location by specifying the path in this field. Changing this location is helpful if another job is using the artifacts of this job and the paths of artifacts to be used are hard-coded into that job, for example.

My workspace location is `D:\MyFirstJenkinsJob`. Now when I run this job, a directory called `MyFirstJenkinsJob` will be created on `D:\` and will be used as the workspace.

## Display Name

A name set here will be shown for the project throughout the Jenkins WebUI. I set the Display Name to `MyJob`. On the dashboard, the job's name is shown as `MyJob`, as shown in Figure 10-8.

## CHAPTER 10 UNDERSTANDING JOBS IN JENKINS



**Figure 10-8.** The job is called *MyJob* on the Jenkins dashboard

Note that the display name is only shown in the WebUI. The workspace folder will still be created with the project name, and not with the display name.

My job's name is `MyFirstJenkinsJob` and the display name is `MyJob`. So in my case, the workspace folder is created with `MyFirstJenkinsJob` and not with `MyJob`.

## Keep the Build Logs of Dependencies

This setting is available in the Display Name option, from the Advanced settings page. If this setting is enabled, all builds referenced here are protected from log rotation. Log rotation is the process of automatic compression, deletion, and mailing of the Jenkins build's logs.

## Source Code Management

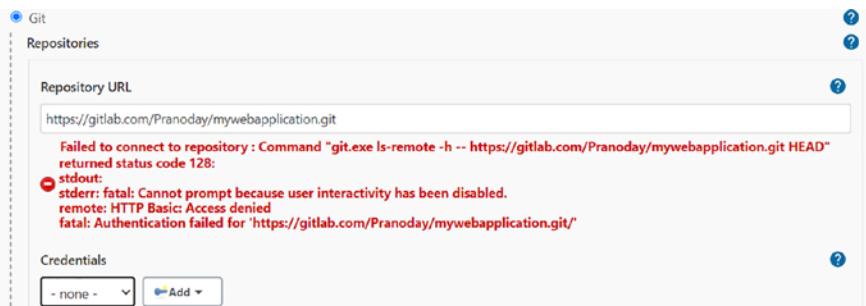
Let's now move to the next section of the General tab on the Job Configuration page, called Source Code Management.

In this section, you will see the Git option only if the Git plugin is installed. For detailed steps on how to install this plugin, see Chapter 6. Usually the Jenkins job responsible for creating new software build works on the latest committed code on the central repository and downloads it first. You need to add the URL of the Git code repository to this field, so that your Jenkins build will download the latest available code.

Here I have added my Git repository URL. When I run my job, it will first download the latest code.

The repository in the Repository URL field is my public repository. A public Git repository does not need authentication.

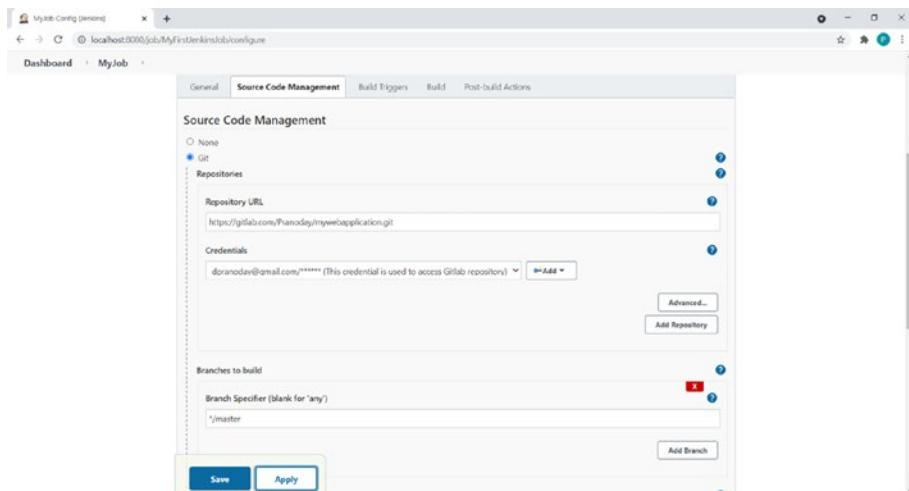
If your Git repository is a private repository, which usually would be the case, you need to provide the required authentication information, such as username/password, SSH private key, or API token, depending on the authentication configured in your Git repository. If you don't provide the required authentication information, you will see the error in Figure 10-9.



**Figure 10-9.** The authentication error when Jenkins tries to access a private repository without the required credentials

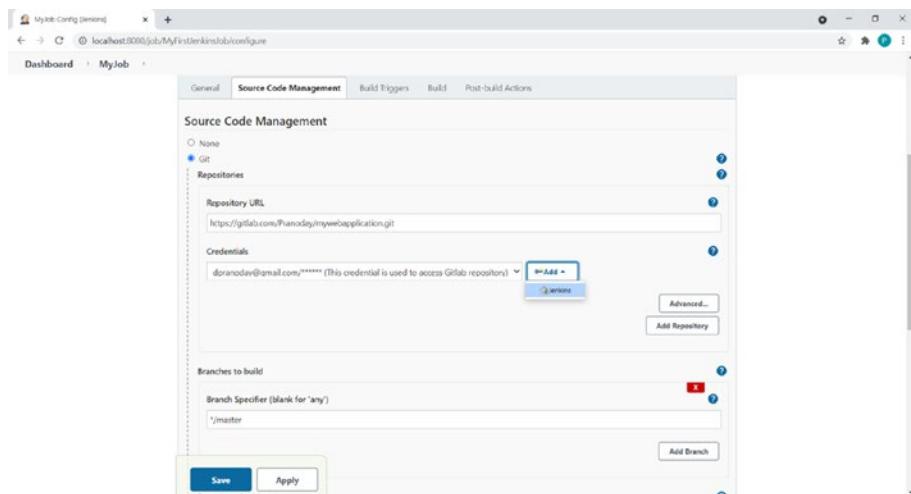
To resolve this error, you need to create a Credentials entry of type Global, which will have the required authentication information. We need to select that entry in the Credentials dropdown and click the Apply button available on this job's configuration page. For more details on credentials and the steps to create a credentials entry, see Chapter 8.

I created a credentials entry of type Global, selected it in the Credentials dropdown, and clicked the Apply button. The error is not shown now; see Figure 10-10.



**Figure 10-10.** Jenkins can successfully access the private repository

If you want to create a credentials entry from this page, you can do so by clicking the Jenkins option available in the dropdown, as shown in Figure 10-11.

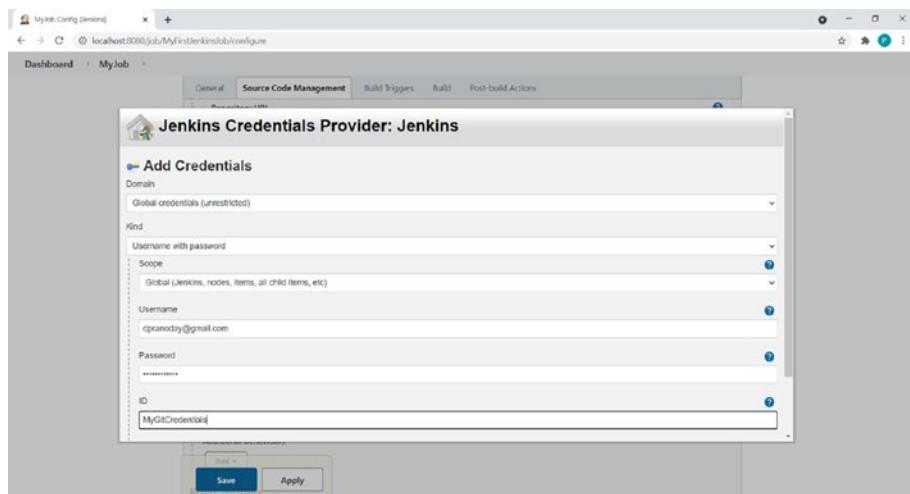


**Figure 10-11.** The Jenkins option is available in the Credentials dropdown

Once you click this dropdown, you will see the Jenkins Credentials Provider: Jenkins window.

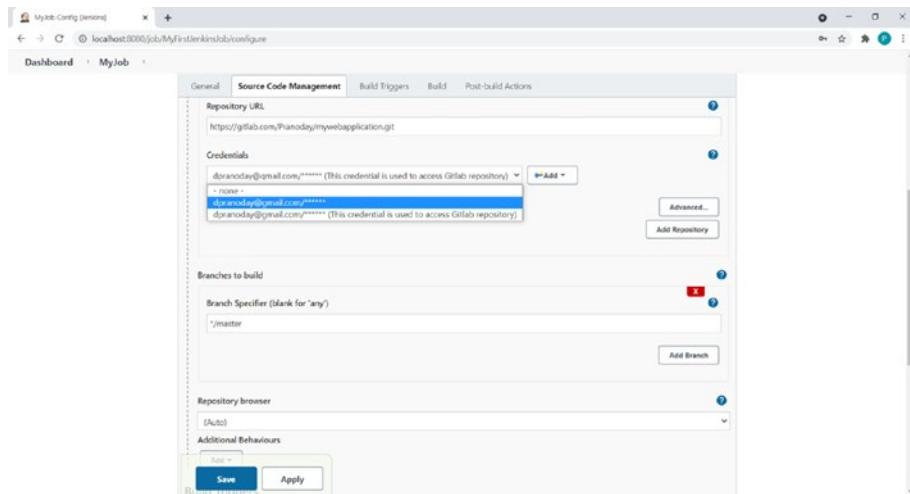
Enter the required details in this window, as shown in Figure 10-12, and click the Add button, which you will see after scrolling down a bit. (All these fields are explained in detail in Chapter 8.)

## CHAPTER 10 UNDERSTANDING JOBS IN JENKINS



**Figure 10-12.** The credentials details filled while creating a new credential

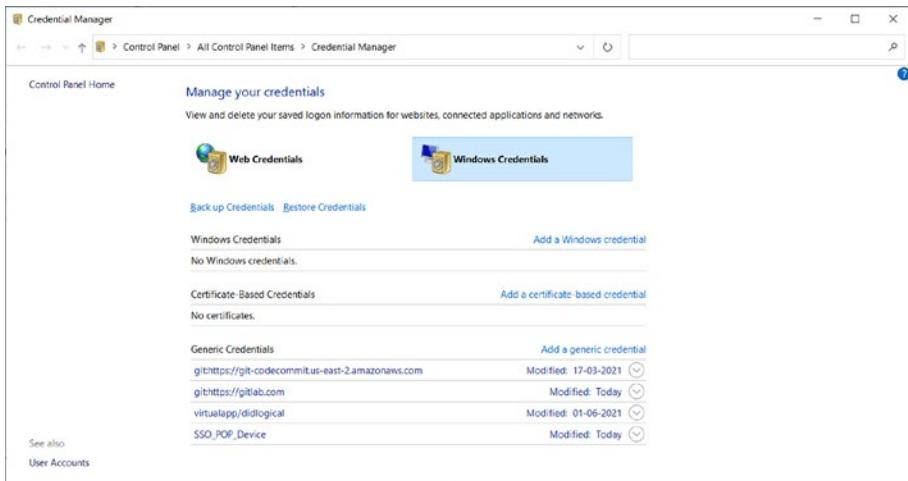
Once you click the Add button, the newly created entry will appear in the Credentials dropdown, as shown in Figure 10-13.



**Figure 10-13.** The newly created credential entry in the Credentials dropdown

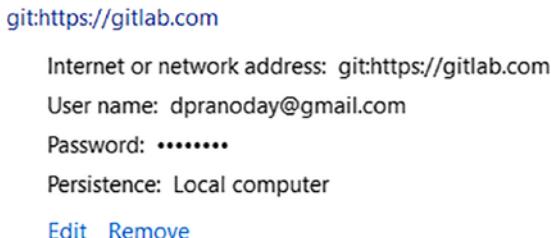
There is one very important point related to accessing private repositories from Jenkins and the Windows Credentials Manager, which usually proves to be difficult to debug.

You may already have your Git username/password stored in Windows Credentials (choose Control Panel ➤ All Control Panel Items ➤ Credential Manager). The Credentials Manager is shown in Figure 10-14.



**Figure 10-14.** The Windows Credentials Manager with all saved credentials

In Figure 10-15, you can see that my Git credentials are saved in Windows credentials. I created these using the Add a Windows Credential link shown in Figure 10-14.



**Figure 10-15.** The GitLab credentials stored in the Windows Credentials Manager

Now you can see that I am not getting an authentication error in the Repository URL field (the error is shown in Figure 10-9), even though the Credentials entry is not selected in the Credentials dropdown, because Jenkins now has my credentials from the Windows Credentials.

If I remove my credentials from Windows credentials, which were saved with URL: <https://gitlab.com>, you can see my Credentials entry is not present in the Windows Credentials, as shown in Figure 10-16.

Generic Credentials	Add a generic credential
git:https://git-codecommit.us-east-2.amazonaws.com	Modified: 17-03-2021
virtualapp/didlogical	Modified: 01-06-2021
SSO_POP_Device	Modified: Today

**Figure 10-16.** My GitLab credentials are no longer present in the Windows Credentials Manager

I get an authentication error (Error 128) in Jenkins, as I have not selected the Credentials entry in the Credentials dropdown and my Git credentials are also not present in the Windows Credentials Manager.

## Branches to Build

By default, Jenkins jobs look for changes in the master branch and will download code from that branch from a remote repository. If you want Jenkins to look for changes in a different branch and download the code from it, you must specify the name of the new branch in this field.

## Build Triggers

Using settings available in this section, you can configure when Jenkins should start running your jobs. Let's look at these settings one by one now.

### Trigger Build Remotely (e.g. from Scripts)

You can trigger a Jenkins job from an external entity like a Git repository. Consider a scenario where you want to trigger a Jenkins job when the code is to be merged into a master code branch by your developers. You have the following two options to trigger the job in this case.

- **Polling Source Code Repository:** In this option, you can configure Jenkins to check for a change in the Source Code Repository after a specific time interval (called the polling time). Jenkins will trigger the job as soon as there's a change in the code repository. This option is described in detail later in this chapter.
- **Triggering Jenkins Job from SCM on a specific event:** In this option, instead of making Jenkins poll a SCM to look for a change, you can configure SCM to trigger a job on a specific event that occurred on the SCM side, such as a merge event that occurs when code is merged into a specific branch in SCM.

If you want to trigger the job using the second option, you need to check the Trigger Build Remotely (e.g. From Scripts) option. Every Jenkins job has an URL from which you can trigger it remotely.

The format of this URL is: `JENKINS_URL/job/<Job Name>/build?token=TOKEN_NAME`.

- **JENKINS\_URL:** This represents the URL used to access the Jenkins instance (e.g., `http://localhost:8080`).
- **JobName:** The name of a Jenkins job. Note that this should be Job Name and not Display Name.
- **TOKEN\_NAME:** This is an access token used to trigger this job. I added my authentication token.

My URL to access this Job is `http://localhost:8080/job/MyFirstJenkinsJob/build?token= abcd123456890`.

You are going to see a practical example of this in upcoming chapters.

## Build After Other Projects Are Built

You can check this option if you want to trigger this job after the execution of another one. We have discussed dependency in jobs, where one job uses artifacts created by another job. So if this job uses artifacts created by another Job called `CompileJavaApplication`, then you should check this option and add the name `CompileJavaApplication` to the Projects to Watch field so that this job is triggered once the `CompileJavaApplication` completes its execution and gives the required artifacts to this job to process them.

You can configure when to trigger this job and when not to depending on the status of dependency job using the following three options:

- **Trigger only if build is stable:** If this option is selected then the job is triggered only if the job it depends on is stable. Stable means that the build executed successfully.

- **Trigger even if the build is unstable:** If this option is selected, the job is triggered even when the dependent job build is unstable. An unstable build means it could complete its task successfully but there are a few publishers who report it as unstable. For example, if a build is compiling an application and the Junit publisher is configured to publish a report of unit testing, and compilation is successful but the unit test fails, then the Junit publisher will mark this build as unstable.
- **Trigger even if build fails:** If this option is selected, the job is triggered even when the dependent job build fails or is broken.

A failed/broken build means one or more of the build steps failed and the build could not complete its task.

## Build Periodically

This option allows you to trigger a build at a particular time and interval, like daily, monthly, weekly, etc. It provides a feature like a job scheduler in Windows or a cron job on UNIX systems. In my opinion, triggering jobs periodically does not follow the CI/CD principles, as CI/CD expects feedback on a software build as soon changes happen in the code and not in a week or month.

An example where this can be useful is when you want to trigger a daily job running e-e tests once the application is built and deployed to the testing environment. You can select this option and write a cron expression representing the schedule in the Schedule field.

Chron expressions are written in the following format:

MINUTE HOUR DOM MONTH DOW

This command has the following five fields separated by tabs or whitespace:

- MINUTE: Minutes in the hour (0-59)
- HOUR: Hour of the day (0-24)
- DOM: The day of the month (1-31)
- MONTH: The month in the year (1-12)
- DOW: The day of the week (0-7), where 0 and 7 represents Sunday

If I want to trigger a job every day, every month, and on all days in a week at 8:45 AM, then I would write the chron expression as follows:

```
45 8 * * *
```

I added this to the Schedule field shown in Figure 10-17.



**Figure 10-17.** The chron expression specified in the Schedule field

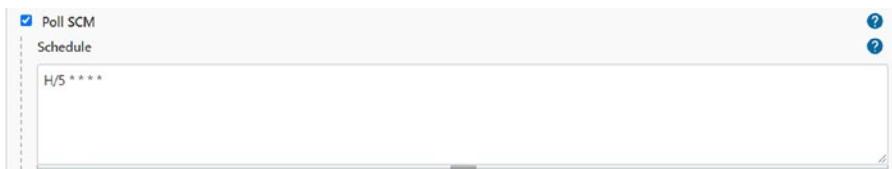
More information about chron expressions can be found at <https://en.wikipedia.org/wiki/Cron>.

## Poll SCM

Using this setting, you can configure a time interval (called the *polling time*) after which Jenkins will keep checking for a change in the code present in the SCM (the URL of which is mentioned in the Source Code Management section).

If I want Jenkins to poll my SCM every five minutes every day, every month, and all days of the week, I add the following cron expression to the Schedule field:

```
H/5 * * * *
```



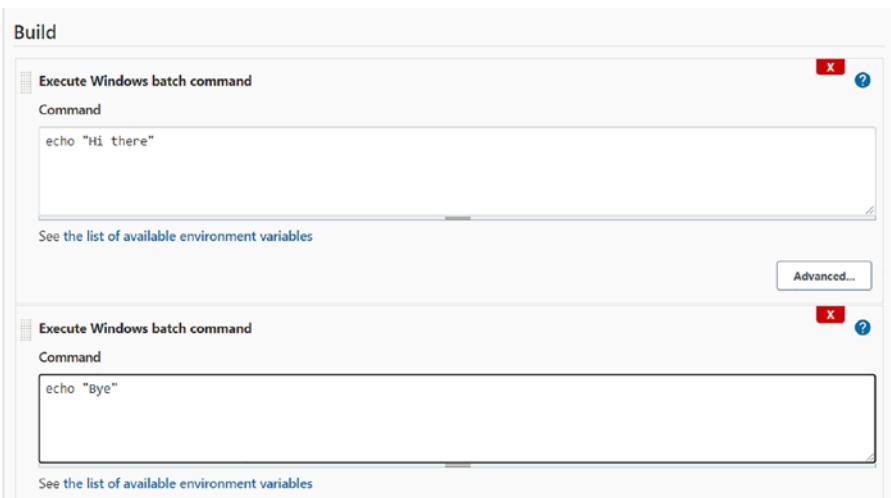
My Jenkins job will keep polling my SCM every five minutes and will trigger the build if SCM has changed since the last poll.

## Build Step

This job performs the assigned task by executing the steps configured in this section. A step could be running a batch file or running any build tool's goal. To add a step to execute the Windows DOS command for example, click the Add Build Step dropdown and select the Execute Windows batch command option.

Add the DOS command to the Command field. To remove a step, click the Red Cross sign next to the step.

We can add more than one step to a single job, which would be executed sequentially from the top, as shown in Figure 10-18.



**Figure 10-18.** Multiple build steps added to a job

## Post-Build Actions

In this section, you can configure actions that you want the Jenkins job to perform once its allotted task is done. For example, you can send an email notification.

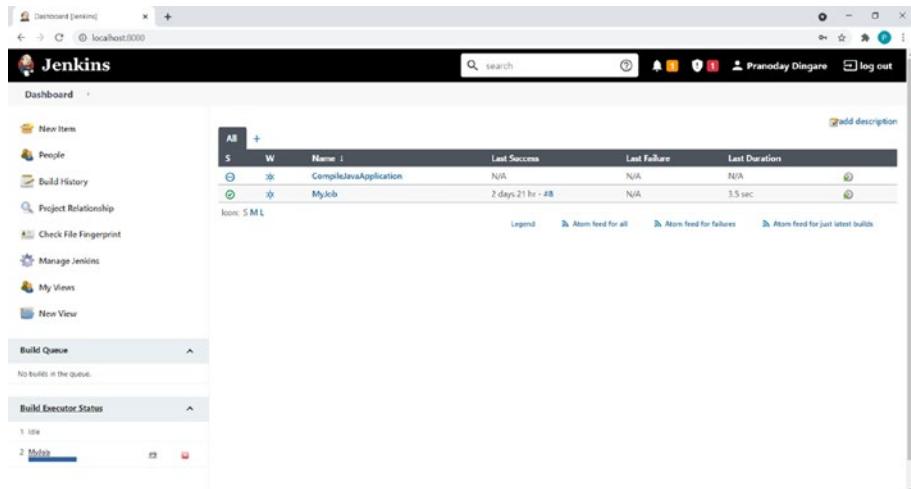
To add actions, click the Add Post-Build Action option and select the required option. You can add multiple actions in this section. You will see a practical use of configuring this section in upcoming chapters.

## How to Run a Job in Jenkins and Check Its Output

Once a Jenkins job has been created, configured, and saved, its entry is shown on the dashboard

To run a specific job manually, click the clock sign for that job entry.

Once a job starts—either manually or it was triggered according to the triggers mentioned in the job's configuration—its execution status can be seen in the Build Executor Status section. You can see the progress bar in Figure 10-19.

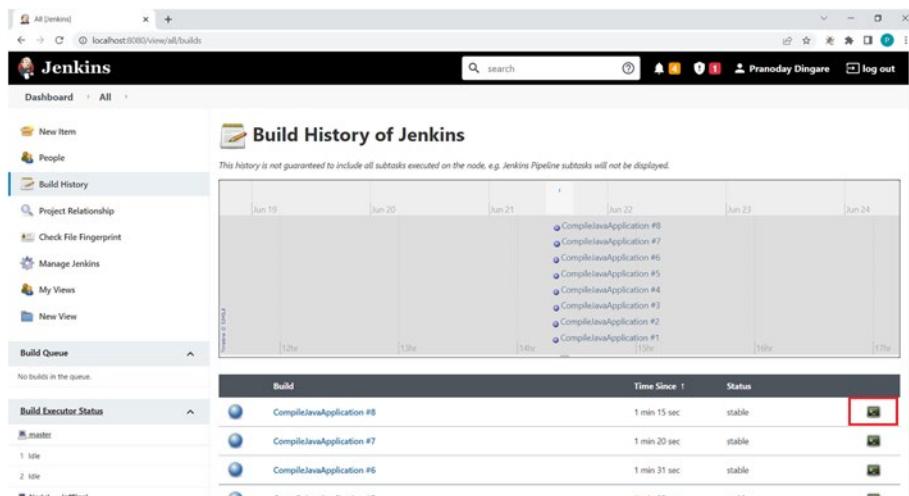


**Figure 10-19.** The progress bar indicates that the job is being executed

Once a job is completed, you can see the history of the build executions by clicking the Build History link shown on left side of the Dashboard page.

Clicking the Build History link will open the Build History of Jenkins page, as shown in Figure 10-20. Note that you may see different build histories depending on the number of times you executed this job. I have executed this job eight times, hence this build history shows build entries from Build #1 to Build #8.

## CHAPTER 10 UNDERSTANDING JOBS IN JENKINS



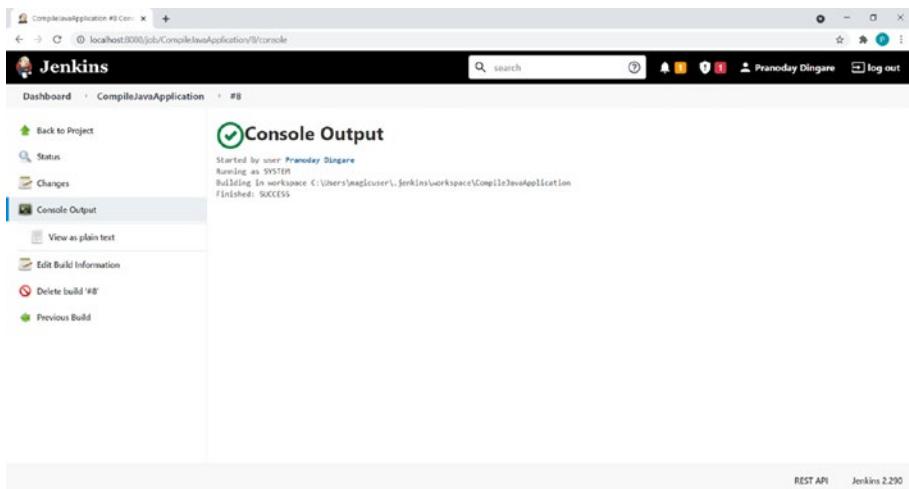
The screenshot shows the Jenkins interface with the 'Build History' sidebar selected. The main area displays the 'Build History of Jenkins' for the 'CompileJavaApplication' job. A note at the top states: 'This history is not guaranteed to include all subtasks executed on the node, e.g. Jenkins Pipeline subtasks will not be displayed.' Below this, a timeline from June 19 to June 24 shows several build entries. The first entry on June 24 is highlighted with a red box. The table below lists the builds:

Build	Time Since 1	Status
CompileJavaApplication #8	1 min 15 sec	stable
CompileJavaApplication #7	1 min 20 sec	stable
CompileJavaApplication #6	1 min 31 sec	stable

**Figure 10-20.** The build history for an executed job

To check the detail build log, click Console Output for a particular build entry from the table. This is highlighted in Figure 10-20.

For example, to see a detail console log of Build #8, you would click the Console Output icon of this build entry. It will show the detail console output of Build #8, as shown in Figure 10-21.



**Figure 10-21.** The detail console output of Build #8

## How to Edit a Job in Jenkins

If you want to edit an existing Jenkins job, click the job's dropdown on the Jenkins dashboard. Click the Configure option from this dropdown. This option is highlighted in Figure 10-22.

## CHAPTER 10 UNDERSTANDING JOBS IN JENKINS

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', etc. The main area shows a table of jobs. One job, 'CompileJavaApplication', is selected, and a context menu is open over it. The menu includes options like 'Changes', 'Workspace', 'Build Now', 'Configure' (which is highlighted in blue), 'Delete Project', and 'Rename'. Below the table, there are sections for 'Build Queue' (empty) and 'Build Executor Status' (2 idle). At the bottom, the URL is visible: 'localhost:8080/jobs/CompileJavaApplication/configure'.

**Figure 10-22.** The *Configure* option in the dropdown menu available for a job

This will take you to the job's configuration page, which is shown in Figure 10-23.

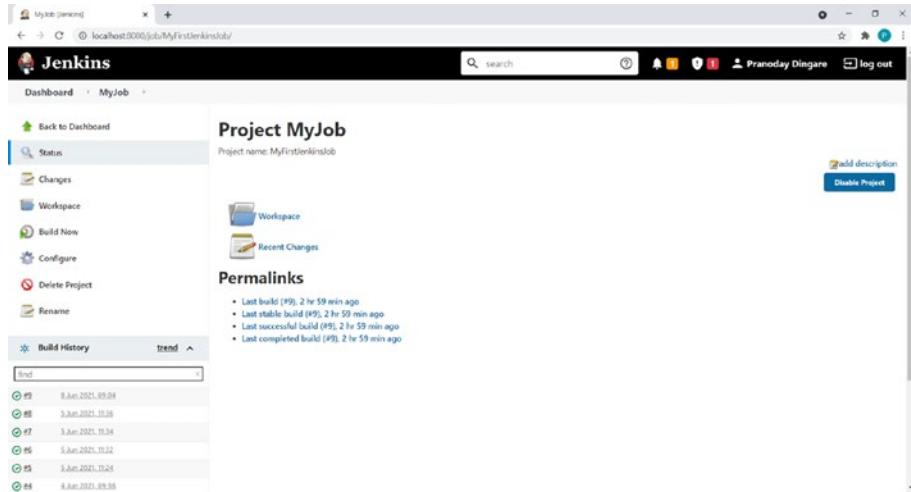
The screenshot shows the 'Configure' page for the 'CompileJavaApplication' job. It has tabs for 'General', 'Source Code Management', 'Build Triggers', 'Build', and 'Post-build Actions'. The 'General' tab is active. It contains fields for 'Description' (empty), 'Preview' (checkboxes for 'Discard old builds', 'This project is parameterised', 'Disable this project', and 'Execute concurrent builds if necessary'), and 'Advanced...' (button). The 'Source Code Management' section shows 'None' selected. The 'Build Triggers' section has a checkbox for 'Trigger builds remotely (e.g., from scripts)' and buttons for 'Save' and 'Apply'. The URL at the top is 'localhost:8080/jobs/CompileJavaApplication/configure'.

**Figure 10-23.** The Job Configuration page for the *CompileJavaApplication* job

Do the required modifications and then click the Save button.

## How to View a Job's Workspace

To view a workspace of a particular job, click the job entry on the dashboard. You will go inside the job page, as shown in Figure 10-24.



**Figure 10-24.** The job page for *MyJob*

Click the Workspace link shown on the left side of the page, which will open the workspace, as shown in Figure 10-25.



**Figure 10-25.** The workspace for *MyJob*

## How to Clear a Job's Workspace

To clear a workspace of a particular job, click the job entry on the dashboard. You will go inside the job page, as shown in Figure 10-24.

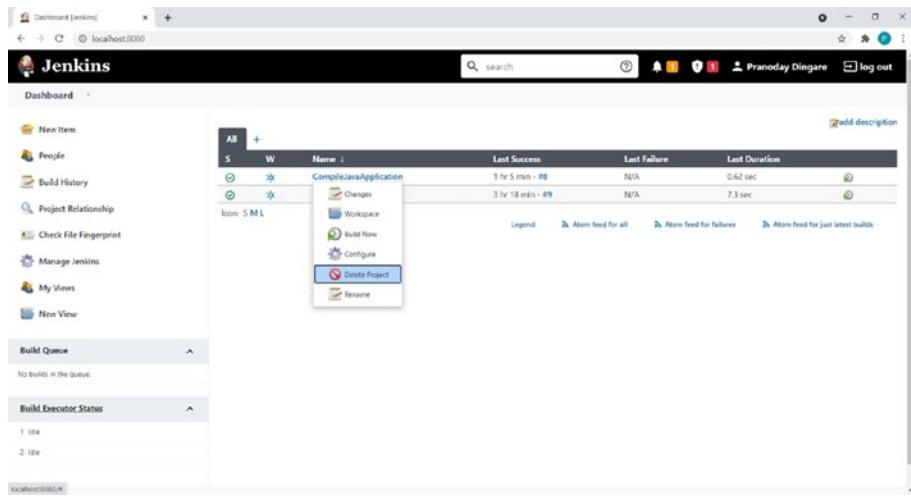
Click the Workspace link shown on the left side of the page. Then click the Wipe Out Current Workspace link, shown in Figure 10-26.



**Figure 10-26.** The Wipe Out Current Workspace menu option under the Workspace menu

# How to Delete a Job

If you want to delete an existing Jenkins job, click the dropdown available for the job entry on the Jenkins dashboard. Then click the Delete Project option from this dropdown, as shown in Figure 10-27.



**Figure 10-27.** The Delete Project menu in the dropdown opened for a job

Click the OK button in the Confirmation alert window.

## Summary

This chapter marked a very important step in your learning journey by introducing an integral concept called *jobs*. You looked at different job configurations by creating a free-style job. You also learned the steps needed to run Jenkins jobs and to view execution logs of a Jenkins build. Subsequent chapters carry this knowledge forward and you will learn real-time implementation of Jenkins jobs to build different kinds of applications, like Java APIs and web applications.

## CHAPTER 11

# Preparing a Java API Project Using Maven

After a thorough explanation of configuring the jobs in Jenkins in the last chapter, this chapter starts looking at Jenkin's real-time use in managing the end-end build lifecycle of different kinds of applications. You see how to configure different kinds of jobs and trigger them in different ways in upcoming chapters.

Different kinds of applications, such as reusable libraries called Application Programming Interfaces (APIs), web applications, RESTful API services, and so on, have their own processes for building and deploying. For example, a Java API project could be bundled in the form of a .JAR file and will be released to an artifact repository, whereas a web application developed in Java could be bundled in a .WAR file and would be deployed in server like Tomcat. To implement end-end build lifecycles of any of these applications in Jenkins, you need to first understand the different build lifecycle phases these applications need to go through, along with the tools that will help you take these applications through those different phases until they reach their final destinations.

This chapter develops a simple Java API project and the different build lifecycle phases it has to go through, along with tools like TestNg and Maven. In subsequent chapters, you'll see how to build different kinds of Jenkins jobs to automate the release of this Java API project.

# Understanding the Maven Build Tool

This section looks at how a very popular build tool called Maven can be used to build a Java API project. Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time.

Before we dive deeply into Maven, you need to understand the typical workflow of the development lifecycle of a Java API and how a build tool helps developers perform them.

## Typical Development Flow of a Java API Project

This section explains the typical tasks a developer needs to perform when developing a Java API project. The actual tasks may vary from project to project.

### Downloading Third-Party Libraries

While developing any application, developers uses binaries, which are bundled API classes available with the development kit. In this case, it's the JDK (Java Development Kit), as we are talking about Java API projects. Along with these native libraries, developers may need libraries developed by the API developers. These are called third-party libraries. These libraries are usually available on different web platforms called artifact or package registries. For example, <https://mvnrepository.com/> is a central repository where Java API developers deploy packaged files (.JAR files) and developers wanting to use them can download them. <https://www.npmjs.com/> is a package registry to release reusable node packages built using JavaScript. The Java API developer in this book needs to download these library files from the mvn repository.

## Adding Downloaded Libraries to the Project Build Path

These downloaded libraries must be added to the reference libraries the project developer is working on. In the context of a Java API project, these libraries are added to the CLASSPATH of the project.

## Coding and Writing Unit Test Cases

Once the required third-party and native libraries are present in the build path of the project, developers can write different API functions. Once the development of a particular unit/function is done, unit test cases are written. If developers are using a test-driven development approach (TDD), they write test cases before implementing a functionality piece.

## Compiling the Application and Unit Test Cases Code

After the code is written, the developer needs to compile the API source code as well as the unit test cases code.

## Running Unit Test Cases

After compiling the application's source code and the unit test cases, developers need to run unit test cases using unit testing tools like TestNG, Pytest, NUnit, etc. These tools depend on the development environment they are using.

## Bundling/Packaging the Application

After the unit testing is done and the code has been merged, the implementation is packaged into a library (.JAR) file.

## Releasing it on the Artifact Repository

Once a particular library version is created, it will be released on the artifact repository so that users can download and use it.

## How the Build Tool Helps Developers

You just saw a typical workflow of a Java API project. In this flow, you saw that developers have to perform a lot of tasks, including downloading the required compile-time libraries (libraries that the developer needs to compile the implementation) and testing libraries (libraries that the developer needs while testing the implementation) and runtime libraries. Then the code is compiled, there is unit testing, packaging, and deployment of the created lib package, and so on. Performing these tasks manually is cumbersome and sometimes error-prone. Hence, developers need a tool to perform these tasks. Tools that developers use to automate these tasks are called *build tools*. There are various build tools available, such as Apache Ant, Maven, Gradle, etc.

## How to Use the Maven Build Tool to Build a Java API Project

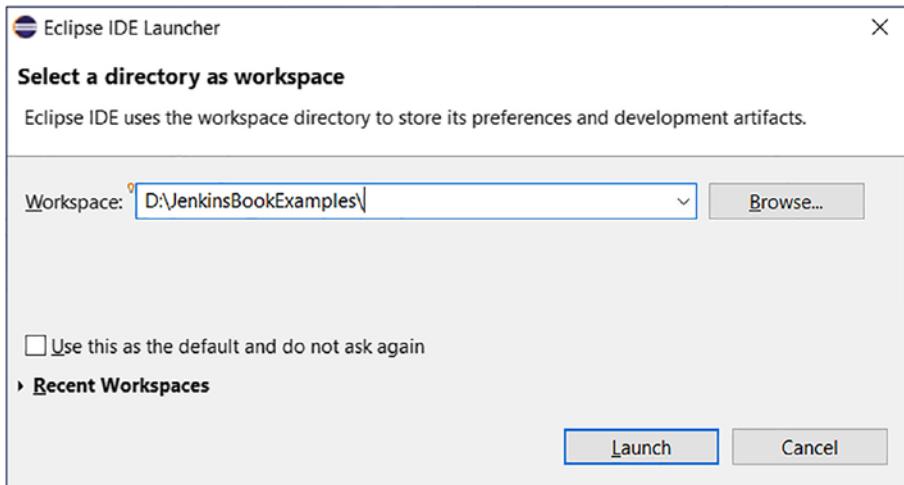
This section explains how you can use the Maven build tool to build a Java API project. You are going to use Eclipse to build the Java API project.

You need to install the following prerequisites before you can move on with this section:

- Java Development Kit (JDK)
- Eclipse (the latest version)

## Step 1: Creating a Workspace in Eclipse

An Eclipse workspace is a directory, a working location where all the projects created in Eclipse are stored. Start Eclipse. It will ask you for a workspace location (see Figure 11-1). You can provide the path of any folder; then click the Launch button.



**Figure 11-1.** The window to select Eclipse Workspace location

Once you click the Launch button, the Eclipse welcome page will appear.

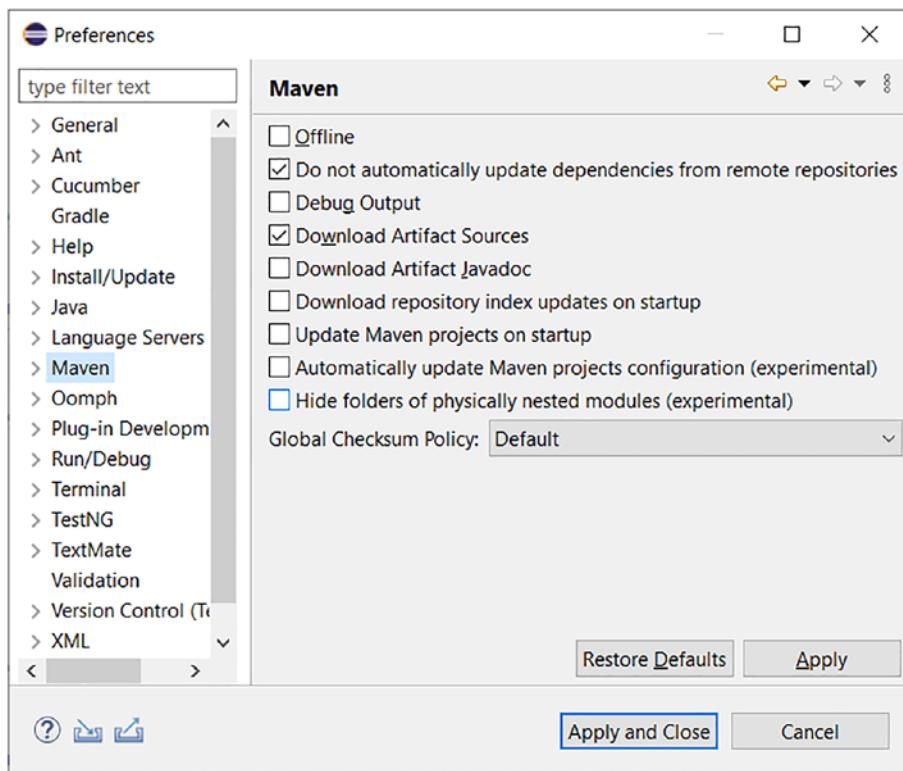
## Step 2: Creating a Maven Project

Follow these steps to create a Maven project:

1. Installing the Maven Eclipse plugin: Maven has a command-line interface (CLI). You can use Maven by running its commands using a shell program. If you want to use Maven from within Eclipse, the Eclipse Maven plugin must be installed inside

the Eclipse environment. In the latest versions of Eclipse, the Maven plugin is already installed. If you are using an older version of Eclipse, you may need to install the Maven plugin explicitly.

2. If Maven is already installed, you can see it in the Preferences window (see Figure 11-2), which opens after you choose the Window ➤ Preferences menu option.



**Figure 11-2.** Maven in Preferences in Eclipse

**Note** To download the required plugins and project dependencies for Maven, you need Internet access. If you are working from a network that has a proxy installed, then you need to create a `settings.xml` file in the  `${use.homer}\.m2` folder and include the proxy details in that file. Refer to the “Understanding Maven’s `settings.xml` File” section in this chapter for more information.

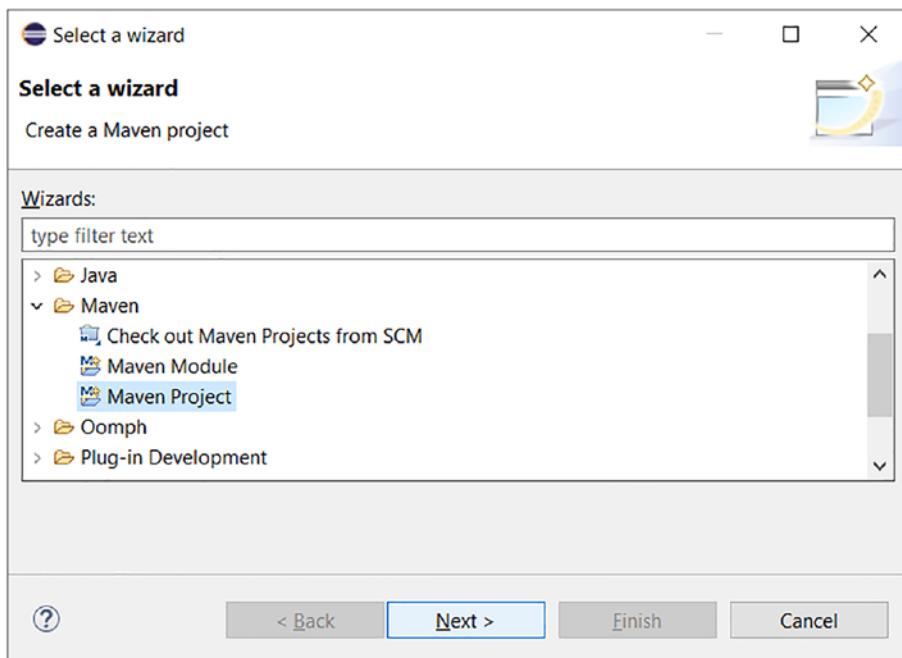
---

3. Creating a Maven project: Choose the `File > New > Other` menu option.

This will open the Select a Wizard window.

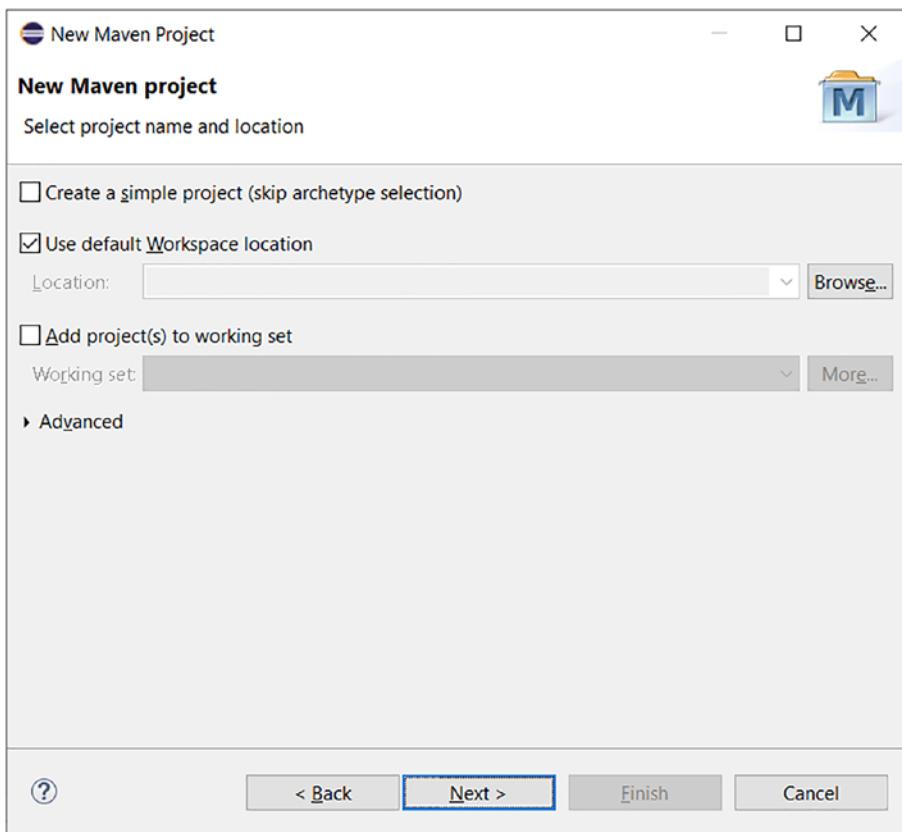
You will see a Maven section there. Open its subsections.

4. Select the Maven Project option and click the Next button (see Figure 11-3).



**Figure 11-3.** The Maven Project option

This will open the New Maven Project window (see Figure 11-4).



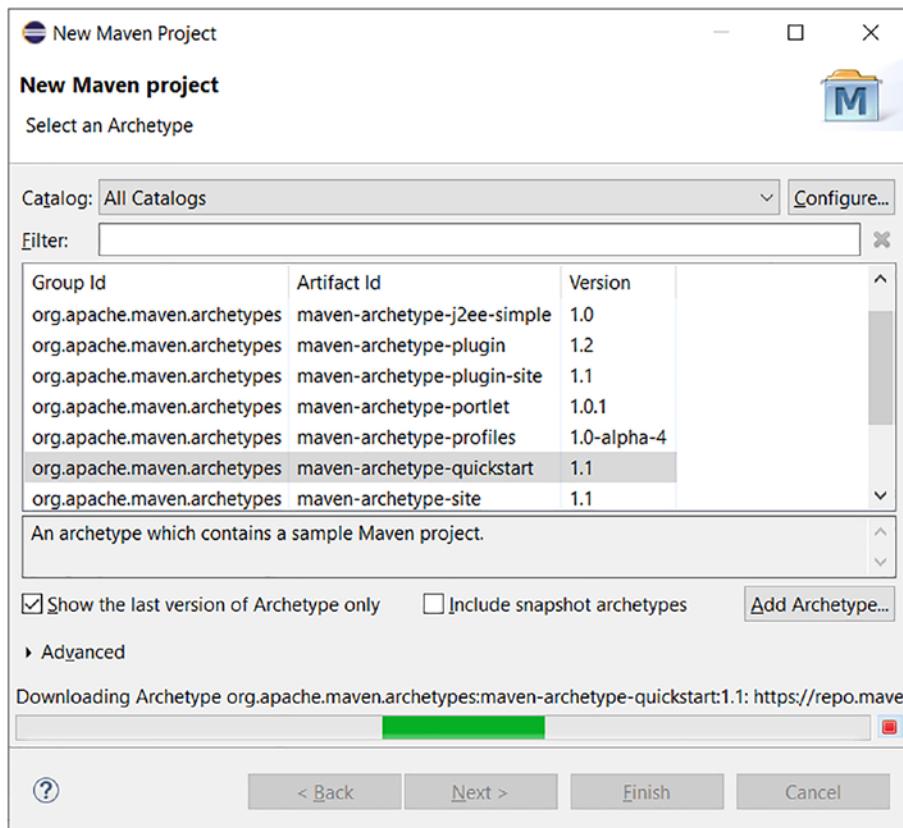
**Figure 11-4.** The New Maven Project window

5. Click the Next button, which will open the Select an Archetype window.

Maven performs each of these tasks using plugins. Different types of applications have their own unique requirements of directory structures. For example, a webapp has a WEB-INF folder and an index.html file and an EJB application has its own requirements of directory structure. Maven provides different archetype plugins which create the appropriate directory structure.

If you are developing a web application, you need the `maven-archetype-webapp`, which will create a required directory structure for your application. We are creating a simple Java API project, so we use the `maven-archetype-quickstart` plugin in this example.

6. Enter `maven-archetype-quickstart` into the Filter field and click the Next button.
7. After clicking the Next button, Maven will start downloading the artifact plugins (see Figure 11-5), which may take some time.



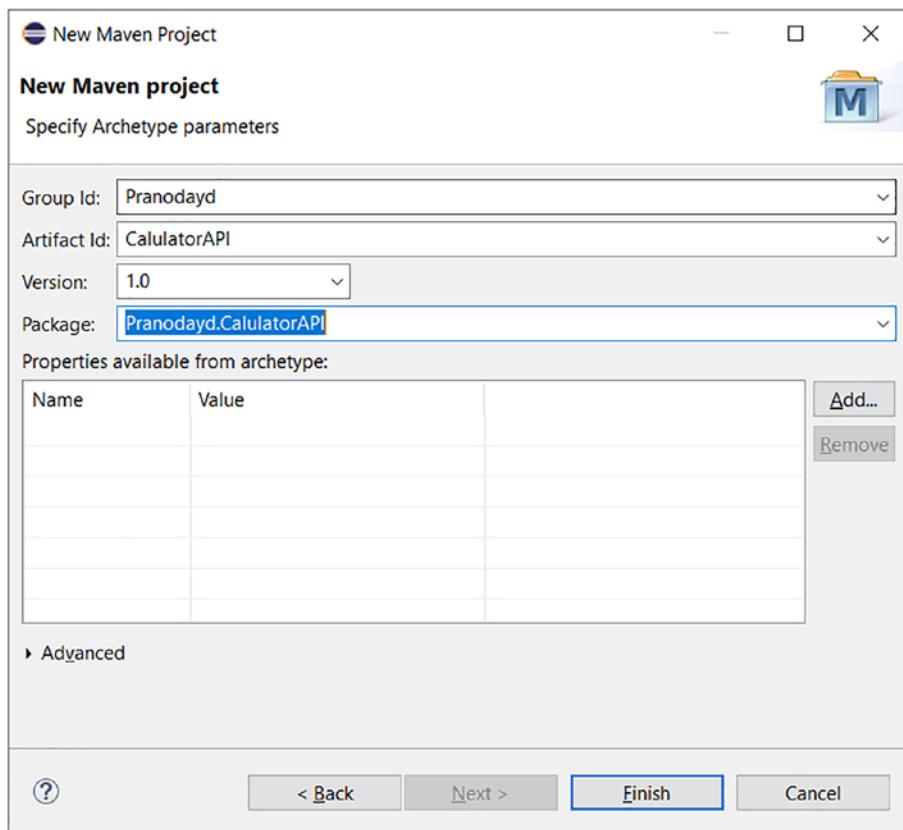
**Figure 11-5.** Downloading the artifact plugins

Once the artifacts are downloaded, you will see the New Maven Project window.

Here you'll see GroupId, ArtifactId, and Version. A Maven project is identified uniquely using these three parameters on any artifact repository, such as a MvnRepository or Nexus repository.

- *GroupId*: Indicates the owner of the project or group under which similar kinds of projects are created.
- *ArtifactId*: The name of a project.
- *Version*: A specific release of a project available to download on the artifact repository.

For example, I used a GroupId of Pranodayd, an ArtifactId of CalculatorAPI, and a Version set to 1.0 (see Figure 11-6).



**Figure 11-6.** The New Maven Project window with the Group ID, Artifact ID, and Version details filled in

8. Click the Finish button, which will start creating the Maven project.

# Understanding the Maven Project Directory Structure

The Maven project directory is created with the name you specify in the ArtifactId field when creating the project. Let's look at the directories and files created in the project.

## src/main/java

`src/main/java` is the default application source code directory in which you are supposed to create your Java package directory structure. It holds the source code files. In this folder, a package directory structure with the GroupId and ArtifactId will be created.

I created a project with the GroupId set to Pranodayd and the ArtifactId set to CalculatorAPI. You can see in the `src/main/java` directory that a directory called Pranodayd has been created. Inside this directory there is a directory called CalculatorAPI. Inside that directory, you will see a file named `App.java`, which is a template demo file that I deleted.

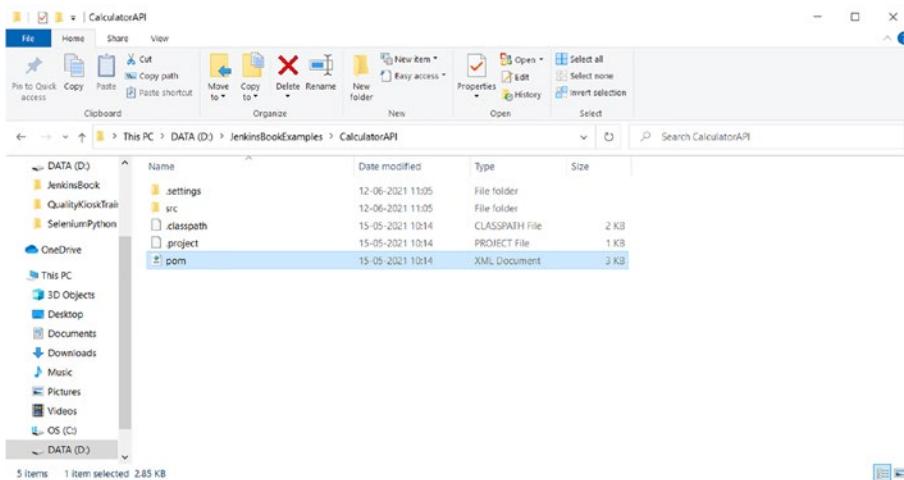
## src/test/java

This directory contains the unit test cases. It also has a package directory structure that's created using the GroupId and ArtifactId.

Inside this directory structure, you will see `Test.java`, which is a template Java file for unit test cases.

Inside the project directory, you will see the `pom.xml` file (see Figure 11-7), which we talk about later in this chapter.

## CHAPTER 11 PREPARING A JAVA API PROJECT USING MAVEN



**Figure 11-7.** The file contents of the Maven project

## Understanding Java API Project Code Files

API source code: Under `src/main/java`, inside the package directory structure, I created a file named `Calculator.java`. This file has basic arithmetic operations like addition, subtraction, multiplication, and division.

API unit test code: Under `src/test/java`, inside the package directory structure, I created Java files containing unit test cases for all arithmetic functions implemented in the API. For example `TestAdditionFunctionality.java` contains unit test cases for the addition function written in the API, `TestSubtractionFunctionality.java` contains test cases for the subtraction function, and so on.

These test cases are executed using a popular unit testing tool in the Java environment, called TestNG.

TestNG is a unit testing tool that controls the flow of unit test cases using different methods annotated with TestNG annotations, including `@BeforeClass`, `@AfterClass`, `@Test`, `@BeforeMethod`, etc. It also generates a test report.

# Understanding the pom.xml File in the Java API Project

POM stands for Project Object Model. A Maven pom.xml file is the heart of any Maven project and it defines different kinds of details of a particular project. Project information is contained in the <project></project> tags.

The information shown in Listing 11-1 defines the identification of the project.

***Listing 11-1.*** Maven Project Identification Information from pom.xml

```
<groupId>Pranodayd</groupId>
<artifactId>CalculatorAPI</artifactId>
<version>1.0</version>
```

The properties section shown in Listing 11-2 defines properties such as which Java version to use to compile the project and which text encoding to use, such as ANSI, UTF8, and so on, when compiling .JAVA files.

***Listing 11-2.*** Maven Project Properties Section from pom.xml

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.
        sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
</properties>
```

The <dependencies></dependencies> section (see Listing 11-3) defines the third-party libraries and at which stage those libraries are required (i.e., at the stage of compilation, unit testing, or when running the app).

***Listing 11-3.*** Maven Project Dependencies Section from pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.4.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

This API project only needs the TestNg dependencies and `<scope>test</scope>` defines that Maven should put TestNg JARs in the project CLASSPATH only when you run unit test cases. Once the unit test cases are done, the TestNg JARs should be removed from the project CLASSPATH. The scope defines at which stage of the lifecycle a particular dependency is required so that it can be downloaded from the central repository inside the local repository and will be added to the project's CLASSPATH.

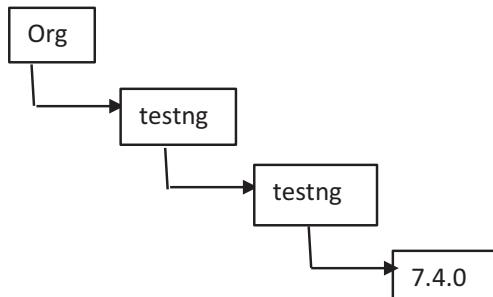
Maven maintains a local repository where it downloads all required plugins and dependencies mentioned in this section.

This local repository Maven creates by default is found at  `${use.homer}\.m2\repository`.

While running particular build phases, Maven checks if the required dependency is available in the local repository. If it is available, it will be added to the project's CLASSPATH. If it is not available, it will be downloaded from the central repository into the local repository and will be added to the project's CLASSPATH.

When Maven downloads any dependency in the local repository it creates a directory structure inside the local repository, according to the GroupId, ArtifactId, and Version of the dependency. For this API project, the TestNg dependency has the following settings: GroupId:org.testng, ArtifactId:testing, and Version:7.4.0.

When downloading this dependency, Maven will create the directory structure shown in Figure 11-8 in the local repository.



**Figure 11-8.** The directory structure created by Maven in the local repository

Inside this, the testng-7.4.0.jar file will be downloaded. You can change the default location of the local repository from \${use.homer}\.m2\repository to another location by creating a settings.xml file inside \${use.homer}\.m2. The settings.xml file should specify a desired local repository location using the <localRepository> tag:

```

<settings>
    <localRepository>D:\MavenRepo</localRepository>
</settings>
  
```

**Listing 11-4.** Maven Project Build Section from pom.xml

```

<build>
    <pluginManagement><!-- lock down plugins versions to avoid
        using Maven defaults (may be moved to parent pom) -->
        <plugins>
            <!-- clean lifecycle, see https://maven.apache.org/ref/
                current/maven-core/lifecycles.html#clean_Lifecycle -->
            <plugin>
                <artifactId>maven-clean-plugin</artifactId>
  
```

```
    <version>3.1.0</version>
  </plugin>
  <plugin>
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.0.2</version>
  </plugin>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.0</version>
  </plugin>
  <plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.22.1</version>
  </plugin>
  <plugin>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.0.2</version>
  </plugin>
  <plugin>
    <artifactId>maven-install-plugin</artifactId>
    <version>2.5.2</version>
  </plugin>
  <plugin>
    <artifactId>maven-deploy-plugin</artifactId>
    <version>2.8.2</version>
  </plugin>
</plugins>
</pluginManagement>
</build>
```

The build tag in `pom.xml` (see Listing 11-4) defines the end-end build lifecycle of an application, along with the Maven plugins used to execute each phase.

## Build Lifecycle Phases and Their Order

**Clean:** Maven uses the `maven-clean-plugin`, which cleans all previously generated compiled files as well as package files and runs a fresh build lifecycle.

**Download resources:** In this phase, all the project's dependencies are downloaded from the artifact repository, like `mvnrepository.com`.

**The Maven plugin** downloads project dependencies along with their circular dependencies.

**Compilation of application source code:** Maven uses the `maven-compiler-plugin`, which internally uses `javac` (a Java compiler) to compile the source code of the application. The compiler plugin compiles all the `.JAVA` files from `src/main/java`, which is the default location, to find the sources of an application.

This compiler plugin takes the Java files as input and produces `.CLASS` files as output of this compilation phase. The compilation phase creates `.CLASS` files for application inside the `$(ProjectDIR)/target/classes` folder.

**Unit Testing:** First `mvn compile:test-compile` phase compiles unit test code from `src/test/java` and puts the class files inside the `$(ProjectDIR)/target/test-classes` folder.

Maven uses the `Maven-surefire plugin`, which triggers unit test cases by invoking JUnit or TestNg frameworks.

**Package:** `Maven-jar-plugin` bundles application's compiled files (i.e. `.CLASS` files created inside the `$(ProjectDIR)/target/classes` folder) in a `.JAR` file. This `JAR` file is created in the `$(ProjectDIR)/target` folder.

**Release Phase:** Maven has an `install` phase in which it installs the `JAR` file into Maven's local repository using `maven-install-plugin`.

If users of this JAR file are remote, this JAR file should be deployed to a central repository available on the web, such as [Mvnrepository.com](http://Mvnrepository.com).

This can be achieved by running `deploy` goal, whereby Maven uses the `maven-deploy-plugin` to deploy a JAR file on the central repository so that it can be downloaded and used by developers.

## How to Use Maven from the CLI

The previous section explained how to use Maven from Eclipse. As previously mentioned, Maven is a command-line tool so it provides different commands to interact with it. Now you learn how to use Maven using CLI in this section.

### Setting Up Maven

Download Maven from the following link given on the Apache Maven website:

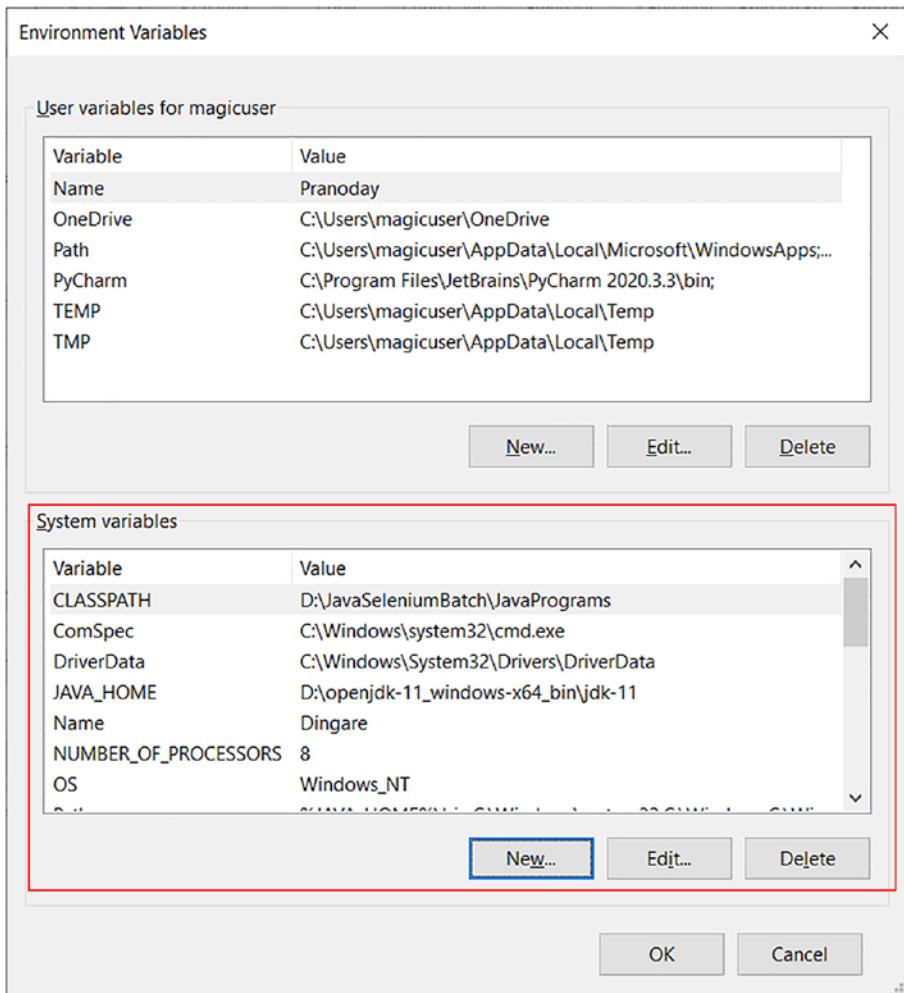
<https://apachemirror.wuchna.com/maven/maven-3/3.8.1/binaries/apache-maven-3.8.1-bin.zip>

Unzip the downloaded ZIP file at your desired location. I unzipped my file to D:\MavenInstallation.

Inside the apache-maven-3.8.1\bin, we have the `mvn` command file. This is the Maven CLI, which you can use from the command prompt. Add this `mvn` command to the PATH environment variable and create `JAVA_HOME` and `M2_HOME` variables in the Environment Variables section.

1. Go to the Environment Variables section: Type Edit System Environment Variables in the Start menu.
2. Select the Edit the System Environment Variables option, which will open the System Properties window.

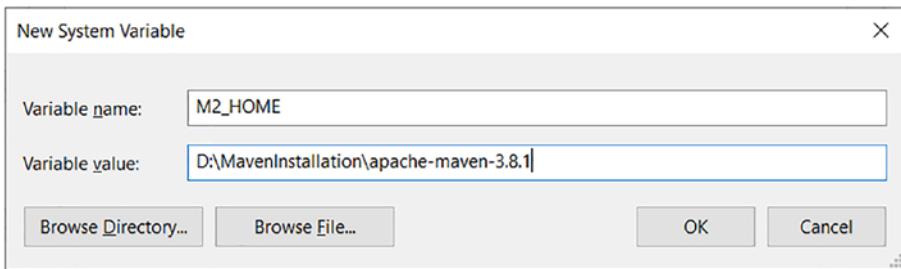
3. Click the Environment Variables button, which will open the Environment Variables window.
4. Create a M2\_HOME environment variable: Click the New button from the System Variables section (see Figure 11-9).



**Figure 11-9.** The Environment Variables window with System Variables section highlighted

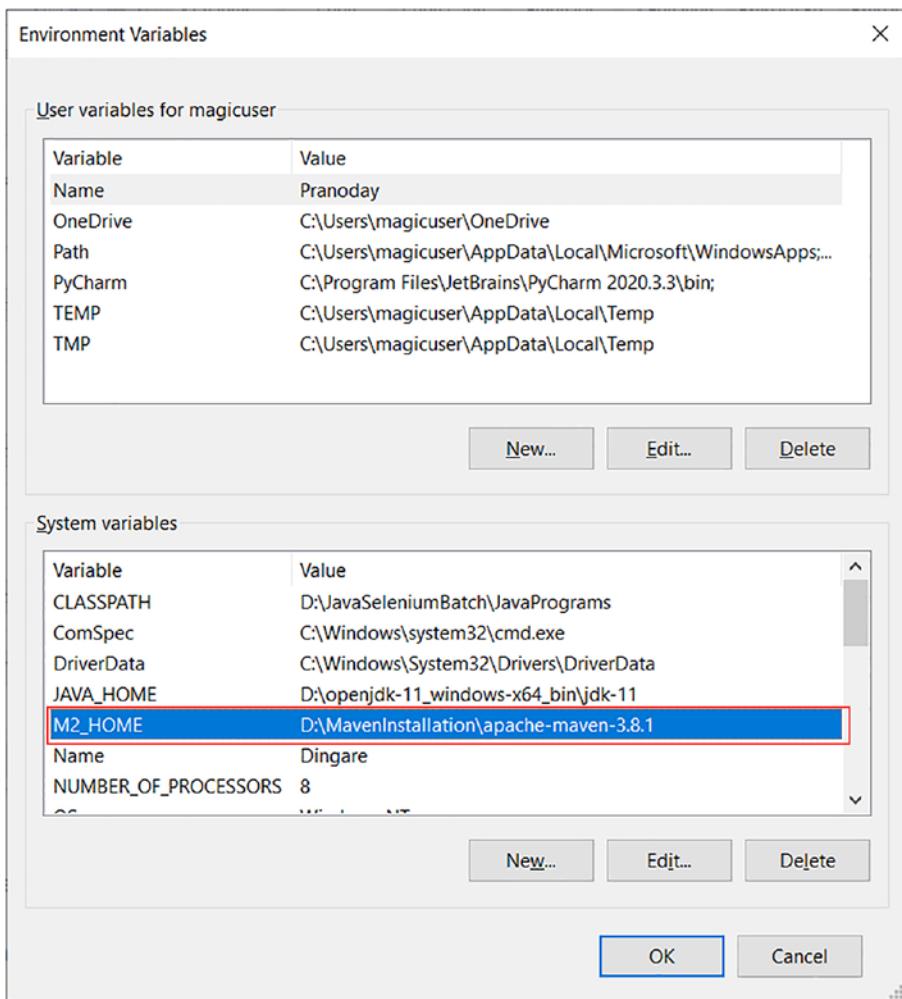
The New System Variable window will open.

5. Enter **M2\_HOME** into the Variable Name field and the location of Maven in the Variable Value field (see Figure 11-10).



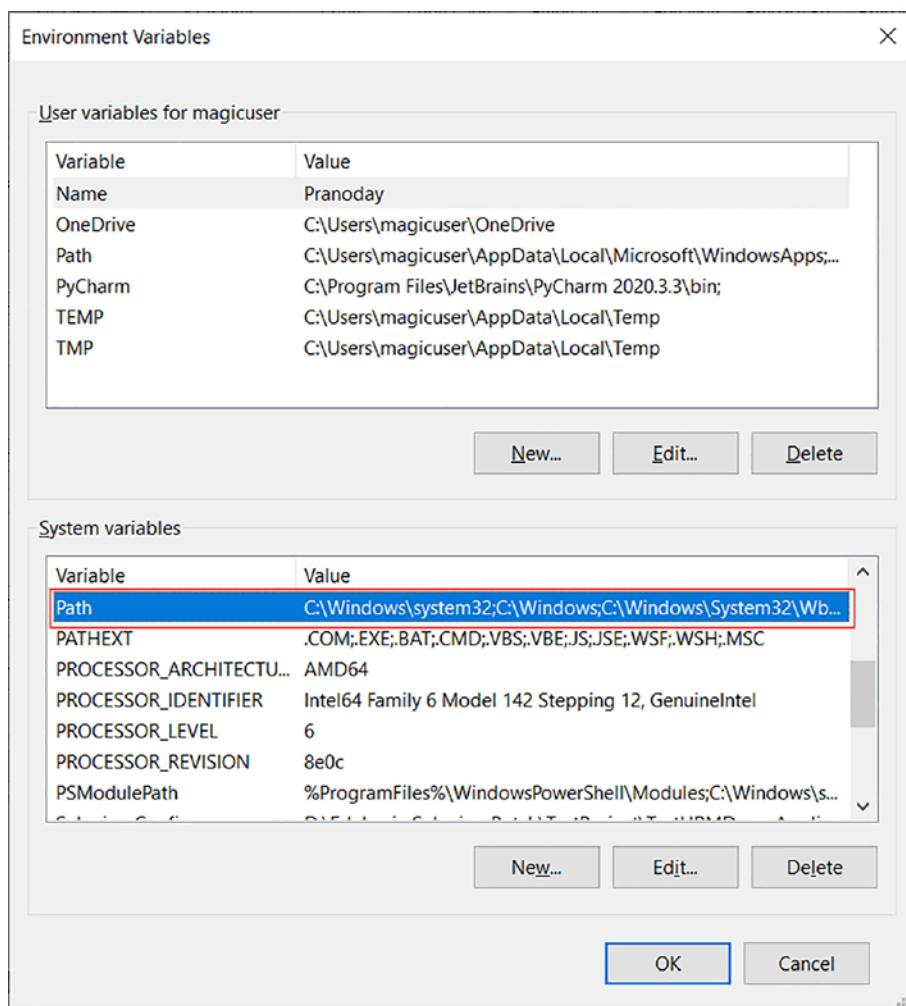
**Figure 11-10.** The Variable Name and Variable Value fields are filled in when creating the M2\_HOME environment variable

6. Click the OK button, which will add the M2\_HOME entry to the System Variables section in Environment Variables (see Figure 11-11).



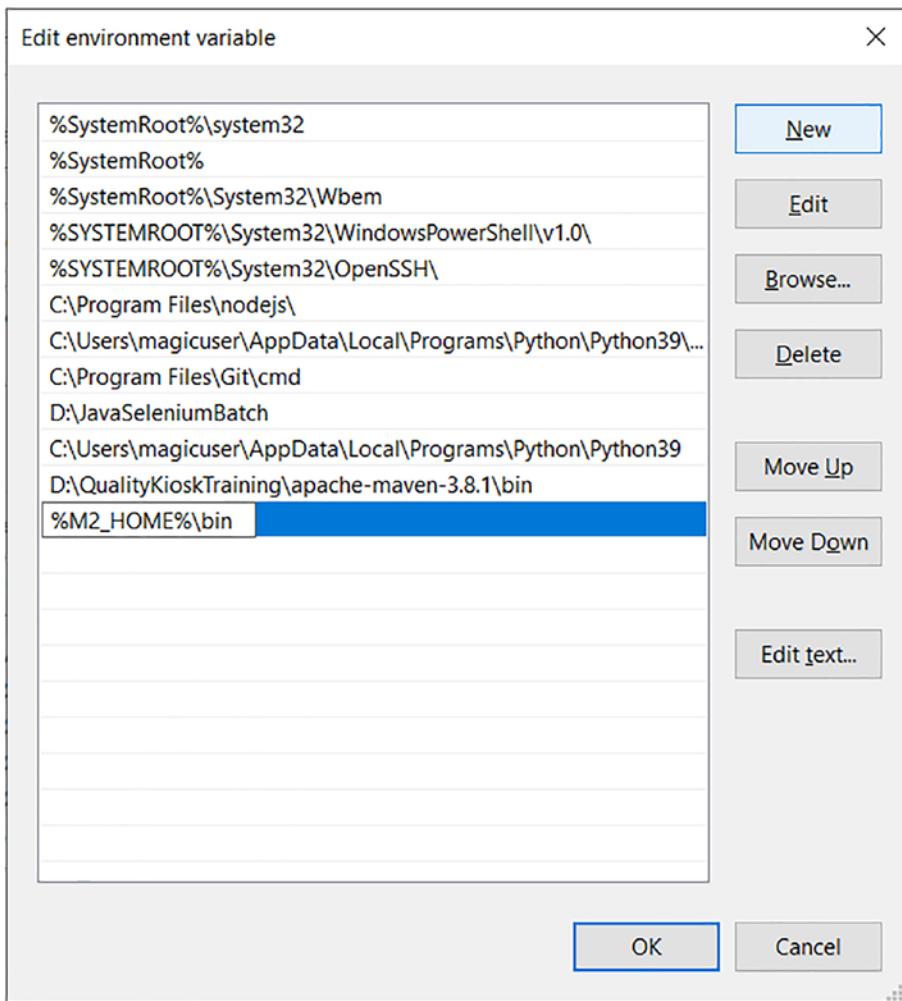
**Figure 11-11.** The M2\_HOME environment variable

7. Add a bin folder to the Maven folder in the PATH environment variable: Locate the PATH environment variable entry in the Environment Variables list under the System Variables section (see Figure 11-12).



**Figure 11-12.** The PATH environment variable in the System Variables section

8. Click the Edit button. Then click the New button and enter %M2\_HOME%\bin in the newly created entry inside the Edit Environment Variable list (see Figure 11-13).

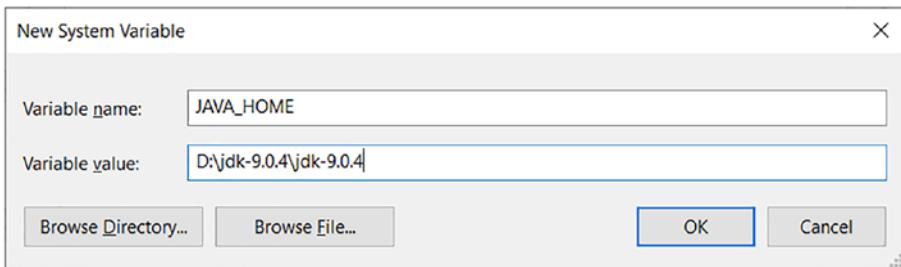


**Figure 11-13.** The bin folder entry added to the PATH environment variable

9. Click the OK button.
10. **Create the JAVA\_HOME environment variable:**  
Click the New button from the System Variables section.

The New System Variable window will open.

11. Enter JAVA\_HOME in the Variable Name field and the location of the Java Development Kit (JDK) in the Variable Value field (see Figure 11-14).



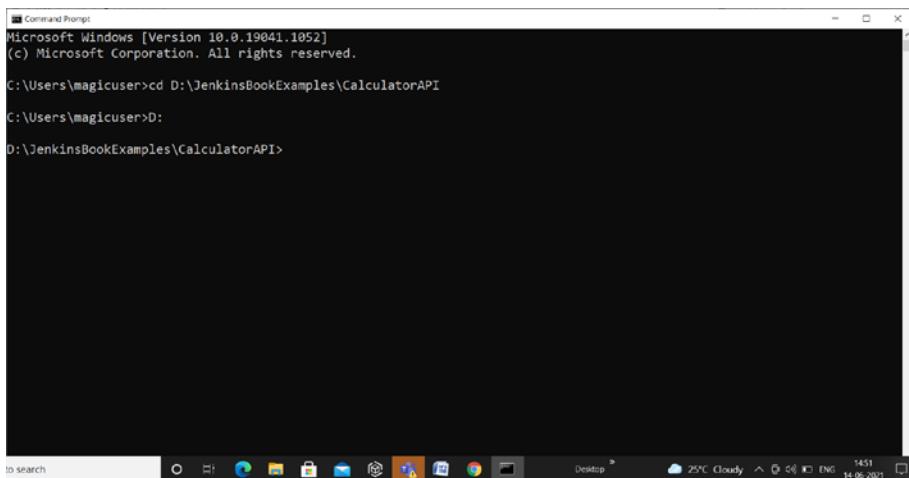
**Figure 11-14.** The Variable Name and Variable Value fields are filled in when creating the JAVA\_HOME environment variable

12. Once all the required configurations are done, check if Maven is configured successfully by running the mvn -version command from the command prompt.

## Using Maven CLI Commands

Once Maven is set up to be used from the command prompt, you can run different Maven commands to control different build lifecycle phases on the API project.

Open the command prompt and change the working directory to the project directory in the command prompt using the cd DOS command (see Figure 11-15).



```
Microsoft Windows [Version 10.0.19041.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\magicuser>cd D:\JenkinsBookExamples\CalculatorAPI
C:\Users\magicuser>D:
D:\JenkinsBookExamples\CalculatorAPI>
```

**Figure 11-15.** The working directory has changed to the project directory

---

**Note** In order to run Maven commands, you need to be in the Maven Project directory, which contains the pom.xml file.

---

To run a particular build phase, run the mvn <Goal Name> command. Let's create a .JAR file of this API project by running the package phase.

Run the mvn package command from the command prompt.

Once you run any Maven goal, it runs all its previous goals as well, For example, when you run the mvn package, it runs the following goals as well:

1. Downloads project dependencies using the maven-resources plugin.
2. Compiles API source code.
3. Compiles unit test cases code.

4. Runs unit test cases using the maven-surefire-plugin.
5. Creates a JAR file using the maven-jar-plugin.

This section explained how to use the Maven build tool to automate a build lifecycle of a simple Java API project. The next section explains how to customize default Maven settings using the `settings.xml` file.

## Understanding Maven's `settings.xml` File

A Maven build tool works with default settings. For example, the local repository's default location is  `${use.homer}\.m2\repository`. If you want to change this location to some other folder or change the central repository URL, you can do that using `settings.xml`.

For example, say you want to change the local repository location and include proxy server settings. To do this, you must create a `settings.xml` file, as shown in the Listing 11-5, and save it in the  `${use.homer}\.m2` directory.

**Listing 11-5.** Maven settings xml with Customized Local Repository and Proxy Settings

```
<settings>
    <localRepository>D:\ MavenRepo</localRepository>
    <proxies>
        <proxy>
            <protocol>http</protocol>
            <host>10.9.1.1</host>
            <port>80</port>
        </proxy>
    </proxies>
</settings>
```

# Summary

This chapter explained the typical build lifecycle phases of a Java API project and described how these build lifecycle phases can be automated using Maven. You configured a Maven project in Eclipse and executed Maven commands to perform all build lifecycle phases and create an artifact (.JAR) file. You also learned how default Maven settings can be customized using `settings.xml`. The next chapter explains how to release this artifact on an artifactory called Nexus by integrating Maven and Nexus. Stay tuned!

## CHAPTER 12

# Integrating Maven with the Nexus Repository and Creating Free-Style Jobs to Release the Java API on the Nexus Repository

In the previous chapter, you created a Java API project and learned how Maven, a build tool, can be used to manage different lifecycle phases during the build process of a typical Java API project. This chapter introduces the use of other CI/CD tools, like Git (a code repository) and Nexus (an artifact repository). You are going to build a Jenkins job to create a release of the Java API project on the Nexus repository.

This chapter discusses how Jenkins can manage to take the raw source code of a Java API project from its source code repository and how, using a build tool, you can take it through all of its build lifecycle phases until the artifact can be deployed to its final destination, which is the Nexus repository.

## Understanding Git

Git is a version control system. Using a version control system, developers can keep different versions of their code and track the changes. They can see how their application evolves by adding new code functions, removing old ones, updating existing ones to fix code defects, and so on. Version control systems also help developers work collaboratively. While working collaboratively, the source code of an application is kept at some central location accessible to all developers, which is called the *central repository*. The code on the central repository is constantly updated with newly developed, tested, and working code. While working in this collaborative environment, developers have to manage a few activities, like merging completed, tested, and working code changes of all the developers into a code kept on the central repository. Every developer needs to sync their copy of the code with the latest changes on the central repository.

We discuss this flow in detail in the next section of this chapter.

There are three types of version control systems:

### 1. Centralized Version Control Systems:

In this type of a system, the code repository is maintained on some server machine and the developers must always connect to that central code repository in order to manage code versioning. If you are not connected to the network, you cannot do version controlling of your source code on a local system. An example of centralized version controlling systems is Tortoise SVN.

## 2. Local Version Control Systems:

In these types of systems, version control happens in a code repository available on the local system. This approach is very common and simple, but at the same time, it is error prone because chances of writing changes into the wrong files are greater.

## 3. Distributed Version Control Systems:

In this type of system there are two types of code repositories:

### i) Local Repository:

Every developer has their own repository on their local system where they can track code changes and maintain different versions of the code.

### ii) Central Repository:

A version of the application code kept on some network location that all developers can access. This is the version of the code to which all developers ultimately contribute by merging their code changes.

In a Distributed Version Control system, developers manage different versions of their code files and keep track of their changes locally on their system when their change is in progress. Once their code changes are completed and tested, they send them to be merged into the ultimate centralized version of the application, which is where the application's build is created.

The advantage of Distributed Version Control systems is that you have version control capabilities when you are not connected to the network.

An example of a Distributed Version Control system is Git, discussed next.

## Installing Git

Download the suitable version of Git according to your computer system from <https://git-scm.com/downloads>. Install Git by running its setup wizard. Once Git is installed, make sure that the Git installation path is present in the PATH environment variable. The Git version control system has a CLI (command-line interface), which includes a bunch of commands.

## Understanding GitHub/GitLab

Many people have confusion around Git and GitLab/GitHub. This section explains GitLab/GitHub and how they are related to Git. Git is a version control system that provides you with ecosystem to leverage benefits of the Distributed Version Control system. Git provides a set of commands to create a local repository, create a code branch, commit code changes, and so on. GitHub/GitLab provides you with central repository, which is created on the web and can be accessed by developers.

## Understanding End-End Use of Git for the API Project

Now that you have Git installed on your system, this section goes step by step through the Git system to perform version control of the API project.

## Step 1: Creating a Local Repository

You first need to create a local repository in your project directory. Open the command prompt and go inside the project directory using the `cd` command.

Run the `git init` command to create a blank local code repository.

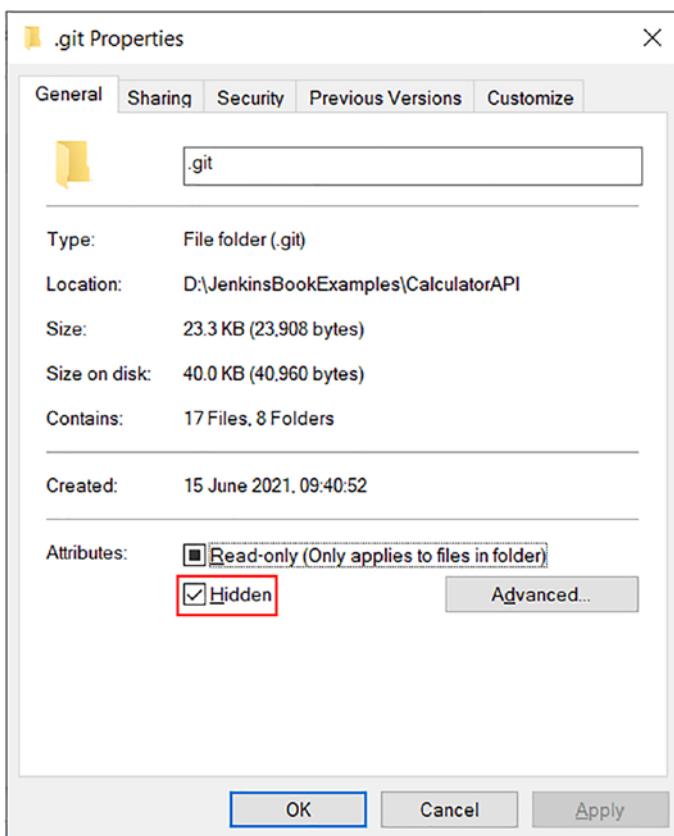
---

**Note** If you get the “git is not recognized as an internal or external command” command when you run the `git init` command, it means the windows command prompt cannot find Git on your system. Go to the System Environment Variables and add the path of the folder where you installed Git in the PATH environment variable. Save the environment variable and restart the command prompt to try again. Don’t forget to restart the command prompt after adding the entry to the PATH environment variable; otherwise, you will receive the same error. After making changes to the system environment variables, you must always restart the software.

---

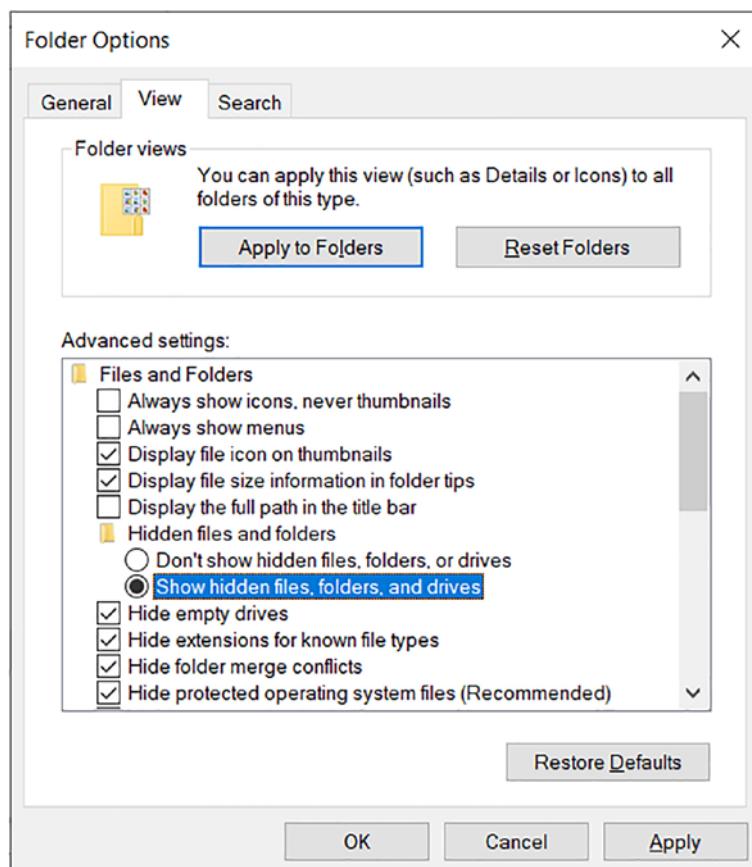
Once the local repository has been created, you will see a `.git` folder in the project directory. Note that by default, the `.git` folder has its Hidden property set to True (see Figure 12-1).

## CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY



**Figure 12-1.** The Hidden checkbox is selected in the .git properties window

Make sure that Show Hidden Files, Folders, and Drives option in the Folder Options window (see Figure 12-2) is selected to see the .git folder.



**Figure 12-2.** The Show Hidden Files, Folders, and Drives option must be selected

## Step 2: Creating a Central Repository on GitLab

Go to [https://gitlab.com/users/sign\\_up](https://gitlab.com/users/sign_up) if you do not have a GitLab account. Sign up by providing the required information on the form.

Once you register on GitLab, you can sign in using [https://gitlab.com/users/sign\\_in](https://gitlab.com/users/sign_in).

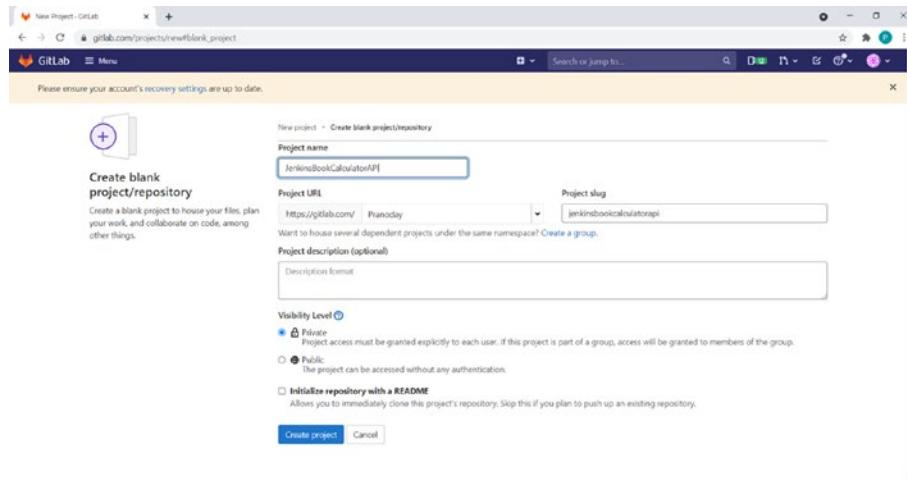
## CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY

Provide a username and password and click the SignIn button. You will end up on the Dashboard page. Click the New Project button.

This will open the Create New Project page.

Click the Create Blank Project/Repository link to create a blank repository.

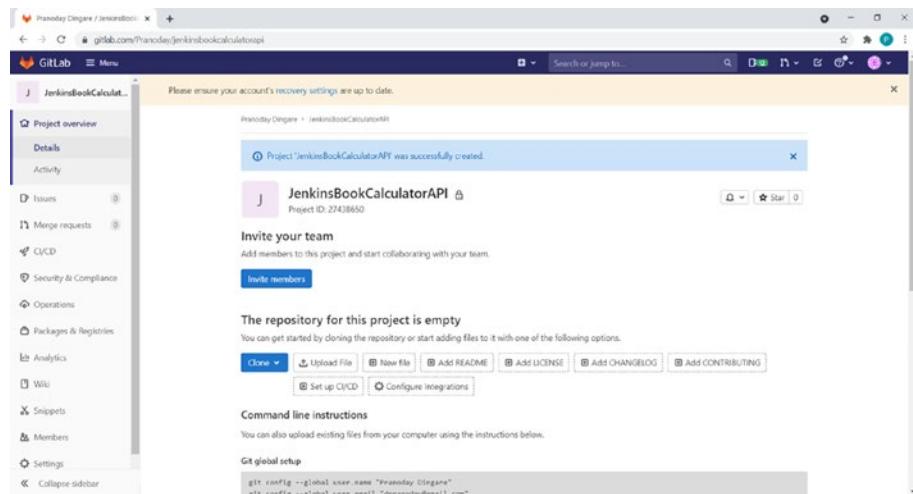
This will open a Create Blank Project/Repository page. Enter the project name in the Project Name field and set the Visibility Level to Private (see Figure 12-3).



**Figure 12-3.** The project details entered in the Create Blank Project/Repository page

Click the Create Project button. It will create a blank project, as shown in Figure 12-4.

## CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY



**Figure 12-4.** The project created with no source files

## Step 3: Committing Code to the Local Repository

Let's create a branch and commit the code to the local repository.

Open the command prompt and go to the project's directory using the `cd` command.

A copy of the code is created and you can make your changes to the branch copy without affecting the main copy. When you create a blank repository, you do not have any branches, which you can check by running the `git branch` command.

Let's create the first branch by running the `git checkout -b <BranchName>` command.

I created a code branch called `FirstBranch` by running the following command:

```
git checkout -b FirstBranch
```

Let's create a file called `.gitignore` in the project directory, with settings shown in Listing 12-1.

***Listing 12-1.*** Settings in the `.gitignore` File

```
.settings  
Target  
.classpath  
.project
```

The `.gitignore` file contains the names of directories and file changes inside which you do not want to track in the Git branches. You want to track changes done only in the `src` folder and the `pom.xml` file, so you have to add the names of other files/directories to this `.gitignore` file.

The Git system has the following three areas:

- **Working directory:** Code that has not been taken up for version control yet and the Git system is unaware of its changes.
- **Staging area:** If you want the Git system to look for changes and keep track of them in the code files, then these files need to be added to the staging area.
- **Commit area:** Once you put your changes in this area, they become permanent parts of the code branch.

Let's add the API's code to the staging area before it can be committed in the branch.

Run the `git add` command, which will add all the directories and files except those added to the `.gitignore` file in the staging area. Whenever you make any changes in such files, Git will notify you about committing them in the branch so as to not lose those changes.

Let's set a username/email for the code commits using the following commands:

```
git config--global user.name <UserName>
```

## CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY

I want to configure Pranodayd as my username for commits, so I execute the following command

```
git config--global user.name Pranodayd
```

I set my email ID using the following command:

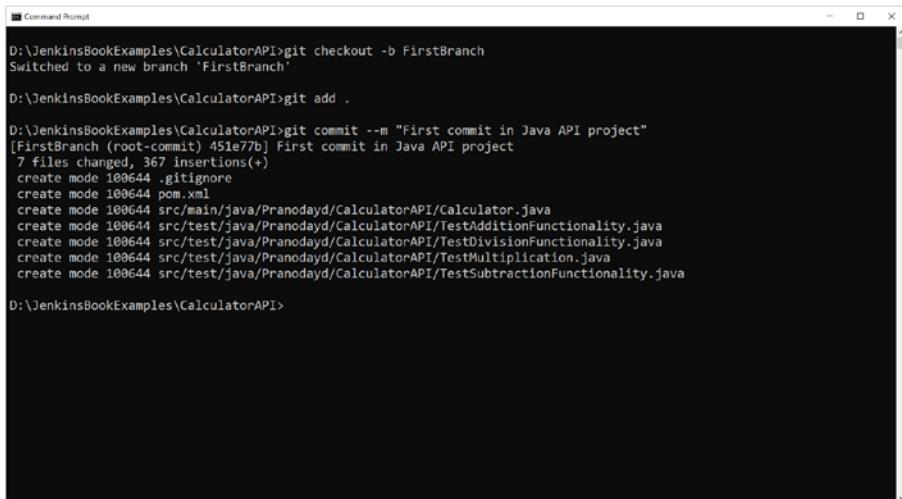
```
Git config--global user.email pranoday.dingare@gmail.com
```

Let's commit changes in a FirstBranch using the commit command. The format of the command is Git commit--m <Commit Message>.

I committed my project with the message using the following command:

```
Git commit --m "First commit in Java API project"
```

Figure 12-5 shows that only the .java files and pom.xml are committed to the branch. The rest were not part of the staging area and so are not part of the branch, as they are listed in .gitignore.



```
D:\JenkinsBookExamples\CalculatorAPI>git checkout -b FirstBranch
Switched to a new branch 'FirstBranch'
D:\JenkinsBookExamples\CalculatorAPI>git add .
D:\JenkinsBookExamples\CalculatorAPI>git commit --m "First commit in Java API project"
[FirstBranch (root-commit) 451e77b] First commit in Java API project
 7 files changed, 367 insertions(+)
   create mode 100644 .gitignore
   create mode 100644 pom.xml
   create mode 100644 src/main/java/Pranodayd/CalculatorAPI/Calculator.java
   create mode 100644 src/test/java/Pranodayd/CalculatorAPI/TestAdditionFunctionality.java
   create mode 100644 src/test/java/Pranodayd/CalculatorAPI/TestDivisionFunctionality.java
   create mode 100644 src/test/java/Pranodayd/CalculatorAPI/TestMultiplication.java
   create mode 100644 src/test/java/Pranodayd/CalculatorAPI/TestSubtractionFunctionality.java
```

**Figure 12-5.** The .java files and pom.xml files have been committed

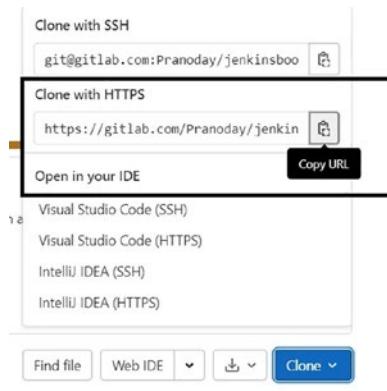
## Step 4: Pushing the Code from the Local Repository to the Central Repository on GitLab

Let's push the code in the central GitLab repository now. To push the branch from the local repository to the central repository, run the following command:

```
git push -u <URL of Gitlab repository> <Branch Name>
```

You will get the GitLab repository URL from GitLab, as shown in the following steps:

Click the Clone button present on the code repository in Gitlab.com. Click the Copy URL button from the Clone With HTTPS section, as shown in Figure 12-6.



**Figure 12-6.** The HTTPS URL of the repository

Use the URL that was copied in the `git push` command:

```
git push -u https://gitlab.com/Pranoday/jenkinsbook  
calculatorapi.git FirstBranch
```

After running this command, you will get a Git Credentials Manager prompt. Enter your Git username and password and click the OK button.

Once the username and password are verified, the code will get pushed to the GitLab repository.

Let's go to the GitLab repository, refresh the page, and verify that the code is there.

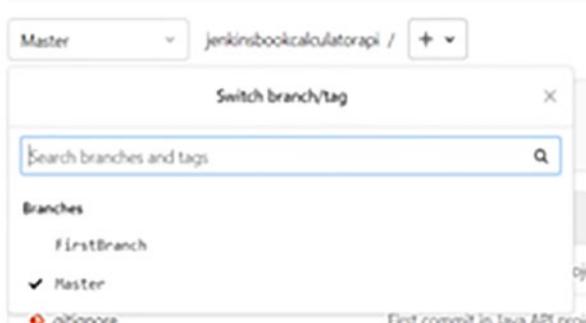
## Step 5: Creating a Master Branch in the Central Repository on GitLab

Now you'll create a new branch called **Master**, which you are going to use as a main branch, which is where all changes from the individual developer branches will be merged after review.

Click the New Branch menu option. It will open the New Branch page.

Enter **Master** in the Branch Name field. Click the Create Branch button, which will create a master branch from the code in the **FirstBranch** branch.

You now have two branches, as shown in Figure 12-7.

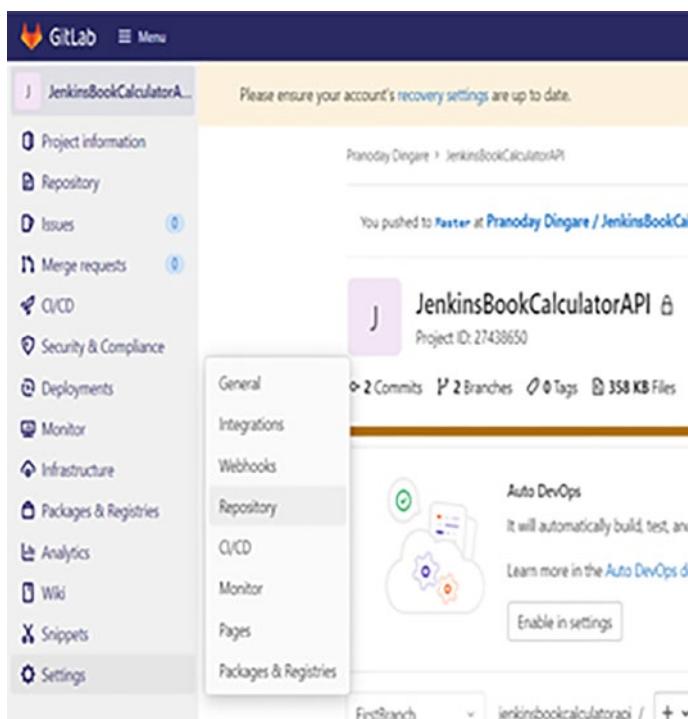


**Figure 12-7.** The two new branches

Let's change the default branch to **Master** from **FirstBranch**, as you are going to create a release from the code merged into the **Master** branch.

Go to Settings ➤ Repository, as shown in Figure 12-8.

## CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY



**Figure 12-8.** The *Settings>Repository* menu

Click the Expand button in the Default Branch section. Select Master in the Default Branch dropdown. Click the Save Changes button.

You can now observe that the master has become the default branch. Let's delete the FirstBranch now. Click the Branches link. Click the Delete Protected Branch link of the FirstBranch. This will open the Delete Protected Branch 'FirstBranch' ? window. Enter FirstBranch into the Input field and click the Delete Protected Branch button.

Now you have only one branch (the Master), which is the default branch and main branch of your application.

## Understanding the Nexus Repository

In the previous section, you learned about the Source Code Repository called Git. Now it's time to look at the Artifact Repository called Nexus.

### What is an Artifact?

An *artifact* is any output created from the build process. For example, if you are working on a web application and you package this application in the form of a .WAR file that you use to deploy the web application, this .WAR file is an artifact. In the case of the Java API project, you are going to bundle the implementation in a .JAR file, which is an artifact generated as part of the build process.

### What is the Nexus Repository?

When you add functionality to the CalculatorAPI, Git will keep different versions of the source code, but along with keeping source code versions, you need a system—an artifact repository—which will keep different versions of artifacts as you create different versions of the CalculatorAPI.jar file. The Nexus repository is such a system; it gives you a platform where you can release the artifacts and your users can download them from this repository as needed. You can deploy the artifacts to the mvnrepository.com available on web as well, but artifacts deployed here can be downloaded and used by anyone. If you want to create an artifact repository that's accessible only by the people in your organization, you should set up a Sonatype Nexus repository in your organization's network.

### Installing the Nexus Repository

Download the Nexus repository .ZIP from this link:

<https://www.sonatype.com/products/repository-oss-download>.

Unzip this folder to some location on your system. I unzipped this folder to D:\NexusRepository on my machine.

It has the following folders:

- **nexus-3.30.1-01:** This folder contains files related to the Nexus system. For example, `Nexus.exe` is present in the `bin` folder and you can use it to start the Nexus system.
- **sonatype-work:** This folder contains data of different artifact repositories you create to manage different versions of the artifacts.

## How to Start the Nexus Repository System

The Nexus repository system provides a server that you use to start on the system at the IP address of the machine and a particular port. You can set up the IP address and port where you want to start the Nexus repository server in the `nexus-default.properties` file present in the `etc` folder inside the Nexus Installation folder.

On my system, this file is at `D:\NexusRepository\nexus-3.30.1-01\etc\nexus-default.properties`.

Let's open this file and set the `application-port` to the port on which we want the Nexus server to start. You can also set `application-host` to the IP address of the machine you want to start Nexus server on. Save the changes in the file.

I set the `application-port` to 8081 and the `application-host` to 192.168.43.10, which is the IP address of my machine. You need to find the IP address of your machine by running the `ipconfig` DOS command.

Once these changes are done, start the Nexus server by running the `Nexus.exe /run` command in the command prompt. `Nexus.exe` is present in the `bin` folder inside the Nexus Installation folder.

## CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY

Open the command prompt and change the working directory to the bin folder present in the Nexus Installation directory.

Run the `Nexus.exe /run` command and wait until the Sonatype Nexus server starts. It may take a few minutes if you are setting up the Nexus repository the first time. During the setup process, a default admin user (UserName: admin) will be created and the password will be stored in the  `${NexusInstallationDir}\sonatype-work\nexus3\admin.password` file.

Click the Allow Access button on the Windows Security Alert window. The server will start successfully.

## Installing Nexus as a Service

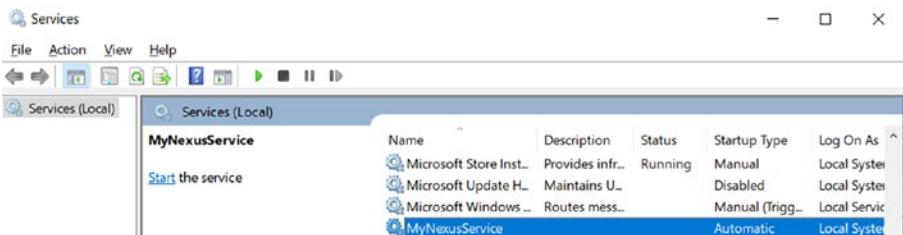
You can install Nexus as a Windows service using the following command:

```
Nexus.exe /install <ServiceName>
```

ServiceName is optional in this command.

Open the command prompt and make  `${NexusInstallationDir}\bin` the working directory. Run the `Nexus.exe /install MyNexusService` command.

You can verify in the Services area that the service is installed, as shown in Figure 12-9.



**Figure 12-9.** The Nexus service is installed with the name MyNexusService

Start the service by clicking the Start Service button.

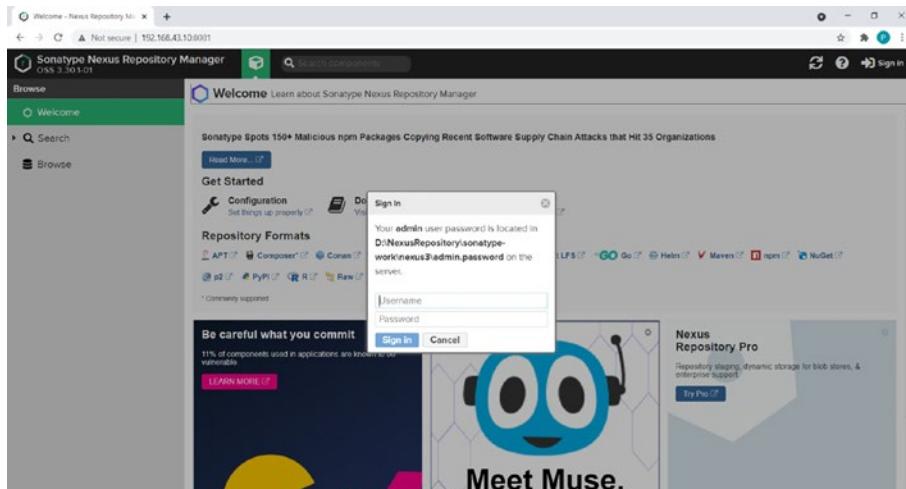
## Accessing the Nexus Repository Manager

Open a browser and access the server by using

`http://<IPAddress>:<Port>`.

I accessed my server with `http://192.168.43.10:8081`. Use your own IP address when accessing Nexus from your browser.

Click the Signin button shown in the top-right corner of the page. This will open Sign In window, shown in Figure 12-10.



**Figure 12-10.** The Sign In window in Nexus

Enter `admin` into the `UserName` field. Then enter the password generated in the `admin.password` file present in the  `${NexusInstallationDir}\sonatype-work\nexus3\` directory.

On my machine, this file is in `D:\NexusRepository\sonatype-work\nexus3`.

I copied the password from the `admin.password` file and entered the Password field in the SignIn page. Click the Signin button.

Click the Next button on the setup wizard. Enter the new admin password and click the Next button.

Select Disable Anonymous Access to allow access only by providing valid credentials. Click the Next button. Click the Finish button to complete the setup.

## Creating a Hosted Repository to Release the Artifacts

Let's create a hosted repository in the Nexus repository. Click the Server Administration and Configuration button. Click the Repositories link. Then click the Create Repository button. This will open the Repositories/Select Recipe page. Select Maven2 (hosted) from the list.

Name the repository in the Name field on the Create Repository:maven2 (hosted) page. I named my repository JenkinsBookCalculatorAPI\_Release.

Scroll down the page and click the Create Repository button. You created the repository and it's shown in the Repositories list on the Repositories Manage Repositories page.

## Integrating Maven and the Nexus Repository

You have configured the Nexus repository that you want to use as the central artifact repository to release the Java API Calculator.jar file. The Maven build tool by default uses <https://repo.maven.apache.org/maven2> as a central repository, which is where the project's artifact (.JAR) file gets deployed. Now you need to tell Maven to use the newly-created Nexus repository to deploy the artifacts instead of using <https://repo.maven.apache.org/maven2>.

## CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY

You need to list the Nexus repository URL in the project's pom.xml file. You can get the repository URL by clicking the repository link present in the repositories list on the Repositories Manage Repositories page.

Copy the URL in the URL field.

Open pom.xml and add this URL under the <distributionManagement></distributionManagement> tag, as shown in Listing 12-2.

**Listing 12-2.** Configured URL of the Repository in pom.xml in the <distributionManagement> Tag

```
<distributionManagement>
    <!-- Publish the versioned releases here -->
    <repository>
        <id>PranodaydNexusRepo</id>
        <name>PranodaydNexusRepo</name>
        <url>http://192.168.43.10:8081/repository/
            JenkinsBookCalculatorAPI_Release/</url>
    </repository>
</distributionManagement>
```

The Nexus repository does not allow anonymous access, so you have to provide the username and password of the Nexus repository in the settings.xml file (see Listing 12-3). Create a settings.xml file if one is not already present in \${use.homer}\.m2 and list the Nexus repository username and password in it.

**Listing 12-3.** Configured Nexus Username and Password in settings.xml

```
<servers>
    <server>
        <!-- this id should match the id of the repo server
            mentioned in pom.xml -->
```

```
<id>PranodayNexusRepo</id>
<username>admin</username>
<password>admin123</password>
</server>
</servers>
```

---

**Note** The `<id>` listed in the `<Repository></Repository>` tag in `pom.xml` should be same as the one listed in `<id>` in `settings.xml` in the `<server></server>` tag.

---

## Releasing the CalculatorAPI.jar File in the Nexus Repository

You have configured the Nexus Repository URL and the credentials in Maven. Now you are all set to release the `CalculatorAPI.jar` in Nexus repository.

Open the command prompt and change the working directory to the Java API project directory using the `cd` command.

Run the `mvn deploy` command.

This command will run all the build lifecycle phases like downloading resources, compiling source code, compiling unit test code, running unit test cases, and packaging class files in `CalculatorAPI1.0.jar`. As a final phase this created `CalculatorAPI1.0.jar` will be deployed to the Nexus repository.

Let's go to the Nexus repository and see the release there. Go to the Nexus Repository dashboard. Click the `Browse` link available on left side of the dashboard repository created in the `Repositories` list on the `Repositories Manage Repositories` page.

Click the repository entry shown in the list on the Browse Assets and Components page.

Hurray! We released the `CalculatorAPI1.0.jar` file to the Nexus repository.

## **Creating a Jenkins Free-style Job to Release the CalculatorAPI.jar in the Nexus Repository**

In this section, you learn how to set up Jenkins and create a free-style job to release a new version of the calculator (`CalculatorAPI2.0.jar`) in the Nexus repository.

### **Step 1: Setting Up Maven in Jenkins**

Start the Jenkins server and sign into Jenkins. Go to the Jenkins dashboard. Choose `Manage Jenkins` ➤ `Global Tool Configuration`. This will take you to the Global Tool Configuration page.

The Default Settings Provider field has a field called `Use Default Maven Settings`, which is the default `settings.xml` file in `~\use.homer\.m2`. This is the file where you set the Nexus repository credentials. So no need to change anything here.

Scroll down the page and click the `Add Maven` button. This will expand the Maven section.

Enter a name in the Name field, uncheck `Install Automatically`, and specify the Maven installation directory path in the `MAVEN_HOME` field. Click the `Save` button.

## Step 2: Adding Git Repository Credentials to Jenkins

Go to the Credentials Manager and create a credentials entry with the Git username and password. Refer to Chapter 8 for detailed steps for creating a credentials entry. I created a credentials entry in the Jenkins Credentials Manager.

## Step 3: Creating a Free-Style Job from the Jenkins Dashboard

Click the New Item link on the Jenkins dashboard. Enter a job name in the Enter an Item Name field and select the Free-style Project option. Click the OK button.

I called my job ReleaseCalculatorAPI and selected the Free-style Project option.

Select the Git radio button in the Source Code Management section and enter the Git code repository HTTPS URL in the Repository URL field. Refer to the “Push the Code from Local Repository to Central Repository on GitLab” section in this chapter to learn the steps required to get the repository URL from the GitLab repository.

Select the credentials entry with the GitLab username/password from the Credentials dropdown and the enter branch name as Master in the Branch Specifier field. This example uses Master for the main Git repository branch.

Scroll down the page to add a build step. Click the Add Build Step dropdown and select the Invoke Top-Level Maven Targets option.

Select the MyMaven option in the Maven Version dropdown. Enter Deploy in the Goals field. Click the Save button.

## Step 4: Add a Subtraction Function and Unit Test Cases to the API Project

Listing 12-4 shows the Subtraction function added to Calculator.java.

***Listing 12-4.*** Subtraction Function from Calculator.java

```
public int Subtraction(int num1,Int num2)
{
    int Res=num1-num2;
    return Res;
}
```

I created a file named TestSubtractionFunctionality.java under the Pranodayd.CalculatorAPI package in the src/test/java folder and added a few unit test cases for the subtraction functionality to it. Refer to Listing 12-5.

***Listing 12-5.*** Unit Test Cases Implemented in TestSubtractionFunctionality.java

```
package Pranodayd.CalculatorAPI;

import org.testng.Assert;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeMethod;
public class TestSubtractionFunctionality
{
    Calculator Cal;
    int Result;
    @BeforeClass
    public void Init()
```

CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY

```
{  
    Cal=new Calculator();  
  
}  
  
@BeforeMethod  
public void ReInitialise()  
{  
    Result=0;  
}  
  
@Test(priority=1,groups= {"RegressionTest"})  
public void TestSubtractionWithPositiveNumbers()  
{  
  
    Result=Cal.Subtraction(50, 10);  
    Assert.assertEquals(Result, 40,"Subtraction does  
    not work with Positive Numbers");  
}  
  
@Test(priority=2)  
public void TestSubtractionWith1Positive1NegativeNumbers()  
{  
  
    Result=Cal.Subtraction(50, -10);  
    Assert.assertEquals(Result, 60,"Subtraction does  
    not work with 1 Positive and 1 Negative Numbers");  
}  
  
@AfterClass  
public void Teardown()  
{  
    Cal=null;  
}  
}
```

Let's change the version from **1.0** to **2.0** in the project's `pom.xml`, as we added new functionality to the API and are now planning to release version 2.0 of `CalculatorAPI.jar`. I changed the version to 2.0, as shown in Listing 12-6.

**Listing 12-6.** GroupId, ArtifactId, and Version Tags from `pom.xml`

```
<groupId>Pranodayd</groupId>
<artifactId>CalculatorAPI</artifactId>
<version>2.0</version>
```

Save the changes in `pom.xml` as well as the `.java` files you just created. The developer will run Unit test cases and see them pass before they move ahead with committing and pushing the change.

## Step 5: Commit Changes in the Local Repository and Push them to the Central GitLab Repository

Open the command prompt and enter into the project directory using the `cd` command.

Let's create a new branch to keep track of this new change, using the following command:

```
Git checkout -b "SubtractionFunction"
```

Let's add the changes in the staging area by running the `git add` command. Let's commit changes in the newly created branch by running this command:

```
git commit --m "Adding subtraction function in Java API
project"
```

Now we'll push the branch in the Git central repository using this command:

```
git push -u https://gitlab.com/Pranoday/jenkinsbookcalculatorapi.git SubtractionFunction
```

---

**Note** Git may not ask for credentials this time because last time, when you pushed the code to the GitLab repository, you provided the Git credentials. They can be saved in the Windows Credentials Manager from the Control Panel. If Git asks for credentials, enter the Git username and password, just the way you did in the previous occasion.

---

## Step 6: Merge the SubtractionFunction Branch with the Master Branch on the Central GitLab Repository

As soon as you refresh the GitLab repository page in the browser, you will see a notification at the top of the page that the SubtractionFunction branch has been pushed.

Click the Create Merge Request button. It will open New Merge Request page.

Enter a description about the new change implemented in the Description field. It is an optional field. Click the Assign to Me link available from the Assignee field to assign this merge request to yourself.

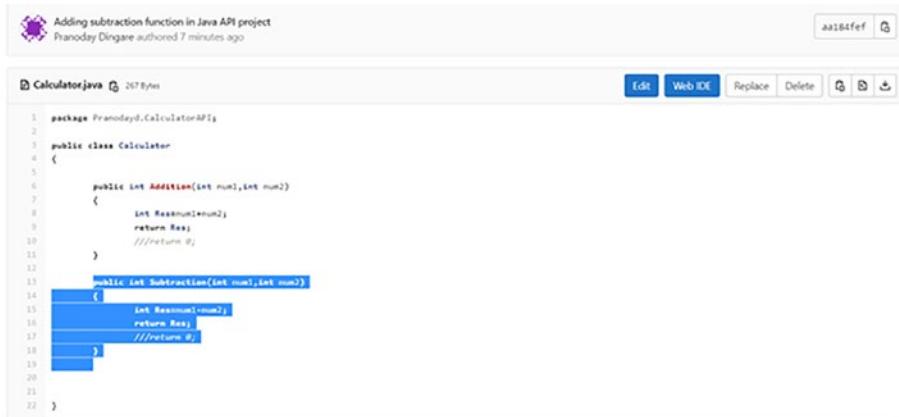
Scroll down the page to see the Create Merge Request button.

Keep the Delete Source Branch When Merge Request Is Accepted checkbox checked. Once the SubtractionFunction branch merges with the Master branch, the SubtractionFunction branch will be deleted from the central repository.

## CHAPTER 12 INTEGRATING MAVEN WITH THE NEXUS REPOSITORY AND CREATING FREE-STYLE JOBS TO RELEASE THE JAVA API ON THE NEXUS REPOSITORY

Click the Create Merge Request button. Click the Merge button to merge changes in the Master branch

Figure 12-11 shows that the changes were merged successfully in the Master branch, because the Subtraction function can be seen in the Calculator class.



The screenshot shows a code editor interface for a Java file named 'Calculator.java' in a project titled 'Adding subtraction function in Java API project'. The file contains two methods: 'Addition' and 'Subtraction'. The 'Subtraction' method is currently selected, indicated by a blue vertical bar highlighting its code block. The code is as follows:

```
1 package PranodeyD.CalculatorAPIs
2
3 public class Calculator
4 {
5
6     public int Addition(int num1,int num2)
7     {
8         int Resnum1num2;
9         return Res;
10        //return 0;
11    }
12
13    public int Subtraction(int num1,int num2)
14    {
15        int Resnum1num2;
16        return Res;
17        //return 0;
18    }
19
20
21
22 }
```

**Figure 12-11.** The Subtraction function in Calculator.java in the GitLab repository

## Running a Jenkins Free-Style Job to Perform a Release of the CalculatorAPI.jar in the Nexus Repository

Click the clock sign next to the job entry on the dashboard. You can see the progress in the Build Executor Status section on left side of dashboard.

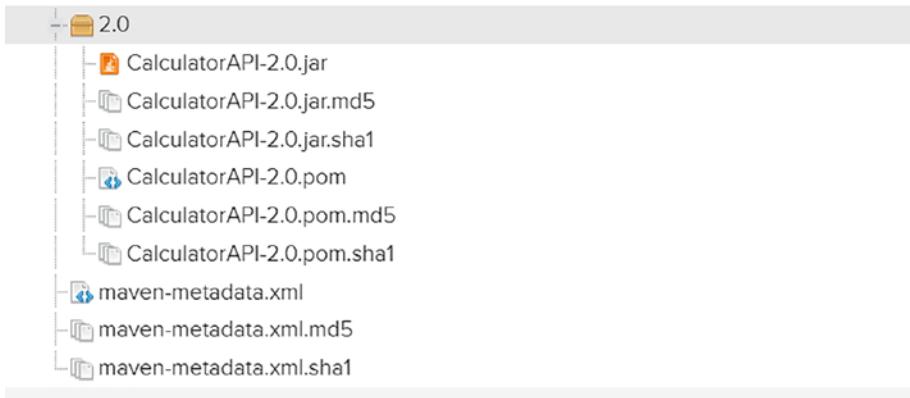
Once the job has executed, you can see its console output (see Figure 12-12).

## Console Output

```
Started by user Pranoday Dingare
Running as SYSTEM
Building in workspace C:\Users\magicuser\.jenkins\workspace\ReleaseCalculatorAPI
The recommended git tool is: NONE
using credential MyGitCredentials
Cloning the remote Git repository
Cloning repository https://gitlab.com/Pranoday/jenkinsbookcalculatorapi.git
> git.exe init C:\Users\magicuser\.jenkins\workspace\ReleaseCalculatorAPI # timeout=10
Fetching upstream changes from https://gitlab.com/Pranoday/jenkinsbookcalculatorapi.git
> git.exe --version # timeout=10
> git --version # 'git' version 2.30.1.windows.1'
using GIT_ASKPASS to set credentials
> git.exe fetch --tags --force --progress -- https://gitlab.com/Pranoday/jenkinsbookcalculatorapi.git +refs/heads,
> git.exe config remote.origin.url https://gitlab.com/Pranoday/jenkinsbookcalculatorapi.git # timeout=10
> git.exe config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git.exe rev-parse "refs/remotes/origin/Master^{commit}" # timeout=10
Checking out Revision 406ab05af2e174d73b1cdcac2f82082d81de34cd (refs/remotes/origin/Master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f 406ab05af2e174d73b1cdcac2f82082d81de34cd # timeout=10
Commit message: "Merge branch 'SubtractionFunction' into 'Master'"
First time build. Skipping changelog.
[ReleaseCalculatorAPI] $ cmd.exe /C "D:\MavenInstallation\apache-maven-3.8.1\bin\mvn.cmd package && exit %ERRORLEVEL%
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building CalculatorAPI 2.0
[INFO] -----
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ CalculatorAPI ---
```

**Figure 12-12.** The console output of the executed job

Go to the Nexus repository to confirm successful release of the CalculatorAPI2.0.jar file. Figure 12-13 shows that the CalculatorAPI 2.0.jar file was released successfully in the Nexus repository.



**Figure 12-13.** The CalculatorAPI 2.0 released in the Nexus repository

## Summary

This chapter explained the distributed version control system called Git. You also looked at different Git commands, like `git init`, `git add`, `git commit`, and `git push`. Then you created a central repository on GitLab and pushed the source code of the Java API code. You learned about the concept of branching and saw how to merge two branches once the developer is finished developing the feature and unit testing it. You also learned about the popular Artifact repository called Nexus. At the end of the chapter, you integrated Maven with Nexus and released a new version of the Calculator API on Nexus by running a Jenkins job. The next chapter explains how to create auto-triggered Jenkins jobs.

## CHAPTER 13

# Creating an Auto-Trigger Free-Style Job to Manage Java API Releases

In the last chapter, you learned about Git, GitLab, the Nexus repository, and how to integrate the Maven build tool with the Nexus repository. You also created a free-style Jenkins job to release the CalculatorAPI2.0 JAR on the Nexus repository. Now that you have the required knowledge, it's time to look at a few more interesting, real-time scenarios.

This chapter explains how to add a new user as a contributor to a private GitLab repository, what SSH authentication is, how to apply SSH authentication to a GitLab repository, how to connect to an SSH authenticated GitLab repository from Jenkins, and how to create a Jenkins job that will poll SCM and trigger the automatic build execution.

## How to Add a New Code Contributor to a Private GitLab Repository

In the last chapter, when you worked with the GitLab repository, you were using the GitLab credentials of a GitLab user who created a GitLab repository. Such a person has administrator privileges on the GitLab code repository. This person could be a developer working on an application. In a development team, multiple developers seldom work with the application. In order to allow these other developers to work with the GitLab repository, the administrator needs to add them as contributors. The administrator can give different roles to different people working with the application, based on the kind of contribution they are going to make. For example, a few team members will only report bugs on an application, so they might need *reporter* privileges. Other team members are going to work as developers so they need more privileges than the previous ones. A GitLab administrator can add team members to the Code repository and assign them different roles.

## How to Invite a Team Member to the Code Repository

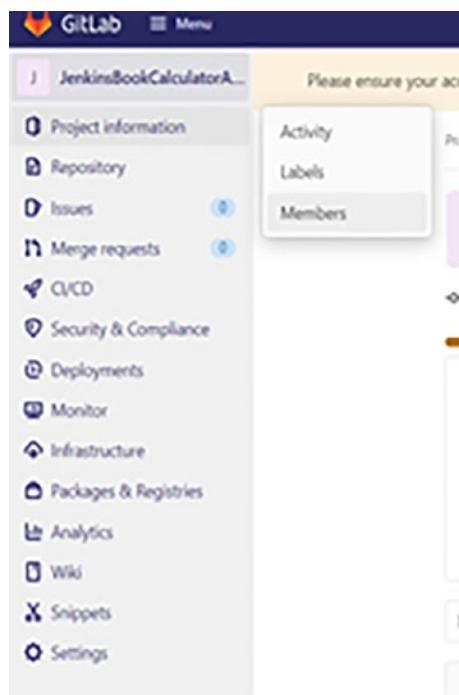
Only the administrator of a repository can invite a new team member. The person who is the administrator wants to invite should have a valid GitLab account.

Here's the step-by-step process for this:

1. Log into the GitLab repository with administrator credentials: Log into the GitLab with the credentials of the GitLab user who created the Code repository. The JenkinsBookCalculatorAPI GitLab repository was created by the dpranoday@gmail.com user. I am logging in with the credentials of this user in GitLab.

After logging in, you will land up on the GitLab dashboard.

2. Go to the code repository: Find the required code repository on the GitLab dashboard. The required code repository is called JenkinsBookCalculatorAPI.
3. Click the GitLab repository link to enter into a repository, which will take you to the repository main page.
4. Go to the Project Members page: Choose Project information ► Members on the left side of the page (see Figure 13-1).



**Figure 13-1.** The Members menu option under the Project Information menu

This will take you to the Project Members page.

5. Add a new member to the Invite Member tab: Type the GitLab user ID of a member you want to invite to the GitLab Member or Email Address field.
6. Select the desired member from the dropdown. I want to invite Pranoday Dingare (@Pranodayd), so I clicked that entry.
7. Set a role to this user now by selecting the role from the dropdown in the Select a Role field. Click the Select a Role field, which will open a list of roles.

I want to invite this member as a developer, so I selected the Developer option from the dropdown.

8. Click the Invite button. The person who has been invited will get an invitation email from their GitLab account email address.

Now this member can contribute their changes to the Java API code repository.

## Understanding SSH Authentication

You can authenticate access to the GitLab code repository using various techniques, like basic authentication, SSH authentication, API Key, etc. In basic authentication, you authenticate the user based on their GitLab username and password, which you saw in the previous chapter. In the previous chapter, you learned how to push the code to the GitLab repository. You also learned how to authenticate Jenkins access to the GitLab repository by creating a credentials entry. In this section, you learn about SSH authentication, which is where the user is authenticated using a public and private key pair.

## Why You Need SSH Authentication

If you authenticated the developer on the GitLab repository using their username/password (i.e., basic authentication), they can access the code repository on any machine (even their personal computers) by sending their valid username and password. But if you want to restrict access to the code repository only from a particular machine (such as an office workstation), you can do so by applying SSH authentication to your GitLab code repository.

## How SSH Authentication Works with GitLab

This technique of access works in the following way:

1. In this technique a developer wanting to access your code repository needs to generate a private key and public key on their machine.

They need to send that generated public key to the administrator of the GitLab code repository.

2. The administrator will then add this public key to the repository they want to grant an access to.
3. When the developer accesses the code repository, a private key stored on their machine will be sent to GitLab. It will check for a public key that matches the private key. If the keys match, access is granted; otherwise, it will be rejected.

## Applying SSH Authentication to the Java API Code Repository

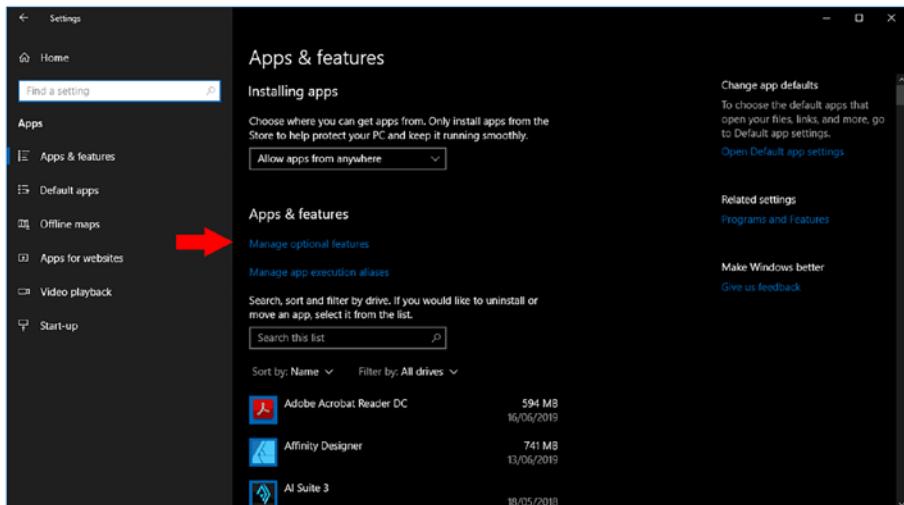
Let's apply SSH authentication to your Java API Code repository. Follow these steps given:

### Step 1: Generating the Public and Private Keys

You need to generate a public and private key pair for the machine that holds the code repository. To generate a key pair, you need to use ssh-keygen tool from the ssh-keygen command.

You need to make sure that OpenSSH is installed on your machine, which you will find installed if you are using Windows 10 (10 April 2018 Update and above). If you are using an older version of Windows, you need to install OpenSSH first.

Launch the Settings app and click the Apps & Features category (see Figure 13-2).



**Figure 13-2.** The Apps & Features window

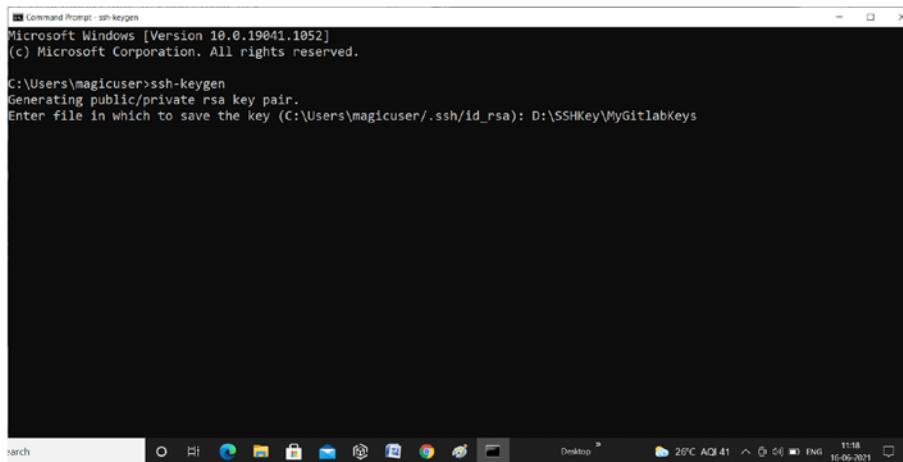
Click the Manage Optional Features link. If you don't see OpenSSH client in the list, click the Add a Feature button and install it. You might need to reboot the machine after installation. Once the OpenSSH client is installed, you are all set to generate the SSH key pair.

Open the command prompt and run the ssh-keygen command.

It will ask for the path of the file where key pair is to be generated. If you press Enter without entering anything, the key pair will be generated in the default path, i.e., \$(user.home)\.ssh\is\_rsa.

I want to generate the SSH key pair in D:\SSHKey\MyGitlabKeys so I enter this path: D:\SSHKey\MyGitlabKeys (see Figure 13-3).

## CHAPTER 13 CREATING AN AUTO-TRIGGER FREE-STYL



The screenshot shows a Windows Command Prompt window titled "Command Prompt - ssh-keygen". The window displays the following text:

```
Microsoft Windows [Version 10.0.19041.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\magicuser>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\magicuser/.ssh/id_rsa): D:\SSHKey\MyGitlabKeys
```

The taskbar at the bottom of the screen shows various icons for applications like File Explorer, Edge, and Google Chrome. The system tray indicates the date as 16-06-2021 and the time as 11:18.

**Figure 13-3.** Executing the ssh-keygen command

It will ask you to enter the passphrase, which is a password you need to provide whenever you use this SSH key.

```
C:\Users\magicuser>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\magicuser/.ssh/id_rsa): D:\SSHKey\MyGitlabKeys
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in D:\SSHKey\MyGitlabKeys.
Your public key has been saved in D:\SSHKey\MyGitlabKeys.pub.
The key fingerprint is:
SHA256:6NlplLdDGKEYvBhoMpCtTY0VqhrYVsZND/1hP9Emg0 magicuser@LPTSEPT12
The key's randomart image is:
+---[RSA 3072]---+
|==.Bo. ...
|+ *o= o..E .
|.o.Bo+ ...*
|+++=oo 0 = 0
|+.=    oS o
|.+    .o.o .
|.    ..0
|     =..
|     ..
+---[SHA256]---+
```

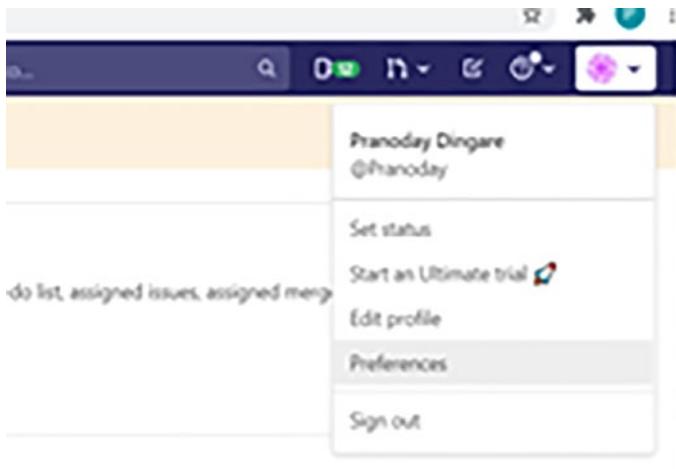
**Figure 13-4.** The public-private key pair was successfully generated

Figure 13-4 shows that the key pair was successfully created inside D:\SSHKeys. The first file contains the private key and the second file contains the public key.

## Step 2: Adding the Public Key to the JenkinsBookCalculatorAPI GitLab Repository

The developer generated this key pair to send a public key to the GitLab repository manager so that they can add it in GitLab. I am adding this public key in GitLab where the JenkinsBookCalculatorAPI repository is.

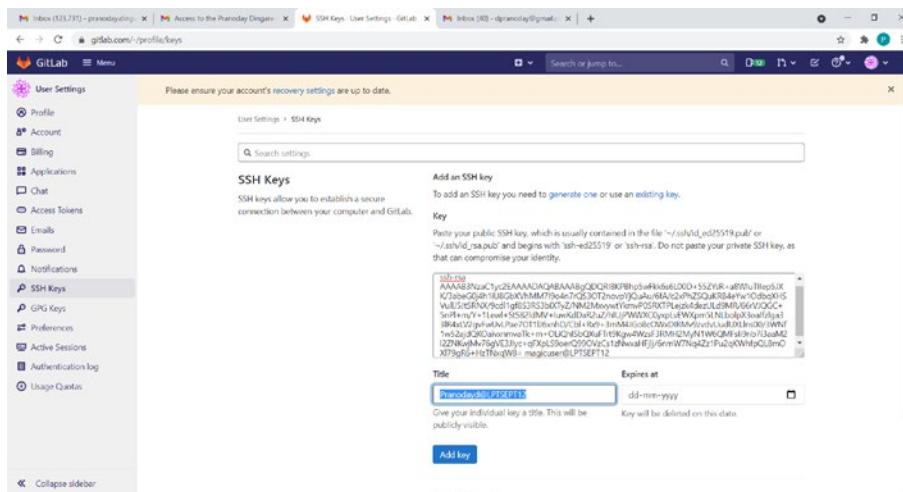
To add the SSH key, click the User Avtar seen on the top-right side of the page. Click the Preferences menu option shown in Figure 13-5.



**Figure 13-5.** The Preferences menu

This will open the Preferences page. Click the SSH Keys menu option available on the left side of the page. This will open the SSH Keys page. Copy the contents of the MyGitLabKeys.pub file and paste it into the Key field.

You can use the title and this key entry will be stored in the Title field. I used Pranodayd@LPTSEPT12 in this field. Figure 13-6 shows all the details filled in.



**Figure 13-6.** The details filled in the Key and Title fields

Click the Add Key button. This will add the key to GitLab successfully.

## Accessing the JenkinsBookCalculatorAPI Repository Using SSH URL and Adding new Arithmetic Functions to the CalculatorAPI

Now the new developer member who has been invited to contribute to the JenkinsBookCalculatorAPI is all set to implement new arithmetic functions in the CalculatorAPI project with their SSH key added to the repository.

## Step 1: Cloning the JenkinsBookCalculatorAPI Repository

This new developer is contributing for the first time to this repository so they have to get the whole remote repository on their machine through the *clone* operation.

A clone operation downloads the whole repository (commit history etc.) along with the contents like source code files etc. on a machine. But before you can clone a repository, you have to do the following settings.

Do you remember that while generating an SSH key pair using the `ssh-keygen` command, that you added a file path to generate a key pair at a different location? If the private key is present in the default location i.e. `$(user.name)\.ssh`, you can run the `git clone` command directly.

The private key is in `D:\SSHKeys\MyGitlabKeys`. You need to perform any of these settings:

- **Option 1:** Register the private key file against a domain for which you want to use it in the `known_hosts` file in the `$(user.name)\.ssh` folder, so that when you try to connect to the `Gitlab.com`, the `ssh` client will send a private key from this `known_hosts` file and access would be granted.
- Run the following command to register the PRIVATE KEY in `known_hosts`:

```
ssh -I D:\SSHKeys\MyGitlabKeys git@gitlab.com
```

This will ask for confirmation, Are you sure you want to continue connecting (yes/no/[fingerprint])?

Type yes and press Enter.

- This will ask for a passphrase. Enter yours.

## CHAPTER 13 CREATING AN AUTO-TRIGGER FREE-STYL

```
C:\Users\magicuser>ssh -i D:\SSHKey\MyGitlabKeys git@gitlab.com
The authenticity of host 'gitlab.com (2606:4700:90:0:f22e:fbec:5bed:a9b9)' can't be established.
ECDSA key fingerprint is SHA256:HbW3g8zUjNSksFbqTiUNPwg2Bqlx8xdGUr1xFzsnUw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'gitlab.com,2606:4700:90:0:f22e:fbec:5bed:a9b9' (ECDSA) to the list of known hosts.
Enter passphrase for key 'D:\SSHKey\MyGitlabKeys':
PTY allocation request failed on channel 0
Welcome to GitLab, @Pranoday!
Connection to gitlab.com closed.
```

**Figure 13-7.** The GitLab has successfully authenticated the user based on the private key

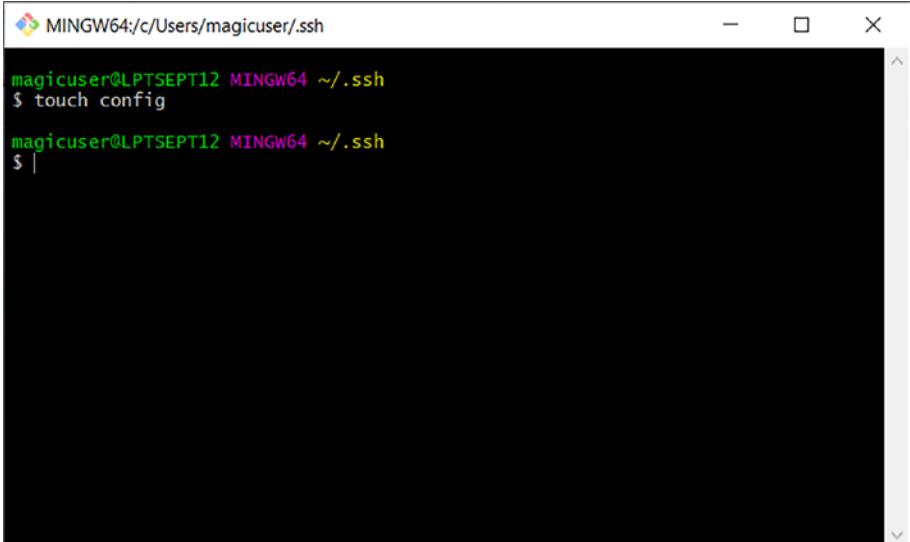
Figure 13-7 shows that the private key was registered successfully.

- Go to the \$(user.name)\.ssh folder. You will see a file named `known_hosts` there.

Open this file to see that the private key from the file is registered against the `git@gitlab.com` domain.

Now you can clone the repository.

- **Option 2:** You have to create a config file in the \$(user.name)\.ssh folder and add the private key file entry to it. Let's look at this process.
- Go to the .ssh folder. On my machine, I have the .ssh folder in `C:\Users\magicuser\.ssh`.
- Right-click and select Git Bash Here to open Git bash.
- Run the `touch config` command (see Figure 13-8), which will create a config file in the .ssh folder.

A screenshot of a terminal window titled "MINGW64:/c/Users/magicuser/.ssh". The window shows a command-line interface with the following text:

```
magicuser@LPTSEPT12 MINGW64 ~/ssh
$ touch config
magicuser@LPTSEPT12 MINGW64 ~/ssh
$ |
```

The terminal has a standard Windows-style title bar with minimize, maximize, and close buttons.

**Figure 13-8.** The touch config command executed in git bash

You can see that the config file is created in C:\Users\magicuser\.ssh.

- Open the file in the notepad editor and enter the following details in it. Save the file:

```
# GITLAB
Host gitlab.com
    HostName gitlab.com
    PreferredAuthentications publickey
    IdentityFile D:\\SSHKey\\MyGitlabKeys
```

- **Option 3:** Open the command prompt and run the following command, which will set the private key file in the git global config file:

```
git config --global core.sshCommand "ssh -i
D:\\SSHKey\\MyGitlabKeys -F /dev/null"
```

- Now get the SSH URL of the repository.
- Click the Clone button available on the repository page on GitLab.
- **Click the Copy URL button available in the Clone with SSH section.**

I created a new blank folder named NewDeveloper on D: to clone the repository and start coding by opening a project.

- Open the command prompt and make D:\NewDeveloper a working directory in the command prompt using the cd command.
- Run the `git clone git@gitlab.com:Pranoday/jenkinsbookcalculatorapi.git` command.

This will ask for confirmation, as the SSH client will not consider Gitlab.com as a known host.

- Type yes and press Enter. Gitlab.com will now be added to the `known_hosts` file in the `(\$user.name)\.ssh` folder.

---

**Note** If you chose Option 1, then while cloning a repository, it will not ask for confirmation, as Gitlab.com must have added in `known_hosts` while applying settings mentioned in Option 1.

---

- Enter the passphrase that you entered while creating the SSH key pair.

The project has been successfully cloned.

## Step 2: Adding a Multiplication Function to the Calculator Class

Let's open the cloned project in Eclipse and add a Multiplication function to the `Calculator.java` file.

I created a blank workspace and now I am importing a cloned Maven project into it. Select the `File > Import` menu option in Eclipse.

Select the `Existing Maven Projects` option under the `Maven` section and click the `Next` button. Add the path of the cloned project to the `Root Directory` field and click the `Finish` button. This will import the project into Eclipse.

Let's open the `Calculator.java` file and add the following code:

```
public int Multiplication(int num1,int num2)
{
    int Res=num1*num2;
    return Res;
}
```

## Step 3: Adding Unit Test Cases for the Multiplication Function

I added a few unit test cases (see Listing 13-1) for the Multiplication function to the `TestMultiplicationFunctionality.java` file under `Pranodayd.CalculatorAPI` package in `src/test/java`.

***Listing 13-1.*** Unit Test Cases Written in `TestMultiplicationFunctionality.java`

```
package Pranodayd.CalculatorAPI;
import java.io.IOException;
import java.nio.file.Files;
```

## CHAPTER 13 CREATING AN AUTO-TRIGGER FREE-STYLED JOB TO MANAGE JAVA API RELEASES

```
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;

import org.testng.Assert;
import org.testng.annotations.*;
public class TestMultiplicationFunctionality
{
    Calculator Cal;
    int Result;
    @BeforeClass
    public void Init()
    {
        Cal=new Calculator();
    }

    @BeforeMethod
    public void ReinitialisingResult()
    {
        Result=0;
    }
    @Test(priority=1,dataProvider="ProvidePositiveInteger
Values",groups= {"RegressionTest"})
    public void TestMultiplicationWithPositiveValues
    (int Number1,int Number2,int expectedResult)
    {
        Result=Cal.Multiplication(Number1, Number2);
        Assert.assertEquals(Result, expectedResult,
        "Multiplication does not work with positive
numbers");
    }
}
```

```
@DataProvider
public Object[][] ProvidePositiveIntegerValues()
{
    /*We want to test functionality with 3 SETs
     *
     * SET1 :1,2,2
     * SET2 :10,20,200
     * SET3 :1000,2000,2000000
     * SET4 :100,200,20000
     */

    Object [][] SetOfValues=new Object[4][3];
    //This is SET 1: 1,2,2

    SetOfValues[0][0]=1;
    SetOfValues[0][1]=2;
    SetOfValues[0][2]=2;

    //This is SET 2: 10,20,200
    SetOfValues[1][0]=10;
    SetOfValues[1][1]=20;
    SetOfValues[1][2]=200;

    //This is SET 3: 1000,2000,2000000
    SetOfValues[2][0]=1000;
    SetOfValues[2][1]=2000;
    SetOfValues[2][2]=2000000;

    //This is SET 4: 1000,2000,2000000
    SetOfValues[3][0]=100;
    SetOfValues[3][1]=200;
    SetOfValues[3][2]=20000;

    return SetOfValues;
}

}
```

## Step 4: Changing the Version Number to 3.0 in pom.xml

Say you are planning to release Version 3.0 of the CalcualtorAPI.jar with this new Multiplication functionality. You need to go to pom.xml and change the version from 2.0 to 3.0, as shown in Listing 13-2.

***Listing 13-2.*** Shows the Version Changed to 3.0 in pom.xml

```
<groupId>Pranodayd</groupId>
<artifactId>CalculatorAPI</artifactId>
<version>3.0</version>
```

## Step 5: Unit Testing the Recent Multiplication Function and Regression Testing for the New Functionalities

Let's run the mvn test command and execute all the unit test cases. This will confirm the newly added Multiplication function is working and has not broken the previously implemented Addition and Subtraction functions.

Open the command prompt and change the working directory to the project directory using the cd command. Then run the mvn test command.

Verify that all the test cases passed.

## Step 6: Committing New Functionality Changes in a Branch Named Multiplication Function in the Local Repository

As you have confirmed that the newly developed function is working along with the old functionalities of the API, you can commit the changes to the new branch.

Let's create a branch named Multiplication Function by running the following command:

```
Git checkout -b "MultiplicationFunction"
```

Add the changes to the staging area with the following command, which will add all the changed files from the current directory as well as any subdirectories to the current directory in the Staging area:

```
git add.
```

Commit the changes to the newly created branch with this command:

```
git commit --m "Multiplication function is added in  
Calculator API"
```

## Step 7: Pushing the MultiplicationFunction Branch to the Remote Repository

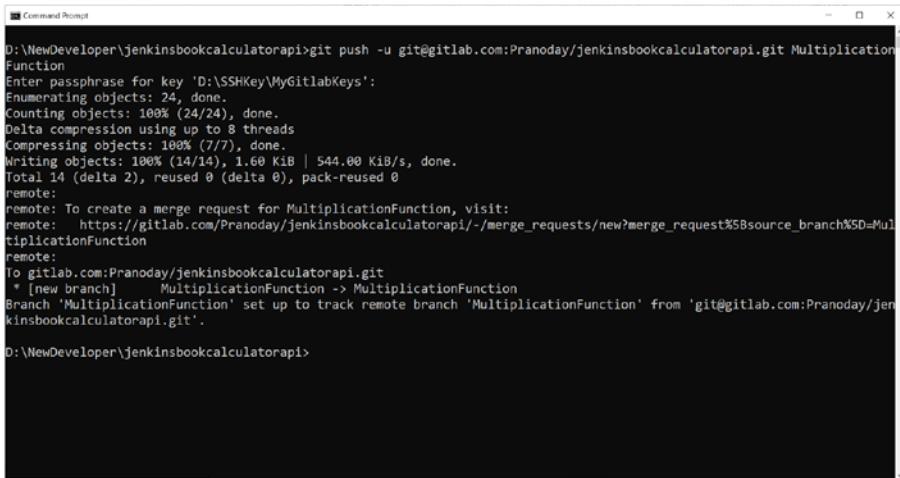
Push this branch to the remote repository using this command:

```
git push -u git@gitlab.com:Pranoday/jenkinsbookcalculatorapi.  
git MultiplicationFunction
```

Note that you are using SSH URL while pushing this branch as well.

Enter the passphrase for the SSH key.

This branch will successfully be pushed to the remote repository (see Figure 13-9).



```
D:\NewDeveloper\jenkinsbookcalculatorapi>git push -u git@gitlab.com:Pranoday/jenkinsbookcalculatorapi.git MultiplicationFunction
Enter passphrase for key 'D:\SSHKey\MyGitlabKeys':
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (14/14), 1.60 KiB | 544.00 KiB/s, done.
Total 14 (delta 2), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for MultiplicationFunction, visit:
remote:   https://gitlab.com/Pranoday/jenkinsbookcalculatorapi/-/merge_requests/new?merge_request%5Bsource_branch%5D=MultiplicationFunction
remote:
To gitlab.com:Pranoday/jenkinsbookcalculatorapi.git
 * [new branch]      MultiplicationFunction -> MultiplicationFunction
Branch 'MultiplicationFunction' set up to track remote branch 'MultiplicationFunction' from 'git@gitlab.com:Pranoday/jenkinsbookcalculatorapi.git'.

D:\NewDeveloper\jenkinsbookcalculatorapi>
```

**Figure 13-9.** The code is successfully pushed to the remote repository

Now refresh the GitLab repository page. You will see that the new branch called MultiplicationFunction has been pushed and is ready to be merged.

## Step 8: Creating a Merge Request for this New Branch

Now create a merge request by clicking the Create Merge Request button. This will take you to the New Merge Request page.

Add details to the Title and Description fields and click the Create Merge Request button.

Now you have to simply click the Merge button to merge changes in the master branch. But you have to do this after creating the Jenkins job because you need to trigger a Jenkins job as soon as you merge the code in the master branch and create a CalculatorAPI.jar file's new version and deploy it on the Nexus repository.

# Creating an Auto-Trigger Jenkins Job with Email Notification

Let's create a free-style job that you are going to trigger automatically to release a new version to the Nexus repository.

## Step 1: Creating a Free-Style Job from Jenkins Dashboard: Click the New Item link on Jenkins Dashboard

Enter a job name in the Enter an Item Name field and select the Free-style project option. Click the OK button.

I named my job ReleaseCalculatorAPIAutoTrigger.

Click the OK button.

Select the Git radio button in the Source Code Management section and enter the Git code repository SSH URL in the Repository URL field.

Now create the SSH credentials entry. Click the Add button in the Credentials field.

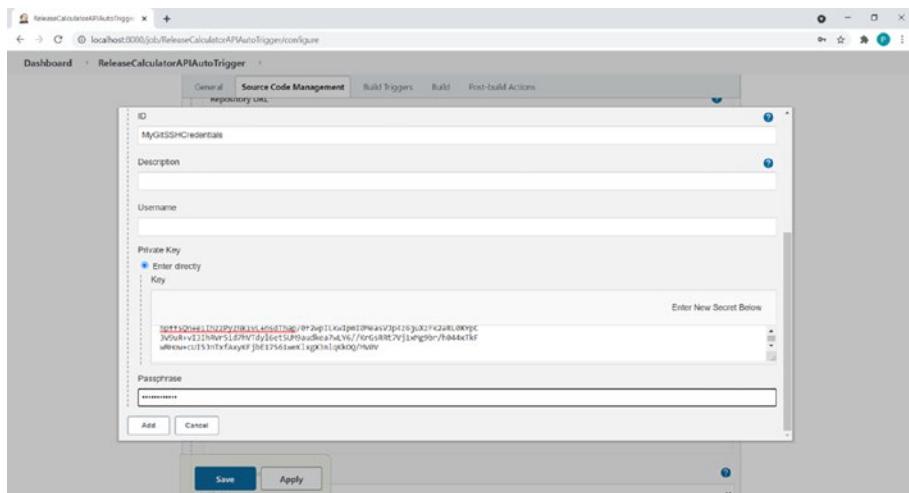
Click the Jenkins option. This will open the Jenkins Credentials Provider: Jenkins page.

Keep the Global Credentials (Unrestricted) option selected in the Domain field. Select the SSH Username with Private Key option in the Kind field dropdown. Enter a value in the ID field. I entered MyGitSSHCredentials.

Select the Enter Directly radio button in the Private Key section. Click the Add button in the Private Key section. Copy the private key from the private key file. Paste the private key in the field. Enter a passphrase of the key in the Passphrase field.

All the required details are shown in Figure 13-10. Click the Add button.

## CHAPTER 13 CREATING AN AUTO-TRIGGER FREE-STYL



**Figure 13-10.** All mentioned details filled in while creating Credentials entry

Let's select the created SSH credentials entry in the Credentials field dropdown to resolve the "Permission denied" error shown in the Repository URL field. This error is because you have not mentioned the SSH private key to authenticate access.

Click the Apply button. You can see that the error is gone now.

Enter the branch name "**Master**" in the **Branch Specifier** field, as you are using **master** as the main Git repository branch.

Scroll down the page to the **Build Triggers** section. Select the **Poll SCM** checkbox.

Choose the polling schedule "After every 5 minutes, every hour, every Day Of Month, every Month, every Day Of Week".

This setting will make Jenkins check the repository's master branch every five minutes for changes. When Jenkins detects changes, it will trigger a build.

Scroll down the page to add a build step. Click the Add Build Step dropdown and select the Invoke Top-Level Maven Targets option.

Select the MyMaven option in the Maven Version dropdown. Enter deploy into the Goals field. See Figure 13-11.



**Figure 13-11.** The Maven Version and Goals option in the Build section

Let's configure Email Notification in the post-build step. Click the Add Post-Build Action button. Select the Email Notification option.

Enter your email ID where you want to receive an email notification. Keep the Send Email for Every Unstable Build checkbox checked to receive email notifications if the build fails.

Click the Save button.

## Setting Up Jenkins to Send Email Notifications

You created a Jenkins job with email notification, but before you execute that job and see the notification, you need to adjust a few settings in Jenkins.

Choose the Manage Jenkins menu on the Jenkins dashboard, which will open the Manage Jenkins page. Click the Configure System option.

This will take you to the System Configuration page. Scroll down to the Jenkins Location section.

Enter an email address in the System Admin Email Address field. Once you receive email notification from Jenkins, you will see this email address in the From column in your inbox. I entered pranoday.dingare@gmail.com in this field.

Scroll down to the Email Notification section.

Configure the SMTP Server first. The SMTP server provides a SMTP service required to send emails. If your organization has an SMTP server setup, you can add its IP address to the SMTP Server field.

I use Gmail's SMTP server, so I entered smtp.gmail.com into the SMTP Server field.

Click the Advanced button, which will open more settings, as shown in Figure 13-12.

The screenshot shows the 'E-mail Notification' configuration page. It includes the following fields:

- SMTP server: smtp.gmail.com
- Default user e-mail suffix: (empty)
- Use SMTP Authentication:
- Use SSL:
- Use TLS:
- SMTP Port: (empty)
- Reply-To Address: (empty)
- Charset: UTF-8

**Figure 13-12.** More settings related to the SMTP server

Select the Use SMTP Authentication checkbox and enter a valid Gmail account username in the User Name field and a Gmail account password in the Password field. By using this UserName and Password, Gmail's SMTP server will authenticate access.

---

**Note** If you added the IP address of an SMTP server from your organization's domain into the SMTP Server field, you have to enter the user credentials into these User Name and Password fields.

---

I provided my Gmail account's username and password. Check the Use SSL checkbox and enter 465 into the SMTP Port field.

Let's test this configuration. Click the Test Configuration by Sending Test Email checkbox. Enter a valid email and click the Test Configuration button.

I get an error, `javax.mail.AuthenticationFailedException: UserName and Password not accepted`. If you get this error, you have to allow less secure apps to send you an email.

I am using [dpranoday@gmail.com](mailto:dpranoday@gmail.com) to send an email, so let's go to Gmail and click the Manage Your Google Account option.

Click the Security option available on the left side of the page. This will open the Security page.

Scroll down to the Less Secure App Access section.

Click the Turn On Access (Not Recommended) link from this section, which will open the Less Secure App Access page. Click the Allow Less Secure Apps:OFF radio button to turn it on.

Now go back to Jenkins and test the configuration again by clicking the Test Configuration button. The email was sent successfully.

## Triggering the New Jenkins Job

You have created the Jenkins job that will poll the master every five minutes and trigger a build as soon as it detects a change.

Now let's go to GitLab and merge the `MultiplicationFunction` branch with the master branch.

Let's go to the GitLab repository page. Click the Merge Requests menu shown on the left side. This will take you to the Merge Requests page.

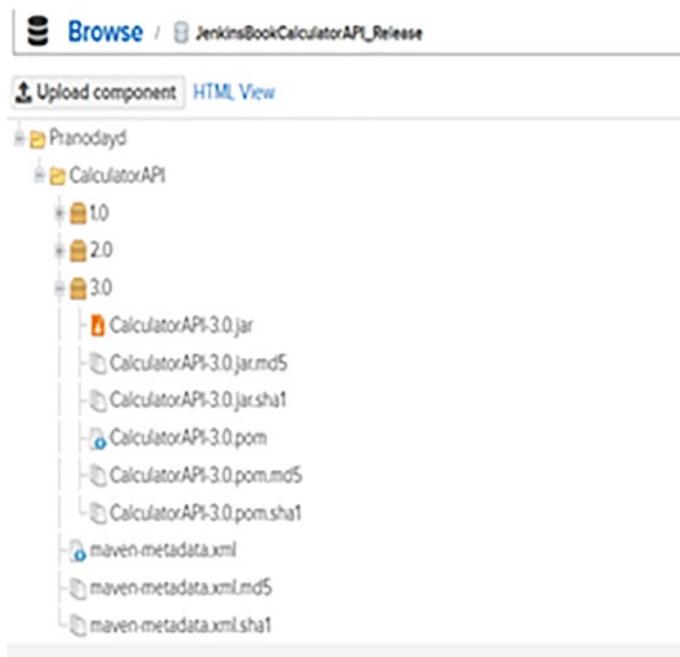
Click the [Multiplication Function is Added in Calculator API](#) link. Click the Merge button to merge the changes in the MultiplicationFunction branch with the master branch.

Now let's go to the Jenkins dashboard. Wait until Jenkins polls the repository at the next polling schedule time. The build is shown in the Build Queue. Wait until the build starts executing. The entry will be shown in the Build Executor status section.

Now go to the console output of this build. Observe at the top of the console log that a job has been triggered due to an SCM change.

Scroll down to see that `CalculatorAPI3.0.jar` was successfully deployed on the Nexus repository.

Let's go to the Nexus repository and confirm the release (see Figure 13-13).



**Figure 13-13.** The 3.0 version of *CalulatorAPI.jar* has been successfully deployed to Nexus

## Understanding the Execution of Jobs with SCM Set

When a build triggers for a job with SCM set, the workspace directory is created in the `$(Jenkins.home)` location, the directory with the job name is created inside this workspace directory, and the configured branch is checked out in this directory.

On my machine, the repository master branch is checked out in this location:

```
C:\Users\magicuser\.jenkins\workspace\Release  
CalculatorAPIAutoTrigger
```

This directory is considered the *working directory*. In this case, the mvn deploy build step is executed from this directory and all Maven build phases are executed and all JAR files are deployed on the Nexus repository.

## Failing the Build to See the Email Notification

You have configured the job to send an email notification on failure of a build by setting the Send Email for Every Unstable Build option in the Post-Build Actions section.

Now I am deliberately changing the expected result of one of the unit test cases in TestAdditionFunctionality.java file to make it fail. I am testing the Addition() function by sending two values as arguments and comparing the result, which is going to fail. See Figure 13-14.

```
@Test(priority=1)
Run | Debug
public void TestAdditionWithPositiveNumbers()
{
    System.out.println("I am in 1 st TestCase");
    Result=Obj.Addition(10,20);
    Assert.assertEquals(Result, 300,"Addition does not work with positive numbers");
}
```

**Figure 13-14.** The code of the TestAdditionWithPositiveNumbers function in the TestAdditionFunctionality.java file

I committed the change in a branch in the local repository and pushed this branch to merge with the remote repository. Then I subsequently merged this branch with the master branch to trigger the build. Triggered Build #6 failed due to the failure of the test case.

You should have seen an email notification about this failure of Build #6. Click the email to see the console output log of the build that failed.

## Summary

This chapter explained how to add a new code contributor to the GitLab repository. You also learned what SSH is, how it works, and how to create public and private key pair and configure it in the GitLab repository. Then you saw how to use the private key to clone the repository/push the code to the remote repository. Then you created a credentials entry in Jenkins with the SSH private key and configured an auto-trigger Jenkins job. You also learned how email notification can be configured and set the Jenkins job to send an email on build failure. You learned how to automate a build process using a Jenkins free-style job. The next chapter explains the Jenkins pipeline and how you can achieve more control over your build processes using it.

## CHAPTER 14

# Understanding the Jenkins Pipeline

In the previous chapter, you learned how to create a Jenkins job that will be triggered automatically. You automated a process of the Java Calculator API release using a free-style job. This job starts by pulling the latest code from the master branch and performs different build lifecycle phases on it, like compiling, unit testing, packaging, and deploying to the artifact repository. This chapter explains an important Jenkins concept—the pipeline.

## What Is a Jenkins Pipeline?

A Jenkins *pipeline* is a script suite of different Jenkins plugins that supports implementation and integration of continuous delivery pipelines into Jenkins.

A continuous delivery pipeline is an automated set of expressions written in the form of code statements that execute the process of getting your application code from the version control system through to the customers and users. Every change you carry out in your application goes through a complex build process on its way to being released.

When you write code, you need an Application Programming Interface (API), which provides a set of functions that you call in your code. For example, say you want to write code to read data from an Excel file using Java. You need a set of functions to interact with the Excel application from your Java code.

In this same manner, to interact with Jenkins plugins from your code, you need a scripting reference. Using these scripting references, you can instruct the Jenkins plugin to perform a task as part of the end-end build process written in the form of code, a Jenkins pipeline.

## Why Use a Jenkins Pipeline?

The following are the advantages of automating a CI/CD process using Jenkins pipelines over Jenkins jobs:

- **Code:** Pipeline allows you to write CI/CD processes of your application in the form of code, which you can check into your source code repository along with the application's source code. If you have build process code in the central repository, it allows you to share it with the rest of the team members more effectively. Say a developer implementing a change in the application code requires a change in a build process. That same developer can implement that change in the build process if it is written and shared as a pipeline code.
- **Durable:** Pipelines can survive both planned and unplanned restarts of the Jenkins controller.
- **Pausable:** Pipeline execution can optionally stop and wait for human input or approval before it continues its execution.

- **Versatile:** Pipeline supports complex CI/CD requirements like forking, looping, and parallelism.
- **Extensible:** Pipeline plugin supports multiple options for integration with other plugins.

## Understanding Different Pipeline Concepts

- *Pipeline:* A *pipeline* is code that defines your entire build process, which includes different stages of building an application, testing, and delivering.
- *Node:* A *node* is a machine capable of executing a pipeline.
- *Stage:* There are different stages that software goes through as part of the build process, such as build, test, deploy, etc. A stage block in a pipeline defines different tasks to be performed as part of a particular stage. For example, the Build stage block includes tasks like compiling the source code, packaging a library, and so on.
- *Step:* A step is a single task that a pipeline performs as part of a particular stage. A stage block is nothing but a collection of multiple steps. For example, a step could execute a batch command, execute a particular Maven goal, and so on.

## Pipeline Syntax Overview

Jenkins supports two types of pipelines—declarative pipelines and scripted pipelines. Let's discuss these two types one by one.

## Fundamentals of Declarative Pipelines

Declarative pipelines contain a pipeline block that defines the entire build process.

```
pipeline
{
    agent any          1
    stages
    {
        stage('Build')      2
        {
            steps
            {
                //          3
            }
        }

        stage('Test')       4
        {
            steps
            {
                //          5
            }
        }
        stage('Deploy')     6
        {
            steps
            {
                //          7
            }
        }
    }
}
```

```
 } //Close of Stages  
 } //Close of Pipeline
```

I numbered important statements in the pipeline code and use these numbers to explain these statements:

- 1:** This statement instructs Jenkins to execute this pipeline on any available agent, i.e. machine.
- 2:** This block defines a build stage.
- 3:** This defines a particular step (i.e., task) related to the build stage.
- 4:** This defines the test stage.
- 5:** This defines a particular step (i.e., task) related to the test stage.
- 6:** This defines the deploy stage.
- 7:** This defines a particular step (i.e., task) related to the deploy stage.

## Fundamentals of Scripted Pipelines

In a scripted pipeline, the entire build process is defined inside a *node* block. Although it is not mandatory to enclose your code inside a node block, it does following things if code is enclosed within it:

- 1:** Schedules the steps contained in a node block to run by adding an item to the Jenkins queue. As soon as the executor is free on the node, the steps will execute.
- 2:** Creates a workspace directory in which files from Source Control Management (SCM) are checked out and worked on.

```
node  
{  
    stage('Build')  
    {  
        //  
    }  
    stage('Test')  
    {  
        //  
    }  
    stage('Deploy')  
    {  
        //  
    }  
}
```

I have numbered important statements in the pipeline code and use these numbers to explain these statements:

- 1:** This statement instructs Jenkins to execute this pipeline on any available agent, i.e. machine.
- 2:** This block defines a build stage. Stage blocks are optional but implementing them in a scripted pipeline provides clearer visualization of each stage's tasks/steps in the Jenkins UI.
- 3:** This defines a particular step (i.e., task) related to the build stage.
- 4:** This defines the test stage.
- 5:** This defines a particular step (i.e., task) related to the test stage.

**6:** This defines the deploy stage.

**7:** This defines a particular step (i.e., task) related to the deploy stage.

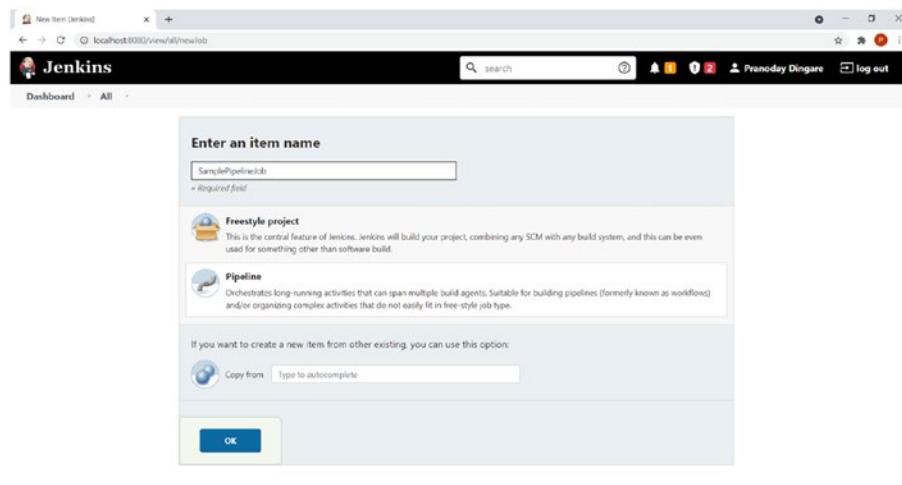
## Defining a Simple Pipeline in Jenkins UI

This section explains how to create a simple pipeline in the Jenkins UI.

1. Log into Jenkins.
2. Install the Pipeline: Job and Pipeline plugins: Refer to Chapter 5 for detailed steps to install this plugin.
3. Go to the Jenkins dashboard.
4. Create a pipeline job. Click the New Item link provided on the left side of the Jenkins dashboard.

This will open the page shown in Figure 14-1.

5. Enter the job name and select the Pipeline option, as shown in Figure 14-1.



**Figure 14-1.** The name for the Pipeline job

6. Click the OK button, which will take you to the Job Configuration page.

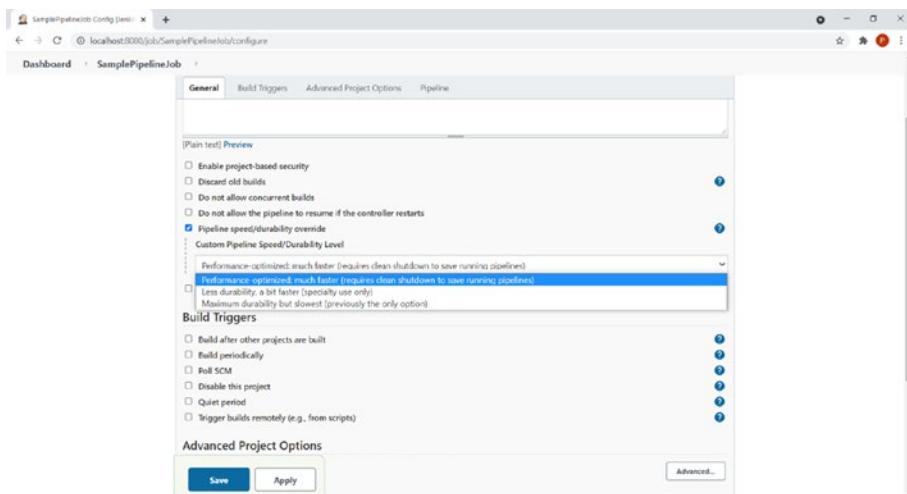
Consider the following settings specific to the Pipeline job:

Do not allow the pipeline to resume if the controller restarts:

If you check this checkbox, it does not resume the execution of the pipeline after the Jenkins controller restarts.

## Pipeline Speed/Durability Override

By default, a running pipeline job writes a lot of data on disk so that the written data pipeline execution can be resumed once Jenkins restarts after failure. But this slows down the execution of the pipeline. Using this setting, you can change this default behavior. Clicking this dropdown field will reveal the options in Figure 14-2.

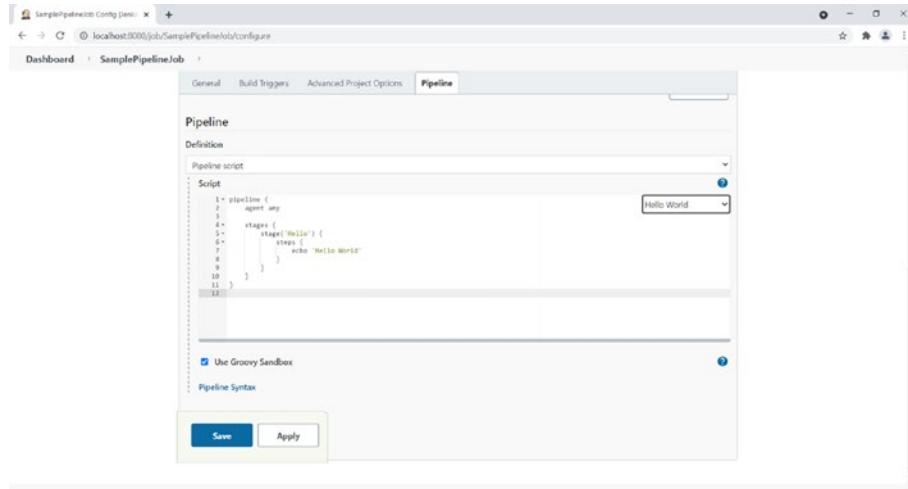


**Figure 14-2.** The options in the Custom Pipeline Speed/Durability Level dropdown

- **Performance-optimized: Much faster (requires clean shutdown to save running pipelines):** If this option is selected then Jenkins pipelines write only the required data. This option allows faster pipeline execution.
- **Less durability, a bit faster (specialty use only):** If this option is selected then Jenkins pipeline does not write much data hence it suffers from less durability.
- **Maximum durability but slowest:** This was the only available option previously. If you select this option, it writes the maximum possible data and provides good durability in case of Jenkins failure as compared to other options. But it makes pipeline execution very slow.

## CHAPTER 14 UNDERSTANDING THE JENKINS PIPELINE

For this example pipeline job, we are not going to select this option. scroll down to the Pipeline section. You have to write pipeline code in the editor. I wrote a simple declarative pipeline in the code editor, as shown in Figure 14-3.



The screenshot shows the Jenkins Pipeline configuration page for a job named "SamplePipelineJob". The "Pipeline" tab is selected. In the "Script" section, there is a Groovy script:

```
1: pipeline {
2:     agent any
3:
4:     stages {
5:         stage('Hello') {
6:             steps {
7:                 echo "Hello World"
8:             }
9:         }
10:    }
11: }
```

Below the script, there is a checkbox labeled "Use Groovy Sandbox" which is checked. At the bottom of the pipeline configuration form are two buttons: "Save" and "Apply".

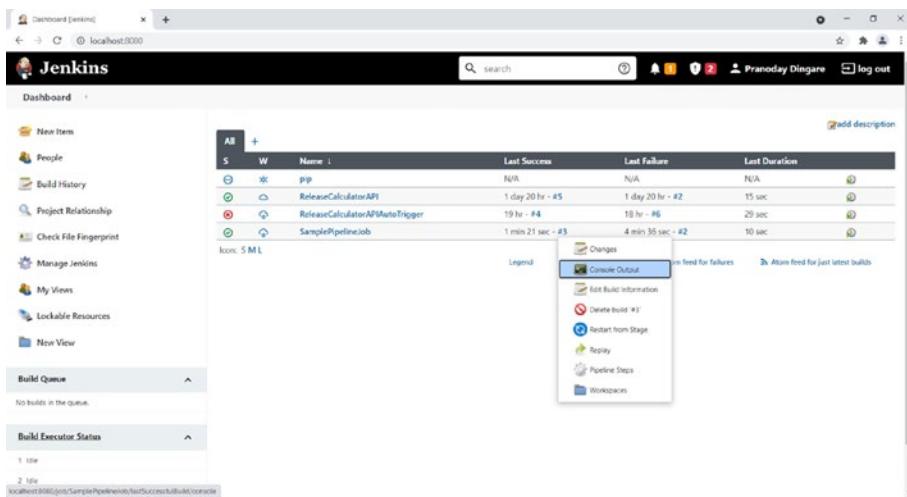
**Figure 14-3.** The a simple declarative pipeline written in the code editor

Click the Save button.

7. Run a pipeline job. Let's go back to the dashboard using the Back to Dashboard link.

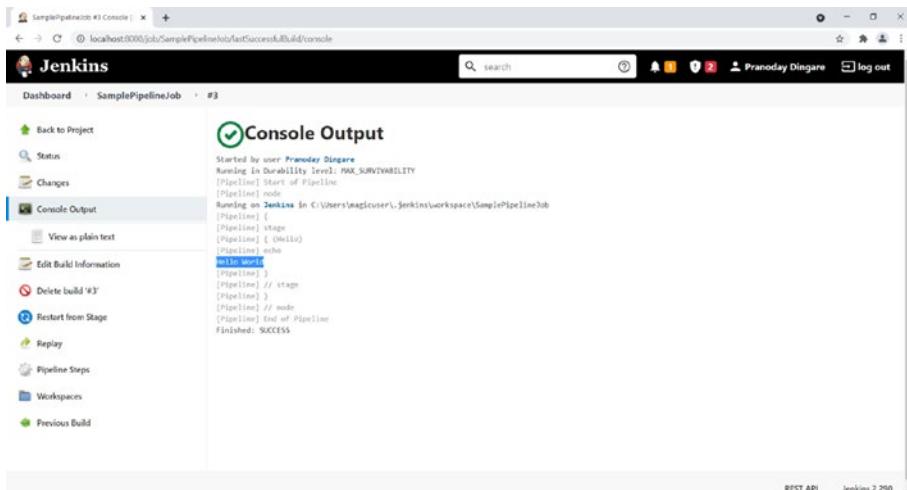
Click the clock sign of the SamplePipelineJob. This will run the pipeline job.

Let's view the console output of this job. Click the Console Output menu to do so (see Figure 14-4).



**Figure 14-4.** The Console Output menu option

This prints "Hello World" on the console (see Figure 14-5).



**Figure 14-5.** The console output after running a pipeline job

# Resolving java.lang. IllegalArgumentException: Unsupported Class File Major Version Error

You will typically encounter the following error after running a pipeline job:

```
java.lang.IllegalArgumentException: Unsupported class file  
major version
```

- **Reason:** You get this error when you use a Java version that's not supported with Jenkins, i.e., Java 1.11 or 1.8.
- **Solution:** Make sure that PATH system environment variable on your machine is pointing to Java version 1.11 or 1.8.

My JAVA\_HOME environment variable is pointing to JDK 11. Don't forget to restart the Jenkins Server after setting the required Java version in Environment Variables.

Otherwise, you can set JDK 11 or JDK 1.8 in the Global Tool Configuration in Jenkins (see Figure 6-10).

## Understanding String Interpolation in the Jenkins Pipeline

String interpolation is the process of replacing variable values with variables present in a string. In the following example, one variable is defined with the name `username` and has the value `Pranoday`. I want to replace this value with a string. This process is called *string interpolation*.

For example:

```
def Username = 'Pranoday'
echo 'Hello Mr. ${Username}'
echo "How are you ? Mr. ${Username}"
```

This will result in the following:

```
Hello Mr. $(Username)
How are you ? Mr. Pranoday
```

If the string is in double quotes, \${VariableName} is the syntax to perform the string interpolation. If string is in single quotes, then \$VariableName is the syntax to perform the string interpolation.

```
echo 'Hello Mr. $Username' would results in
Hello Mr. Pranoday
```

## String Interpolation Example

The Jenkins pipeline exposes different environment variables via global variables, including BUILD\_ID, BUILD\_NUMBER, JENKINS\_URL, JOB\_NAME, etc.

To see a full list of the environment variables that are accessible from the Jenkins pipeline, enter the following URL in your browser:

```
 ${YOUR_JENKINS_URL}/pipeline-syntax/globals#env
```

\${YOUR\_JENKINS\_URL} is the Jenkins hostIPAddress:Port.

My Jenkins starts at localhost:8080 so this URL to see all environment variables is <http://localhost:8080/pipeline-syntax/globals#env>.

Say you want to print the BUILD\_ID and JENKINS\_URL variables to the console using an echo statement in the Jenkins pipeline. This is where you need to use string interpolation.

```
pipeline
```

```
{  
    agent any  
    stages  
    {  
        stage('Example')  
        {  
            steps  
            {  
                echo "Running ${env.BUILD_ID} on  
                ${env.JENKINS_URL}"  
            }  
        }  
    }  
}
```

## Creating a Pipeline Job to Release the Java API

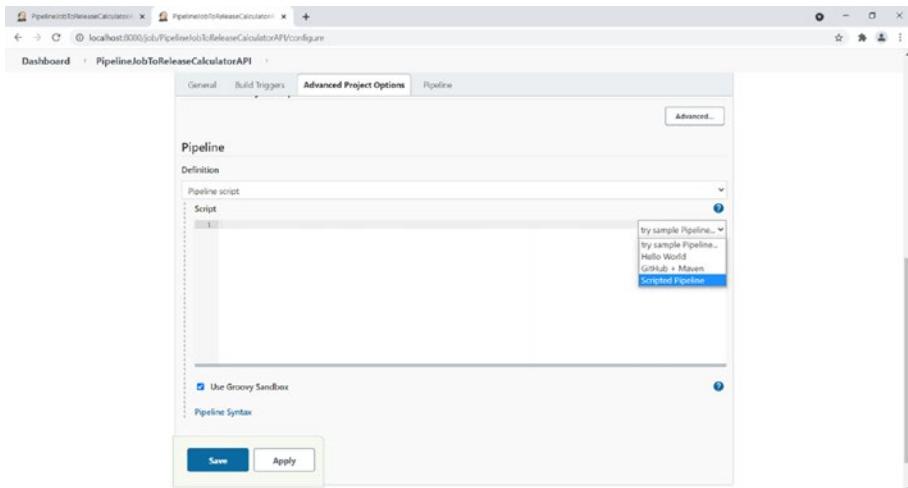
You previously created a free-style job to release the CalculatorAPI JAR in the Nexus Repository. Now you'll see how to create a pipeline job that will perform the same tasks as the free-style job to release a new version of CalculatorAPI.jar to the Nexus repository.

1. Go to the Jenkins dashboard.
2. Create a pipeline job. Click the New Item link.

Name the job and select the Pipeline option.

3. Click the OK button.
4. Scroll down to find the Pipeline section.

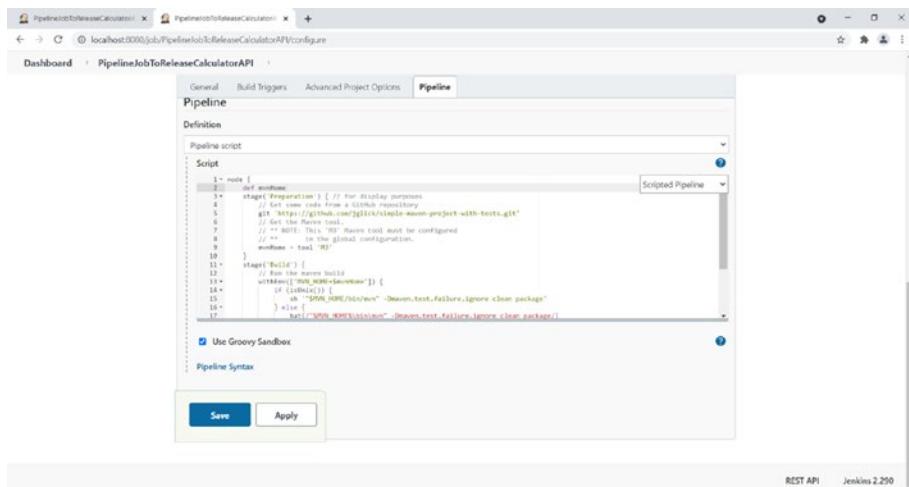
5. Click the Scripted Pipeline option (see Figure 14-6) to generate a basic template of the pipeline script in the editor.



**Figure 14-6.** The options available in the Try Sample Pipeline dropdown

After clicking this option, you will get a template scripted pipeline script, as shown in Figure 14-7.

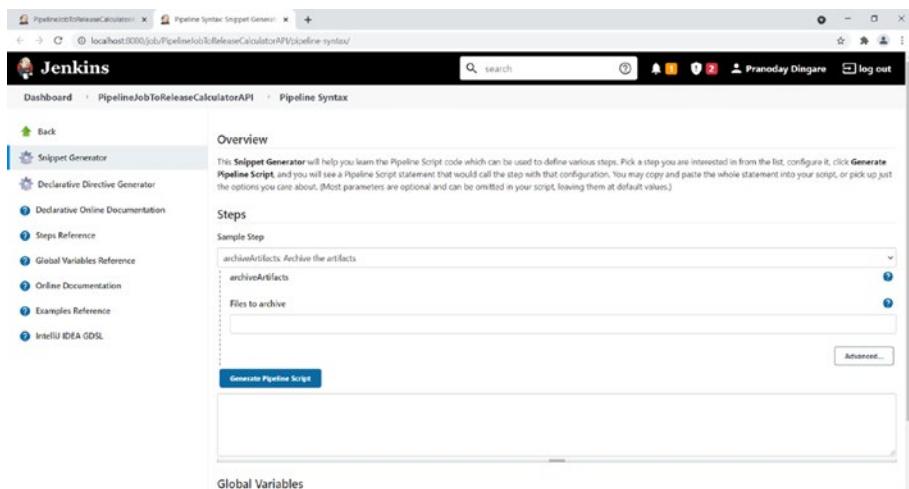
## CHAPTER 14 UNDERSTANDING THE JENKINS PIPELINE



**Figure 14-7.** The sample scripted pipeline code

You are going to delete the code from this template and write your own.

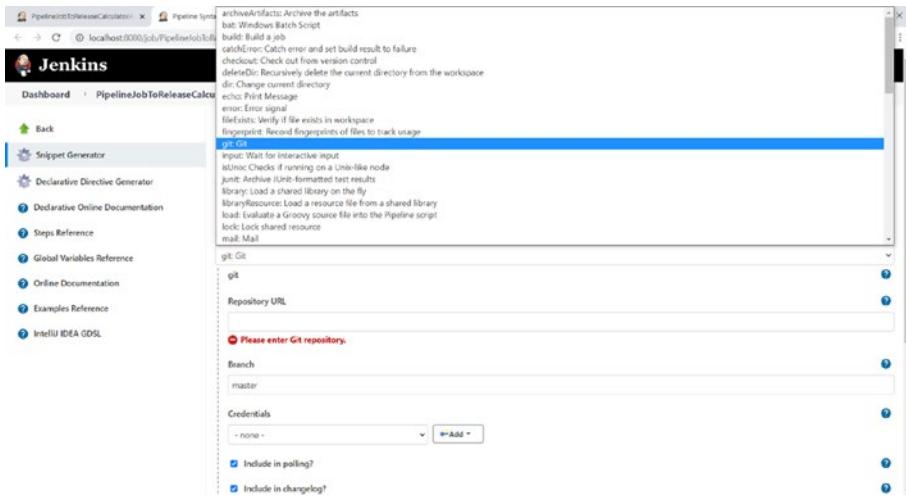
Instead of writing each code step manually, you can use the Pipeline Syntax:Snippet Generator provided by Jenkins. Click the Pipeline Syntax link available below the code editor to open the Pipeline Syntax:Snippet Generator, as shown in Figure 14-8.



**Figure 14-8.** The Snippet Generator

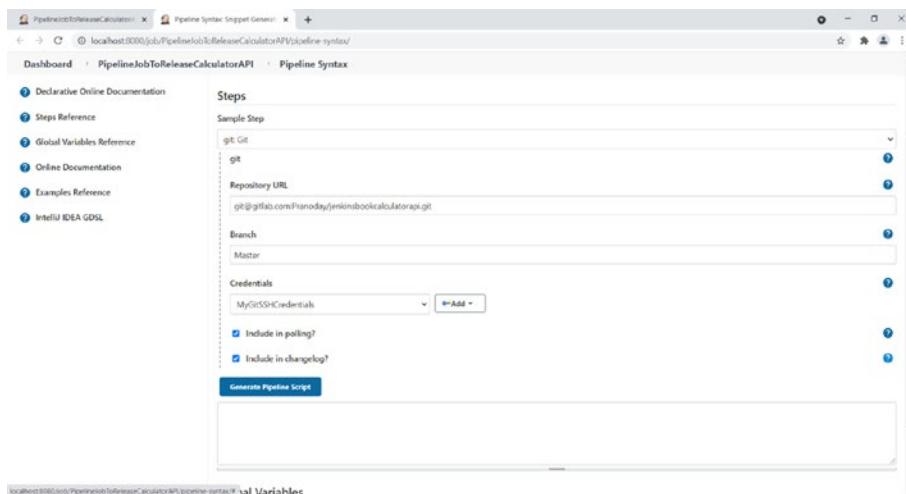
Click the Sample Step dropdown, which will show a list of pipeline steps you can use in the pipeline according to your requirements.

First check out the master branch from the GitLab repository, so select the Git step from the list (see Figure 14-9).



**Figure 14-9.** The git:Git option in Sample Step dropdown and other Git-related fields

After selecting the Git option, you can see fields to enter details of your Git repository, a branch to checkout, the credentials entry with the authentication information, and so on. I added the Git repository SSH URL to the Repository URL field, the master branch to the Branch field, and the SSH credentials entry to the Credentials field (see Figure 14-10).



**Figure 14-10.** All Git repository settings in the Snippet Generator

Click the Generate Pipeline Script button. Clicking this button generates a pipeline script statement based on the details you provided.

Copy the generated line and then go back to the pipeline job and do the changes in the generated template pipeline script.

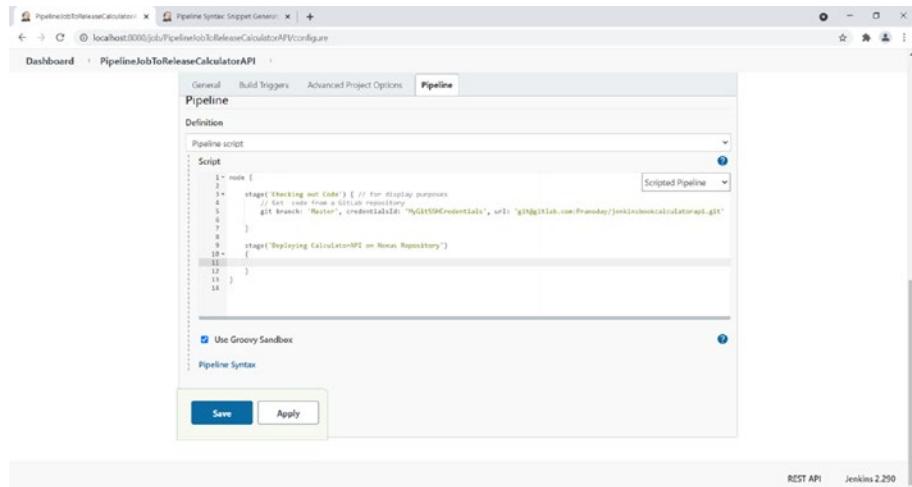
I deleted the unwanted code from this template and added Stage: Checking out Code in pipeline code. I pasted the Git step inside it. I also added a comment describing what this stage does to the pipeline, using double slashes (//). Multiline comments need to be enclosed within /\*\*/.

In this pipeline, the first stage is to pull code from the repository's master branch using the Git step:

```
git branch: 'Master', credentialsId: 'MyGitSSHCredentials',
url: 'git@gitlab.com:Pranoday/jenkinsbookcalculatorapi.git'
```

```
git step has 3 arguments branch: <NameOfBranch>,
credentialsId:<ID of our Credentials entry>, url: <Git
repository URL>
```

Now add the next stage, “Deploying the CalculatorAPI on the Nexus Repository” (see Figure 14-11).



**Figure 14-11.** The Checking Out Code and Deploying CalculatorAPI on Nexus Repository stages added to the code

In this stage, you want batch command to run Maven deploy goal. Go to Pipeline Syntax: Snippet Generator again to generate this step.

Now select bat step from the list. I wrote %MVN\_HOME%\bin\mvn deploy to the Batch Script edit field. Click the Generate Pipeline Script button, which will generate the script step.

Copy this step and paste it inside the newly added stage. I added a few more lines to the pipeline. Look at the code in Listing 14-1.

### ***Listing 14-1.*** Pipeline Code

```

node
{
    def mvnHome
    stage('Checking out Code') { // for display purposes
        // Get code from a GitLab repository

```

## CHAPTER 14 UNDERSTANDING THE JENKINS PIPELINE

```
git branch: 'Master', credentialsId:  
'MyGitSSHCredentials', url: 'git@gitlab.com:Pranoday/  
jenkinsbookcalculatorapi.git'  
  
}  
  
stage('Deploying CalculatorAPI on Nexus Repository')  
{  
    mvnHome = tool 'MyMaven'  
    withEnv(["MVN_HOME=$mvnHome"])  
    {  
        bat '%MVN_HOME%\bin\mvn deploy'  
    }  
}  
}  
}
```

def mvnHome: This statement defines a variable called mvnHome.

stage('Checking out Code'): This Git step from this stage pulls code from the repository.

mvnHome = tool 'MyMaven': This statement sets the value of mvnHome to the PATH of the Maven installation, which you added to the Global Tools Configuration and called it MyMaven (see Figure 14-12).



**Figure 14-12.** The Maven configuration created in the Global Tools and Configuration

So the mvnHome variable is set to D:\MavenInstallation\apache-maven-3.8.1.

A following block sets the value of mvnHome to a system variable MVN\_HOME and the bat step inside the withEnv block uses this environment variable to get to the mvn command inside the bin folder of the Maven installation to run deploy goal.

```
withEnv(["MVN_HOME=$mvnHome"])
{
    bat '%MVN_HOME%\bin\mvn deploy'
```

Click the Save button.

## Running a Pipeline Job and Release the Calculator API

You created a pipeline job. Let's run it now to release a new version of the CalculatorAPI. I am not adding anything new to the code, just correcting the expected result of the TestAdditionWithPositiveNumbers() test case, which I changed to the wrong value in the last chapter to demonstrate job failure. the code after this correction is shown in Listing 14-2.

***Listing 14-2.*** The TestAdditionWithPositiveNumbers() Test Case Method Code with Correct Expected Result Value

```
@Test(priority=1)
public void TestAdditionWithPositiveNumbers()
{
    System.out.println("I am in 1st TestCase");
    Result=Obj.Addition(10,20);
    Assert.assertEquals(Result,30,"Addition does not work
    with positive numbers");
}
```

I also changed the version of the API to 7.0 in pom.xml, as shown in Listing 14-3.

**Listing 14-3.** The Version Changed to 7.0 in pom.xml

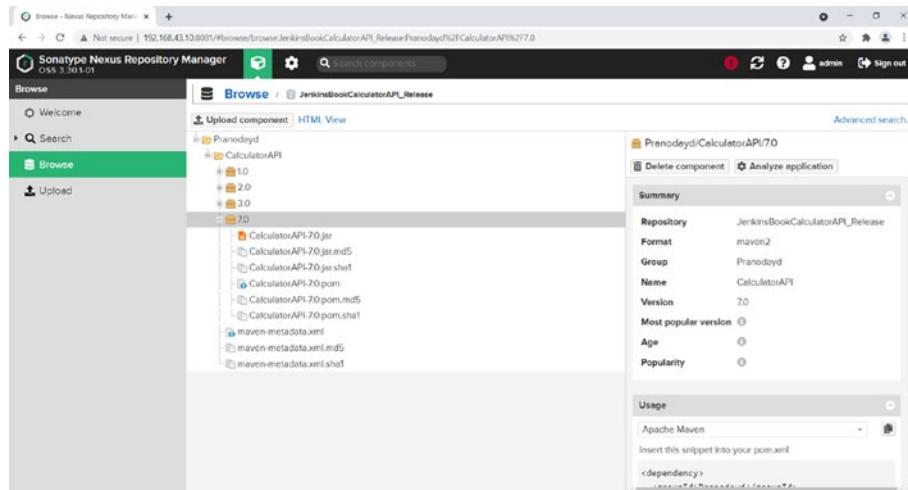
```
<groupId>Pranodayd</groupId>
<artifactId>CalculatorAPI</artifactId>
<version>7.0</version>
```

I committed a change in a new local branch and pushed this branch to the remote repository where I merged this into the master branch.

Now run PipelineJobToReleaseCalculatorAPI manually by clicking the clock sign. The progress of the build is shown in the Build Execution Status section present on the left side of the page.

Let's check the console output. Scroll down to see the successful release of the CalculatorAPI 7.0.jar.

Now go to the Nexus Repository to confirm the successful release (see Figure 14-13). Yes, you got it!



**Figure 14-13.** The CalculatorAPI 7.0.jar released in the Nexus repository

## Summary

This chapter explained what Jenkins pipelines are and their advantages over Jenkins free-style jobs. You also learned what declarative and scripted pipelines are. You learned how to generate the pipeline steps using the Snippet Generator. At the end of the chapter, you created a pipeline to run all the build lifecycle phases on your Java API and released a new version of it in the Nexus repository. After learning how Jenkins can be used to automate the end-end build lifecycle of an API project, you are all set to see how Jenkins can automate a build lifecycle of a web application, which is the topic of the next chapter.

## CHAPTER 15

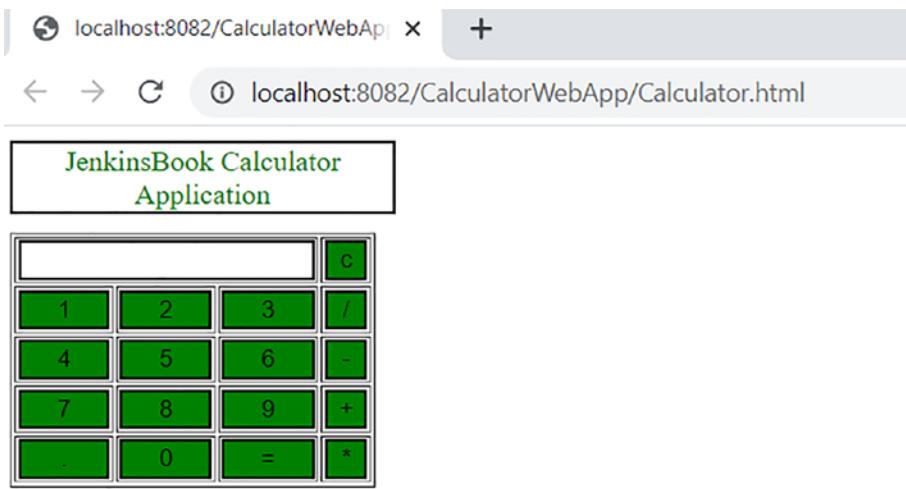
# Creating Jenkins Jobs to Manage a Web Application Project

In the previous chapter, you learned how to use Jenkins jobs to manage the release of an API project developed using Java. You created free-style as well as pipeline jobs, which pull the latest application code from the remote GitLab repository, and then run the different phases of the build lifecycle, like compilation, unit testing using TestNG, packaging tested implementation in the form of .JAR files, and finally deploying the .JAR file to the Nexus repository.

This chapter shows you how to manage the release lifecycle of a simple calculator web application using Jenkins jobs. This web application is developed in the Maven project and consists of a single .HTML file. You are going to implement an end-end build process that will consist of operations such as pulling source code from the repository, deploying the app in an IIS web server, running automated E-E tests developed in Python, and using the UI automation library called Selenium WebDriver.

# Understanding the Calculator Web Application Source Code

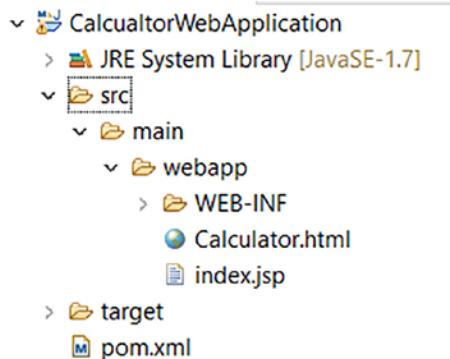
After loading this application in a browser, it looks Figure 15-1.



**Figure 15-1.** The Calculator web application running inside a browser

It performs simple arithmetic operations like addition, subtraction, multiplication, and division. Users can perform any of these operations by interacting with this application using the button controls and the result of the operation will be shown on the Input control

The UI interface of this application is designed using simple .HTML and CSS. The logic of this application is written using JavaScript embedded inside the .HTML file. The project source code directory structure is shown in Figure 15-2.



**Figure 15-2.** The directory structure of the Calculator web application project

The `Calculator.html` file contains the application code, as shown in Listing 15-1.

**Listing 15-1.** `Calculator.html` Source Code File from the Calculator Web Application

```
<html>
<head>
<script>
    //function that display value
    function dis(val)
    {
        document.getElementById("result").value+=val
    }

    //function that evaluates the digit and return result
    function solve()
    {
        let X = document.getElementById("result").value
        let z = eval(X)
        document.getElementById("result").value = z
    }
}
```

```
//function that clear the display
function clr()
{
    document.getElementById("result").value = ""
}
</script>
<!-- for styling --&gt;
&lt;style&gt;
    .title{
        margin-bottom: 10px;
        text-align:center;
        width: 210px;
        color:green;
        border: solid black 2px;
    }

    input[type="button"]
    {
        background-color:green;
        color: black;
        border: solid black 2px;
        width:100%
    }

    input[type="text"]
    {
        background-color:white;
        border: solid black 2px;
        width:100%
    }
&lt;/style&gt;
&lt;/head&gt;</pre>
```

```
<!-- create table -->
<body>
<div class = title >JenkinsBook Calculator Application</div>
<table border="1">
<tr>
<td colspan="3"><input type="text" id="result"/></td>
<!-- clr() function will call clr to clear all value -->
<td><input type="button" value="c" onclick="clr()"/></td>
</tr>
<tr>
<!-- create button and assign value to each button -->
<!-- dis("1") will call function dis to display value -->
<td><input type="button" value="1" onclick="dis('1')"/></td>
<td><input type="button" value="2" onclick="dis('2')"/></td>
<td><input type="button" value="3" onclick="dis('3')"/></td>
<td><input type="button" value="/" onclick="dis('/')"/></td>
</tr>
<tr>
<td><input type="button" value="4" onclick="dis('4')"/></td>
<td><input type="button" value="5" onclick="dis('5')"/></td>
<td><input type="button" value="6" onclick="dis('6')"/></td>
<td><input type="button" value="-" onclick="dis('-')"/></td>
</tr>
<tr>
<td><input type="button" value="7" onclick="dis('7')"/></td>
<td><input type="button" value="8" onclick="dis('8')"/></td>
<td><input type="button" value="9" onclick="dis('9')"/></td>
<td><input type="button" value="+" onclick="dis('+')"/></td>
</tr>
<tr>
<td><input type="button" value"." onclick="dis('.')"/></td>
<td><input type="button" value="0" onclick="dis('0')"/></td>
```

```
<!-- solve function call function solve to evaluate value -->
<td><input type="button" value="/" onclick="solve()"/></td>
<td><input type="button" value="*" onclick="dis('*')"/></td>
</tr>
</table>
</body>
</html>
```

## Building the Calculator Web Application

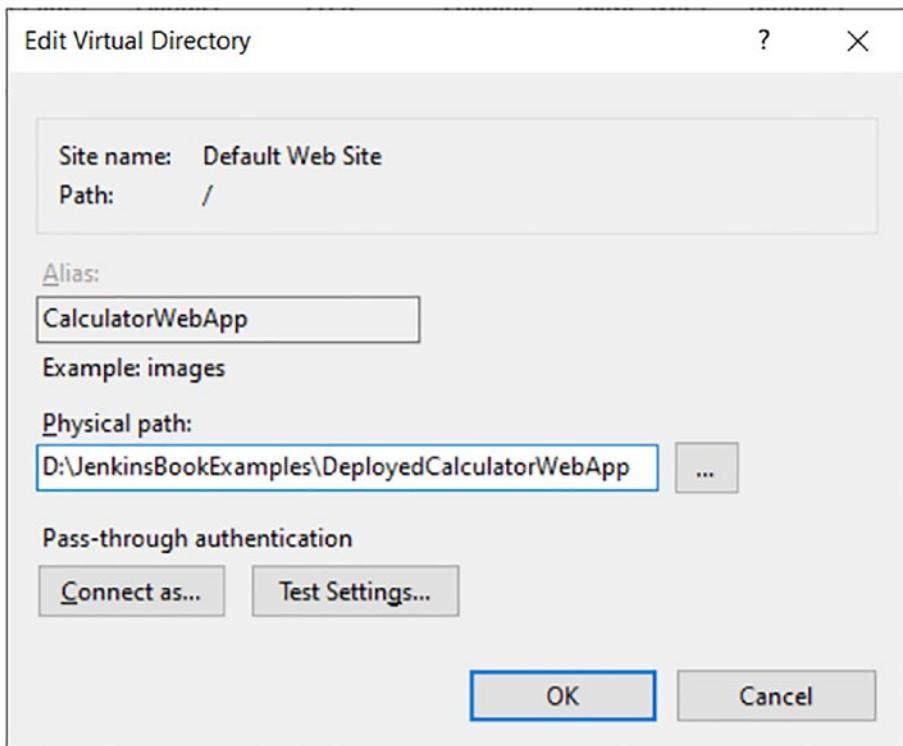
This web application contains only an .HTML file, so building this application does not require compilation or packaging phases that other applications usually need. I want to keep the example simple so that you can concentrate on the Jenkins features instead of the technical details of the application.

## Deploying the Calculator Web Application

To deploy this application, you simply need to copy the `Calculator.html` file to the directory being pointed to by the web alias created in the IIS web server.

I created an alias in the IIS web server called `CalculatorWebApp` and it points to the `D:\JenkinsBookExamples\DeployedCalculatorWebApp` directory on my machine.

You can add the alias configuration in the Edit Virtual Directory window, as shown in Figure 15-3.



**Figure 15-3.** The Edit Virtual Directory window from IIS

The deployment process includes one step to copy Calculator.html from the project's webapp directory to D:\JenkinsBookExamples\DeployedCalculatorWebApp so that it can be accessed using a web alias.

## Accessing the Calculator Web Application

I have my default website running on port 8082 in the IIS web server. If this is true for you, you can access the Calculator web application using the following URL:

`http://localhost:8082/CalculatorWebApp/Calculator.html`

# Understanding E-E Testing the Calculator Web Application Using the Selenium WebDriver

E-E testing consists of testing the business flows of an application. It is about running end-end positive as well as negative business flows from the user's perspective and checking if the application behaves correctly. These flows test the functionality of all the components, such as the interaction between the components along with interaction of the application components with third-party software like databases, web servers, application servers, and so on.

## What Is UI Automation Testing

UI automation testing is a type of testing in which an automation tool simulates end-user interactions with applications by interacting with its UI elements and tests in different end-end business scenarios. These kinds of end-end flows are usually automated using different licensed UI automation tools like UFT, TestComplete, Ranorex, etc. There are various open source UI automation tools available as well, like Selenium, Protractor, WebDriver IO, etc. This chapter uses Selenium to perform UI automation testing on the web application.

## How UI Automation Tools Work

These tools come with a set of libraries for UI interactions like typing, clicking, selecting options from dropdowns, and so on. You can give instructions to the UI automation tool using supported scripting languages. The tool needs the following two instructions to be able to interact with the application's UI control/element:

- **Identifying the UI element:** Before the tool can perform an interaction on the UI control/element, it needs to identify the UI element from the rest of the elements available on the application page/screen. This can be done with the help of unique attributes set to the UI control, like ID, name, class, etc. UI automation libraries support sets of APIs that will identify the UI controls based on the attribute values provided.
- **Performing interactions with the UI element:** Once the element is identified, you can instruct a tool to perform required interactions, like typing, clicking, etc., with the help of a set of APIs available in the tool libraries.

## What Is a Selenium WebDriver

A Selenium WebDriver is an UI automation tool that has libraries available in different programming languages, like Java, C#, JavaScript, Python, etc.

I developed UI tests for the CalculatorWebApplication using Selenium WebDriver's Python library.

## Understanding the Selenium Tests Written in Python

TestCalculatorWebApplicationUsingSeleniumPython is a Python project created in the D:\JenkinsBookExamples directory in which I have implemented UI tests. This section explains a few important files from this project in brief.

## EnvVars.csv

This file contains all the environment variables settings like paths, application URLs, etc. This file is located inside the automation test project directory. It contains a path of chromedriver.exe, which the Selenium tool uses to open in the Chrome browser. Chromedriver.exe is present in Drivers directory, which is inside the project directory. EnvVars.csv contains a path of ObjectRepositories folder along with the web application URL. You need to change this file according to paths and URL that you use.

## CalculatorPage.csv

This file is in the ObjectRepositories folder in the project directory. This file contains all the identification data of the elements you interact with in the UI tests.

## AutomationFramework Package

This package contains the automation framework code with the functions that will read the environment variables from EnvVars.csv, read the identification data from CalculatorPage.csv, and will interact with the browser and UI elements from the application.

Utils.py has a function called InitializeEnvVars() (see Listing 15-2) that reads the EnvVars.csv. You need to change the path (D:\\Jenkins BookExamples\\TestCalculatorWebApplicationUsingSeleniumPython \\EnvVars.csv) accordingly if you do not keep the project in the D:\\ JenkinsBookExamples folder.

***Listing 15-2.*** The InitializeEnvVars Function from the Utils.py File

```

@classmethod
def InitialiseEnvVars(cls):
    #Opening a csv file in Readmode using open function
    with open('D:\\JenkinsBookExamples\\TestCalculator
    WebApplicationUsingSeleniumPython\\EnvVars.csv') as
    csv_file:
        csv_reader=csv.reader(csv_file,delimiter=',')
        #Using this for loop we are reading the contents of
        EnvVars file row by row
        for row in csv_reader:
            #for the 1st iteration of loop:row=Chrome
            DriverPath,D:\\XoriantPythonSeleniumPostman
            Training\\Drivers\\chromedriver.exe
            #row(ChromeDriverPath,D:\\XoriantPythonSelenium
            PostmanTraining\\Drivers\\chromedriver.exe)
            Utils.EnvVars[row[0]]=row[1]

```

**CalculatorWebApp\_Pages Package**

This contains `BasePage.py`, which has a `BasePage` class implemented and `CalculatorPage.py`, which has a `CalculatorPage` class implemented.

The `CalculatorPage` class has different functions, like `DoAddition()` and `DoSubtraction()`, that are implemented and automate the flows of different calculator web application functionalities.

**CalculatorWebApp\_TestCases Package**

This package contains Python files with different test cases of the Calculator web application. Test cases of every functionality to be tested are implemented in separate files. The `test_AdditionFunctionality.py` file has test cases to test the addition functionality of the web application.

Test cases are grouped into different groups based on the scope of testing they handle, such as like RegressionTest, SmokeTest, etc.

---

**Note** *Smoke testing* is a type of testing done when testers receive a new build from developers, to check if the build has the required minimum characteristics and is eligible for more rigorous testing.

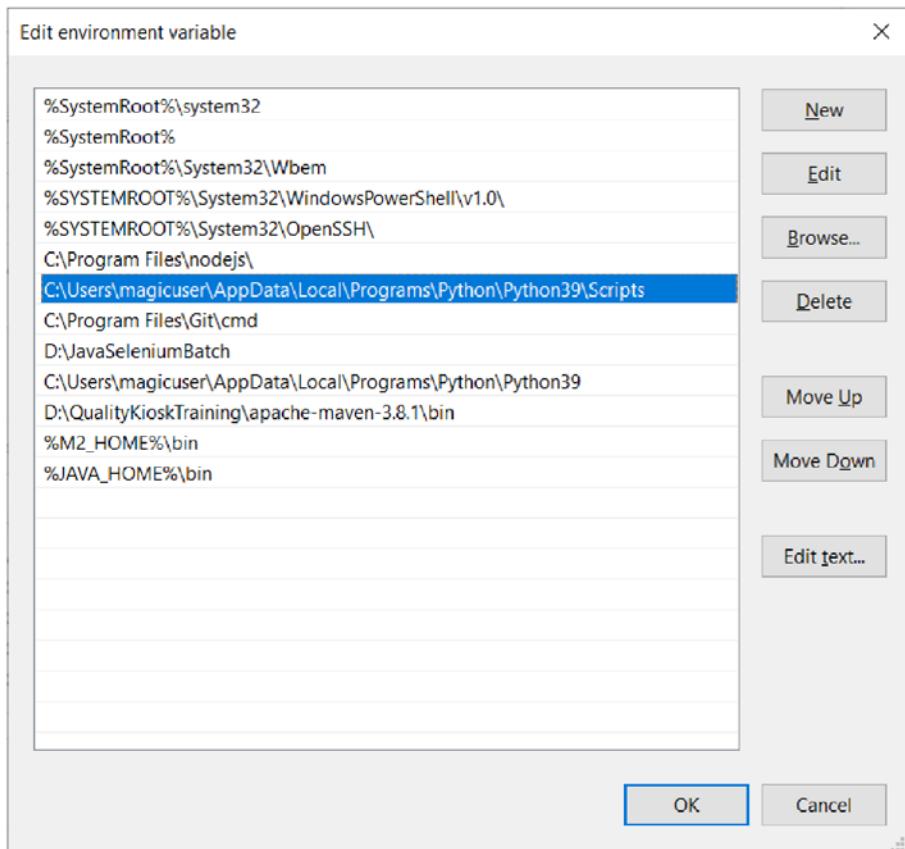
*Regression testing* is a type of testing done to confirm that none of the existing application functionalities have been adversely impacted by the changes implemented in an application.

---

## Software Setup to Run these Test Cases

Python: You need to install Python and add it to the PATH environment variable in the environment variables.

Pip: Pip is a Python package manager that installs Python packages from the pip registry. The Pip registry is a Python package registry available on the web, and it holds reusable Python packages. It's like mvnrepository.com for Java. You need pip to install the Python Selenium library. Pip also needs to be in the PATH environment variable. Pip gets installed along with Python in the Scripts folder inside the Python installation directory.



**Figure 15-4.** The Edit environment variable window showing the PATH environment variable

Both Python and pip have been added to the PATH environment variable in Figure 15-4.

Chrome browser: You need the Chrome browser installed on your machine.

Chrome Driver: Chrome Driver is a Selenium WebDriver implementation for the Chrome browser that converts Selenium API calls in Chrome native UI Automation APIs and gets the UI interactions from the test performed on applications running inside the Chrome browser. You can download the Chrome Driver version that's compatible with your Chrome browser using this link:

<https://chromedriver.chromium.org/downloads>

Selenium WebDriver Python libraries: You can install Selenium Python libraries by running the following command using the terminal:

```
pip install selenium
```

Pytest (unit testing framework): Pytest runs the Selenium test cases and generates an HTML report of the test execution.

You can install Pytest by running the following command using the terminal:

```
pip install pytest
```

## Running Selenium Python Tests

### Step 1: Open the Command Prompt and go to the TestCalculatorWebApplicationUsing SeleniumPython Project Directory

**You** need to open the command prompt and use the cd command to go inside the automation project directory.

### Step 2: Create a Virtual Python Environment and Activate It

**You** have to install the virtualenv Python package, which helps in creating the Python virtual environment, using this command:

```
pip install virtualenv
```

The Python virtual environment means that you create your own environment for the project instead of using the Python available globally on the system.

This will allow you to install the required version of Selenium libraries, pytest, and other required libraries without disturbing other projects that are using these libraries from the global Python path.

Also you don't need to carry the Python libraries required for the automation tests on different machines. Once you pull the automation code from the repository, the Jenkins job will run commands to create a virtual environment, and you can install the required libraries and invoke tests without manual intervention.

The command to create the virtual environment is:

```
virtualenv<NameOfVirtualEnvironment>.
```

I want to create a virtual environment called  
**“TestCalculatorWebApplication”**

I executed this command:

```
virtualenvTestCalculatorWebApplication
```

To activate this environment, use this command:

```
TestCalculatorWebApplication\Scripts\activate
```

## Step 3: Install the Required Python Packages

Let's install pytest, the Python Selenium library, and pytest-html (an HTML report generator) in this virtual environment using the following pip commands:

```
pip install pytest
```

You need to be inside the following directory to run this pip command:

```
TestCalculatorWebApplicationUsingSeleniumPython
```

Install the Python selenium library by running the following command from the same directory:

```
pip install selenium
```

Install the pytest-html reporter by running the following command from the same folder.

```
pip install pytest-html
```

Now that everything is set, you can run the tests using the following command:

```
pytest --html=TestsResult.html
```

This command will run functions whose names start with "test" in all Python files having names starting with "test\_" and will create a report of execution in the TestsResult.html file in the current directory.

Test execution will clear the browser caches first. Then it will run tests by opening a browser and interacting with the calculator web application as shown in Figure 15-1. Let's see the result by opening a TestsResult.html file.

## Pushing the WebApplication and Automation Project to the GitHub Repository

You learned in previous chapters how to work with the GitLab repository. Now we are going to look at how to work with the GitHub repository. Commands and Git concepts you learned about while working with the GitLab repository are applicable with GitHub.

Github.com provides you with the web platform to create remote repositories like Gitlab.com.

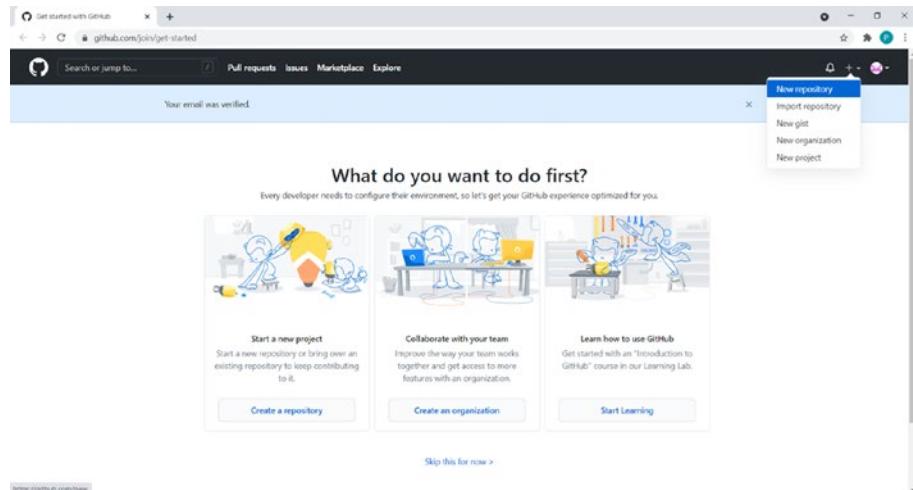
## Step 1: Sign up on Github.com

If you do not have a GitHub account, you need to sign up and create one.

Go to <https://github.com/> and complete the signup process. It's a simple process where you need to provide your desired username and password and other required information.

## Step 2: Creating a New Repository

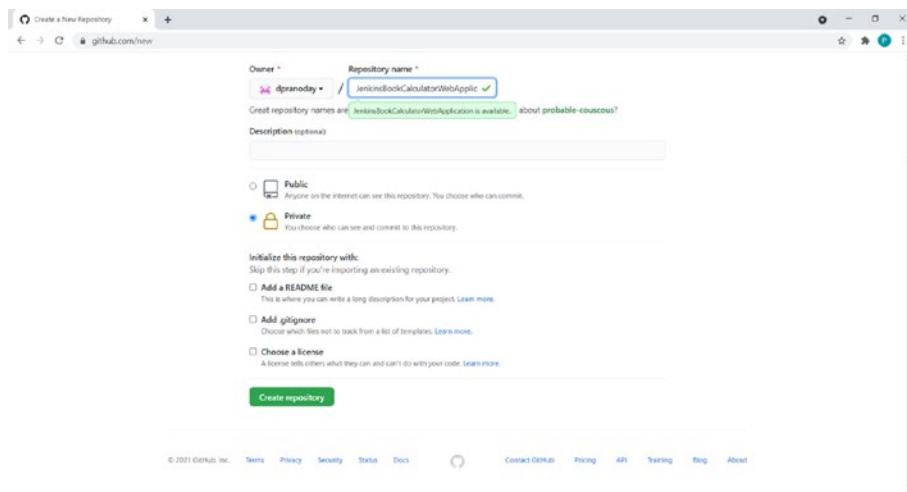
Sign into your GitHub account using your credentials. You will land on the GitHub dashboard page. Click the + sign available in top-left corner and select the New Repository menu option, as shown in Figure 15-5.



**Figure 15-5.** The New Repository menu option on the GitHub dashboard

It will open the Create a New Repository page. Enter the name of the repository in the Repository Name field. Select the Private radio button to create a Private GitHub repository, as shown in Figure 15-6.

Scroll down page to find the Create Repository button and click it.



**Figure 15-6.** The Create a New Repository page from GitHub after entering the required details

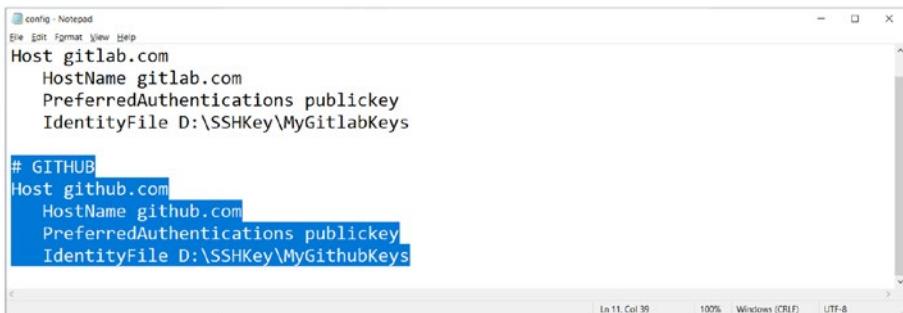
This will create a blank repository.

## Step 3: Creating a New SSH Key Pair for the GitHub Repository

You need to create a SSH key pair using the `ssh-keygen` command. Refer to Chapter 13 for details on creating SSH key pairs.

## Step 4: Include the SSH Private Key File Path in the `$(user.name)\.ssh\config` File

You need to include the path of the private key file in the config file present in `$(user.name)\.ssh\`, where you included the GitLab private key path. Refer to the highlighted part of Figure 15-7.



```
config - Notepad
File Edit Format View Help
Host gitlab.com
  HostName gitlab.com
  PreferredAuthentications publickey
  IdentityFile D:\SSHKey\MyGitlabKeys

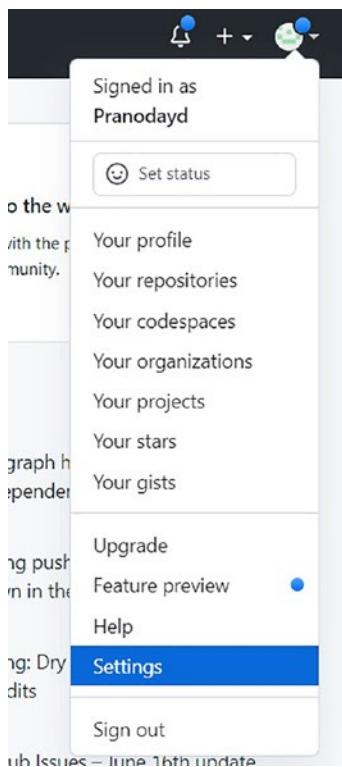
# GITHUB
Host github.com
  HostName github.com
  PreferredAuthentications publickey
  IdentityFile D:\SSHKey\MyGithubKeys

In 11, Col 39 100% Windows (CR/LF) UTF-8
```

**Figure 15-7.** The \$(user.name)/.ssh/config file after adding the GitHub private key file path

## Step 5: Adding an SSH Public Key to the GitHub Repository

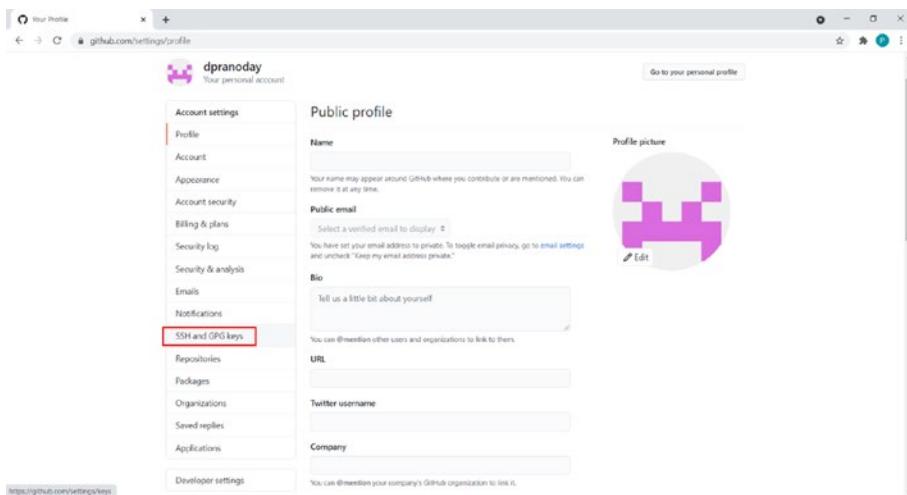
Click the Avtar shown in the upper-right corner of Github.com and select the Settings menu, as shown in Figure 15-8.



**Figure 15-8.** The Settings menu on GitHub

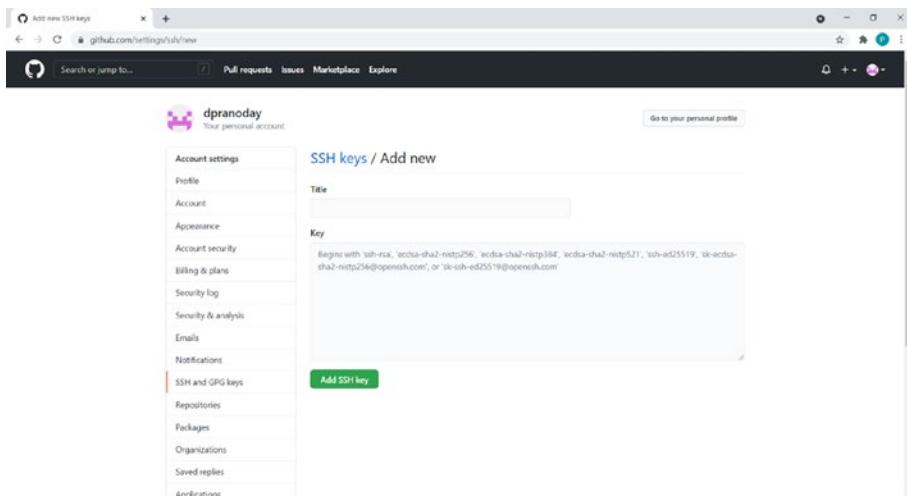
Click the SSH and GPG key menu option (highlighted in Figure 15-9) available on the left side of the Public Profile page.

## CHAPTER 15 CREATING JENKINS JOBS TO MANAGE A WEB APPLICATION PROJECT



**Figure 15-9.** The Public Profile page from GitHub

This will open the page that includes the New SSH Key button to add a key. Click the button to get the page shown in Figure 15-10.



**Figure 15-10.** The page from GitHub where you add the public key

Paste the public key into the Key field and click the Add SSH Key button.

## Step 6: Committing the Calculator Web Application to the Master Branch in the Local Repository

You need to enter into the project directory using the `cd` command and execute all the commands from the project directory. Create a blank Git repository using the `git init` command.

Add the whole current directory in the staging area using the `git add` command.

Commit the project to the master branch in the local repository using the command:

```
git commit --m "<commit message>"
```

I executed the following command to commit changes:

```
git commit --m "Committing CalculatorWebApp project"
```

When you do not specify the branch, the changes are committed to the master branch by default.

## Step 7: Pushing the Master Branch to the Remote Repository

Run the `git push <SSH URL of our repository>` command. To get the URL of the repository, go to the repository page on [Github.com](#). Click the SSH button available on the repository page. Click the Copy button to copy the URL.

Now go back to the command prompt to run the `git push` command:

```
git push git@github.com:dpranoday/  
JenkinsBookCalculatorWebApplication.git master
```

Enter your key passphrase. You should be able to push your code successfully.

Now go to the GitHub repository, refresh the page, and see if the code has been pushed successfully.

---

**Note** If, after entering the GitHub private key file path in the `(user.name)\.ssh\config` file, you get an error while trying to run the `git push` command using the SSH URL (such as `Permission denied`), go to `(user.name)\.gitconfig` and delete the `[core] sshCommand = ssh -i $HOME/.config/ssh/id_rsa -F /dev/null` statement if it exists. The path mentioned after `-i` could be different in your case.

---

## Pushing the Selenium Python Automation Project to GitHub

Let's create a new private repository on GitHub and push your automation project by following the same process as for the WebApplication project.

I created a repository called `TestWebApplicationWithSeleniumPythonTests` and pushed the automation project.

## Creating Parameterized Auto-Trigger Free-Style Jenkins Jobs

In this section you are going to create the following two parameterized Jenkins Freestyle jobs:

- **BuildAndDeployCalculatorWebApplication**  
(upstream job)
  - This job will be triggered when a change is pushed to the master branch of the **JenkinsBookCalculatorWebApplication** repository.
  - It will pull the latest change and copy **Calculator.html** in the IIS web server to deploy the latest web application and then it will call another parameterized job **TestCalculatorWebApplication** by sending it a parameter that defines which UI tests are to be executed.
- **TestCalculatorWebApplication** (downstream Job):
  - This job will be called by the first job called **BuildAndDeployCalculatorWebApplication**. It will pull the latest automation code from the master branch of the GitHub repository called **TestWebApplicationWithSeleniumTests**, run a specific group of Selenium tests based on the value received from the upstream job, and will email the test execution report .HTML file.

## Step 1: Installing the Parameterized Trigger Plugin

As you need to send parameter values from one job to another, you need to install the plugin called Parameterized Trigger. Before installing this, install the Maven Integration plugin, which is a dependency plugin for the Parameterized Trigger plugin.

## Step 2: Creating a BuildAndDeployCalculator WebApplication Job

I entered the name of the first free-style job and clicked the OK button.

This is a parameterized job, so you have to check the This project is Parameterized checkbox.

This job is going to copy `Calculator.html` to a folder pointed to by web alias named `CalculatorWebApp` from the IIS web server. Instead of hard-coding this folder path, we create a parameter to hold this path so that if in future you want to copy the `Calculator.html` to some different folder then you won't need to modify the build step.

To create a parameter, click the Add Parameter button.

You want to save a path of directory, so you need the String Parameter.

Click the String Parameter option and enter the required parameter details.

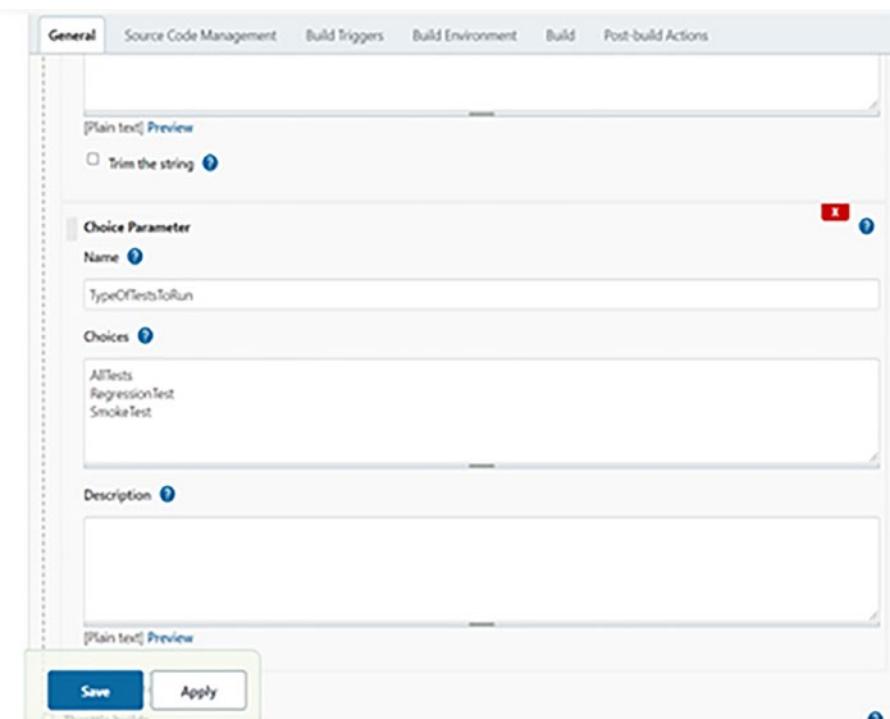
I entered `CalculatorWebApplicationDeploymentDirectory` in the Name field and `D:\JenkinsBookExamples\DeployedCalculatorWebApp` in the Default Value field.

Whenever you want to change the directory, you need to make the corresponding change in the Default Value field.

In this job, we want to have a choice parameter called `TypeOfTestsToRun` with three values: `RegressionTest`, `SmokeTest`, and `AllTests`. Based on value selected, this job will ask the downstream job to run a particular set of tests. The default value would be `AllTests`. If you want to run only `RegressionTests` or `SmokeTests`, select the values accordingly.

Let's create the choice parameter by following the steps you executed while creating String parameter. This time you need to select the Choice Parameter from the Add Parameter dropdown.

I have included three choices in Figure 15-11. The All Tests option is mentioned on the first line as it's the default value.



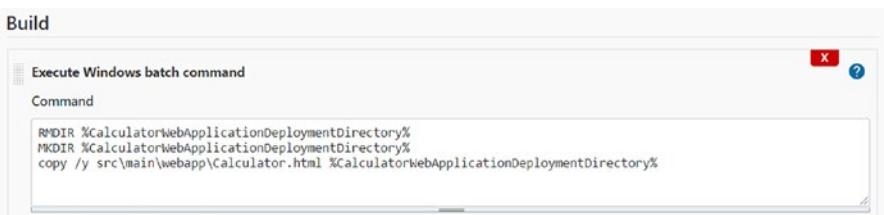
**Figure 15-11.** The values set for the `TypeOfTestsToRun` choice parameter

Scroll down and select the Git radio button, enter the SSH URL of the `JenkinsBookCalculatorWebApplication` GitHub repository, and select the SSH credentials entry created for GitHub in the Credentials dropdown. Keep the master in the Branch field.

Now scroll down and check the **Poll SCM** checkbox and set the polling schedule to `H/5 * * * *`.

Scroll down and add a build step. Click the Add Build Step button and select the Execute Windows Batch Command option.

I added three Windows batch commands in the Edit field shown in Figure 15-12.



**Figure 15-12.** The batch commands written to deploy the Calculator web application

- The first command removes the existing directory.
- The second command creates a new directory.
- The third command copies **Calculator.html** from **src\main\webapp** in the newly created directory.

As discussed in the last chapter, when you run a job having SCM set, the remote repository gets cloned in the workspace directory and this workspace directory is considered the working directory for job execution. Hence, you access `Calculator.html` using the relative path in the third command.

Note that, to access the value of the `CalculatorWebApplicationDeploymentDirectory` parameter, you must enclose it in `%` signs in all three batch commands.

Add a post-build step to call the downstream job. Click the Add Post-Build Action button and select the Trigger Parameterized Build On Other Projects option.

Add the name of the downstream job to the **Projects to Build** field. It will give an error near this field as you have not created the downstream project, and you can ignore it for now.

Select the Stable or Unstable But Not Failed value in the Trigger When Build Is field, as you want to run tests if the build is successful, irrespective of its stability.

Now add a parameter that will send a value to the parameterized job.

Click the Add Parameters button and select the Predefined Parameters option. This will show you the Predefined Parameters section.

I created a variable named `TestType=$(TypeOfTestsToRun)` which will be the selected value from the Choice parameter and will pass it to the downstream job.

## Step 3: Creating the TestCalculatorWeb Application Job

On the dashboard I entered the name of the second freestyle Job and clicked the OK button. This is a parameterized job so you have to check the This Project Is Parameterized checkbox

Click the Add Parameter button and select the String Parameter option.

Set the name of parameter to `TestType`. It has to have the same name as the parameter to which we are setting value to be sent in the Upstream job. Set `AllTests` in the Default Value field.

Scroll down and select the Git radio button, enter the SSH URL of the `TestWebApplicationWIthSeleniumPythonTest` GitHub repository, and select the SSH credentials entry created for GitHub in the **Credentials** dropdown. Let's keep the master in the Branch field.

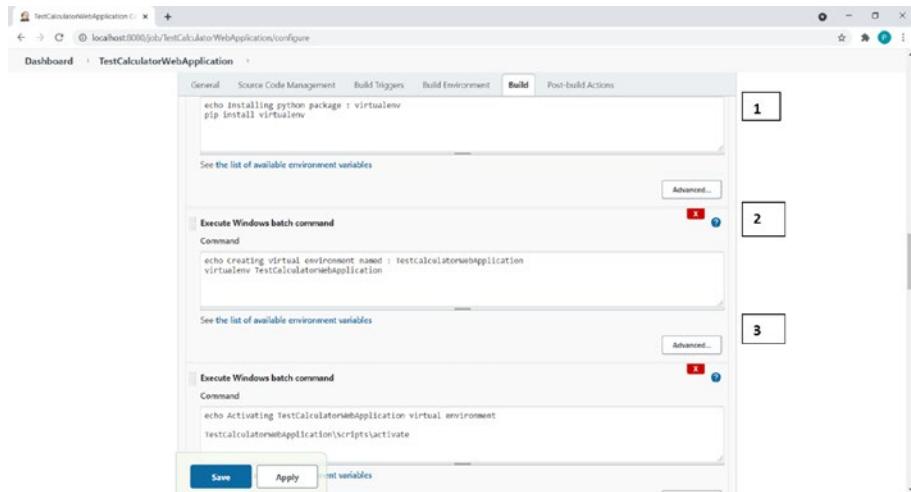
Note even though we used the SCM repository, we are not checking the Poll SCM checkbox because we do not want to trigger this job when there's a merge in the master branch. This job is going to be called from our Upstream job

Scroll down and add the build step. Click the Add Build Step button and select the Execute Windows Batch Command option.

I created multiple batch command steps by clicking the Add Build Step button and selecting Execute Windows Batch Command multiple times.

## CHAPTER 15 CREATING JENKINS JOBS TO MANAGE A WEB APPLICATION PROJECT

The first batch command marked with No.1 in Figure 15-13 installs the virtualenv Python package.

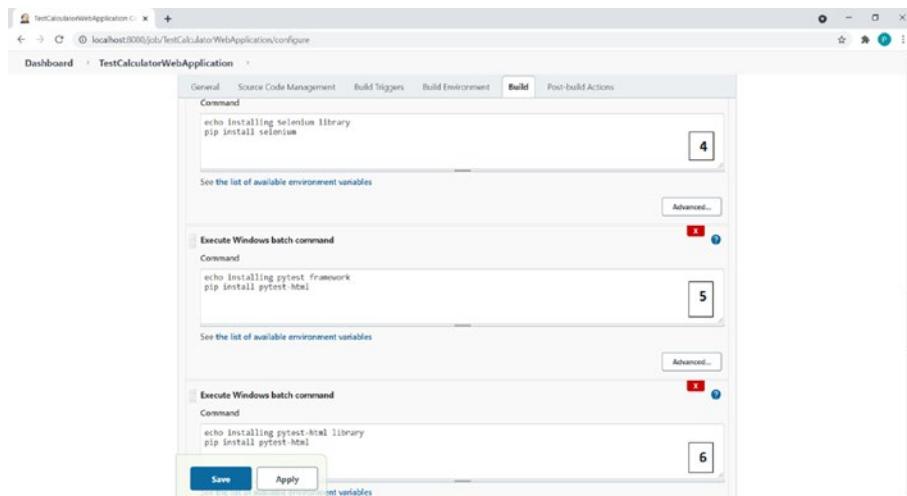


**Figure 15-13.** The first three commands required to run Selenium tests

The second batch command marked with No.2 in Figure 15-13 creates a virtual environment called TestCalculatorWebApplication.

The third batch command marked with No.3 in Figure 15-13 activates the created virtual environment.

## CHAPTER 15 CREATING JENKINS JOBS TO MANAGE A WEB APPLICATION PROJECT



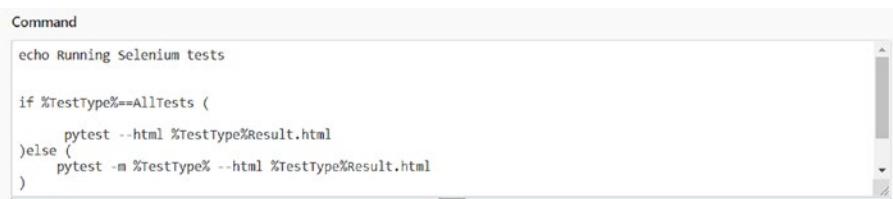
**Figure 15-14.** The next three commands required to run Selenium tests

The fourth batch command (marked with No.4 in Figure 15-14) installs the Selenium Python library.

The fifth batch command marked with No.5 in Figure 15-14 installs the Pytest framework library.

The sixth batch command (marked with No.6 in Figure 15-14) installs the pytest-html library.

The seventh batch command shown in Figure 15-15 checks the value of the TestType variable. If the value is AllTests then it runs all tests and saves the report in a file named as AllTests.html. If the value is something other than AllTests then it runs tests by using the pytest -m command and saves the result in a file with the corresponding name.



```
echo Running Selenium tests

if %TestType%==AllTests (
    pytest --html %TestType%Result.html
) else (
    pytest -m %TestType% --html %TestType%Result.html
)
```

**Figure 15-15.** The batch code that will trigger different suites of Selenium tests

---

**Note** The ( bracket after the if statement and else statement should be on the same line as the if or else. One blank space ‘ ’ is also mandatory between the condition statement and (‘ . No spaces ‘ ’ should be present between the else word and )’ of the if block.

---

Click the Save button.

## Configuring an Extended Email Notification

In this section, you learn how to send a test execution report of the Selenium E-E tests through email. In the last chapter, you learned how to configure Jenkins to send email notifications through default email settings provided by the Mailer plugin. Using that plugin, you cannot send customize email messages or send email attachments. In order to send detailed customized email messages along with email attachments, you need to use the Email Extension Jenkins plugin and then configure the Extended Email-notification settings.

## Step 1: Installing the Email Extension Plugin

You need to install the Email Extension plugin in Jenkins. For detailed steps on how to install plugins, refer to Chapter 5.

## Step 2: Configuring Extended Email Extension settings

Click the ManageJenkins ➤ Configure System menu option. Scroll down the page to the Extended E-mail Notification section, which is available only when you have the Email Extension plugin installed.

You are going to configure the same Gmail settings that you configured for the basic email notification in last chapter. Figure 15-16 shows these settings again for your quick reference.

Extended E-mail Notification	
SMTP server	smtp.gmail.com
SMTP Port	465
SMTP Username	pranoday.dingare@gmail.com
SMTP Password	[REDACTED]
<input checked="" type="checkbox"/> Use SSL	
<input type="checkbox"/> Use TLS	

**Figure 15-16.** The email settings

Let's look at a few more frequently used settings from this section. You can configure a default list of email IDs that you want to send email notification using the Default Recipients field. I configured my two email IDs in this field.

If you have email IDs configured in the Default Recipients list but do not want to send email notifications to a few of them, you can add those email IDs to the Excluded Recipients list.

The Default Subject field lists the Jenkins Environment Variables PROJECT\_NAME (which returns the job name), BUILD\_NUMBER (the build number of the current build being executed), and BUILD\_STATUS (the status of build like success or failure).

The Default Content field allows you to configure a default message in an email.

Click the Save button.

---

**Note** Settings we configure in this section are global settings so are applied to all jobs having email notifications. Do not configure any job-specific details or details of an application you are building in any of your Jenkins jobs.

---

## Step 3: Adding an Email Notification Step in the Post-Build Section of the TestCalculatorWeb Application Job

Go inside the TestCalculatorWebApplication job by clicking the Configure menu. This will take you inside settings of the job.

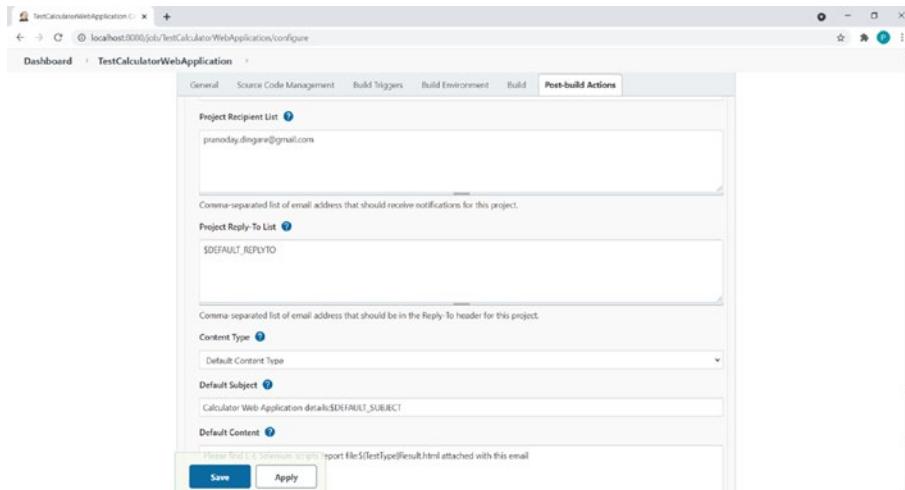
Scroll down to the Post-Build Actions section (see Figure 15-17). Click the Add Post-Build Action button and select Editable Email Notification.

Here you can configure Jenkins job-specific details to be sent in the email notification.

I configured my email ID to receive email notification from this job in the Project Recipient List field. Note that if you set a list of email IDs here it will override the global Default Recipient setting you configured previously.

## CHAPTER 15 CREATING JENKINS JOBS TO MANAGE A WEB APPLICATION PROJECT

Include the name of the calculator application concatenated with the default subject mentioned in the global settings in the Default Subject field.



**Figure 15-17.** The Project Recipient List, Default Subject, and Default Content fields with customized values

I set the message to be displayed in the email body using the Default Content field. The default content field contains: "Please find E-E Selenium scripts report file: \${TestType}Result.html attached with this email"

Note that the parameter variable is referred to as \${TestType} to include its value in message displayed in the email body.

In the attachments field you have to mention the path of files to be sent as an email attachment. This path should be relative to the Workspace directory. I used "\*\*/\${TestType}Result.html" in the attachments field.

I added the Selenium test execution report HTML file using the following pattern.

In the pattern, `**/` means look for the file in the current directory (the Workspace directory) as well as its subdirectories. The name of our result file contains the `TestType` parameter instead of hard-coding the name of the file.

Click the Advanced Settings button.

Now click the **Add Trigger** button in the **Triggers** section to configure the email notification trigger. To send an email notification regardless of success or failure, select the Always option. Click the Save button.

## Running a Parameterized Free-Style Job Manually

Go to the dashboard and run `BuildAndDeployCalculatorWebApplication` by clicking the clock sign.

This will take you to the next page where the job will ask you to select a parameter value from the dropdown.

The job is waiting for the user to put values in the parameters. Select `RegressionTest` from the `TypeOfTestsToRun` dropdown. Click the Build button. This will start this job execution.

After completion, it will trigger the downstream job. It will run UI automation tests. After completion of the downstream job, it will send you an email notification.

Click it to see the email and the `RegressionTestResult.html` attached to the email.

# Auto-Triggering a Parameterized Free-Style Job

Let's make one change to the `Calculator.html` and then push it to GitHub master branch, which will trigger the upstream job followed by the downstream Job.

I changed the title color to red (it was previously green). Refer to the highlighted portion of Figure 15-18.



```
<style>
  .title{
    margin-bottom: 10px;
    text-align:center;
    width: 210px;
    color:red;
    border: solid black 2px;
  }

  input[type="button"]
```

**Figure 15-18.** The part of `Calculator.html` where you need to change the title color

The button background color has changed to red too, which was also green previously. Refer to the highlighted portion of Figure 15-19.



```
input[type="button"]
{
  background-color:red;
  color: black;
  border: solid black 2px;
  width:100%
}

input[type="text"]
```

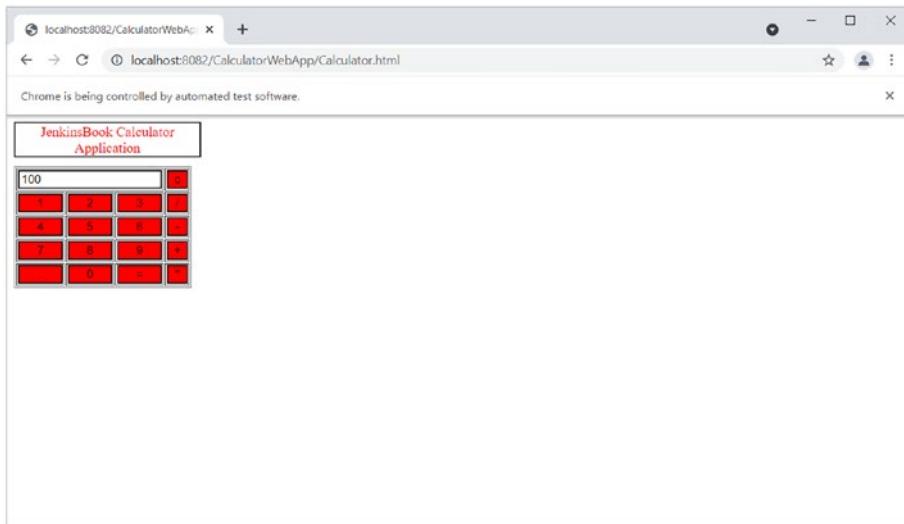
**Figure 15-19.** The part of `Calculator.html` where you need to change the button color

Let's commit the change in the master branch in the local repository and then push it to the remote GitHub repository in the master branch.

Once you push the code to the remote repository, it will trigger the upstream job with the default values.

Once it's done with the upstream job, it will trigger the downstream job.

This will run the UITests on the updated application. You can see the title's foreground color and buttons' background color have changed to red, as shown in Figure 15-20.



**Figure 15-20.** The changed Calculator web application opened by the Selenium test

After completing the downstream job, you will get an email with the AllTestsResult.html file attached.

# Creating a Parameterized Pipeline Job

This section explains how to build a parameterized pipeline job. This section creates the following two parameterized pipeline jobs:

- **BuildAndDeployCalculatorWebApplicationPipeline**  
(upstream job): This pipeline job will be triggered on a push of change in the master branch of the **JenkinsBookCalculatorWebApplication** repository.  
It will pull the latest change and copy **Calculator.html** in the IIS web server to deploy the latest web application and then it will call another parameterized pipeline job **TestCalculatorWebApplicationPipeline** and send it to the type of tests to be executed as an argument.
- **TestCalculatorWebApplicationPipeline**  
(downstream job): This pipeline job will be called by the first job called **BuildAndDeployCalculatorWebApplicationPipeline**. It will pull the latest automation code from the master branch of the GitHub repository called **TestWebApplicationWithSeleniumTests**, run the Selenium tests based on the values received from the upstream job, and will email the test execution report .HTML file to the recipients.

# Creating a Build and Deploying the Calculator Web Application Parameterized Pipeline Job

Step 1: Creating a pipeline job from the dashboard: Go to the dashboard and click the New Item menu.

Enter `BuildAndDeployCalculatorWebApplicationPipeline` for the job name and select the Pipeline option. Click the OK button.

Select the `This Project Is Parameterized` checkbox and create two parameters—a string parameter to hold the directory path where `Calculator.html` file is copied and a choice parameter, which will define which group of Selenium tests are executed once the application is deployed.

Select the `Poll SCM` checkbox and configure the schedule as you did for the free-style job in previous chapters. In the Pipeline editor, write your pipeline script with the help of the Snippet Generator.

Place the generated script line in the Pipeline script editor inside the Checking Out Calculator Web Application Repository stage. Listing 15-3 shows that stage implemented in the pipeline script.

***Listing 15-3.*** The Pipeline Script with the Checking out Calculator Web Application Repository Stage

```
node{
    def mvnHome
    stage('Checking out Calculator Web Application
repository.')
    {
        git credentialsId : 'MyGithubCredentials' , url:
            'git@github.com:dpranoday/JenkinsBookCapculatorWeb
Application.git'
    }
}
```

Once the application code is pulled, you need to copy `Calculator.html` from `$(Workspace)\src\main\webapp\` to the path present in the `PathOfCalculatorAppDeploymentDirectory` parameter.

Now create a Windows batch step using the Snippet Generator. Copy the generated script code and put it inside the Pipeline script editor, inside the Deploying Web Application stage block, as shown in Listing 15-4.

***Listing 15-4.*** The Pipeline Script with the Deploying Web Application Stage

```
stage('Deploying Web Application')
{
    bat
    """
        RMDIR %PathOfCalculatorAppDeploymentDirectory%
        MKDIR %PathOfCalculatorAppDeploymentDirectory%
        copy /y src\\webapp\\Calculator.html
        %PathOfCalculatorAppDeploymentDirectory%
    """
}
```

Note that I replaced the hard-coded directory path with `%PathOfCalculatorAppDeploymentDirectory%`.

Once you deploy your application by copying `Calculator.html`, you are going to call the second pipeline job, called `TestCalculatorWebApplicationPipeline`.

Let's create the build: build a job step using the Snippet Generator. You'll see the error below the Project to Build field because you have not yet created the second job.

Let's create the third stage block, the Testing Calculator Web Application stage, and paste the generated step into it. Add the parameters argument to this build step.

This stage contains the following build step:

```
build job:'TestCalculatorWebApplicationPipeline',parameters:[$class: 'StringParameterValue', name: 'TestType', value: params.TypeOfTests]
```

We are calling the `TestCalculatorWebApplicationPipeline` **job** and sending it a parameter of class: `StringParameterValue` and name: `TestType`. **The** value of this parameter would be the one selected in the Choice parameter. Observe this build job step in Listing 15-5 to see how a job parameter is referenced using `params.TypeOfTests` in order to evaluate it and then send the evaluated value through the parameter.

***Listing 15-5.*** The Pipeline Script with the Testing Calculator Web Application Stage

```
stage('Testing Calculator Web Application')  
{  
    build job:'TestCalculatorWebApplicationPipeline',parameters:[$class: 'StringParameterValue', name: 'TestType', value: params.TypeOfTests]  
}
```

Click the Save button.

## Creating a TestCalculatorWebApplication Pipeline Job

Go to the dashboard and click the New Item menu. Enter `TestCalculatorWebApplicationPipeline` for the job name and select the Pipeline option. Click the OK button.

Select the This Project Is Parameterized checkbox.

You need to create a parameter of type String with the same name as the one used to send a value to the upstream job. In the upstream job, you created parameter called TestType in the Build step.

Using the following steps, you can call this job from the UPSTREAM job.

```
build job:'TestCalculatorWebApplicationPipeline',parameters:[$class: 'StringParameterValue', name: 'TestType', value: params.TypeOfTests]
```

Observe that the name of the parameter is TestType, so you need to create a parameter called TestType in this downstream job.

Let's write the pipeline script in this job. Listing 15-6 shows the script added to the Pipeline editor.

***Listing 15-6.*** Pipeline Testing Calculator Web Application

```
try
{
stage('Pulling Test automation code')
{
    gitcredentialsId: 'MyGithubCredentials', url:
        'git@github.com:dpranoday/TestWebApplicationWith
        SeleniumPythonTests.git'
}
stage('Running tests')
{
    if(params.TestType.equals("AllTests"))
    {
        bat "pytest --html ${params.TestType}
        Result.html"
    }
}
```

```

        else
        {
            bat "pytest -m ${params.TestType} --html
            ${params.TestType}Result.html"
        }
    }

}
finally
{
    emailextattachmentsPattern: '**/*'+params.TestType+'Result.
    html', body: 'Please find E-E Selenium scripts report
    file:' +params.TestType+'Result.html attached with this email',
    subject: 'Calculator Web Application details:$DEFAULT_SUBJECT',
    to: 'pranoday.dingare@gmail.com'
}

```

Let's look at the different blocks of this script in detail.

In the previous block, all the stages are enclosed between try blocks so that even if a stage fails and you get an exception, the script code should send you an email notification.

**stage('Pulling Test automation code')**: Checks out the master branch from the GitHub repository.

**stage('Running Tests')**: At this stage, you are checking the value of the pipeline parameter. If it is AllTests then you are calling the bat step which will run all pytesttests. Otherwise, it will run a particular group of tests like RegressionTest or SmokeTest, etc., by using -m.

Observe how we referred to the parameter in the script code. If the block reference is params.TestType, you need to refer to the pipeline script parameters using params.

In `bat` step, the entire batch command is enclosed between double quotes (“) and the script parameter is interpolated using  `${params.TestType}` .

`finally` block: In the `finally` block, you are sending an email notification using the `email text` step.

```
emailextattachmentsPattern: '**/*'+params.TestType+'Result.html', body: 'Please find E-E Selenium scripts report file:' + params.TestType +'Result.html attached with this email', subject: 'Calculator Web Application details:$DEFAULT_SUBJECT', to: 'pranoday.dingare@gmail.com'
```

The first argument of this step includes the relative path of the .HTML file in first argument.

It also includes the script parameter using `params`. But this reference is without curly braces ('{'). Here, you are not interpolating but are concatenating the variable value using `+`.

The `'**/*'+params.TestType+'Result.html'` statement evaluates the variable and concatenates its value with `**/*` and `Result.html`.

The second argument is `body` which displays a message in an email body. It concatenates the value of the pipeline parameter using the `+` sign.

The third argument displays the `subject`.

Note the reference of the `DefaultSubject` variable. It is interpolated using the `$` sign but no curly brace is used. If you want to interpolate any variable, whether environment variable or pipeline parameter in a string enclosed within single quotes ('), you need to use `$` without the curly braces. Click the `Save` button to save the configuration of the pipeline job.

## Running the Parameterized Pipeline Job Manually

Click the clock image shown for the BuildAndDeployCalculatorWebApplicationPipeline job.

Then select the SmokeTest option from the dropdown and click the Build button.

This will trigger the upstream pipeline, which will deploy the application and call the downstream pipeline, which will run the tests and send an email notification containing the test report.

## Automatically Triggering the Parameterized Pipeline Job

Let's make a change to the Calculator web application and push it to the central GitHub repository. I changed the Title color and button controls background color to Green in Calculator.html.

Commit the master branch in the local repository and then push it to the remote GitHub branch. Now push it to the GitHub repository.

Once you push the code to the remote repository, it will trigger the upstream pipeline with the default values. It will call the downstream job and start the test execution.

The tests will be executed on the updated application, in which the color has changed to green, and it will send the test results via email.

## Summary

This chapter showed you how to set up the UI automation tool called Selenium using Python libraries. You automated an end-end build lifecycle of the CalculatorWeb application using upstream and downstream free-style jobs. You also learned how to send parameters from an upstream job to a downstream one and execute the parameterized jobs manually by triggering them automatically. In the closing sections of the chapter, you implemented the same upstream and downstream parameterized pipeline jobs and executed them manually by triggering them automatically. You also configured customized email notifications in the jobs. The next chapter discusses pipelines as code.

## CHAPTER 16

# Understanding Pipeline as Code

In the last chapter you learned about GitHub and integrating Jenkins with GitHub. You also learned about parameterized free-style and pipeline jobs. You learned about E-E testing using the UI automation tool called Selenium and managed the web application's release using upstream and downstream jobs. You also saw how to set up customized email notifications that will send UI automation test reports.

In this chapter, you learn about API authentication, how to set up API authentication in the GitHub repositories, and how to integrate them with GitHub repositories having API authentication set from Jenkins. You are also going to learn about pipeline as code and how to trigger this pipeline from GitHub using Webhooks, when an event occurs on the GitHub side like a merge, push, etc.

## What Is API Authentication

In earlier chapters of this book, you learned how to set up basic authentication (username/password) and SSH authentication (public key-private key pair) in the GitLab/GitHub repositories and how to interact with these repositories using authentication credentials.

Along with basic and SSH authentication, there is another popular authentication technique called *API authentication*. In this technique, you have to create a secret text called an API access token that you can use in place of a password to access the Git remote repositories. This technique of authentication is mostly used when accessing the Git remote repositories, either through API endpoints or the command-line.

## How to Apply API Authentication to GitHub Repositories

In order to apply an API authentication technique to the GitHub repositories, you need to create an API token in GitHub. Here are the steps to create an API access token in GitHub.

Step 1: Logging into GitHub: Log into your GitHub account using the GitHub username and password. After logging in, you will see the GitHub dashboard.

Step 2: Creating an API access token: Click the User Avtar shown in top-right corner of the dashboard and select the Settings menu.

This will take you to the Public Profile page. Click the Developer Settings link provided on the left side of the page.

This will take you to the GitHub Apps page. Click the Personal access Tokens link.

It will take you to the Personal Access Tokens page. Click the Generate New Token button.

This will open the New Personal Access Token page. Enter a string in the Note field that will indicate the purpose of the access token and then select the Repo checkbox inside the Select Scopes section to get full control of the private repositories (see Figure 16-1).

## New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### Note

[AccessTokenToAccessPrivateRepositories](#)

What's this token for?

### Expiration \*

30 days

The token will expire on Wed, Jul 20 2022

### Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows

**Figure 16-1.** The repo checkbox selected to select all permissions

Scroll down the page and click the Generate Token button, which will take you to the next page. Click the Generate Token button.

Click the copy button highlighted in Figure 16-2 to copy the token and then paste it into a file to use it in the future, whenever required.

✓ ghp\_K9YChRgV55v6tICOj8Tf2Ax8EdJlje4BJlx4 

**Figure 16-2.** The copy button highlighted along with the generated token

Once you navigate away from this page, for security reasons this generated API token will not be shown again. So make sure to copy this token and keep it in a file before you leave this page.

## How to Use an API Token to Access a Private GitHub Repository

Now that you have your API token, you can use it to access the JenkinsBookCalculatorWebApplication repository. You will clone the repository, make a few changes, and push these changes to the remote repository by accessing it using the API token you just generated.

Step 1: Cloning the JenkinsBookCalcualtorWebApplication repository: Create a blank folder to clone the repository.

I created a blank folder named CloningRepoUsingAPIToken on D:.

Open the command prompt and go inside this newly created directory using the cd command. Enter the git clone command using the HTTPS URL of the repository

I entered the following command:

```
git clone  
https://github.com/dpranoday/JenkinsBookCalculator  
WebApplication.git
```

After pressing the Enter key it will appear in the Sign In window of GitHub. Enter your access token into the Personal Access Token field.

Click the Sign in button. It will clone the repository.

---

**Note** Once you access your repository using the username and password/access token, these credentials are stored in the Windows Credentials section on your system.

---

Got to the Windows Credentials section in the Control Panel. Type Windows Credentials in the Start menu.

Select the Manage Windows Credentials menu, which will open the Manage Your Credentials window. You can see that my credentials are saved under the Generic Credentials section (see Figure 16-3).



**Figure 16-3.** The credentials entry with the GitHub username and access token

In subsequent access attempts to any of the GitHub private repositories, the credentials saved here will be used and the GitHub SignIn window will not be shown.

Step 2: Make code changes and push them to the GitHub remote repository: Open the `Calculator.html` file from the `src/main/webapp` directory from the cloned repository.

Let's change the title and button background colors from green to red (the lines to be changed are shown in bold in Listing 16-1).

**Listing 16-1.** CSS Code from `Calculator.html` from Calculator Web Application

```
<style>
  .title{
    margin-bottom: 10px;
    text-align:center;
```

```
width: 210px;  
color:green;  
border: solid black 2px;  
}  
  
input[type="button"]  
{  
background-color:green;  
color: black;  
border: solid black 2px;  
width:100%  
}
```

Now save the changes and commit them to the local repository using the following two commands:

```
git add .  
git commit -m "Changing title color and button control  
background color to red"
```

The changes will be committed to the default branch, which is `master`.

Push the changes to the remote repository using the `git push` command:

```
git push  
https://github.com/dpranoday/JenkinsBookCalculatorWeb  
Application.git master
```

As discussed, it will not prompt you to enter the access token again and will push your changes to GitHub repository.

After refreshing the GitHub repository page, you can see your change in the remote repository.

# Creating an API Access Token in GitLab

This section explains how to create an API access token in the GitLab repository.

Step 1: Log into GitLab: Log into your GitLab account using your GitLab username and password. After logging in, you will see the GitLab dashboard.

Step 2: Create an API access token: Click the User Avtar shown in the top-right corner of the dashboard and select the Preferences menu.

This will take you to the Preferences page. Click the Access Tokens link provided on the left side of the page.

Clicking the Access Tokens link will take you to the Personal Access Tokens page.

Enter a string that will indicate the purpose of access token in the Token Name field and select the Write\_Repository checkbox in the Select scopes section to get full control over the private repositories.

Scroll down the page to the ***Create Personal Access Token*** button. Clicking it will take you to the **Access Token** page.

Click the ***Copy Personal Access Token*** button to copy the token. Paste it into a file to use it in the future whenever required.

Once you navigate away from this page, for security reasons this generated API token will not be shown again. So make sure you copy this token and keep it in a file before you leave this page.

## How to Use the API Token to Access a Private GitLab Repository

Now you have the API token. Let's see how to use it to access the JenkinsBookCalculatorAPI repository. You will clone your repository, make a few changes, and push these changes to the remote repository by accessing the API token you just generated.

Step 1: Cloning the JenkinsBookCalculatorAPI repository: Create a blank folder to clone the repository.

I created a blank folder named

CloningCalculatorAPIRepoUsingAPIToken on D:.

Open the command prompt and go inside this newly created directory using the cd command. If you have credentials stored in the Windows Credentials Manager, delete them before cloning the repository. Let's go to the Windows Credentials in the Manage Your Credentials window.

You can see the GitLab credentials (Username-Password.API access token). Click the Remove button for the GitLab credentials entry.

Click the Yes button on the Delete Generic Credential confirmation.

Enter the git clone command using Https URL of the GitLab repository. I entered the following command:

```
git clone  
https://gitlab.com/Pranoday/Jenkinsbookcalculatorapi.git
```

After pressing Enter, you get the Git Credential Manager dialog.

Enter the GitLab account's username in the User Name field and the GitLab access token in the Password field then click the OK button.

After clicking the OK button, the repository will be cloned.

Step 2: Making code changes and pushing them to a GitLab remote repository: Open the Calculator.java file from src/main/java/Pranodayd/CalculatorAPI from the cloned repository.

Change name of the variable from Res to R in the Addition function. (See this change in Listing 16-2.)

**Listing 16-2.** The Name of the Res Variable Is Changed to R in the Addition Function from Calculator.java

```
public int Addition(int num1,int num2)  
{  
    int R=num1+num2;
```

```
    return R;  
    //return 0;  
}
```

Save the changes and commit them to the local repository. These changes will be committed to the default branch, master. I committed changes using the following two commands:

```
git add .  
git commit -m "Changed variable name in Addition function"
```

Push the changes to the remote repository using the `git push` command:

```
git push  
https://gitlab.com/Pranoday/Jenkinsbookcalculatorapi.git master
```

As discussed, it will not prompt you to enter the access token again and will push your changes to the GitLab repository.

After refreshing the GitLab repository page, you can see the change in the remote repository.

## How to Access a GitHub/GitLab Repository Using an API Token in Jenkins

In this section, you create a Jenkins free-style job to release the `CalculatorAPI.jar` file by accessing the `JenkinsBookCalculatorAPI` repository from GitLab.

Let's first set up Jenkins and create a free-style job to release a new version of the `CalculatorAPI` in the Nexus repository.

## Step 1: Setting Up Maven in Jenkins

Refer to the Understanding Global Tool Configuration Settings: section in Chapter 6. You can skip this step if you have been following this book from start and have already configured Maven in Jenkins.

## Step 2: Creating a Free-Style Job from the Jenkins Dashboard

Click the New Item link on the Jenkins dashboard. Enter the job name in the Enter an Item Name field and select the Free-style project option.

I called the job ReleaseCalculatorAPI\_APIToken and selected the Free-style Project option then clicked the OK button.

Click the OK button to open the next page.

Select the Git radio button in the Source Code Management section and enter the Git code repository https URL in the Repository URL field. Refer to the “Push the Code from Local Repository to Central Repository on GitLab” section in Chapter 12 for these steps.

Click the Add button displayed in the Credentials field and click the Jenkins option. This will open the Jenkins Credentials Provider: Jenkins window. Select the Username with Password option in the Kind dropdown.

Enter your GitLab username in the Username field and the GitLab API token in the Password field. Enter a unique string as a credentials entry ID in the ID field.

---

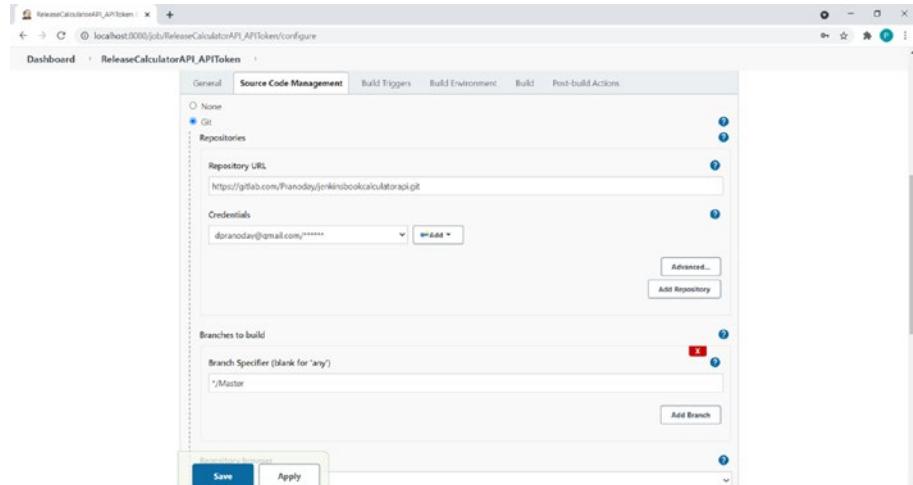
**Note** To access GitLab/GitHub repositories using API authentication from Jenkins, you need to create a credentials entry of type Username with Password and not of type Secret Text. Secret Text is used when you are not required to provide a username. But to access GitLab/GitHub repositories, you need to provide a username. If you want to access using API token, you need to use the username and API token in place of your login password.

---

Click the Add button to add this credentials entry. Click the Credentials field dropdown to select the entry.

Enter the branch name master in the Branch Specifier field, as you are using master as the main Git repository branch.

Figure 16-4 shows all these fields from the Source Code Management section filled with the details we just explained.



**Figure 16-4.** The Repository URL, Credentials, and Branches to builds fields filled

Scroll down the page to add a build step. Click the Add Build Step dropdown and select the Invoke top-level Maven targets option.

Select the MyMaven option in the Maven Version dropdown. You will see a name that you configured the Maven settings with in Global Tools Configuration page. I have my Maven configured with the MyMaven name. Enter Deploy in the Goals field. Click the Save button.

## Step 3: Change the Version in pom.xml of the Cloned API Project

Listing 16-3 shows this change in pom.xml. I changed the version to 8.0.

**Listing 16-3.** Version Changed to 8.0 in pom.xml

```
<groupId>Pranodayd</groupId>
<artifactId>CalculatorAPI</artifactId>
<version>8.0</version>
```

## Step 4: Commit Changes in the Local Repository and Push them to the Central GitLab Repository

Open the command prompt and enter into the project directory using the cd command. Now add changes to the staging area by running the git add command.

Commit the changes in the master branch by running this command:

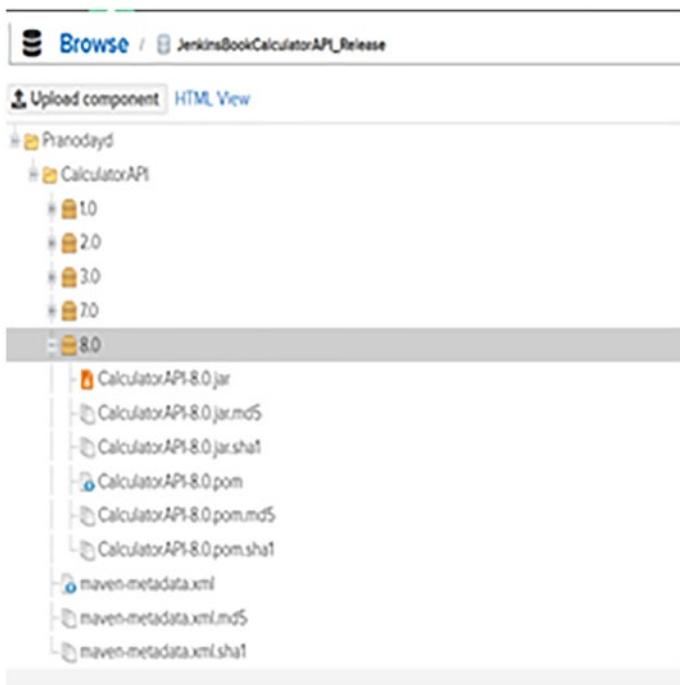
```
git commit --m"Changing version in pom.xml"
```

Push the change in the Git central repository using the following command:

```
git push
https://gitlab.com/Pranoday/jenkinsbookcalculatorapi.git master
```

# Running Free-Style Jobs Accessing the GitLab Repository with the API Token

There are no changes to the way you run a Jenkins job, which accessed the Git repository using the API token. To run a job, go to the dashboard and click the clock sign, which will start build. You will get the console output of this job. Figure 16-5 shows that the CalculatorAPI8.0 JAR file has been released to the Nexus repository.



**Figure 16-5.** The CalculatorAPI-8.0.jar has been released to the Nexus repository

## Understanding Pipeline as Code (*Jenkinsfile*)

Now that you are familiar with the Jenkins pipeline concept, you are going to learn how a pipeline is written outside the Jenkins UI in a file called a *Jenkinsfile*. This is called as *pipeline as code*.

### What Is a Jenkinsfile

A Jenkinsfile is a text file that contains the pipeline script code. It gets checked into the source code repository of an application along with the application's source code. This file can be implemented using scripted syntax (Groovy script) or descriptive syntax (DSL).

### Advantages of Pipeline as Code

There are several advantages of writing pipelines in Jenkinsfiles and overwriting it in the Jenkins UI:

- You can create pipelines for all branches and execute pull requests with just one Jenkinsfile.
- You can review your code in a Jenkinsfile the way you exercise code review of your application's source code.
- Writing pipeline code in a separate file and checking it in a version control system like SVN, Git etc. will allow you to keep track of periodic changes done in the build process.
- Jenkinsfile serves as a single source of pipeline code that can be shared among multiple developers.
- If an application developer has carried out a change in application source code which needs to be implemented in the build process, that same developer can implement this build process change in the Jenkinsfile.

## Writing a Pipeline in a Jenkinsfile

In this section you are going to learn how to write a pipeline in a Jenkinsfile. The pipeline in a Jenkinsfile can be written with both scripted or declarative syntax. You learned in the previous chapters how to write Jenkins pipeline using scripted syntax, you this chapter implements a Jenkinsfile using declarative syntax.

### Scripted vs Declarative Syntax

Before you start implementing a pipeline in declarative syntax, let's look at differences between scripted and declarative syntax:

- Declarative syntax is a relatively new feature in Jenkins.
- A scripted pipeline is a traditional way of writing pipeline code.
- Scripted pipeline uses stricter Groovy based syntaxes.
- Declarative pipelines are introduced to offer a simpler and more optioned Groovy syntax.
- Scripted pipelines are defined in a block called a node.
- Declarative pipelines are defined in a block called a pipeline.

The structure of scripted and declarative syntaxes are discussed in Chapter 14.

### Creating a Jenkins Job to Run a Jenkinsfile

Let's create a pipeline job that will run the build process of the CalculatorAPI. The difference between this job and the jobs you created previously for the same purpose is that you are going to write a pipeline in declarative syntax. The pipeline is going to be written in a Jenkinsfile and not in the Jenkins UI. You need to install the Pipeline plugin.

Step 1: Creating a pipeline job: Click the New Item link from the Jenkins dashboard to create a pipeline job.

I called mine DeployCalculatorAPIUsingPipelineAsACode and selected the Pipeline option. Click the OK button.

Scroll down page to find the Pipeline section. You can create the pipeline script using a Script Editor which you can copy in Jenkinsfile afterward.

The pipeline you are now going to build requires interaction with Git and Maven, so start by getting the corresponding script template by selecting the GitHub+Maven option from the dropdown.

Selecting this option will give you a declarative script template which you can edit according to your requirements.

I defined an environment block (see Listing 16-4) that's used to define variables which you want to share in all stages of this pipeline.

#### ***Listing 16-4.*** Environment Section from the Pipeline

```
pipeline
{
    environment
    {
        FAILED_Stage=""
    }
}
```

The first step is to clone the CalculatorAPI repository. Let's get the step generated from Snippet Generator. This process is explained in detail in Chapter 14.

I changed the template to add the Cloning Calculator API repository stage (see Listing 16-5) and pasted the Git step generated using the Snippet Generator.

This stage also includes a script block which assigns the pipeline environment variable FAILED\_Stage with the name of the current stage. The current stage name is returned by the Jenkins environment variable STAGE\_NAME.

***Listing 16-5.*** The Cloning Calculator API repository Stage in the Pipeline

```
Stage('Cloning Calculator API repository')
{
    steps
    {
        script
        {
            FAILED_Stage=env.STAGE_NAME
        }
        git branch : 'Master', credentialsId: 'MyGitlabAPIToken',
        url: 'https://gitlab.com/Pranoday/jenkinsbook
calculatorapi.git'
    }
}
```

Add another stage (called Deploying CalculatorAPI) (see Listing 16-6) with the batch command mvn deploy. This stage also saves its name in the Pipeline environment variable.

***Listing 16-6.*** The Deploying CalculatorAPI Stage in the Pipeline

```
Stage('Deploying CalculatorAPI')
{
    steps
    {
        script
```

```
{  
    FAILED_Stage=env.STAGE_NAME  
}  
bat 'mvn deploy'  
}  
}
```

The requirement is to send email notification at the end if the process fails. If process is successful, then you want to archive the artifact.

This can be achieved using a post-block in a declarative pipeline. Let's look at the post-block before you use it in the pipeline.

A post-block contains the steps to be performed after the completion of all stages or of a particular stage depending on its position in the pipeline script. A post-block contains different condition blocks, such as success, failure, and always. These condition blocks allow the execution of steps inside each condition, depending on the completion status of the entire pipeline or of a particular stage.

The following code includes a post-block written after the stage block.

```
stage('Display message on console')  
{  
    bat 'echo Hi'  
}  
post  
{  
    success  
    {  
        bat 'echo Display message stage is successful'  
    }  
}
```

```
failure
{
    bat 'echo Display message stage is successful'
}
}
```

In this example, you have the Display Message on Console stage, which displays “Hi” on the console. After completion of this stage, the success block or the failure block will be executed based on the completion status of the stage. If no error occurred, it completes successfully and the success block is executed; otherwise, the failure block is executed.

The following example shows where a post-block is mentioned after all stages.

```
stages
{
    stage('Display welcome message on console')
    {
        bat 'echo Hi'
    }
    stage('Display good bye message on console')
    {
        bat 'echo Good Bye'
    }
}
post
{
    success
    {
        bat 'echo Both stages are successful'
    }
}
```

```

failure
{
    bat 'echo some stage is not successful'
}
}

```

In this example, you have two stages mentioned inside the stages block and the post-block is mentioned after that. This post-block will be executed after completion of the last stage mentioned inside the stages block. If both the stages are successful, the success block will be executed. If one of the stages fails, the failure block will be executed.

```

stages
{
    stage('Display welcome message on console')
    {
        bat 'echo Hi'
    }
    stage('Display good bye message on console')
    {
        bat 'echo Good Bye'
    }
}
post
{
    always
    {
        bat 'echo It will always get executed'
    }
}

```

If the post-block uses the always block, it executes irrespective of the completion status of the stages.

Now that you understand the purpose of the post-block, you can resume implementation of the pipeline. In the pipeline, you want to archive an artifact generated if both stages pass. Otherwise, you want to send a failure notification through email with the name of the failed stage and the console log of the build attached.

Let's implement this using a post-block:

```
post
{
    success
    {
        archiveArtifacts 'target/*.jar'
    }

    failure
    {
        emailext attachLog: true, body: 'Stage: '+FAILED_
        Stage+' from Build : $BUILD_NUMBER of $JOB_NAME
        failed.Hence release of new build could not be
        done on Nexus repository.Please find detailed
        console log attached with this email.', subject:
        'CalculatorAPI details:$DEFAULT_SUBJECT', to:
        'pranoday.dingare@gmail.com'
    }
}
```

I added this post-block after the stages block of the pipeline so that it would be executed after the Deploying Calculator API stage completed.

In the emailext step, you are setting attachLog: true to send the console log an email. The rest of the parameters of the emailext step you are already familiar with. Note how we used the pipeline environment variable in the body parameter to include the failed stage name in the email body. This would be executed if one of the stages failed to complete.

In the success condition block is the archiveArtifacts step. Let's look at this step in more detail.

Build artifacts are usually created inside the workspace, which might get deleted once you execute subsequent builds. But if you want to preserve artifacts of the build, these need to be copied outside the workspace. The archiveArtifacts step saves these build artifacts in the \${JENKINS\_HOME} directory. The second stage has an mvn deploy batch step, which creates a .JAR file that you want to preserve outside the workspace directory.

## Saving the Pipeline Code in a Jenkinsfile and Pushing it to the GitLab Repository

You have created the pipeline script in the Jenkins UI, but you want this to be part of the GitLab repository. Let's create a file named `Jenkinsfile` and paste this script inside it. I created a `Jenkinsfile.txt` file inside a directory where I cloned the remote repository.

Inside this file, I pasted the pipeline script we created using the Jenkins UI. See Listing 16-7.

**Listing 16-7.** The Entire Pipeline Code Created Using the Jenkins UI

```
Pipeline
{
    environment
    {
        FAILED_Stage=' '
    }
    agent any
```

```
stages
{
    stage('Cloning Calculator API repository')
    {
        steps
        {
            script
            {
                FAILED_Stage=env.STAGE_NAME
            }

            // Get the code from Gitlab repository
            git branch: 'Master', credentialsId:
            'MyGitlabAPIToken', url: 'https://gitlab.com/
            Pranoday/jenkinsbookcalculatorapi.git'
        }
    }

    stage('Deploying Calculator API')
    {
        steps
        {
            script
            {
                FAILED_Stage=env.STAGE_NAME
            }
            bat 'mvn deploy'
        }
    }
}
```

```
post
{
    success
    {
        archiveArtifacts 'target/*.jar'
    }

    failure
    {

        emailext attachLog: true,body: 'Stage: '+FAILED_
        Stage+' from Build : $BUILD_NUMBER of $JOB_NAME
        failed.Hence release of new build could not be
        done on Nexus repository.Please find detailed
        console log attached with this email.', subject:
        'CalculatorAPI details:$DEFAULT SUBJECT', to:
        'pranoday.dingare@gmail.com'

    }
}

}
```

I committed this file to the local repository using the following commands:

```
git add .
git commit --m "Adding Jenkinsfile"
```

and then pushed it to the GitLab repository using the following command:

```
git push  
https://gitlab.com/Pranoday/jenkinsbookcalculatorapi.git
```

You can see the `Jenkinsfile.txt` inside your GitLab repository now. Let's go inside the Jenkins job.

Click the Configure link. Scroll down to the Pipeline section. Open the Definition dropdown. Click the Pipeline Script from SCM option. Open the SCM dropdown and select the Git option. Enter the URL of the JenkinsBookCalculatorAPI repository in the Repository URL field and select the credentials entry in the Credentials dropdown. Add master to the Branch Specifier field.

In the Script Path field, you need to add the relative path of the pipeline file. In this case, the pipeline is written in a file named `Jenkinsfile.txt` and it is present inside the repository's root folder, so you only have to include `Jenkinsfile.txt` in the Script Path field. Click the Save button.

## Triggering a Jenkins Job Using a GitLab Webhook

You have learned how to trigger jobs using different types of triggers like Chron expressions and polling SCMs. In this section, you are going to learn about a new mechanism to trigger a Jenkins job. When you configure a Jenkins job to poll SCM, then Jenkins checks for a new change in the corresponding Git repository and triggers the build. In this case, the build trigger comes from the Jenkins side and it only gives a trigger when there is a new change pushed to the remote repository.

Whenever you interact with the Git remote repository by adding new comments, pushing code branches, or opening/closing an issue, it triggers different events. If you want to trigger the Jenkins job as a result of such events, you need to use a concept called *Webhooks*.

Webhooks are nothing but configurations that contain URLs to send a POST request to as a result of some event. If you create Webhook configuration with a URL of your Jenkins job, you can trigger a build based on an event that occurred on the GitLab repository. To get your Jenkins job triggered using Webhooks, you need to create a Webhook in the GitLab repository and configure Jenkins to get a build trigger from GitLab. Before you configure a Webhook in GitLab, let's go over two important concepts:

- **Public IP:** A machine connected to a network has two types of IPs. A Local IP is used in a local network (LAN) to communicate with a machine. A public IP is used by machines outside of the local network to communicate with a machine. Gitlab.com is not inside a LAN so it needs a public IP of the Jenkins machine to trigger a job.

A public IP of a machine can be determined using this website:

<https://www.whatismyip.com/what-is-my-public-ip-address/>

You can go to this website and it will show you public IPv4 address of your machine. Go to this website from the machine that has the Jenkins server.

You have to use this IP while configuring a Webhook URL in GitLab.

- **Port forwarding (opening ports):** In this scenario, the Jenkins server is going to be started on the local (LAN) IP address and going to be listening to some port,. You have started Jenkins on port 8080. Gitlab.com is going to send a request to the Jenkins machine using an URL with a public IP of your machine, right? This request should be forwarded to a local IP address where Jenkins is started. This concept of forwarding requests sent to a public IP to a LAN IP addresses is called *port forwarding* or *opening a port*.

The following steps explain how to configure port forwarding using a DLINK router. In your case, navigation to the pages will be different, but conceptually the process is the same.

1. Go to the D-link router configuration page: Open <http://192.168.0.1>.
2. Enter admin for the username and password.  
Click the OK button, which will open the D-link configuration page.
3. Click the Advanced button, which will open the next page.
4. Click the FIREWALL button shown on the top horizontal bar, which will open the next page.
5. Click the PORT FORWARDING link shown on the left side.

**Local IP Address:** Enter your machine's local IP address (which you get after running Ipconfig command) in the Local IP Address field. In my case, it is 192.168.0.100.

**Local Port Range:** You need to specify a port range on which you have started the Jenkins server. If you want to start Jenkins on any port between 8080 to 8090, you can add this port range. As I started my Jenkins server on 8080, I enter 8080 in both fields.

**Protocol:** Keep the Both value selected in this dropdown.

**Remote IP Address:** In this field, enter your machine's public IPv4 address, which you have determine from whatismyip.com website. My machine's public IP is 1.23.253.174.

**Remote Port Range:** You can add a range of ports here as well. But I am entering 8080 in both the fields.

6. Click the Save& Apply button to save the configuration.

Once you are done with the settings mentioned here, you're ready to create a Jenkins job to be triggered using a GitLab Webhook URL.

Step 1: Start the Jenkins Server on a local IP address: You need to start the Jenkins server on your local IP.

Step 2: Configure the Jenkins URL in the Jenkins configuration: Go to Manage Jenkins ➤ Configure System and set the machine's public URL in the Jenkins URL field.

Click the Save button.

Step 3: Install the GitLab plugin in Jenkins: You need to install the GitLab plugin in Jenkins. Refer to Chapter 5 for detailed steps on how to do that.

Step 4: Configure the Jenkins job to get a trigger from GitLab: Before you go to the GitLab repository to create a Webhook, go inside the DeployCalculatorAPIUsingPipelineAsACode Jenkins job from the dashboard and configure it to receive a trigger from GitLab.

Scroll down to the Build Triggers section. After installing the GitLab plugin, you will see the Build When a Change Is Pushed to GitLab option.

Copy the GitLab Webhook URL from the Build When Change Is Pushed to GitLab option. Check the Build When Change Is Pushed To GitLab checkbox.

Check the Push Events checkbox and uncheck the others, as you want to trigger this job once. You will push changes to the remote repository. Click the Advanced button. This will open other settings related to this option. Scroll down to get to the Secret Token field.

Click the Generate button. This will generate a secret token for this job which will authenticate the access of this job from GitLab when it triggers it.

Copy this token and paste it in a file before you save the configurations. Click the Save button.

Step 5: Creating a Webhook in GitLab: Go to GitLab repository you want to trigger the Jenkins job from, to create a Webhook.

In this example, you trigger it on push of the code inside the JenkinsBookCalculatorAPI repository.

Got to the JenkinsBookCalculatorAPI repository. Choose the Menu ➤ Settings ➤ Webhooks menu. It will take you to the Webhooks page.

Enter the project URL in the URL field and the project secret token into the Secret token field.

Since you want to trigger a job on push to the “master” branch, check the Push Events checkbox in the Trigger section and enter master in the Edit field shown below the Push Events checkbox.

Scroll down the page and click the Add Webhook button. This will save the Webhook.

Step 6: Changing the pom.xml of the CalculatorAPI project and pushing the change: I changed the version to 9.0 (see Listing 16-8).

***Listing 16-8.*** The Version Changed to 9.0 in pom.xml

```
<groupId>Pranodayd</groupId>
<artifactId>CalculatorAPI</artifactId>
<version>9.0</version>
```

Use `git add` and `git commit` and then push the changes to the master branch using the `git push` command. If you do all this, you’ll see that the build has been triggered.

## Creating a Pipeline Job to Trigger Using GitHub Webhooks

In this section, you are going to create a Webhook in GitHub the way you created it in GitLab. You are going to work with different settings to trigger a Jenkins job using a GitHub Webhook. Let’s get started by following these steps.

Step 1: Creating the JenkinsBookCalculatorAPI repository on GitHub: I created a private repository on GitHub called JenkinsBookCalculatorAPI and pushed the CalculatorAPI project to it.

This is the same CalculatorAPI project you have been using throughout this book. In the previous section, you created the `Jenkinsfile.txt` file in the project directory and pushed it to the GitLab repository. I made only one change before pushing it to the GitHub repository; instead of keeping the pipeline script in the `Jenkinsfile.txt` file, inside project root directory, I named the Pipeline script file `BuildCalculatorAPI.txt` and shifted it inside the `BuildScript` directory.

Step 2: Installing the GitHub plugin in Jenkins: You need to install the GitHub plugin to get the configuration options in the Jenkins JOB. Refer to Chapter 5 for detailed steps on how to install plugins in Jenkins.

Step 3: Setting a Hook URL in the Jenkins configuration: Go to Manage Jenkins ➤ Configure System. Click the Advanced button under the GitHub section.

Check the Specify Another Hook URL for GitHub Configuration checkbox and add the public IP of the machine.

Click the Save button.

Step 4: Creating a pipeline job: Click the New Item link from the Jenkins dashboard to create a pipeline job.

I called it `DeployCalculatorAPIUsingPipelineAsACodeFromGitHub` and selected the Pipeline option. Click the OK button

Go to the Build Triggers section and check GitHub Hook Trigger for GITScm Polling. This option is only available when you have the GitHub plugin installed.

In the Pipeline section, select the Pipeline Script from SCM option in the Definition field. Select the Git option in the SCM dropdown. Enter the HTTP URL of your recently created GitHub repository in the Repository URL field. I created a new credentials entry of type UserName with Password with the GitHub API token. Select this credentials entry in the Credentials dropdown. Enter the master in the Branch Specifier field.

In the Script Path field, add BuildScript/BuildCalculatorAPI.txt as the pipeline is written in a file named BuildCalculatorAPI.txt which is inside the BuildScript folder from the project's root directory. Click the Save button.

Step 5: Creating a Webhook in the GitHub repository: Go to the JenkinsBookCalculatorAPI repository page. Click the Settings tab. Click the Webhooks menu on the left side, which will open the Webhooks page.

Click the Add Webhook button. Enter this URL in the following format inside the Payload URL field:

```
http://<Public IP of Jenkins server machine>:<port>  
/github-webhook/
```

My Jenkins is started on the 8080 port on the public IP machine with an 1.23.253.174 public address, so I entered <http://1.23.253.174:8080/github-webhook/>.

Select the Just the Push Event radio control and then click the Add Webhook button.

## Triggering Pipeline Jobs Using GitHub Webhook

Let's change the version to 10.0 in the pom.xml file (see Listing 16-9).

***Listing 16-9.*** The Version Changed to 10.0 in pom.xml

```
<groupId>Pranodayd</groupId>  
<artifactId>CalculatorAPI</artifactId>  
<version>9.0</version>
```

Commit the change in the local repository using the git add and git commit commands. Once these changes are committed, push the changes to the master branch in the GitHub repository using the git push command, which will trigger the build on a push event by the GitHub Webhook.

## Summary

In this chapter, you learned about API authentication and its advantages over using a password when logging into the GitHub/GitLab repositories. You also learned how to generate these API tokens in GitHub and GitLab platforms. You used these generated tokens while working with GitHub/GitLab repositories using git commands. You also saw how to generate a credentials entry containing these API tokens in Jenkins. After a detailed understanding of API tokens, you learned how to write pipeline in a code file (not in the Jenkins UI) and execute these pipelines which are pushed into the SCM along with source code of your application. At the end of the chapter, you learned about Webhooks and learned how to trigger Jenkins jobs on a push event using Webhooks configured in GitHub and GitLab. The next chapter covers the Jenkins distributed builds.

## CHAPTER 17

# Jenkins Distributed Builds

You have learned how to implement various Jenkins jobs as free-style jobs and as pipeline jobs throughout this book. You have seen execution of these jobs by triggering builds, either manually or through some external trigger like Webhooks, SCM polling, etc. To execute these jobs, you used one Jenkins machine that had all the necessary hardware/software requirements. This kind of setup does not help in building a huge application with multiple time-consuming phases. Sequential execution of huge build phases will take a lot of time and ultimately will delay the release of the build. Frequent releases of such time-consuming builds are difficult to trigger and manage too. Another important problem is that you need to have all the necessary software installed on that single machine, which may not be possible due to hardware constraints. If that build machine breaks down, then replicating all the installations on a new build server would be a huge task. Modern applications demand testing to be done on different hardware/software combinations, which will not be possible if you have only one machine taking care of the entire build process.

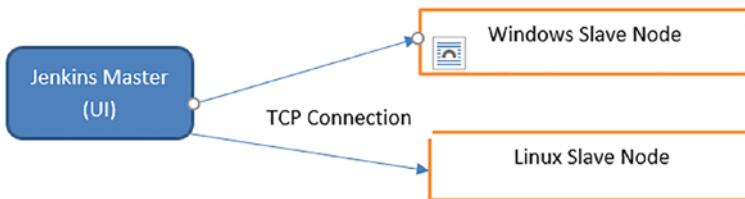
To solve all these problems, Jenkins comes with a great feature called *distributed builds*. This chapter explains how Jenkins distributed builds work. It discusses how to set up Jenkins to run distributed builds and how to set up free-style and pipeline jobs to run distributed builds.

## Jenkins Distributed Architecture

Jenkins uses a master-slave architecture to manage distributed builds. Let's look at the master-slave concept:

**Jenkins master:** This is a machine with Jenkins installed on it. This Jenkins instance sends different phases of the build to different machines for execution. This distribution will seldom be based on hardware/software requirements that a particular build phase needs.

**Jenkins slave:** Also called a Jenkins agent, this is another machine with network access to the Jenkins Master. This machine has the Jenkins agent installed and is responsible for executing the build phases assigned by the Jenkins Master. One or more Jenkins slaves can be responsible for executing multiple build phases in parallel and complete the entire build process. An architectural diagram of a Jenkins master-slave is shown in Figure 17-1.



**Figure 17-1.** The Jenkins master-slave architecture

## Ways to Connect the Master and Slaves

In order to execute the distributed build, the master and slaves need to be connected. There are two ways to do this:

**Master to agent:** In this way, the agents (slaves) are configured to get connection requests from the master. In this configuration, the slave machines need to be minimally configured. They just need to have JDK/JRE installed. The master will connect with the agent machine through a SSH port, copy the `remoting.jar` on the agent machine, and will start running it using the JDK/JRE available on that machine. Through this `remoting.jar` agent, the machine will run the Jenkins job. In this case, the master should be able to send requests to the slave machine.

**Agent to master:** If the master does not have access to the agent machines, it will not be able to start the agent process. In this case we need to use a different type of agent configuration, called JNLP (Java Network Launch Protocol). With this approach, you need to set the Fixed or Random radio control for the TCP Port for Inbound Agents option on the Manage Jenkins ➤ Global Security page on the Jenkins master machine.

If you select the Fixed option, you need to set a port in the field. This is the port that agents use to connect to the master through JNLP. I am setting this port to 7070. Click the Save button.

Once this is set, you need to open an agent page inside a browser on the agent machine. This will show a JNLP Launch icon. Click that icon to install the JNLP agent and start the slave machine.

Once the JNLP agent is installed, you can install it as a Windows service so that you do not need to start the agent interactively again.

This is convenient when the master cannot connect to the agents because it's outside of the firewall, for example.

## Understanding the Configuration to Connect the Master to the Agent Using SSH

This section goes through the step-by-step process to configure the master-slave machines to start the Jenkins agent using an SSH connection. I use two different machines to explain this configuration.

- **Master:** the IP address of this machine is 192.168.43.10. This machine has a full Jenkins installation.
- **Slave:** the IP address of this machine is 192.168.43.185. This machine does not need a Jenkins installation. It has JDK11 installed.

### Step 1: Install the SSH Build Agent's Plugin

Log into Jenkins on the master machine and install the SSH Build Agents plugin using the Plugin Manager.

### Step 2: Install Java on the Slave Node

Install Java on the slave machine if it is not already installed. The Java installation steps are outside the scope of this book.

## Step 3: Create an SSH Public-Private Key Pair

On the master machine, create a public-private key pair using the `ssh-keygen` command.

I created a key pair in the default path  `${CURRENT_USER}\.ssh`. I did not specify a passphrase for the key. You can create a key pair at a different location and may provide a passphrase too.

## Step 4: Add a Credentials Entry with a Private Key to the Master Machine

Go to the Jenkins dashboard on the master machine. Then go to the Manage Jenkins ➤ Manage Credentials menu. The Credentials page will appear.

Click the Jenkins link shown in the Stores Scoped to Jenkins section, which will open the System page.

Click the Global Credentials (Unrestricted) link to open the Global Credentials (Unrestricted) page. Click the Add Credentials menu shown on the left side.

Select the SSH Username with Private Key option from dropdown. Enter the username in the Username field

Note that the Username field should have a username of a authenticated user from the slave machine. The Jenkins master will connect to the slave using the username mentioned in this field. The ID and Description fields can have any value.

Click the Enter Directly radio button. I entered ADMIN into the Username field here. (Remember this name, as you need in future steps.)

Click the Add button and paste the created Private Key value copied from the  `${CURRENT_USER}\.ssh\id_rsa` file inside the field.

I keep the Passphrase field blank, as I did not specify a passphrase when creating the key pair. If you added a passphrase when you created the key pair, you need to include it here too.

Click the OK button.

## Step 5: Add a Node Entry to the Jenkins Master

On the Jenkins master, click the Manage Jenkins menu. Click the Manage Nodes and Clouds link. This will open a page on which you can create new nodes.

Click the New Node link available on the left side. The next page will open, which is where you add the node details.

Enter a name in the Node Name field; I entered Node1. Click the Permanent Agent radio button.

Click the OK button. This will open page on which you can specify other details of the node.

Let's review all the parameters on this page before you fill them in:

- **Name:** The name of the slave, which should be unique.
- **Description:** This field is optional, but if mentioned, it can be helpful for other team members.
- **# of executors:** The maximum number of concurrent builds that Jenkins may perform on this agent. I used one executor for testing purposes. You can check the server stat and then define the number of executors.
- **Remote root directory:** An agent needs to have a directory dedicated to Jenkins. Specify the path to this directory on the agent. It is best to use an absolute path, such as c:\jenkins\node1. This should be a path local to the agent machine. There is no need for this path to be visible from the master. The Jenkins agent will create a workspace in the directory mentioned in this field.

- **Labels:** Labels (or tags) are used to group multiple agents into one logical group. Multiple labels must be separated by a space.
- For example, the `linux docker` would assign two labels to the agents `linux` and `docker`.
- **Usage:** This controls how Jenkins schedules builds on this node. Utilize this node as much as possible. This is the default and normal setting. In this mode, Jenkins uses this node freely.
- **Launch method:** This controls how Jenkins starts this agent. The following three options are available in this dropdown:
  - Launch agent by connecting to the master
  - Launch agent via execution of the command on the controller
  - Launch agents via SSH
- **Launch agent via Java Web Start:** This allows the slave to be launched using [Java Web Start](#). In this case, a JNLP file must be opened on the agent machine, which will establish a TCP connection to the Jenkins master. This means that the agent need not be reachable from the master; the agent just needs to be able to reach the master. If you have enabled security via the Configure Global Security page, you can customize the port on which the Jenkins master will listen for incoming JNLP agent connections.

- By default, the JNLP agent will launch a GUI, but it's also possible to run a JNLP agent without a GUI, e.g., as a Window service. The next section of this chapter includes an example and detailed steps.
- **Launch the agent via execution of the command on the master:** This starts the slave by having Jenkins execute a command from the master. Use this when the master is capable of remotely executing a process on another machine, e.g. via SSH or RSH. Usually, a utility like Psexec.exe can be used to start the agent on a remote machine.
- **Launch the slave agents via SSH:** This starts a slave by sending commands over a secure SSH connection. The slave needs to be reachable from the master, and you have to supply an account that can log into the target machine. For this, no root privileges are required. This is the one that I am using for my slave configuration.
- If you select this option in the Launch Method dropdown, you get two additional fields on the form—Credentials and Host Key Verification Strategy.
  - **Credentials:** In the Credentials dropdown, you need to select the credentials entry with the authentication information to be used to authenticate the user on the agent node.
  - **Host Key Verification Strategy:** This controls how Jenkins verifies the SSH key presented by the remote host while connecting. This dropdown has the following options:
    - Known hosts file verification strategy

- Manually provided key verification strategy
- Manually trusted key verification strategy
- Non verifying verification strategy
- **Known hosts file verification strategy:** Checks the `known_hosts` file (`~/.ssh/known_hosts`) for the user Jenkins is executing, to see if an entry exists that matches the current connection. If you get a SSH Host Key Verification error as shown here:

No entry currently exists in the Known Hosts file for this host. Connections will be denied until this new host and its associated key is added to the Known Hosts file.

Key exchange was not finished, connection is closed.

`java.io.IOException: There was a problem while connecting to node2.scmquest.com`

It could be a problem with the SSH lib used by Jenkins, which does not support newer ciphers like `ecdsa-sha2-nistp256`. Just delete the `known_hosts` entry and create a new one using the following command:

`ssh -o HostKeyAlgorithms=ssh-rsa node2.scmquest.com` (where `node2.scmquest.com` is the hostname of the slave server)

This will solve your problem.

- **Manually provided key verification strategy:** This ensures that the key provided by the remote host matches the key set by the user who configured this connection.
- **Manually trusted key verification strategy:** Requires a user with `Computer.CONFIGURE` permission to authorize the key presented during the first connection to this host before the connection is allowed to be established.

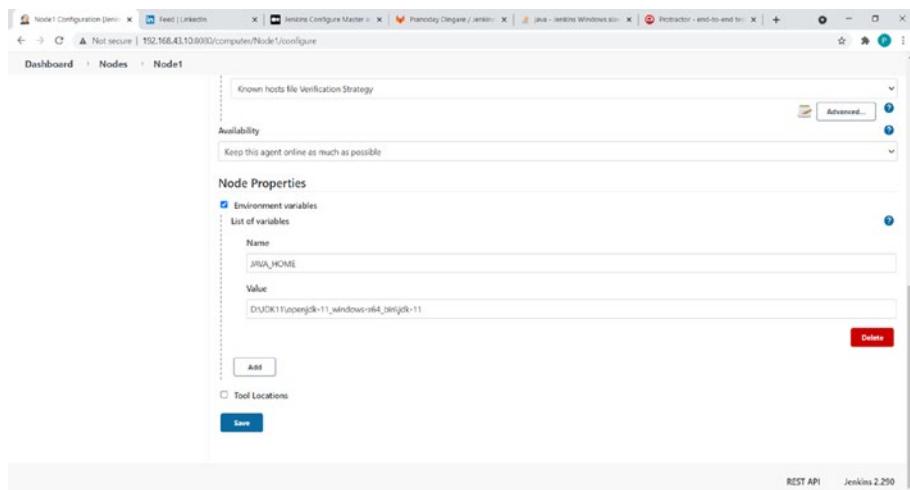
- **Non verifying verification strategy:** Does not perform a verification of the SSH key presented by the remote host, allowing all connections regardless of the key they present. It's not advisable to select this, as it may open the path for attackers.
- **Let Jenkins control this Windows slave as a Windows service:** This starts a Windows slave by [a remote management facility](#) built into Windows. This is suitable for managing Windows slaves. Slaves need to be IP reachable from the master.
- **Availability:** Controls when Jenkins starts and stops this agent.
- **Keep this slave online as much as possible:** This is the default setting. In this mode, Jenkins tries to keep the slave online as much as possible. If Jenkins can start the slave without user assistance, it will periodically attempt to restart the slave if it is unavailable. Jenkins will not take the slave offline.
- **Take this slave online when in demand and offline when idle:** In this mode, if Jenkins can launch the slave without user assistance, it will periodically attempt to do so while there are unexecuted jobs; otherwise, the slave will be taken offline by Jenkins.

You can now fill in these fields to create the node entry:

1. In the Name field, I entered Node1.
2. In the Number Of executors field, I have used value 1 as I do not want to run more than one builds on a single agent.

3. In the **Remote Root Directory**, I have used C:\JenkinsJobExecution. Once you run the job in this directory, the workspace of the job will be created. I used the directory from C: because this is my SystemRoot and once the master connects to the slave it will enter the user directory from SystemRoot (C: in my case). If you include a directory from a different root, such as D:\, Jenkins will throw an error while running the job.
4. In the Labels field, I entered CalculatorAPI\_Node.
5. In the Usage field, I kept the Use This Node As Much As Possible value selected.
6. In the Launch method, I selected the Launch Agents via SSH value.
7. In the Host field, I used 192.168.43.185, which is the IPv4 address of my slave node.
8. In the Credentials field, I selected the credentials entry we created in this chapter.
9. In the Host Key Verification Strategy field, I selected the Known Hosts File Verification Strategy value.
10. I selected the Environment Variables checkbox under the Node Properties section and created the JAVA\_HOME environment variable by entering JAVA\_HOME into the Name field. The Value field contains the path of the JDK on the slave machine. You need this because the job you are going to run on the slave runs mvn commands and Maven needs the JAVA\_HOME environment variable pointing to the JDK (see Figure 17-2).

## CHAPTER 17 JENKINS DISTRIBUTED BUILDS



**Figure 17-2.** The JAVA\_HOME environment variable in Node Properties

11. Click the Save button to save the node configuration.

After you perform these steps on the master machine, you can move to the slave/agent machine for further steps.

## Step 6: Add a Public Key to the authorized\_keys File on the Slave Machine

Let's go to the {CURRENT\_USER} directory on the agent machine. In my case, this directory is C:\Users\ADMIN. Create a directory called .ssh. Right-click to get the context menu. Click the Git Bash Here menu option, which will open the Git bash window

Create a file named authorized\_keys by running the following bash command in Git bash:

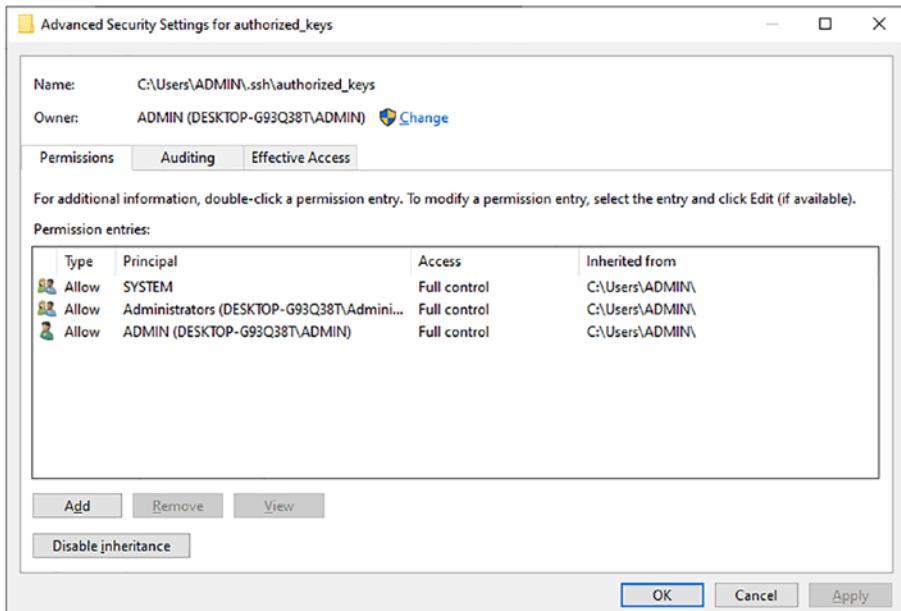
```
touch authorized_keys
```

This command will create a file named authorized\_keys in the .ssh folder under the \${CURRENT\_USER} directory. Open this file and paste the public key from the key pair you generated on the master machine.

## Step 7: Change the Permissions of the authorized\_keys File on the Slave Machine

Right-click the authorized\_keys file. Select the Properties menu option, which will open the Properties window.

Click the Security tab. Then click the Advanced button. This will open the Advanced Security Settings for authorized\_keys window (see Figure 17-3).

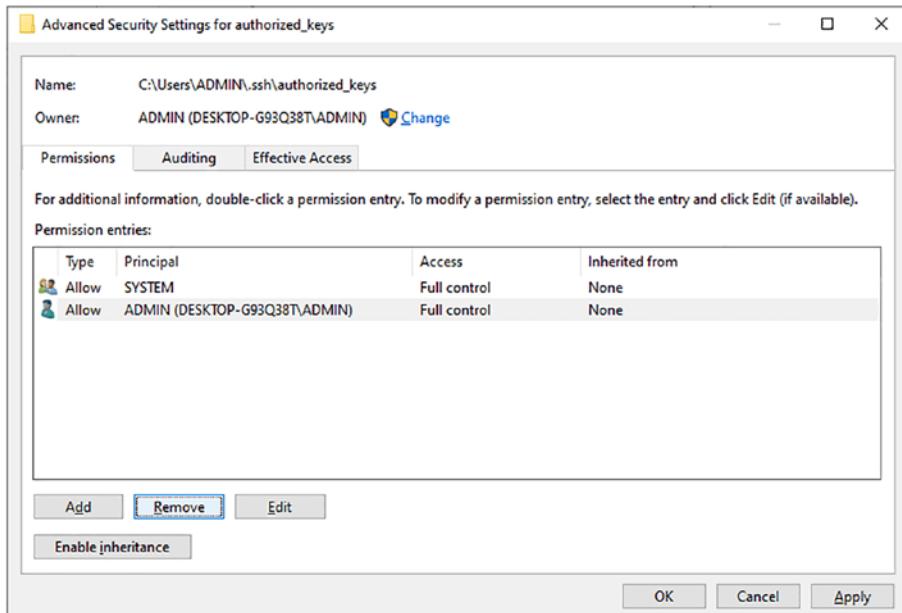


**Figure 17-3.** The advanced security settings of the authorized\_keys file

Click the Disable Inheritance button. Then select the Convert Inherited Permissions into Explicit Permissions on this Object option (i.e., the first option).

Keep only the system and your current user in the list; delete any other users by clicking the Remove button while being parked on the user entry you want to delete.

Figure 17-4 shows that I have kept only two users: ADMIN (which is my current user and the one for which you created the credentials entry in Step 4) and SYSTEM. See Figure 17-4.



**Figure 17-4.** Only the ADMIN and SYSTEM user entries are left

Click the OK button on this window and the OK button on the authorized\_keys properties window.

## Step 8: Restart the sshd Service

Go to the services window and restart the service named OpenSSH SSH Server.

## Step 9: Check the Connection to the Agent Machine from the Master

Open the command prompt on master machine and run the following command:

**ssh UserName@IPAddress of Slave**

I executed the following command:

**ssh ADMIN@192.168.43.185**

After pressing the Enter key, you will get the prompt, Are you sure you want to continue connecting(yes/no/[fingerprint])?

Type yes and press Enter, after which the SSH connection will be successful. A file called known\_hosts will be created inside the \${CurrentUser}\.ssh folder. Open this file; it will have a key entry added for your agent machine.

## Step 10: Launch the New Node from the Master Machine

Click the Manage Jenkins ➤ Manage Nodes and Clouds menu option. Click the Node1 entry and then click the Relaunch Agent button. You can see that the master was successfully connected to the agent.

During this connection process, the master copies remoting.jar to the agent machine inside the folder that's set as the remote root directory.

## Creating a Free-Style Job to Run on the Node1 Agent

After successful configuration of the master and the agent using SSH, it's time to create a job to run on the node you just configured.

I created a free-style job that will deploy the CalculatorAPI JAR on the Nexus repository.

I checked the Restrict Where This Project Can Be Run checkbox and entered Node1 in the Label Expression field. This setting is responsible for running the job on a specified node. It has Git URL set in the Source Code Management section. The build step is set to mvn deploy. Click the Save button.

## Running the New Free-Style Job on the Node1 Agent

Before running a job on an agent connected using SSH, you need to launch an agent which you have already created. Run the job; it will be executed on the agent machine.

## Understanding the Configuration to Connect the Agent to the Master Using JNLP

### Step 1: Configure the Jenkins Master to Receive JNLP Agent Connection Requests

On the master machine, click the Manage Jenkins menu. Scroll down the page and click the Configure Global Security menu.

Scroll down the page again and, under the Agents section, select the Fixed radio button available for TCP Port for Inbound Agents option and enter 7070 as the port number.

You can use any available port number in this field. I use 7070 as it is available on my machine. Scroll down the page and check the Enable Agent ➤ Control Access Agent checkbox under the Agent ➤ Controller Security section. Click the Save button.

## Step 2: Set the Jenkins URL

Click the Manage Jenkins menu. Click the Configure System menu. Scroll down the page to see the Jenkins Location section.

Enter `http://<IP Address of Master machine:port>` in the Jenkins URL field. My Jenkins master starts on the IP address 192.168.43.10 and port 8080 so I entered `http://192.168.43.10:8080` here.

Note: Do not use `http://localhost` as you will not be able to access this master machine from the agent machine when the localhost is used.

Click the Save button.

## Step 3: Create a New Node Entry from the Agent (Slave) Machine

Open Jenkins on the slave machine using the Jenkins Master URL. Click the Manage Jenkins ➤ Manage Nodes and Clouds menu. Click the New Node menu entry available on the left side. Enter the name of the node in the Node Name field. I entered Node3. Click the OK button.

Here are the other fields and their entries:

- **Name:** Node3
- **Number of executors:** 1
- **Remote root directory:** C:\JNLPNode

- **Labels:** JNLPNode
- **Usage:** Use this node as much as possible

In the Launch Method dropdown, keep the Launch Agent by Connecting to Master option selected. See the Node Properties section in Figure 17-2. Click the Save button. Then click the agent.jar link highlighted in Figure 17-5 to download the agent.jar file.

The screenshot shows the Jenkins Node Properties page for 'Agent Node3'. In the 'Labels' section, 'JNLPNode' is listed. To the right, there's a 'Actions' column with a blue hyperlink labeled 'agent.jar'. The main content area provides instructions for connecting to Jenkins via Java Web Start or from the command line, including a specific URL and a secret key.

**Figure 17-5.** The agent.jar hyperlink

After downloading the agent.jar file, keep it at your desired location. I have kept it in the D:\Agentjar folder.

Select the command below the Run from Agent Command Line section. My command is as follows:

```
Java -jar agent.jar -jnlpUrl http://192.168.43.10:8080/
computer/node3/jenkins-agent.jnlp -secret
76f01ce9855280b9b743b9afc1cce4af908675f5a6ed229efd8fb4909aa
74b -workdir "C:\JNLPNode"
```

Copy it. Modify the path of the agent.jar accordingly. I kept my agent.jar file in the D:\AgentJar folder. The following is the command, modified for me:

```
Java -jar D:\AgentJar\agent.jar -jnlpUrl  
http://192.168.43.10:8080/computer/  
node3/jenkins-agent.jnlp -secret  
76f01ce9855280b9b743b9afc1cce4af908675f5a6ed229efd8fb4909aa74b  
-workdir "C:\JNLPNode"
```

After modifying the agent.jar path in the command, execute it from the command prompt.

Note that the value of the workdir parameter in this command has been taken from the Remote root Directory, field which you configured when configuring a node instance. The value of the secret parameter will be different for each node.

## Creating a Job to Run on the JNLP Node

I created a job called JNLPNodeJob, which will pull the code from the CalculatorAPI repository and will build the Jar file and deploy it on Nexus (same as the other job you created in this chapter). The only difference is that the Label Expression field has the Node3 value.

Run the job by clicking the clock sign from the dashboard. You can see that the job is getting executed on Node3.

## Summary

In this chapter, you learned what distributed builds are and their advantages. You also learned about the different ways to connect the Jenkins master and the Jenkins slave nodes. You learned detailed steps to connect master-slave using SSH and using JNLP. You also configured your Jenkins jobs to run on the slave nodes. The next chapter explains how to create an EC2 instance on the AWS platform and deploy the web application on it through the Jenkins pipeline.

## CHAPTER 18

# Integrating Jenkins with AWS

Nowadays applications are deployed on machines available on the cloud instead of from in-premise machines. An understanding of cloud platforms like AWS (Amazon Web Services) and Azure is vital to the success of the DevOps professional.

This chapter talks about using Jenkins to deploy a web application on a machine (EC2 Instance) available on an AWS cloud and then run the Selenium tests on this web application.

## Understanding an EC2 Instance on AWS

An EC2 instance is a virtual server in Amazon Web Services terminology. EC2 stands for *Elastic Compute Cloud*. It is a web service where an AWS subscriber can request and provision a compute server in the AWS cloud. Users can rent virtual server use per hour and use it to deploy software applications.

The instance will be charged per hour with different rates based on the type of instance chosen. You can create instances based on your CPU and memory requirements and use them as long as you need them. You can terminate the instance when it is no longer used and save on costs.

## Creating an EC2 Instance on AWS

Before you can involve Jenkins in deploying web applications on an EC2 instance, you need to create one EC2 instance on AWS. Use the steps in the following sections to do just that.

### Step 1: Sign Up on AWS

Go to the <https://aws.amazon.com/account/sign-up> link and click the Create a Free Account button, which will show the Sign Up for AWS page.

Fill in the required details and click the Continue (step 1 of 5) button.

Fill in the required details again. Click the Continue (step 2 of 5) button, which will show the Billing Information page. Enter the required payment details on the Billing Information page.

Select the Text Message radio button, enter your mobile number, and click Continue (step 4 of 5). After clicking this button, you will receive an SMS text on your mobile. Enter this code on the appropriate page.

Click the Continue (step 5 of 5) button, which will open the Select a Support Plan page. Click the Complete Sign Up button. Click the Go to AWS Management Console button. Select the Role and Interest dropdowns.

Finally, click the Submit button.

### Step 2: Sign in to AWS

Click the Sign in to the Console button. Enter an email address in the Root User Email Address field and click the Next button.

Enter the CAPTCHA in the field and click the Submit button.

Enter the password you selected during the signup process in the Password field.

Click the Sign in button. Click the All Services link to open all available services. Then click the EC2 link, which is the first option in the Compute section.

## Step 3: Create an EC2 Instance

Scroll down the page and click the Launch Instance dropdown; then click the Launch Instance option.

To create an EC2 instance with Windows OS, type **Windows** and press the Enter key, which will show the available EC2 instances with Windows OS.

While creating an EC2 instance, you have options like t2micro, t2medium, t2 large, etc., which differ in the kind of diskspace, processor speed, and RAM they offer. You need to select a type based on your diskspace, RAM, and processor speed requirements. Note that a few types of EC2 instances are free, like t2micro, but others are paid. In this chapter, select the first entry, which is free.

Click the Review and Launch button. Next, click the Launch button.

You will see the Select an Existing Key Pair or Create a New Key Pair page. From this page, you can create an authentication key pair to access the EC2 instance.

From the Choose an Existing Key Pair dropdown, select the Create a New Key Pair option. Enter a name in the Key Pair Name field. I entered MyEC2Instance in this field.

Click the Download Key pair button. The MyEC2Instance.pem file will be downloaded. Click the Launch Instances button. Then scroll down the page to find the View Instances button; click it.

After clicking that button, the Instances page will open.

## Step 4: Start an EC2 Instance

Click the checkbox shown in the first column for instance entry and click the Actions button to open the dropdown. Click the Connect dropdown option. Click the RDP Client tab. Click the Get Password link.

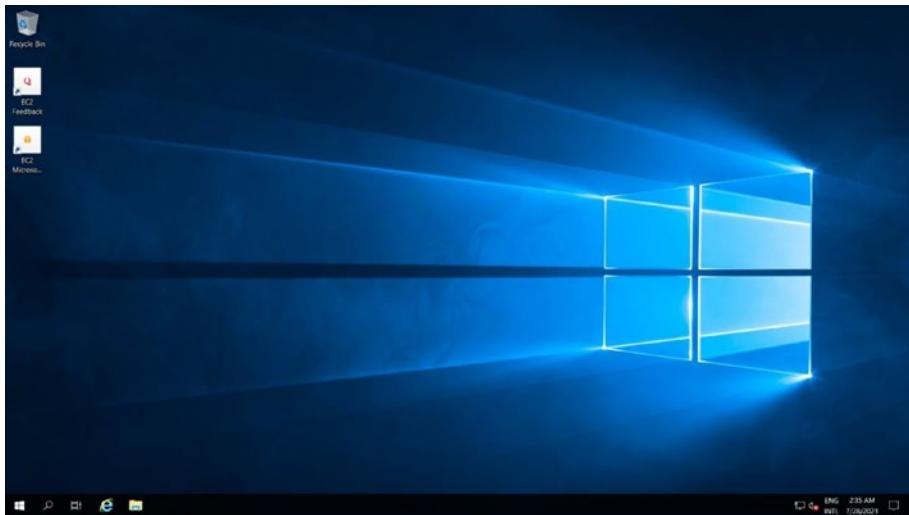
Click the Browse button and select the private key (MyEC2Instance.pem) file.

After you click the Open button from the File dialog, the private key contents will be shown in the text field. Click the Decrypt Password button. It will generate a password below the Password: label. Copy the password by clicking the button provided to the right of the generated password.

Click the Download Remote Desktop File button. Double-click the downloaded remote desktop file and click the Connect button.

Paste the password you copied inside the Password field and click the OK button.

Click the Yes button, which will open the RDP to EC2 instance you launched, shown in Figure 18-1.



**Figure 18-1.** Desktop of an EC2 instance

# Configuring the EC2 Instance to Deploy the Calculator Web Application

## Step 1: Install the IIS Web Server

You need to install the IIS web server so that you can deploy the Calculator web application in it. Follow these steps to install the IIS web server on server-type Windows machines.

Use the Window key to open the Windows menu search and type Server Manager. Select the Server Manager menu entry, which will open the Server Manager Dashboard.

Click the Add Roles and Features link and then click the Next button.

Keep the Role-Based or Feature-Based Installation radio button selected and click the Next button, which will open the Select Destination server window. Click the Next button.

Select the Web Server IIS checkbox, which will open the Add Features That Are Required for Web Server (IIS) window.

Click the Add Features button, which will open the Select Server Roles window.

Click the Next button to open the Select Features window.

Click the Next button to open the Web Server Role (IIS) window.

Click the Next button again, which will open the Select Role Services window.

Click the Next button a final time to open the Confirm Installation Selections window.

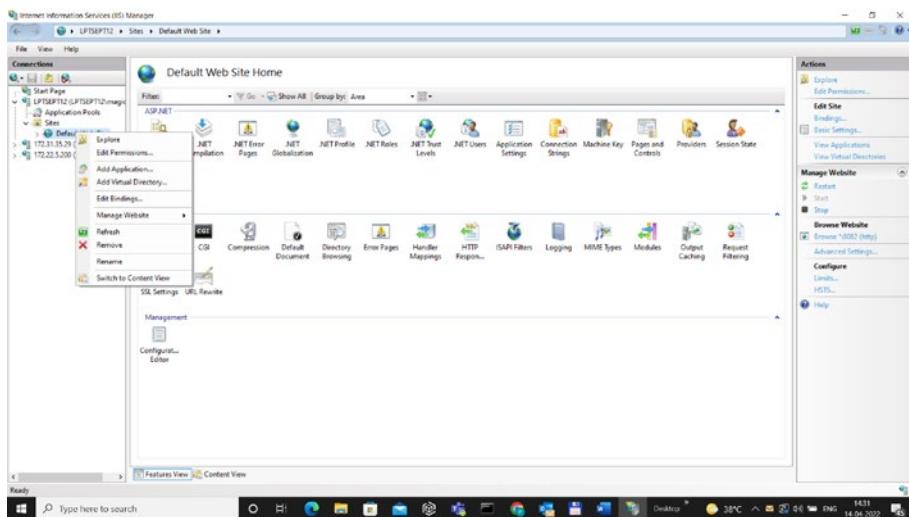
Click the Install button. Wait until the installation completes. Once installation is completed, click the Close button.

## Step 2: Configure the IIS Web Server

Create a blank directory named DeployedCalculatorApp on C:\.

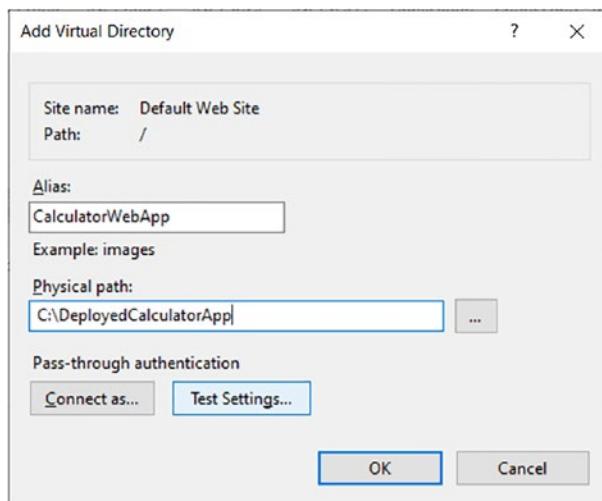
Now you'll create a web alias (a virtual directory) in IIS which will point to this directory. Go to the Windows menu and select the Internet Information Services (IIS) Manager option, which will open the Internet Information Services (IIS) Manager window.

Open the tree shown in the IIS console and right-click Default Web Site as shown in Figure 18-2.



**Figure 18-2.** The menu opens after clicking the Default Web Site

Select the Add Virtual Directory menu option to open the Add Virtual Directory window. Fill in the alias name as CalculatorWebApp in the Alias field and enter C:\DeployedCalculatorApp in the Physical Path field, as shown in Figure 18-3. Click the OK button to create a virtual directory inside the default website.



**Figure 18-3.** The details filled in to create an alias in the default website

## Step 3: Configure the SSH Connection Between the Local Computer and the EC2 Instance

As part of the Calculator web application's deployment process, you need to copy `Calculator.html` from your local computer to the EC2 instance inside the directory `C:\DeployedCalculatorApp` (to which your virtual directory in IIS is pointing). To copy this file from your local computer to an EC2 instance, you need to establish an SSH connection with the EC2 instance. Follow these steps to configure the required settings:

1. Generate an SSH key pair on a local computer: Use the `ssh-keygen -t rsa` command to generate an SSH public-private key pair.

This will generate public and private key files on the local computer in the `$(CURRENT_USER)\.ssh` folder.

2. Install the OpenSSH Server on an EC2 instance:  
On the EC2 instance, choose the Windows Start ➤ Settings menu to open the Settings page.
3. Click the Apps option, which will open the Apps & Features window.
4. Click the Manage Optional Features link.
5. Click the Add a Feature button.
6. Locate the OpenSSH Server option in the list.
7. Click the Install button.
8. Create an `authorized_keys` file on the EC2 instance and paste the public key into it: Go to the EC2 instance, open the command prompt from the `C:\users\administrator` directory, and run the `mkdir .ssh` command to create the `.ssh` folder.
9. From the command prompt, enter into the `.ssh` folder using the `cd` command and create a blank file named `authorized_keys`. Do this by running the following command:

```
fsutil file createnew authorized_keys 0
```

This will create the `authorized_keys` file in the `.ssh` folder.

10. Open this file and paste the public key you generated from the key pair.
11. Save the changes.

12. Set the permissions of the `authorized_keys` file to the Administrator user: Right-click the `authorized_keys` file and click the Properties menu.
13. Click the Security tab.
14. Click the Advanced button.
15. Click the Disable Inheritance button.
16. Select the Convert Inherited Permissions Into Explicit Permissions on this Object option.
17. Click the OK button.
18. Click the OK button from the `authorized_keys` Properties window.
19. Modify the `sshd_config` file on the EC2 instance: Go to `C:\ProgramData\ssh`.

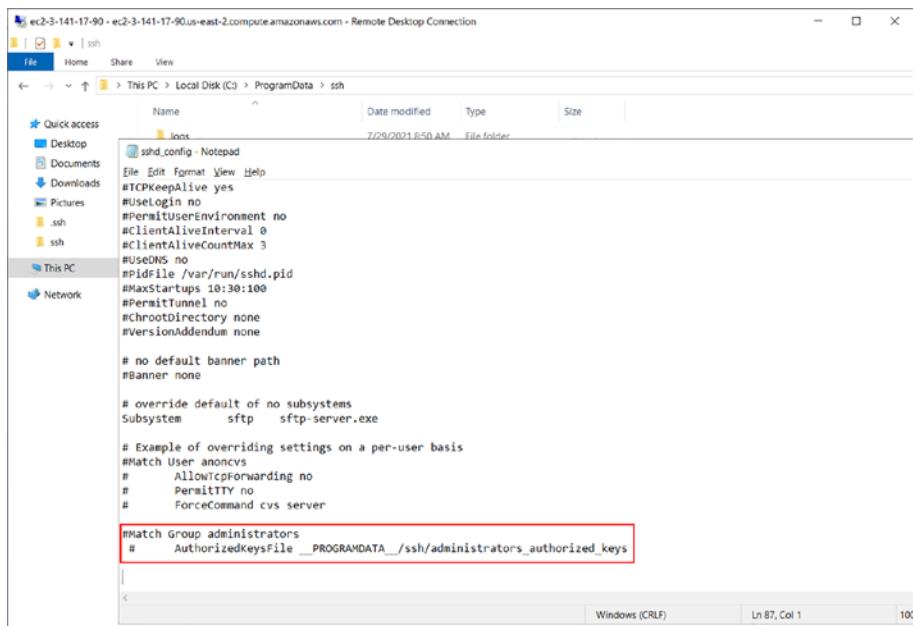
---

**Note** If the ProgramData folder is not shown, make hidden files/folders visible from the View menu.

---

20. Open the `sshd_config` file, scroll down, and comment out the last two lines by adding # to the start of these lines, as shown in Figure 18-4.

## CHAPTER 18 INTEGRATING JENKINS WITH AWS



```
File ec2-3-141-17-90 - ec2-3-141-17-90.us-east-2.compute.amazonaws.com - Remote Desktop Connection
File Home Share View
This PC > Local Disk (C) > ProgramData > ssh
Name Date modified Type Size
Quick access
Desktop
Documents
Downloads
Pictures
.ssh
ssh
This PC
Network
File Edit Format View Help
#TCPKeepalive yes
#SelLogin no
#PermitUserEnvironment no
#ClientAliveInterval 0
#ClientAliveCountMax 3
#UseDNS no
#Pidfile /var/run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none
# no default banner path
#Banner none

# override default of no subsystems
Subsystem sftp sftp-server.exe

# Example of overriding settings on a per-user basis
#Match User anonymous
#   AllowTcpForwarding no
#   PermitTTY no
#   ForceCommand cvs server

#Match Group administrators
#   AuthorizedKeysFile _PROGRAMDATA_/_ssh/administrators_authorized_keys
```

**Figure 18-4.** The last two lines commented out from the sshd\_config file

21. Save the file. Go to the Services window by choosing the Windows Start ➤ Services menu and restarting the Open SSH Server service.
22. Allow the SSH connection between the local computer and the EC2 instance: Before you can connect to the EC2 instance using SSH, you need to allow this connection by creating an inbound rule in the EC2 instance settings on AWS.
23. Go to the AWS console and click the Instances link inside the Instances section on the left side.  
This will open the Instances page.

24. Click the checkbox shown for the running instance
25. Click the Security tab.
26. Click the link shown under the security groups label inside the Security tab.
27. Click the Edit Inbound Rules button.
28. After clicking this button, you will see the Edit Inbound Rules page.
29. Click the Add Rule button.
30. Select SSH in the Type column dropdown for the newly created entry.
31. Specify 0.0.0.0/0 in the Source column.
32. Click the Save Rules button.

This rule will be added to the existing rules in the security group.

This rule will allow the SSH connection to be made with the EC2 instance using port 22 (the default SSH port).

33. Test the SSH connection between the local computer and the EC2 instance: Now you have all the necessary configurations in place. Let's test the connectivity using SSH between the local computer and the EC2 instance by running the following command:

```
ssh <UserName>@<EC2Instance_PublicIPV4DNS>
```

The command I am using is as follows:

```
ssh Administrator@ec2-3-141-17-90.us-east-2.  
compute.amazonaws.com
```

34. You can get the publicIPv4DNS from the Details page of the EC2 instance.
35. I executed the following command from the command prompt:

```
ssh Administrator@ec2-3-141-17-90.us-east-2.  
compute.amazonaws.com
```

This will ask, Are You Sure You Want to Continue Connecting (Yes/No/[Fingerprint])? Prompt. Choose Yes and press Enter.

You should now be able to successfully connect to the EC2 instance through SSH.

## Copying the Calculator.html File to the EC2 Instance

Before you create a Jenkins job that will deploy the CalculatorWebApplication by copying Calculator.html to the EC2 instance, you need to test the copying functionality by running the scp command.

You need to use the following command for this purpose:

```
scp -i /directory/to/privatekey /your/local/file/to/copy  
user@ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com:path/to/file
```

I want to copy the `Calculator.html` file to this folder:

D:\JenkinsBookExamples\CalcualtorWebApplication\src\main\webapp to C:\DeployedCalculatorApp on the EC2 instance. Therefore, my command is as follows:

```
scp -I  
C:\Users\magicuser\.ssh\id_rsa  D:\JenkinsBookExamples\  
CalcualtorWebApplication\src\main\webapp\Calculator.html  
Administrator@ec2-3-141-17-90.us-east-2.com  
pute.amazonaws.com:C:\DeployedCalculatorApp
```

After running this command from the command prompt, the `Calculator.html` file should be copied successfully to the EC2 instance inside the `C:\DeployedCalculatorApp` folder.

## Accessing Calculator.html Copied to the EC2 Instance from a Browser on a Local Computer

Let's now access the `Calculator.html` file from the browser on the local computer. Before you access the EC2 instance using the HTTP URL, you need to create an inbound rule to allow this connection. Follow the same steps you performed previously to create an inbound rule for SSH. The only change is that you need to select HTTP in the Type dropdown.

Go to the EC2 instance, choose the Windows Start ➤ Services menu, and restart the World Wide Web Publishing service. Open a browser on your local computer and access this URL:

<http://<EC2InstancePublicIP4DNS>/CalculatorWebApp/Calculator.html>

I used this URL:

<http://ec2-3-141-17-90.us-east-2.compute.amazonaws.com/>  
[CalculatorWebApp/Calculator.html](http://ec2-3-141-17-90.us-east-2.compute.amazonaws.com/CalculatorWebApp/Calculator.html)

# Creating a Jenkins Job to Deploy the Calculator Web Application on an EC2 instance

I created a free-style job called BuildAndDeployCalculatorWebApplicationOnEC2Instance. This job contains an Execute Windows batch command build step using the scp command shown here:

```
scp -i C:\Users\magicuser\.ssh\id_rsa src\main\webapp\Calculator.html Administrator@ec2-3-141-17-90.us-east-2.compute.amazonaws.com:C:\DeployedCalculatorApp
```

We have already discussed this command. This job will pull changes from the CalculatorWebApplication SCM. Using the scp command, it will copy Calculator.html from the workspace directory to the Amazon EC2 instance inside the C:\DeployedCalculatorApp folder.

After the build step is complete, this job will call the downstream job TestCalculatorWebApplication, which will start the Selenium tests. This job is mentioned in the Projects to Build field in the Post-Build Actions section. Click the Save button.

## Changing the Calculator Application URL in Selenium Framework

The Selenium framework accesses the Calculator.html file from your local computer. Now you want that framework to access the Calculator.html file deployed on the EC2 instance and run tests on it.

You need to change the ApplicationURL environment variable in EnvVars.csv to include the EC2 instance's calculator.html URL.

Commit and push this change to the Selenium framework repository. I pushed the change to my framework repository:

[gitlab.com/Pranoday/JenkinsBookSeleniumPythonFramework.git](https://gitlab.com/Pranoday/JenkinsBookSeleniumPythonFramework.git)

This is where the Selenium framework and tests will be pulled and executed.

## Running the BuildAndDeployCalculatorWebApplication OnEC2Instance Job

Let's run the BuildAndDeployCalculatorWebApplicationOnEC2Instance job by clicking the clock sign on the Jenkins dashboard. Click the Build button.

You can see the console log of the build of the BuildAndDeployCalculatorWebApplicationOnEC2Instance job.

After completing its tasks, it will trigger the TestCalculatorWebApplication job.

This job will start testing the Calculator web application, which its upstream job has deployed on the EC2 instance.

## Summary

This chapter explained what an EC2 instance is and how to create one. Then you installed the IIS web server on an EC2 instance and created a web alias to deploy the Calculator web application. You configured an SSH public key on the EC2 instance so that it can be connected over SSH. You also learned how to create inbound rules in the Security group attached to an EC2 instance to be able to connect it to HTTP and SSH. Finally, you created a Jenkins job which pulled the Calculator application and copied the Calculator.html file on the EC2 instance using the scp command. As of now, you have interacted with Jenkins through its UI. In the next chapter, you learn how to control Jenkins using its CLI (command-line interface).

## CHAPTER 19

# Miscellaneous Topics Part 1

This and the following chapter cover a few important topics such as Jenkins CLI, Jenkins Rest APIs. You are already aware of how to control Jenkins manually using its UI. But sometimes you may need to control Jenkins programmatically through scripting languages or shell programs. In that case, you need to know how to use the programming interface provided by Jenkins and its command-line interface. This chapter focuses on how to interact with Jenkins using its CLI.

You are going to learn about some different common use cases where you are required to interact with Jenkins programmatically instead of manually.

## Understanding the Jenkins CLI

Jenkins provides two kinds of interfaces—the Graphical User Interface(GUI), which we have already covered in this book, and the Command-line Interface (CLI), which you are going to learn about in this chapter.

Jenkins provides a different set of shell commands that you can run from shell programs, like the command prompt on Windows or the console shell on Linux-based systems. You can write batch programs or shell scripts to automate access to the Jenkins server. Let's see in detail

the different CLI commands Jenkins provides, how can we interact with the Jenkins server using these commands, and most importantly, how to provide authentication information to the Jenkins server to authenticate the Jenkins server access using CLI.

Jenkins provides us with many CLI commands, a list of which is available on the Jenkins CLI page.

Click the Manage Jenkins link available on the left side of the Jenkins dashboard. This will open the Manage Jenkins page.

Scroll down the page to find the Jenkins CLI link under the Tools and Actions section. Click the Jenkins CLI link, which will take you to the Jenkins CLI. This page shows a list of different commands and the purpose of each command.

## How to Interact with Jenkins Using its CLI

This section explains how to perform basic operations through the CLI, like creating a job, running a build of a specific job, retrieving a list of all jobs, retrieving console output of a build, disabling a particular job, deleting a particular job, and more.

Before you start using the Jenkins CLI, you need to download `jenkins-cli.jar`. This JAR file provides the CLI client so you can access the CLI interface.

Click the `jenkins-cli.jar` link from the Jenkins CLI page to download the `jenkins-cli.jar` file. After you click the link, the `jenkins-cli.jar` file will be downloaded.

**Note** I started my Jenkins server from my machine's IPv4 address (192.168.43.10) and the default port 8080. Therefore, my Jenkins URL is `http://192.168.43.10:8080`, which I use to access my Jenkins server through CLI. Your Jenkins URL will contain either the localhost or your IP address along with the port. I keep the `jenkins-cli.jar` file in the `D:\JenkinsBookExamples` folder, so the CLI commands have the `jenkins-cli.jar` path, written as `D:\JenkinsBookExamples\jenkins-cli.jar`. You need to use your path on your machine.

---

## How to Create a Job Using the Jenkins CLI

To create a new job in Jenkins, you need to specify the job name and the configuration XML based on the new job. A *view* is a way to organize jobs and content into tabbed categories on the Jenkins dashboard. By default, Jenkins jobs are created in the All view; this is the tab that you see when you go to the Jenkins dashboard

To create a new job in Jenkins, run the following command:

```
java -jar {pathofJenkins-cli.jar} -s ${JENKINS_URL} -webSocket  
create-job ${JobName} < ${Configuration.xml}
```

---

**Note** While demonstrating different CLI commands in this chapter, placeholders use `{}$`, which should be replaced with actual values.

---

To create a new job through CLI, you need to specify its specifications using an .XML file, which would be taken as a template for the new job.

I want to create new job called CLIJob1 using the configuration of an existing job named Demo. The configuration file is {JENKINS\_HOME}\jobs\Demo\Config.xml. You get this configuration file of a demo job by default after Jenkins is installed. I copied this file to the D:\JenkinsBookExamples folder and renamed it Demo.xml so my command is as follows:

```
java -jar D:\JenkinsBookExamples\jenkins-cli.jar -s  
http://192.168.43.10:8080/ -webSocket create-job CLIJob1 < D:\  
JenkinsBookExamples\Demo.xml
```

Open the command prompt and run this command:

```
java -jar D:\JenkinsBookExamples\jenkins-cli.jar -s  
http://192.168.43.10:8080/ -webSocket create-job CLIJob1 < D:\  
JenkinsBookExamples\Demo.xml
```

It will give the following error:

```
ERROR: anonymous is missing the Overall/Read permission
```

You get an error here because Jenkins considered this command coming from an anonymous user, as you have not provided authentication yet. Anonymous users are not authorized to create jobs according to the security configurations of the Jenkins server. Now let's see how to provide the authentication information of a user who is authorized to create a job in this CLI.

## Authenticating Users Using Basic Authentication (Username-Password/API Token)

When you perform any task using the Jenkins CLI, you need to provide authentication information to perform that task using the -auth argument. You can authenticate users using basic authentication as well as SSH authentication. Let's see first how to send Basic Authentication data using

the `-auth` command-line argument. In basic authentication, you have to send a username-password or a username-API token. Sending an API token is the preferred and more secure option.

Log in using the authorized user's credentials in the Login window. Once you are logged in, you can go to the page to create an API token.

I logged in using my Jenkins administrator credentials.

Click the User Name shown in the top-right corner of the dashboard page.

Click the Configure link to go to the next page, where you can configure API token for the user.

Click the Add New Token button inside the API Token section to generate a new token.

Enter a name in the field and click the Generate button.

Copy the generated token using the Copy This Token button and paste it in a file to preserve it for future use. Click the Save button.

Let's authenticate the user using this token and create the job again using CLI with the following command:

```
Java -jar ${Jenkins-cli.jar file path} -s {JENKINS_URL} -auth  
${UserName}:${API_TOKEN} -webSocket create-job ${JOB_NAME} <  
${Configuration.xml}
```

I am using the following command after replacing all the placeholders with values:

```
java -jar D:\JenkinsBookExamples\jenkins-cli.jar -s  
http://192.168.43.10:8080/ -auth Pranodayd:119737275fd13  
2a08d5a3b457ed56649a2 -webSocket create-job CLIJob1 < D:\  
JenkinsBookExamples\Demo.xml
```

Run the command at the command prompt. Then go to your Jenkins Dashboard page and refresh it. You can see that new job with the name CLIJob1 was created.

Let's go inside CLIJob1 to see the build step having the Execute Batch command step with the echo Hi, which you also have in the Demo job.

---

**Note** If the template XML file has any syntax issues, the create-job CLI command returns this error: cannot access the file because it is being used by another process. This is quite misleading and not related to the cause of the problem.

---

If a job with the same name already exists, the create-job command gives the Job Already Exists error.

## Authenticating Users Through SSH While Using CLI Commands

You have already seen how to authenticate users using the username-API Token. Now you'll see how to authenticate Jenkins users using the SSH authentication technique. You are familiar with SSH authentication, which was discussed in previous chapters of this book. The SSH authentication you need is a private key-public key pair.

First generate a new key pair using the following command:

```
ssh-keygen -m PEM -t rsa
```

Note that the `-m PEM` argument is used in the `ssh-keygen -m PEM -t rsa` command. `-m PEM` will generate a private key in RSA format, whereas simple the SSH-Keygen command, without the `-m PEM` argument, generates a private key in the OpenSSH format. The Jenkins CLI client supports private keys in the RSA format, so don't forget to mention the `-m` argument in the command. Once the key pair is generated, you need to specify the public key on the Jenkins User Configuration page. Follow these steps to do this:

1. Click the username shown on top-right side of the Dashboard page and click the Configure link.

This will open a page on which you can configure details for the user, like the API token, the public key, etc.

2. Scroll down the page to find the SSH Public Keys section. Paste the public key you generated into the SSH Public Keys field.
3. Click the Save button.

## Configuring an SSH Server in Jenkins

To access Jenkins using SSH authentication, you need to configure the SSH Server in Jenkins. First choose the Manage Jenkins ➤ Configure Global Security menu, which will take you to the Configure Global Security page.

Scroll down the page to find the SSH Server section near the end.

You need to set a port for the SSH connection. You have two options—Fixed and Random. If you select the Random radio control, the Jenkins server will select a random port to accept SSH connections. If we have a Firewall set up on the machine and need to allow incoming connections made on a specific port, then setting up an inbound rule for the random port will be difficult. If you select the Fixed radio button, you can use the port number of your choice. Jenkins will accept SSH connections on a given port. As the port is fixed, it would be easy to set up an inbound rule in a firewall. I selected the Fixed radio button and set it to 9090. Click the Save button.

## Configuring an Inbound Rule in a Firewall

If a firewall is set up on the machine, you need to create an inbound rule so that incoming connections made at particular port/incoming connections from a particular program will be allowed. In this example, we are going to create an inbound rule for the SSH server port 9090.

1. Go to the Windows Defender Firewall with Advanced Security menu option from the Windows start menu.
2. Click the Inbound Rules option on left side.
3. Click the New Rule link on the right side, which will open the Rule Type window.
4. Select the Port radio button.
5. Click the Next button.
6. Select the Specific Local Ports radio button and enter **9090**.
7. Click the Next button, which will open the Action window.
8. Click the Next button to see the Profile window.
9. Click the Next button to see the Name window.
10. Enter a name for the rule and click the Finish button.

## How to Build Jobs with the Jenkins CLI Using SSH Authentication

This section explains how to build a Jenkins job using the Jenkins CLI command `build`.

The build command has the following optional command-line options:

-c: If you pass this option to the build command it checks if there are any changes on the SCM configured in the job and triggers the build only if there are changes.

-f: If you pass this option to the build command, it returns the exit code based on the outcome of the build. If the build is successful, it returns an exit code of 0.

-p: Using this option, you can pass parameters to a build in the Key=Value format.

-s: If you pass this option, then the build command will wait until the build completes/gets interrupted. If you do not set this option, the build command will trigger the build but will not wait until its completion.

-v: This option will print the console output of the build.

-w: This option will wait until the start of the command.

While running the build command, you need to authenticate the Jenkins user who will use the SSH authentication technique. To use Jenkins CLI with SSH authentication, you have the following two options:

- Using an SSH client like OpenSSH/Putty
- Using Jenkins-cli.jar

## Using the OpenSSH Client to Run Jenkins CLI Commands

Let's use the OpenSSH client to trigger a build using Jenkins CLI build command.

Open the command prompt and run the following command to run the build of the ReleaseCalculatorAPI job:

```
ssh ${JenkinsUserName}@${JenkinsServerIPAddress} -p  
${JenkinsSSHPort} -i ${PrivateKeyFilePath} build ${JOB_NAME}
```

I replaced these placeholders with actual values and created the following command:

```
ssh Pranodayd@192.168.43.10 -p 9090 -i D:\SSHKey\JenkinsCLI  
build ReleaseCalculatorAPI
```

---

**Note** 9090 is a port we configured under the SSH Server section in the Jenkins settings.

---

Pranodayd is a Jenkins user we are authenticating. D:\SSHKey\JenkinsCLI is a file with the private key.

Run the following command from the command prompt:

```
ssh Pranodayd@192.168.43.10 -p 9090 -i D:\SSHKey\JenkinsCLI  
build ReleaseCalculatorAPI
```

Go to the Jenkins dashboard page. You will see that the build for the ReleaseCalculatorAPI job was triggered.

Let's run the build command again, but this time using the build command's options.

```
ssh Pranodayd@192.168.43.10 -p 9090 -i D:\SSHKey\JenkinsCLI  
build ReleaseCalculatorAPI -s -v -c
```

Go to the Jenkins Dashboard page and note that this time the build is not triggered, as we sent the `-c` option, which checks for changes in the SCM. Since there are no changes in the Git repository, the build is not triggered.

I changed version to 10.0 in `pom.xml` of the CalculatorAPI project and pushed this change to the Gitlab repository. Then I executed the same Jenkins CLI build command.

You can observe on the Jenkins dashboard that this time, the build was triggered and the console log of the build command was listed because you used the `-v` option.

## **Using the jenkins-cli.jar Client to Run Jenkins CLI Commands with SSH**

You are already familiar with `Jenkins-cli.jar`, which you used to run Jenkins CLI commands using the HTTP connection mode. The previous section covered the SSH connection mode and you saw how the SSH client called OpenSSH to run the Jenkins CLI commands. This section explains how to use `Jenkins-cli.jar` to run Jenkins CLI commands using SSH authentication (SSH Connection mode).

If you are starting the Jenkins server on an IP address (i.e. non-default), then you need to start the Jenkins server with an argument named `org.jenkinsci.main.modules.sshd.SSHD.hostName`, which will include the IP address of the Jenkins machine as its value.

If you are using the `Jenkins.war` file to start the Jenkins server, you need to use the following command:

```
Java -Dorg.jenkinsci.main.modules.sshd.SSHD.
hostName=<IPAddress> -jar <PATH of Jenkins jar> --httpListen
Address=<IPAddress>
```

## CHAPTER 19 MISCELLANEOUS TOPICS PART 1

I am using the following command:

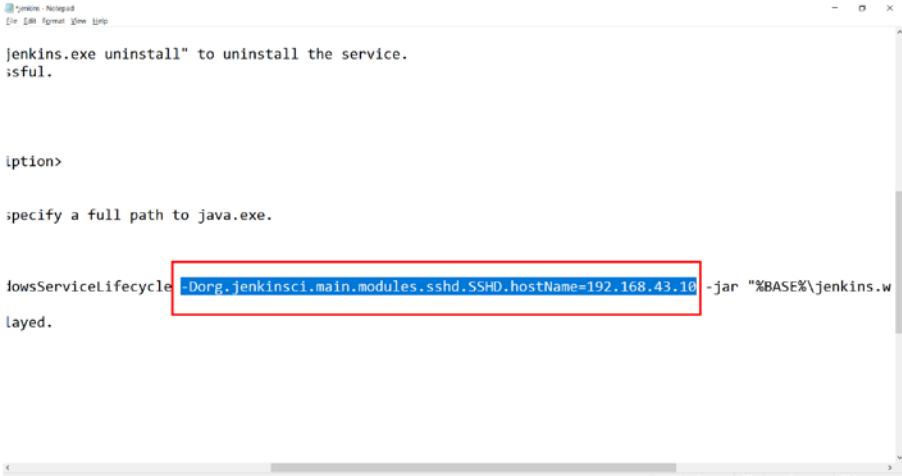
```
Java -Dorg.jenkinsci.main.modules.sshd.SSHD.  
hostName=192.168.43.10 -jar D:\jenkins\jenkins.war  
--httpListenAddress=192.168.43.10
```

---

**Note** There should not be whitespace between - and D and -D and org.jenkinsci.main.modules.sshd.SSHD.hostName.

---

If you are starting Jenkins as a service, you need to go to the `Jenkins.xml` file and add this argument inside the `<arguments>` tag, as shown in Figure 19-1.



The screenshot shows a Windows Notepad window with the file name "Jenkins.xml". The content of the file is as follows:

```
Windows Notepad  
File Edit Format View Help  
  
jenkins.exe uninstall" to uninstall the service.  
;sful.  
  
<option>  
  
specify a full path to java.exe.  
  
lowsServiceLifecycle -Dorg.jenkinsci.main.modules.sshd.SSHD.hostName=192.168.43.10 -jar "%BASE%\jenkins.w  
layed.
```

A red rectangular box highlights the line `-Dorg.jenkinsci.main.modules.sshd.SSHD.hostName=192.168.43.10`. The status bar at the bottom of the Notepad window shows "In 41, Col 140" and "100% Unix (LF) UTF-8".

**Figure 19-1.** The `Dorg.jenkinsci.main.modules.sshd.SSHD.hostName` in `Jenkins.xml`

Restart the Jenkins service. Once the Jenkins server starts, either by using `Jenkins.war` or a Jenkins service, you can use Jenkins-client.jar to run the Jenkins CLI commands. You need to use the following command:

```
java -jar ${PATH of Jenkins-cli.jar} -s ${JENKINSURL} -i
${PrivateKeyPATH} -ssh -user ${UserName} ${Jenkins CLI command}
${CLI options}
```

I am using the following command to build the job named Demo:

```
java -jar D:\JenkinsBookExamples\jenkins-cli.jar -s
http://192.168.43.10:8080 -ssh -user Pranodayd -i D:\SSHKey\
JenkinsCLI -p 9090 build Demo
```

Note that when you are using SSH connection mode, you need to specify -ssh in the command. If you are using -ssh then you must send the user ID with the -user option. By default, jenkins-cli.jar connects to the host and port that the Jenkins server is running on to get the SSH connection. But if you have set up a different port for SSH in the Jenkins server configuration (refer to previous sections of this chapter to learn how to set the SSHD port), as we set it to 9090, you need to specify the -p option in the command.

## How to Export All Jobs

If you want to shift your Jenkins server from one machine to another, you need to export Jenkins jobs from the old machine before you discontinue it. Listing 19-1 is batch code that exports all Jenkins jobs in .XML files.

**Listing 19-1.** Batch Code to Export all Jenkins Jobs

```
set JenkinsCLIJarLocation=D:\JenkinsBookExamples\
jenkins-cli.jar
set JenkinsURL=http://192.168.43.10:8080
FOR /F "tokens=*" %%g IN ('java -jar %JenkinsCLIJarLocation%
-s %JenkinsURL% -auth Pranodayd:Pranodayd@10 list-jobs') do
(java -jar %JenkinsCLIJarLocation% -s %JenkinsURL% -auth
Pranodayd:Pranodayd@10 get-job %%g > %%g.xml)
```

The batch code defines the variable JenkinsCLIJarLocation with the location of Jenkins-cli.jar. It defines the variable JenkinsURL as having the Jenkins URL. The following command returns all jobs using the Jenkins CLI command list-jobs:

```
java -jar %JenkinsCLIJarLocation% -s %JenkinsURL% -auth  
Pranodayd:Pranodayd@10 list-jobs
```

The following command runs in a loop and iterates over job names returned by list-jobs. It exports the configuration of each job using the get-job Jenkins CLI command. It exports the configuration returned for each job in the .XML file using the > (the redirection symbol).

```
java -jar %JenkinsCLIJarLocation% -s %JenkinsURL% -auth  
Pranodayd:Pranodayd@10 get-job %%g > %%g.xml
```

## How to Import All Jobs

Once the jobs are exported in the form of .XML files from the old machines, you can copy the folder with all jobs exported to a new Jenkins server machine and run the batch shown in Listing 19-2.

### ***Listing 19-2.*** Batch Code to Import All Jenkins Jobs

```
set JenkinsCLIJarLocation=D:\JenkinsBookExamples\  
jenkins-cli.jar  
set JenkinsURL=http://192.168.43.10:8080  
set JobsExportLocation=D:\PD  
FOR /F "delims=*" %%a IN ('dir /s /b %JobsExportLocation%\*.  
xml') do java -jar %JenkinsCLIJarLocation% -s %JenkinsURL%  
-auth Pranodayd:Pranodayd@10 create-job %%~na< %%a
```

The batch code defines the variable JenkinsCLIJarLocation as having the location of Jenkins-cli.jar. It defines the variable JenkinsURL as having the Jenkins URL of the new machine

It defines the variable JobsExportLocation as having the path of the folder in which you exported the job XML files.

The following line returns an absolute location of each .XML file from the folder and will store it in a variable:

```
FOR /F "delims=*" %%a IN ('dir /s /b  
%JobsExportLocation%\*.xml')
```

The following line runs the Jenkins CLI command create-job\${JOB\_NAME} < \${CONFIGURATIONFILE} in a loop.

JOB\_NAME is extracted from the full path stored in variable a using ~na and CONFIGURATIONFILE is sent using variable a.

```
do java -jar %JenkinsCLIJarLocation% -s %JenkinsURL% -auth  
Pranodayd:Pranodayd@10 create-job %%~na< %%a
```

## Summary

This chapter explained how to interact with Jenkins server using its CLI (command-line interface) and how to use the Jenkins-cli.jar to interact with the Jenkins server by connecting using HTTP URL and SSH authentication. You also learned, with the help of code snippets, how to perform regular tasks like exporting Jenkins jobs using the Jenkins CLI.

## CHAPTER 20

# Miscellaneous Topics Part 2

The last chapter discussed how to interact with the Jenkins server using its CLI. You learned about using `Jenkins-cli.jar` to interact with the Jenkins server through its CLI. Sometimes interacting with the Jenkins server through its CLI is not sufficient. Users may need extra programmatic control over the Jenkins server, which is possible only by using programming languages. They may need to interact with the Jenkins server using a programming language like Java or Python, for example. Jenkins provides Rest APIs, through which you can interact with the Jenkins server using different programming languages.

This chapter explains how to control the Jenkins server through REST APIs provided by Jenkins, using Python. This chapter also covers creating reusable libraries to be used in multiple pipelines.

# Understanding Jenkins Remote Access API

Jenkins provides remote access API to its functionalities. Currently, it comes in the following three flavors:

- XML
- JSON
- Python

Remote access API is offered in REST-like style. REST APIs are light-weight web services. These are the APIs that can be called over the HTTP protocol. Jenkins APIs are available under .../api/ URL where the ... portion is the data that it acts on. For example, if my Jenkins server is started on <http://192.168.43.10:8080>, then <http://192.168.43.10:8080/api/> will show the top-level API features available, primarily listing of all configured jobs. If I want to access information about the last successful build of the Demo job, then I go to <http://192.168.43.10:8080/job/Demo/lastSuccessfulBuild>.

Before you look at all things you can do with Jenkins Remote APIs (REST-API), let's consider a few fundamentals of REST APIs. REST APIs provide different methods such as GET, POST, PUT, and DELETE. If you wanted to get information about some resource on the server, you would use GET. If you wanted to create a new resource on the server, like a Jenkins job, you would use POST.

You can send parameters to REST APIs in the following three ways:

- **Query parameters:** Values sent to the REST API as a part of URL in the form of Key=Value. These Key-Value pairs appear after the ? in the URL.

For example, <https://example.com/articles?sort=ASC&page=2>

There are two parameters here—sort with a value of ASC and page with a value of 2.

- **Path parameters:** These are the values sent as part of a URL endpoint. These are usually within curly braces, i.e., {}.
- **Header parameters:** These are the parameters sent in the request headers. Authentication information such as username-password and API tokens are typically sent using this type of parameter.

## Using Jenkins Remote Access API

This section discusses how to use Jenkins REST APIs to perform different tasks. Before you start using Jenkins REST APIs, you need a REST client to access REST APIs. Web browsers acts like clients when you access any HTML file from the server, right? Similarly, in order to access resources available on the server using REST APIs, you need software/utility that will act as a REST client. There is a popular REST client available called Curl. Using Curl, you can access REST APIs through command-line programs like the command prompt on a Windows machine.

Download Curl from <https://curl.se/windows/>. Clicking this link will download a .ZIP file. Unzip the file at your desired location.

Add this bin folder path to the PATH environment variable so that Curl.exe can be accessed from any working directory in the command prompt.

## Getting the Configuration of Existing Jenkins Jobs Using Jenkins Remote API

Now you have Curl set up. Let's start with the actual work now. We are going to fetch a configuration of the existing Jenkins jobs using the following Jenkins REST API.

```
curl -X GET http://192.168.43.10:8080/job/Demo/config.xml -u  
Pranodayd:119737275fd132a08d5a3b457ed56649a2 -o D:\PD\  
mylocalconfig.xml
```

Let's look at this API request in more detail:

-X GET: Defines the type of method of the REST API. This is a GET request as you are fetching details of some resource that's already present on the server.

<http://192.168.43.10:8080/job/Demo/config.xml>: This is the URL of the config.xml file for the Demo job, which you want to fetch. As discussed previously in this chapter, every job's configuration is stored in the file named config.xml in the \${JENKINS\_HOME}\Jobs\\${JOB\_NAME} folder.

-u Pranodayd:119737275fd132a08d5a3b457ed56649a2: After -u, you are sending the user's authentication information. Authentication information contains UserName:APIToken. This is the same API token used previously in this chapter.

-o D:\PD\mylocalconfig.xml: Using -o, you are writing the configuration contents you are receiving as a response in a file path.

Note that when using REST APIs, you need to provide the authentication information of the user who is authorized to perform the task in question.

Run the following command from the command prompt:

```
curl -X GET http://192.168.43.10:8080/job/Demo/config.xml -u
Pranodayd:119737275fd132a08d5a3b457ed56649a2 -o D:\PD\mylocalconfig.xml
```

Go to folder D:\PD. You will see that the mylocalconfig.xml file has been created.

## Creating New Jenkins Jobs Using Jenkins Remote API

This section creates a new Jenkins job using the following Jenkins remote API. To create a new job, you need a job name and a configuration XML file that will be used as a template.

```
curl -X POST http://192.168.43.10:8080/createItem?name=
RestAPIJob -u Pranodayd:119737275fd132a08d5a3b457ed56649a2
--data-binary @D:\PD\Demo.xml -H "Content-Type:text/xml"
```

Let's look at this API request in more detail:

-X POST: Defines the type of the method of the REST API. This is a POST request, as you are creating a new resource on the server.

[`http://192.168.43.10:8080/createItem?name=RestAPIJob`](http://192.168.43.10:8080/createItem?name=RestAPIJob): <http://192.168.43.10:8080/>  
`createItem` is the URL endpoint that you want to access to create a new job. `name=RestAPIJob` after the ? is a query string parameter containing the name of the new job.

**-u Pranodayd:119737275fd132a08d5a3b457ed5664**

9a2: After -u you are sending the authentication information of the user. The authentication information contains UserName:APIToken. This is the same API token you used previously in this chapter.

**--data-binary @D:\PD\Demo.xml:** You are sending a path of the job configuration XML that needs to be taken as a template to create a new job. The contents of this xml will be sent as a request data and is used to create new job.

**-H "Content-Type:text/xml":** You are sending information about the type of request data using the Content-Type request header.

Run the following command from the command prompt:

```
curl -X POST http://192.168.43.10:8080/
createItem?name=RestAPIJob -u Pranodayd:119737275fd132a08d5a
3b457ed56649a2 --data-binary @D:\PD\Demo.xml -H "Content-
Type:text/xml"
```

Go to the Jenkins Dashboard. You can see that the new job with the name RestAPIJob has been created.

## Triggering Parameterized Jenkins Job Using Jenkins Remote API

This section talks about the Jenkins remote API, which you can use to trigger a Jenkins parameterized job and explains how to send parameters to this job.

I want to trigger a build of the parameterized job called `BuildAndDeployCalculatorWebApplication`. This job takes the following two parameters:

- `CalculatorWebApplicationDeploymentDirectory`
- `TypeOfTestsToRun`

I use the following curl command to trigger this parameterized job.

```
curl -X POST http://192.168.43.10:8080/job/BuildAnd
DeployCalculatorWebApplication/buildWithParameters -u Pranoday
d:119737275fd132a08d5a3b457ed56649a2 -d "CalculatorWeb
ApplicationDeploymentDirectory=D:\JenkinsBookExamples\Deployed
CalculatorWebApp&TypeOfTestsToRun=AllTests"
```

Let's look at this command in more detail:

`-X POST`: Defines the type of method of the REST API. This is a POST request.

`http://192.168.43.10:8080/job/BuildAndDeployCalculatorWebApplication/buildWithParameters`: This is the URL endpoint that you want to access to trigger a build. This URL endpoint is in this format:

**`${JENKINS_URL}/job/${JOB_NAME}/buildWithParameters`**

`-u Pranodayd:119737275fd132a08d5a3b457ed56649a2`: After `-u`, you are sending the authentication information of the user. The authentication information contains `UserName:APIToken`. This is the same API token you used previously in this chapter.

`-d"CalculatorWebApplicationDeploymentDirectory=D:\JenkinsBookExamples\DeployedCalculatorWebApp\&TypeOfTestsToRun=AllTests": This is sending the required two job parameters in the format VariableName=VariableValue. Note that there is a & in between both parameters.`

Run the following command from the command prompt:

```
curl -X POST http://192.168.43.10:8080/job/BuildAndDeployCalculatorWebApplication/buildWithParameters -u Pranodayd:119737275fd132a08d5a3b457ed56649a2 -d "CalculatorWebApplicationDeploymentDirectory=D:\JenkinsBookExamples\DeployedCalculatorWebApp\&TypeOfTestsToRun=AllTests"
```

Go to the Jenkins Dashboard. You can see that a build for the job named BuildAndDeployCalculatorWebApplication/buildWithParameters has been triggered.

During the build process, the web application would be tested through the Python Selenium tests.

## **Triggering Normal (Non-Parameterized) Jenkins Job Using Jenkins Remote API**

Let's trigger a non-parameterized job using a Jenkins REST API. There is a very simple REST API call to trigger a non-parameterized job. Say you want to trigger the job named ReleaseCalculatorAPI. You would use the following command:

```
curl -X POST http://192.168.43.10:8080/job/ReleaseCalculatorAPI/build -u Pranodayd:119737275fd132a08d5a3b457ed56649a2
```

Let's look at this command in more detail:

**-X POST:** Defines the type of method of the REST API. This is a POST request.

`http://192.168.43.10:8080/job/`

`ReleaseCalculatorAPI/build`: This is the URL endpoint that you want to access to trigger a build. This URL endpoint is in this format:

`${JENKINS_URL}/job/${JOB_NAME}/build`

Note that it uses `build` at the end here to trigger a non-parameterized job.

`-u Pranodayd:119737275fd132a08d5a3b457ed56649a2`: After `-u`, you are sending the user's authentication information. The authentication information contains `UserName:APIToken`. This is the same API token you used previously in this chapter.

Run the following command from the command prompt:

```
curl -X POST http://192.168.43.10:8080/job/  
ReleaseCalculatorAPI/build -u Pranodayd:119737275fd132a08d5a  
3b457ed56649a2
```

On the Jenkins dashboard, you can see a new build where the job named `ReleaseCalculatorAPI/` is triggered.

# Working with the Jenkins Server Using Python-Jenkins

You learned how to use Jenkins REST APIs to perform different tasks. You used the REST client called *Curl* for this purpose. If you want to work more effectively with REST APIs from different programming languages like Java, C#, Python, Ruby, and so on, you need language wrappers created over a particular REST API endpoint. If you use these language wrappers, the response returned by a REST API is returned after serializing it in the form of a variable type that a particular language supports. For example, if you are using a Java wrapper over a REST API, the returned response would be serialized in the Java object and you could deal with it the way you deal with any other Java object.

This section explains how Python wrappers implemented over Jenkins REST APIs can be used to perform different tasks. You can try other Ruby and Java wrappers as per your language preferences.

There are various Python wrappers, including JenkinsAPI, Python-Jenkins, api4jenkins, and aiojenkins. They are all object-oriented Python wrappers for Python REST API, which provides the Pythonic way to control the Jenkins server. Of these four Python wrappers, you are going to learn how to use Python-Jenkins to perform different tasks.

Before you can start using a Python wrapper, you need to install the following things if they are not installed on your system:

- **Python Interpreter:** Download the appropriate Python version from <https://www.python.org/downloads/>
- **Pycharm:** Download the Professional version (free trial for 30 days) or Community edition (free version) of Pycharm, a code editor for Python.

- **Python-Jenkins package:** Once you install the Python interpreter, you need to add Pip and Python to the PATH system environment variable. This is explained in Chapter 15.

Once Pip is present in the PATH environment variable, you can install the Python-Jenkins package using the `pip install Python-jenkins` command.

## Using Python-Jenkins Package Libraries

I created a Python project in pycharm and created a Python file called `WorkingWithJenkinsJobs.py` and wrote code in it. The code written in this file is shown in Listing 20-1.

### ***Listing 20-1.*** Python Code to Connect to a Jenkins Server

```
import jenkins
#Connecting to a Jenkins server
server = jenkins.Jenkins('http://192.168.43.10:8080',username="Pranodayd",password="119737275fd132a08d5a3b457ed56649a2")
```

The code in Listing 20-1 calls a constructor of class with the URL of the Jenkins server, username, and password parameters. It creates a reference variable named `server` to programmatically access the Jenkins server instance.

### ***Listing 20-2.*** Python Code to Get the Information About the Logged-in User in Jenkins

```
#Getting information aboutloggedin user
user = server.get_whoami()
version = server.get_version()
print('Hello %s from Jenkins %s' % (user['fullName'], version))
```

The code in Listing 20-2 calls an API called `get_whoami()`, which returns information about the logged-in user. This code prints this user information along with the version of the Jenkins server.

***Listing 20-3.*** Python Code to Retrieve the configuration.xml File of the Job Named Demo

```
#Getting the configuration of existing job named "Demo"  
DemoJobConfiguration=server.get_job_config("Demo")
```

The code in Listing 20-3 retrieves the configuration XML of an existing job named Demo and saves it in a variable named `DemoJobConfiguration`.

***Listing 20-4.*** Python Code to Create a New Job Named JobCreatedUsingPythonJenkins

```
...  
Creating a new job named "JobCreatedUsingPythonJenkins"  
using configuration retrieved and saved in  
variable DemoJobConfiguration  
...
```

```
server.create_job('JobCreatedUsingPythonJenkins',  
DemoJobConfiguration)
```

The code in Listing 20-4 calls the `create_job` API to create a new job named `JobCreatedUsingPythonJenkins` and the configuration is stored in the `DemoJobConfiguration` variable. The text within the quotes '' is commented code.

***Listing 20-5.*** Python Code to Print Information of All Jobs Available in Jenkins

```
#Printing information of all jobs,Each job information is
returned in the form of Dictionary
jobs = server.get_jobs()
print(jobs)
```

The code in Listing 20-5 returns the information about all existing jobs. Information about each job is returned in the form of a dictionary object having the name, url, color, and fullname keys.

***Listing 20-6.*** Python Code to Trigger a Build of a Job Not Accepting Any Parameters (Simple Non-Parameterized Job)

```
#Triggering a build of Non-parameterized job
server.build_job('ReleaseCalculatorAPI')
```

The code in Listing 20-6 triggers the build of a non-parameterized job named ReleaseCalculatorAPI.

***Listing 20-7.*** Python Code to Delete a Specific Jenkins Job

```
#Deleting a job created
server.delete_job('JobCreatedUsingPythonJenkins')
```

The code in Listing 20-7 deletes the job named JobCreatedUsingPythonJenkins.

***Listing 20-8.*** Python Code to Trigger a Build of a Job That Accepts Parameters (a Parameterized Job)

```
# build a parameterized job
# Building our job "BuildAndDeployCalculatorWebApplication"
with required 2 parameters
```

```
server.build_job('BuildAndDeployCalculatorWebApplication',  
{'CalculatorWebApplicationDeploymentDirectory':  
'D:\\JenkinsBookExamples\\DeployedCalculatorWebApp\\',  
'TypeOfTestsToRun': 'AllTests'})
```

The code in Listing 20-8 builds the parameterized job named BuildAndDeployCalculatorWebApplication. It calls build\_jobapi with two parameters—the first parameter is the job name and the second parameter is the dictionary object having both parameters sent using Key:Value.

***Listing 20-9.*** Python Code to the Retrieve the Build Number of the Last Build Executed for a Specific Job

```
#Retrieving build number of last build executed for job  
"BuildAndDeployCalculatorWebApplication"  
last_build_number = server.get_job_info('BuildAndDeployCalculatorWebApplication')['lastCompletedBuild']['number']
```

The code in Listing 20-9 retrieves the number of the last completed build for the job named BuildAndDeployCalculatorWebApplication.

---

**Note** I uploaded the whole Python project with this code to the GitLab public repository at [https://gitlab.com/Pranoday/python\\_jenkins.git](https://gitlab.com/Pranoday/python_jenkins.git).

---

# How to Use Shared Libraries in the Jenkins Pipeline

Multiple jobs often have common build steps in their pipelines. Instead of repeating these steps in each of these jobs, you can implement them in one Groovy file and share this Groovy file with the multiple jobs. This approach reduces code repeatability and introduces maintainability in pipelines.

You created a pipeline that deploys the `CalculatorAPI.jar` file. This pipeline pulls the code of the Calculator API from the Git repository and runs all the required build phases, like compile, test, package, and deploy. Now let's assume that you have another Java API .JAR file to build using the pipeline and all the steps applicable to the `CalculatorAPI` jar are also applicable to this Java API. Creating different pipeline scripts with the same steps in two different pipeline jobs is not a good idea. What you can do is implement these steps in one Groovy file and use it as a library file in both of the Jenkins jobs. There would be a few project-specific parameters like Git URL, credentials, that you could send to this common library as an argument. Let's create a shared library and use it in the Jenkins pipeline to achieve this.

## Step 1: Creating a Shared Library in the .Groovy File

I created a blank folder called `SharedLibrary` on my machine. Inside that, I created a folder called `vars` and in that I created a file named `buildJavaAPI` (camel case for the filename is mandatory here) with the `.groovy` extension.

**Note an important point here: The .groovy script file should be inside a directory named src or vars.** It does not work otherwise.

The code for this file is in Listing 20-10.

***Listing 20-10.*** Call Function Written in the buildJavaAPI.groovy File

```
def call(String RepoUrl,StringBranch, String Credentials) {  
    pipeline {  
        agent any  
  
        stages {  
  
            stage("Checkout Code") {  
                steps {  
                    git branch: "${Branch}", credentialsId: "${Credentials}",  
                    url: "${RepoUrl}"  
                }  
            }  
            stage("Cleaning workspace") {  
                steps {  
                    bat 'mvn clean'  
                }  
            }  
            stage("Running Testcase") {  
                steps {  
                    bat 'mvn test'  
                }  
            }  
            stage("Packing Application") {  
                steps {  
                    bat 'mvn package -DskipTests'  
                }  
            }  
            stage("Deploying Application") {  
                steps {  
                    bat 'mvn deploy -DskipTests'  
                }  
            }  
        }  
    }  
}
```

```
        }  
    }  
}  
}
```

It contains the build pipeline written in a function named call. A shared library file should contain the function named call and the reusable code should be written inside this function. Whenever a Jenkins job refers to a shared library, Jenkins calls the function named call from within the Groovy file. The call function is like a main function in Java. The call function written in this .groovy file has three parameters that will require project-specific information—Giturl, credentials entry name, and branch.

## Step 2: Push the Created Shared Library File to the GitLab Repository

Let's create a new GitLab repository and push the .groovy file to that.

I created a private repository called JenkinsSharedLibrary on GitLab and pushed the .groovy file in it.

## Step 3: Configure the Shared Library in Jenkins

Go to the Jenkins UI and choose Manage Jenkins ➤ Configure System. This will open the next page. Scroll down to find the Global Pipeline Libraries section.

Click the **Add** button in the **Global Pipeline Libraries** section, which will open additional fields.

Enter a name in the **Name** field to use in the @Library annotation in the job pipelines to make a reference to the reusable library. I entered BuildJavaAPIs.

In the Default Version field, enter the name of the branch where the reusable library is pushed to on the Git repository. The groovy script is pushed into the master branch in the repository, so I used master in this field.

In the Retrieval Method section, select the Modern SCM radio control and then inside the Source Code Management subsection, select the Git radio control.

Specify the Git repository URL of the reusable library in the Project Repository field and select the appropriate credentials entry in the Credentials dropdown.

Click the Save button.

## Step 4: Create a Pipeline Job to Use the Shared Library

I created a new pipeline job named PipelineJobUsingSharedLibrary. Scroll down the page to find the Pipeline section and write the script shown in Listing 20-11.

### ***Listing 20-11.*** Pipeline Script Calling the Shared Library

```
@Library('BuildJavaAPIs')
buildJavaAPI('https://gitlab.com/Pranoday/
jenkinsbookcalculatorapi.git','Master','MyGitCredentials')
```

Line 1:@Library imports the shared library to the Jenkins job. (Recall that BuildJavaAPIs is the name we used when configuring shared libraries in Manage Jenkins.)

Line 2: Calls the `buildJavaAPI` (a name of the shared library `.groovy` file) with three parameters—the Calculator API GitLab URL, the branch in which you keep the Calculator API's latest source code, and the name of the credentials entry with the GitLab authentication information stored inside Jenkins.

Click the Save button.

## Step 5: Running the Pipeline Job

Go to Jenkins dashboard and run the `PipelineJobUsingSharedLibrary` job by clicking the clock sign. Check the console output.

You can see that it will check out the shared library from its SCM and run the pipeline code inside it by using the values received as parameters.

This reusable library can be used to build other Java API projects as well. You need to simply send project-specific details as arguments to it.

## Summary

In this chapter, you learned how to get the configuration of an existing Jenkins job and trigger Jenkins jobs using Jenkins REST APIs in Python. You also learned how to create a reusable library and use it in the Jenkins pipeline.

Throughout this book, you learned about all important features that a Jenkins administrator or a Jenkins user needs to know. I hope this book has shown you how to use Jenkins with your real-time projects.

# Index

## A

Addition and Subtraction functions, 224  
Add Parameter button, 285  
Agile development lifecycle model, 1  
Agile methodology, 2, 7, 9  
Amazon Web Services (AWS), 359–362  
Apache Tomcat, 10  
API access token  
    creation, 308–310  
    free-style jobs, 319, 320  
    GitLab repository, 313  
    Jenkins, 315–318, 338  
    private GitHub  
        repository, 310–312  
    private GitLab  
        repository, 313–315  
API authentication  
    aim, 308  
    GitHub repositories, 307–310  
API token, 63  
Application Programming Interface (API), 238  
ApplicationURL environment, 372

Arithmetic operations, 262  
Artifact, 191  
Artifact/package registries, 148  
Assigning roles to users in Jenkins  
    Assign Roles screen, 100  
    creating user roles, 97–99  
    Role-Based Authorization Strategy Plugin, 93–95  
    Role-Based Strategy, 96  
    steps, 99  
    View role, 101  
Authentication, 55, 59, 64  
Authentication information, 393  
Authorization, 55, 60  
Auto-Triggering, 296  
AWS platform, 357

## B

Basic authentication, 307  
BuildAndDeployCalculatorWeb Application, 404  
Build in Jenkins, 116  
buildJavaAPI, 405  
Build step, 116  
Build tools, 150

## INDEX

### C

CalcualtorAPI.jar, 224  
CalculatorAPI repository, 322  
Calculator.html file, 263, 373  
Calculator.java file, 221  
CalculatorPage class, 271  
Calculator web application, 4, 373  
Chrome browser, 273  
Chromedriver.exe, 270  
Chron expressions, 135, 136  
CI/CD process, 9  
CI/CD workflow  
    Addition() function code, 6  
    code development, 5, 6  
    developing/running unit test cases, 4, 5  
    rerun the unit test case, 6  
    run e-e tests, 7  
    web application and deploy, 7  
Cloning Calculator API repository stage, 322, 323  
Command-line interface (CLI), 151, 166, 375  
CompileJavaApplication, 134  
Configure Global Security page, 55, 56, 96, 106, 109, 111, 345  
Configure System menu option, 292  
Console Output menu option, 247  
Console stage, 325  
Continuous Integration (CI), 2, 4  
    code to the central repository, 3  
    running unit tests locally, 2  
    unit and integration tests, 3

Credentials, 69  
Credentials ID, 70  
Cross-Site Request Forgery (CSRF), 62  
Crumb, 62  
Curl, 400

### D

Declarative pipelines, 240, 246  
Declarative syntax, 321  
Default Subject, 293  
Default website running, 267  
DemoJobConfiguration, 402  
Deploying CalculatorAPI, 323  
Deployment process, 267  
Development flow of a Java API Project  
    coding and writing unit test cases, 149  
    downloaded libraries, 149  
    third-party libraries, 148  
    unit test cases, 149  
DingarePranoday user, 103, 105, 108–112  
Distributed Version Control system, 179  
Domain Specific Language (DSL), 11

### E

EC2 instance, AWS  
    creation, 361  
    sign in, 360, 361

- sign up, 360
- starting, 362
- virtual server, 359
- EC2 Instance configuration,
  - calculator web application
  - accessing Calculator.html File, 371
  - copying Calculator.html File, 370, 371
  - IIS web server, 363, 364
  - Jenkins Job, 372
  - running application, 373
  - selenium framework, 372
  - SSH connection, 365–370
- Eclipse workspace, 151, 166
- Edit Virtual Directory window, 266
- E-E testing, 261, 268
- Elastic Compute Cloud, 359
- Email Extension plugin, 292
- Email notification, 234
  
- F**
- Free-style jobs, 116, 145
  
- G**
- Generate Pipeline Script button, 254
- Git, 178, 179
  - end-end use for the API Project, 180–190
  - GitLab/GitHub, 180
  - installing, 180
- GitHub account, 277
- GitHub branch, 305
- GitHub repository, 286, 297, 305, 337
- GitHub Webhook, 335–337
- GitLab, 216
- GitLab code repositories, 71
- GitLab credentials, 208
- GitLab/GitHub, 180
- GitLab private key path, 278
- GitLab repository, 209, 211, 253
- GitLab repository page, 226
- GitLab Webhook
  - local IP address, 333
  - local port range, 333
  - port forwarding, 332, 333
  - protocol, 333
  - Public IP Webhooks, 332
  - remote IP address, 333
  - remote port range, 333
  - triggering procedures, 334, 335
  - URL, 332
- Git repository, 127, 133, 253
- Global Tool Configuration
  - page, 45, 46
- Global Tool Configuration Settings, 45, 47, 48
- Graphical User Interface (GUI), 375
  
- H**
- Header parameters, 393
- HTTP Proxy Configuration
  - section, 38, 39

## INDEX

- |
- Inheritance Strategy dropdown, 111, 112
  - Installation of Jenkins
    - configuration files and directory structure, 17
    - as a Docker Image, 17
    - hardware/software requirements, 15, 16
    - settings in Jenkins.xml, 18, 19
    - using a .WAR file, 16
    - using the MSI Installer, 16
    - on Windows, 15
  - Internet access, 38
- ## J, K
- JAR file, 328
  - Java API Project, 147–150, 174, 175
    - code files, 160
    - pom.xml File, 161–165
    - third-party libraries, 148
    - using Maven build tool, 148, 150–158
  - Java Configuration, 53, 54
  - JAVA\_HOME environment variable, 248
  - Java Network Launch Protocol (JNLP), 341
  - Jenkins
    - anonymous user access, 59
    - architecture, 12–14
    - assigning project-based role to the user, 102–104
    - authentication, 55
    - automation server, 57
    - as a CI/CD automation server, 69
    - CI/CD process, 9, 12
    - configuring a credentials provider, 84–88
    - configuring global security, 55–64
    - configuring Jenkins (*see* Jenkins configuration)
    - creating project-based roles, 101–105
    - creating users, 91, 92
    - credential entries, 72, 74–78
    - credentials entry in a different domain, 79–84
    - credentials management, 71
    - history, 10
    - implementing CI/CD, 11, 12
    - installing Jenkins (*see* Installation of Jenkins)
    - Jenkins Credentials plugin, 70
    - LDAP, 64
    - matrix authorization strategy, 109–112
    - matrix-based security option, 106–109
    - own user database, 57
    - Role-Based Authorization Strategy Plugin, 93–95
    - Role-Based Strategy option, 96
    - scope and domains, 71, 72
    - server, 391

- settings for the Authorize Project plugin, 65
- software development process, 10
- updating a credentials entry, 76–79
- JenkinsBookCalculatorAPI repository, 215, 216, 335
- JenkinsBookCalculatorWeb Application repository, 310
- Jenkins CLI, 375
  - authenticating users, SSH, 380, 381
  - basic authenticating users, 378–380
  - exporting all Jenkins jobs, 387, 388
  - GUI, 375
  - importing all Jenkins jobs, 388, 389
  - inbound rule configuration, 382
  - interaction, 376
  - jenkins-cli.jar client, 385–387
  - job building, 382, 383
  - job creation, 377, 378
  - OpenSSH client, 384, 385
  - shell commands, 375
  - SSH server configuration, 381
- jenkins-cli.jar link, 376
- jenkins-cli.jar client, 385–387
- Jenkins configuration
  - adding a new user, 31
  - global settings and paths, 21
- opening the browser and signing in, 24
- resetting the username and password, 27–30
- signing, 21
- starting the Jenkins server, 22, 23
- System Configuration page, 25
- using the .WAR file on Linux, 23
- Jenkins controller, 244
- Jenkins creation, 315–318
- Jenkins Credentials Provider, 85, 88
- Jenkins dashboard, 396, 399
- Jenkins distributed builds
  - adding node entry, 344–350
  - adding public entry, 350, 351
  - architecture, 340
  - changing permissions, 351, 352
  - connecting configuration, 342
  - connecting master and slaves, 341, 342
  - connection checking, 353
  - credentials entry, 343, 344
  - launching new node, 353
  - SSH build agent’s plugin, 342
  - sshd service, 353
  - SSH public-private key pair, 343
- Jenkinsfile
  - advantages, 320
  - definitions, 320
- Jenkins creation, 321–328
- Jenkins UI, 320
- saving pipeline code, 328–331

## INDEX

- Jenkinsfile (*cont.*)
  - scripted *vs.* declarative
    - syntax, 321
  - triggering, GitLab
    - Webhook, 331–335
  - writing pipeline, 321
- JENKINS\_HOME, 17
- Jenkins home directory, 26
- Jenkins installation directory, 17
- Jenkins job (*see* Job in Jenkins)
- Jenkins job configuration
  - build triggers, 133–138
  - Credentials dropdown, 129, 130
  - Credentials Manager, 131, 132
  - description, 118
  - discard old builds, 118–120
  - display name, 125, 126
  - executing concurrent builds, 122, 123
  - GitlabCredentials
    - Domain, 82, 83
  - job parameters, 121–123
  - log rotation, 126
  - quiet period, 123, 124
  - Source Code Management, 127
  - workspace, 125
- Jenkins machine, 339
- Jenkins master, 340
- Jenkins master-slave
  - architecture, 340
- Jenkins pipeline, 245, 249
  - CI/CD process, 238
  - code and use, 241
  - concepts, 239
- definition, 237
- scripting references, 238
- stages, 239
- types, 239
- Shared libraries, 405, 407
- Jenkins plugins
  - advanced tab, 38–40
  - disabling, 38
  - Email Extension plugin, 34
  - Git plugin, 34
  - installation, 35
  - installed tab, 38
  - Plugin Manager page, 36
  - troubleshooting installation problems, 40–44
  - updated versions, 36
  - available tab, 36
- Jenkins Project, 115
- Jenkins remote access API
  - creating new jobs, 395, 396
  - fetching configuration, existing jobs, 394, 395
- Jenkins server, Python-Jenkins, 400, 401
- PATH environment variable, 393
- Python-Jenkins Package
  - Libraries, 401–404
- REST APIs, 392
- REST-like style, 392
- shared libraries, 405–409
- triggering non-parameterized jobs, 398, 399
- triggering parameterized jobs, 396–398

types, 392  
 web browsers, 393  
**Jenkins** Remote APIs (REST-API),  
 375, 392  
**JenkinsSharedLibrary**, 407  
**Jenkins slave**, 340  
**Jenkins system**, 24, 100  
**Jenkins.xml file**, 18, 20  
**JNLP agent**, 342  
**Job configuration page**, 118,  
 127, 142  
**JobCreatedUsingPython**  
 Jenkins, 402  
**Job in Jenkins**  
 build history for an executed  
 job, 139, 140  
 build step, 116  
 clear a job's workspace, 144  
 create a job, 117, 118  
 delete an existing Jenkins  
 job, 145  
 edit an existing Jenkins  
 job, 141–143  
 execution status, 139  
 job configuration page, 118,  
 127, 142, *See also Jenkins*  
 job configuration  
 trigger, 115  
 view a job's workspace, 143, 144

**L**

**Lightweight Directory Access  
 Protocol (LDAP)**, 64–67

Local network (LAN), 332  
**Local repository**, Maven's, 162,  
 163, 174

**M**

**Master to slave configuration**, JNLP  
 connection requests, 354  
**Jenkins URL**, 355  
**JNLPNodeJob**, 357  
 new node entry  
 creation, 355–357  
**Maven build tool**, 148, 150, 174  
**Maven CLI commands**, 166, 172  
**Maven configuration**, 48–53, 256  
**Maven Integration plugin**, 34  
**Maven project directory**, 159, 160  
**Maven Projects option**, 221  
**Maven settings xml**, 174  
**Multiplication function**, 221, 225  
**MultiplicationFunction**  
 branch, 231  
**MyMaven**, 256

**N**

**New free-style job**, node1  
 agent, 354  
**New Merge Request page**, 226  
**New Repository menu option**, 277  
**Nexus repository**, 191, 192, 226,  
 227, 250, 258, 261  
 accessing, 194  
**CalculatorAPI.jar**, 197

## INDEX

- Nexus repository (*cont.*)  
    free-style job, 198–205  
    hosted repository, 195  
    installation, 191  
    installing Nexus as a Windows service, 193  
    integrating Maven, 195, 196  
    repository system, 192, 193
- O**  
ObjectRepositories, 270  
OpenSSH, 213
- P**  
Parameterized checkbox, 301  
parameterized Jenkins Freestyle jobs, 283  
Parameterized job, 288  
Parameterized pipeline job, 298  
Parameterized Trigger plugin, 284  
PATH environment variable, 166, 169–171, 401  
Path parameters, 393  
PEM argument, 380  
Pipeline, 11  
Pipeline as code (*see* Jenkinsfile)  
Pipeline execution, 238  
Pipeline job, 244  
Pipeline script, 301  
Pipeline section, 246  
Pipeline Syntax, 252, 255  
Plugin, 33, *See also* Jenkins plugins  
    Plugin Manager, 342  
Post build actions, 116, 234  
Preferences page, 215  
Project-based matrix authorization, 109, 111, 113  
Project Members page, 210  
Project Object Model (POM), 161  
Push Event radio control, 337  
Pycharm, 400  
Pytest, 274  
Python files, 271  
Python Interpreter, 400  
Python-Jenkins Package Libraries, 401–404  
Python libraries, 275  
Python package, 289  
Python package manager, 272  
Python project, 269  
Python REST API, 400  
Python selenium library, 272, 275, 276  
Python Selenium tests, 398  
Python virtual environment, 275  
Python wrappers, 400
- Q**  
Query parameters, 392
- R**  
Random numbers, 64  
Rapid Application Development (RAD), 1

Remote GitLab repository, 261  
 Remote repository, 225  
 Repo checkbox, 308  
 Repository master branch, 233  
 Repository page, 282  
 RESTful API services, 147  
 Role-Based Authorization Strategy  
     Plugin, 93–95

## S

Scripted pipeline, 241, 251,  
     252, 321  
     node block, 241  
     stage blocks, 242  
 Selenium, 307  
 Selenium E-E tests, 291  
 Selenium Python library, 290  
 Selenium WebDriver, 261, 269  
 Selenium WebDriver Python  
     libraries, 274  
 Sequential execution, 339  
 Shared libraries, Jenkins Pipeline  
     CalculatorAPI.jar file, 405  
     configuration, 407, 408  
     groovy file, 405, 406  
     pipeline job creation, 408, 409  
     running, 409  
 Simple pipeline, 243  
 SmokeTest option, 305  
 SMTP Authentication  
     checkbox, 230

Snippet Generator, 259, 299  
 Source code management (SCM)  
     tools, 33, 124, 127, 133, 136,  
     137, 241  
 SSH authentication, 69, 211, 307,  
     378, 389  
     apply, 212  
     public and private key, 212  
     public-private key pair, 214  
     ssh-keygen command, 214  
     technique, 212  
 SSH credentials, 228  
 SSH key pair, 213, 217, 278  
 SSH public-private key pair, 343  
 SSH server, 63  
 Stable code, 3  
 String interpolation, 248, 249  
 String parameter, 285  
 System Configuration page, 229  
 SystemRoot, 349

## T

TestCalculatorWebApplication,  
     293, 372  
 Test-driven development (TDD), 2  
 TestNG, 160  
 Third-party libraries,  
     148, 149, 161  
 Tomcat, 147  
 Troubleshooting installation  
     problems, 40–44

## INDEX

### **U**

UI automation testing, 268  
UI automation tool, 307  
UI interface, 262  
User Credentials Provider, 85, 86

### **V**

Version control systems, 178  
Virtual Directory, 364  
Virtual environment, 289

### **W, X, Y, Z**

Web alias, 364  
Web application,  
    147, 266, 276, 283  
Web Application Archive files  
    (WAR), 15  
Webhooks, 331  
Windows Credentials, 311  
Workspace, 125  
World Wide Web Publishing  
    service, 371