

Mitesh Soni

Jenkins Essentials

Second Edition

Setting the stage for a DevOps culture



Packt

Jenkins Essentials

Second Edition

Setting the stage for a DevOps culture

Mitesh Soni

Packt >

BIRMINGHAM - MUMBAI

Jenkins Essentials

Second Edition

Copyright © 2017 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: July 2015

Second edition: June 2017

Production reference: 1280617

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78847-106-0

www.packtpub.com

Credits

Author
Mitesh Soni

Copy Editor
Saafis Editing

Reviewers
Stefan Lapers
Devin Young
Javier Delgado

Project Coordinator
Shweta H Birwatkar

Commissioning Editor
Vijin Boricha

Proofreader
Safis Editing

Acquisition Editor
Divya Poojari

Indexer
Mariammal Chettiyar

Content Development Editor
Deepti Thore

Graphics
Tania Dutta

Technical Editor
Sneha Hanchate

Production Coordinator
Nilesh Mohite

About the Author

Mitesh Soni is an avid learner with 10 years' experience in the IT industry. He is an SCJP, SCWCD, VCP, IBM Urbancode, and IBM Bluemix certified professional. He loves DevOps and cloud computing and he also has an interest in programming in Java. He finds design patterns fascinating. He believes "a picture is worth a thousand words."

He occasionally contributes to etutorialsworld.com. He loves to play with kids, fiddle with his camera, and take photographs at Indroda Park. He is addicted to taking pictures without knowing many technical details. He lives in the capital of Mahatma Gandhi's home state.

Mitesh has authored the following books with Packt:

- DevOps Bootcamp
- Implementing DevOps with Microsoft Azure
- DevOps for Web Development
- Jenkins Essentials
- Learning Chef

"I've missed more than 9,000 shots in my career. I've lost almost 300 games. 26 times, I've been trusted to take the game-winning shot and missed. I've failed over and over and over again in my life. And that is why I succeed."—Michael Jordan.

I've always thanked a lot of people who have been instrumental in contributing to my life's journey up to now, but I guess it's time to really acknowledge that One person who has been with me as long as I can remember.

With this book, I would like to thank the one and only invisible yet omnipresent Almighty. We share a mutual love and hate relationship and I really value it. You were always there equally during my good and bad times and without you, I wouldn't have made it this far!

Last but not the least, I want to thank all who taught me how to love myself, first!

About the Reviewers

Stefan Lapers started his career almost 20 years ago as an IT support engineer and quickly grew into Linux/Unix system engineering and software development.

Over the years he accumulated experience deploying and maintaining hosted application solutions while working for great customers such as MTV, TMF, and many more. In more recent years, he was involved in multiple development projects and their delivery as a service on the internet.

In his spare time, he enjoys his family and building/flying RC helicopters.

Devin Young graduated with a BS in sports management from Ohio State University and somehow wound up working as a software engineer shortly afterwards. He specializes in DevOps and site reliability and is particularly fond of automation and real-time applications.

He grew up as a competitive jump roper, leading him to create the mobile app RopeRacer, which was launched on iOS in March 2015. The app has become a success in the world of jump rope and is now used in tournaments around the United States.

Devin is currently an MBA candidate at the TCU Neeley School of Business.

Javier Delgado is an automation fanatic, continuous tasks (inspection, testing, delivery), evangelist, and perpetual new-knowledge addict.

He works as DevOps and Jenkins internal information well at Stratio Big Data Inc.

www.PacktPub.com

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www.packtpub.com/mapt>

Get the most in-demand software skills with Mapt. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Customer Feedback

Thanks for purchasing this Packt book. At Packt, quality is at the heart of our editorial process. To help us improve, please leave us an honest review on this book's Amazon page at <https://www.amazon.com/dp/1788471067>.

If you'd like to join our team of regular reviewers, you can e-mail us at customerreviews@packtpub.com. We award our regular reviewers with free eBooks and videos in exchange for their valuable feedback. Help us be relentless in improving our products!

I would like to dedicate this book to lot of people who gave me a ray of hope amidst darkness. I would like to dedicate this book to Shreyansh (Shreyu – my sister Jigisha's baby boy) who showed me the power of innocence and smiles, Vinay Kher for his blessing, my parents who are always there silently praying for me, Simba (Priyanka Agashe) for supporting and encouraging me all the time and forcing me to believe in myself Indian Army and all brave soldiers in uniform for protecting us.



Table of Contents

Preface	1
<hr/>	
Chapter 1: Exploring Jenkins	7
Introduction of Jenkins 2	8
Features	10
Installation of Jenkins 2	11
Jumpstart tour of the Jenkins dashboard	22
Configuration settings in Jenkins	24
Overview of the CI/CD pipeline	27
Summary	29
Chapter 2: Installation and Configuration of Code Repository and Build Tools	30
<hr/>	
Overview of Jenkins	31
Installing Java and configuring the environment variables	32
Installing Java on Windows 10	32
Installing Java on CentOS	32
Configuring environment variables	34
Installing and configuring Ant	35
Configuring Ant in Windows	35
Configuring Ant in CentOS	35
Installing Maven	36
Configuring Ant, Maven, and JDK in Jenkins	36
First job in Jenkins	40
Installing and configuring the Git repository on CentOS	51
Creating a new build job in Jenkins with Git and GitHub	55
Eclipse and Jenkins integration	59
Summary	65
Chapter 3: Managing Code Quality and Notifications	66
<hr/>	
Jenkins 2.x integration with Sonar 6.3	68
Quality Gate plugin	91
Email notifications on build status	97
Summary	102
Chapter 4: Continuous Integration with Jenkins	103
<hr/>	
Dashboard View plugin	104

Creating and configuring a build job for a Java application with Ant	106
Creating and configuring a build job for a Java application with Maven	111
Summary	117
Chapter 5: Continuous Delivery - Implementing Automated Deployment	118
An overview of Continuous Delivery and Continuous Deployment	119
Installing Tomcat	120
Deploying a war file from Jenkins to Tomcat	120
Deploying a WAR file from Jenkins to AWS Elastic Beanstalk	126
Deploying a war file from Jenkins to Microsoft Azure App Services	136
Summary	144
Chapter 6: Continuous Testing - Functional and Load Testing with Jenkins	145
Functional testing with Selenium	146
Load testing with Apache JMeter	167
Summary	174
Chapter 7: Build Pipeline and Pipeline as a Code	175
Build Pipeline	176
Upstream and downstream jobs	178
Overview of pipeline as a code	189
Pipeline as a code - implementation	190
Promoted builds	209
Summary	211
Chapter 8: Managing and Monitoring Jenkins	212
Managing Jenkins master and slave nodes	213
Monitoring Jenkins with JavaMelody	220
Managing job-specific configurations - backup and restore	224
Managing disk usage	229
Build job-specific monitoring with the Build Monitor plugin	231
Audit Trail plugin-overview and usage	234
Workspace Cleanup plugin	235
Conditional Build Step plugin	236
EnvInject plugin	238
Summary	240
Chapter 9: Security in Jenkins	241
User management	242
Role-based security	248
Project-based security	252

Summary	256
Index	<u>257</u>

Preface

Jenkins has been used specifically for Continuous Integration over the years. Continuous Integration systems are a vital part of Agile teams because they help enforce the principles of Agile development.

Jenkins, a continuous build tool, enables Agile teams to focus on work and innovations by automating the build, artifact management, and deployment processes, rather than worrying about manual processes. However, the release of Jenkins 2.0 and later versions are focused on Continuous Delivery.

Jenkins is an open source automation server. Continuous Integration is a significant part of DevOps culture and hence many open source and commercial tools for Continuous Delivery utilizes Jenkins for a complete product.

DevOps is a buzzword in 2015 and for coming years as per Market trends by various research firms. Continuous Integration is a significant part of DevOps culture and hence the trend to use Jenkins will increase in future. If Continuous Integration is the base, then Continuous Delivery is the topping. Jenkins supports and focus more on end-to-end automation of application life cycle management system.

What this book covers

Chapter 1, *Exploring Jenkins*, describes in detail the basics of Continuous Integration and an overview of Jenkins. It describes the recent growth of importance of Continuous Integration as a practice to cultivate DevOps culture. This chapter also describes the installation and configuration of Jenkins. We are going to take a whistle-stop tour through some of the key features of Jenkins, and plugin installation as well.

Chapter 2, *Installation and Configuration of Code Repositories and Build Tools*, describes in detail how to prepare the runtime environment for application life cycle management and configure it with Jenkins – the open source continuous integration tool. It will cover how to integrate Eclipse and Jenkins, so builds can be run from Eclipse as well.

Chapter 3, *Managing Code Quality and Notifications*, will cover how to integrate static code analysis behavior into Jenkins. Code quality is an extremely vital feature that impacts on applications' effectiveness and by integrating it with Sonar, users get insights into problematic portions of code. This chapter also covers email notifications on build status.

Chapter 4, *Continuous Integration with Jenkins*, describes in detail how to create and configure build jobs in Java, how to run build jobs, unit test cases using Ant, and Maven build tools. It covers all aspects of running a build to create a distribution file or war file for deployment.

Chapter 5, *Continuous Delivery - Implementing Automated Deployment*, provides insights into how functional testing and load testing can be performed and how they can be integrated with Jenkins to adopt the Continuous Testing practices of DevOps culture.

Chapter 6, *Continuous Testing - Functional and Load Testing with Jenkins*, takes one step forward in the DevOps pipeline by deploying artifacts to local or remote application servers. It gives insights into automated deployment and Continuous Delivery processes, and also covers how to deploy applications to a public cloud platform using Jenkins.

Chapter 7, *Build Pipeline and Pipeline as a Code*, will cover how to orchestrate build job to execute them in a specific sequence. We will cover the Build Pipeline plugin and the Pipeline as a Code feature that is available in Jenkins 2 and later.

Chapter 8, *Managing and Monitoring Jenkins*, gives insight into the management of Jenkins nodes and monitoring them with Java Melody, in order to provide details on the utilization of resources. It also covers how to monitor build jobs configured for Java-or .NET-related applications, and managing those configurations by keeping backups of them.

Chapter 9, *Security in Jenkins*, will cover security management options available in Jenkins. This will help to perform user management, authentication, and authorization, including matrix-based security and role-based access.

What you need for this book

This book is for beginners. This book assumes that you are familiar with at least the Java programming language. Knowledge of core Java and JEE is essential if you are to use this book to gain better insights. Having a strong understanding of programming logic will provide you with the background to be productive with Jenkins, while using plugins or writing commands for the shell.

As the application development life cycle will cover a lot of tools, it is essential to have some knowledge of repositories such as svn, git, and so on, IDE tools such as Eclipse, and build tools such as Ant and Maven.

Knowledge of code analysis tools will make jobs easier to configur and integrate, however, it is not required in order to complete the exercises presented in this book. Most of the configuration steps are mentioned clearly. SonarQuve version 6.3 is used for code analysis.

You will be walked through the steps required to install Jenkins on a Windows and Linux-based host. In order to be immediately successful, you will need administrative access to a host that runs a modern version of Windows and Linux; Windows 10 will be used for demonstration purposes. If you are a more experienced reader, then a recent release of almost any distribution will work just as well (but you may be required to do a little bit of extra work that is not outlined in the book).

Additionally, you will need access to the internet to download plugins that you do not already have, as well as an installation of Jenkins. Any normal hardware configuration is good enough, such as 4 GB RAM and 500 GB hard drive.

Who this book is for

This book assumes that you are familiar with at least java programming language. Knowledge of core java and JEE is essential considering this book to gain better insight. Having a strong understanding of program logic will provide you with the background to be productive with Jenkins while using plugins of writing commands for shell.

As application development life cycle will cover lot of tools in general; it is essential to have some knowledge of repositories such as svn, git etc; IDE tools such as Eclipse; build tools such as ant and maven.

Knowledge of code analysis tools will make job easier in configuration and integration, however it is not extremely vital to perform exercises given in the book. Most of the configuration steps are mentioned clearly.

Additionally, you will need access to the Internet to download plugins that you do not already have, as well as an installation of the Jenkins.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:

A block of code is set as follows:

```
#run the model  
model <- OneR(train_data, frisked ~ ., verbose = TRUE)  
#summarize the model  
summary(model)  
#run the sql function from the SparkR package  
SparkR::sql("SELECT sample_bin , count(*)  
\nFROM out_tbl group by sample_bin")
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
#note we are specifying the SparkR filter, not the dplyr filer  
head(SparkR::filter(out_sd1,out_sd1$sample_bin==1),1000)
```

Any command-line, (including commands at the R console) input or output is written as follows:

```
> summary(xchurn)
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen."

Warnings or important notes appear in a box like this.



Tips and tricks appear like this.



Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/Jenkins-Essentials-Second-Edition>. We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output.

You can download this file from

https://www.packtpub.com/sites/default/files/downloads/JenkinsEssentialsSecondEdition_ColorImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

1

Exploring Jenkins

Jenkins is an open source automation server (after Jenkins 2.0 was released) written in Java. It was one of the most popular **Continuous Integration (CI)** tools used to build and test different kinds of projects. Now, it is also used for **Continuous Delivery (CD)** after Jenkins 2.0. This chapter describes in detail the basics of CI and overview of Jenkins 2. It describes the importance of CI and CD as a practice to cultivate DevOps culture in recent times.

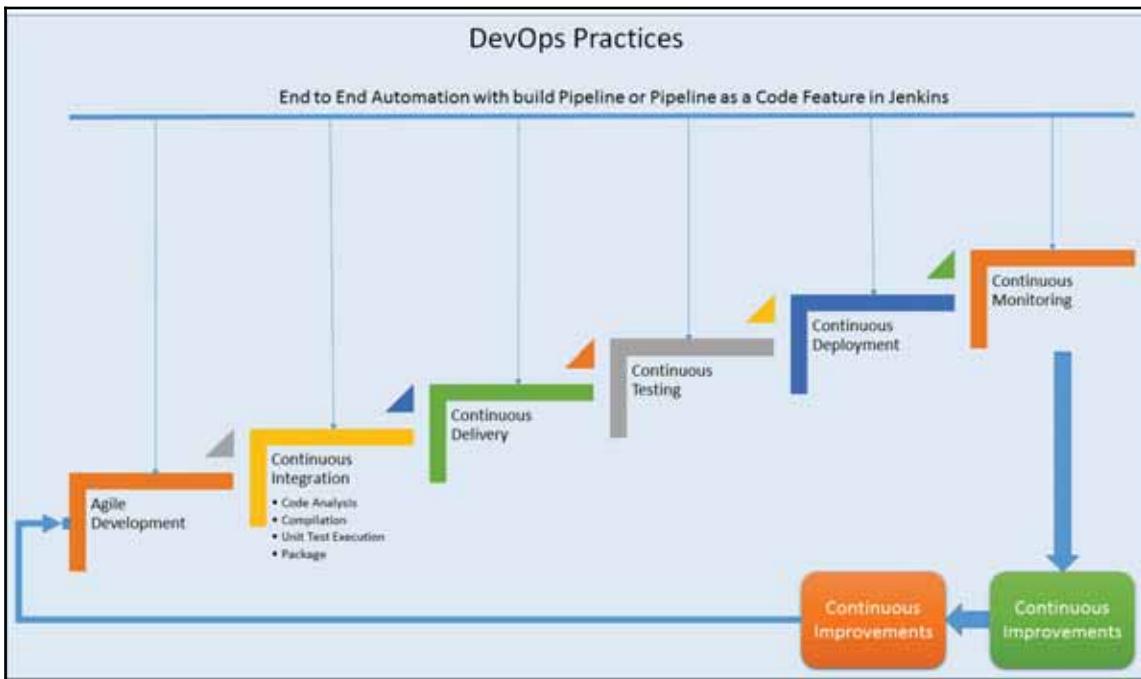
This chapter also describes installation and configuration of Jenkins 2. We are going to take a jumpstart tour through some of the key features of Jenkins and plugin installations as well.

To be precise, we will discuss the following topics in this chapter:

- Introduction of Jenkins 2 and its features
- Installation of Jenkins 2
- Jumpstart tour of Jenkins dashboard
- Configuration settings in Jenkins
- Overview of CICD pipeline

Lets get started! On your marks, get set, go!

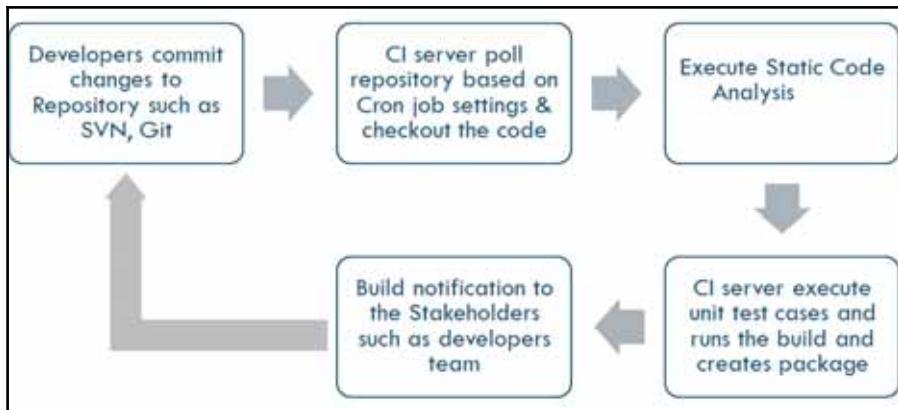
In this chapter, we will see what Jenkins 2 is and the new features introduced regarding CI as a part of our DevOps journey. We will cover the following steps to complete our DevOps journey. Each chapter is a stepping stone to reach the next one. It is always good to have an incremental approach so we can measure our success and feel the pain points as well to realize the value of this journey:



At the end of this chapter, we will know essential things about Jenkins 2 and how it is a game changer in terms of CD. It is no longer a CI server. It is on its way to becoming a mature product in the category of automation servers by focusing on Continuous Delivery after Jenkins 2.0 is released.

Introduction of Jenkins 2

Let's first understand what CI is. CI is one of the most popular application development practices in recent times. Developers integrate bug fixes, new feature development, or innovative functionality in a code repository. The CI tool verifies the integration process with an automated build and test to detect issues with current sources of an application and provide quick feedback:



Jenkins is a simple, extensible, and user friendly open source tool that provides continuous integration services for application development. Jenkins supports SCM tools such as Git, Subversion, Star Team, and CVS, AccuRev. Jenkins can build Apache Ant and Apache Maven-based projects.

The concept of plugins makes Jenkins more attractive, easy to learn, and easy to use. There are various categories of plugins available, such as the following:

- Source code management
- Slave launchers and controllers
- Build triggers
- Build tools
- Build notifiers
- Build reports
- Other post-build actions
- External site/tool integrations
- UI plugins
- Authentication and user management
- Android development
- iOS development
- .NET development
- Ruby development
- Library plugins

Jenkins defines interfaces or abstract classes that model a facet of a build system. Interfaces or abstract classes agree on what needs to be implemented, and Jenkins uses plugins to extend those implementations.

With Jenkins 2, the focus is also on CD where the application is deployed in the specific environment using an automated approach. Jenkins 2 is a clear signal regarding the focus on both CI and CD best practices of DevOps culture and not on CI only.

Features

Jenkins is one of the most popular automation servers in the market, and the reasons for its popularity are some of the following features:

- Easy installation on different operating systems, Arch Linux, FreeBSD, Gentoo, MacOS X, openBSD, openSUSE, RedHAT/Fedora/CentOS, Ubuntu/Debian, Windows, and it is also available for Docker and as generic Java packages too.
- Easy upgrades: Jenkins has very speedy release cycles (long-term support and weekly releases).
- Simple and easy to use user interface in Jenkins 2.x -.
- Set of suggested plugins at the time of installation.
- Improved new item page.
- Improved job configuration page with easy navigation.
- Jenkins 2 supports pipelines as code that uses **domain-specific language (DSL)** to model application delivery pipelines as code; we can utilize the pipelines as code and keep them in a repository and maintain versions in a similar way to source code.
- Easily extensible with the use of third-party plugins: there are over 400 plugins.
- Easy to configure the setup environment in the user interface. It is also possible to customize user interface as you wish.
- Master slave architecture supports distributed builds to reduce the load on CI servers.
- Build scheduling based on cron expressions.
- Shell and Windows command execution that makes any command-line tool integration in the pipeline very easy.
- Notification support related to build status.

Installation of Jenkins 2

Let's start with Jenkins 2.x installation. Go to <https://jenkins.io/> and click on the **Download** button to download packages for installation of Jenkins:



At <https://jenkins.io/download/>, we get two sections. One is for **Long-term Support (LTS)** that is selected after every 12 weeks from regular releases as the stable release from that duration.

Another section is **Weekly** release, which has bug fixes and is made available to users and developers.

We will use the **Generic Java Package (.war)** file in our installation of Jenkins as shown in the following screenshot.

The reason for selecting the **.war** file for the Jenkins installation is its ease of use across operating systems. Jenkins is written in Java, and in any case, to execute Jenkins we need the latest Java version installed on our system. For Jenkins Installation, Java 8 is recommended. It is recommended to have 1-GB of memory.

Verify the Java installation by using the `java -version` command in the command prompt or terminal based on the operating system.

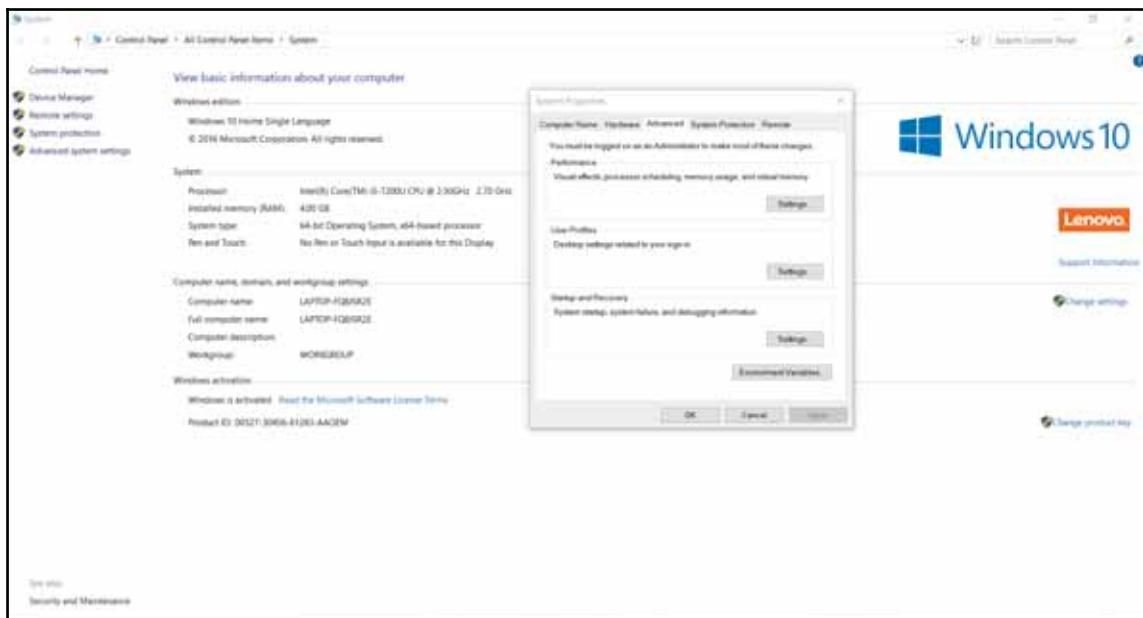
Download **Generic Java Package (.war)** from <https://jenkins.io/download/> on the local system as follows:

The screenshot shows the Jenkins homepage with the title "Getting started with Jenkins". It highlights two release lines: "Long-term Support (LTS)" and "Weekly". The "Weekly" section is described as producing bug fixes and features weekly. Below each section is a "Changelog" link and a "Past Releases" link. A large sidebar on the right lists various download options for Jenkins 2.46.3, categorized under "Download Jenkins 2.46.3 for:". The categories include Arch Linux, Docker, FreeBSD, Gentoo, Mac OS X, OpenBSD, openSUSE, Red Hat/Fedora/CentOS, Ubuntu/Debian, Windows, and "Generic Java package (.war)".

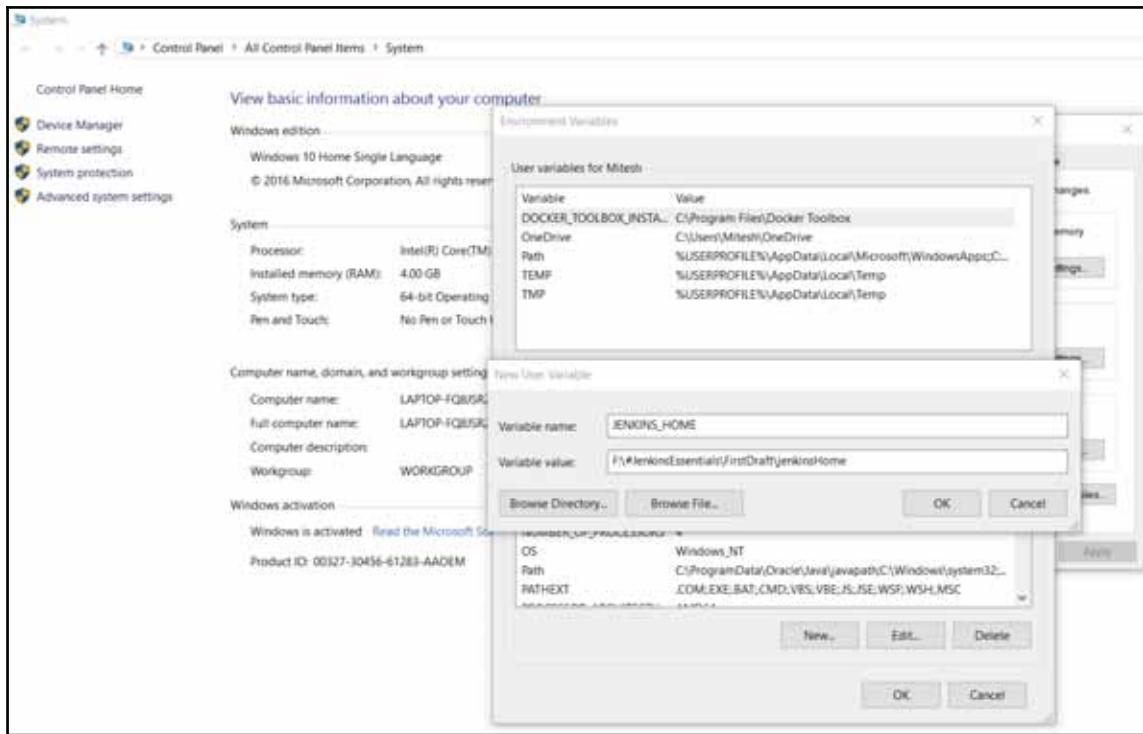
Download Jenkins 2.46.3 for:	
Arch Linux	
Docker	
FreeBSD	
Gentoo	
Mac OS X	
OpenBSD	
openSUSE	
Red Hat/Fedora/CentOS	
Ubuntu/Debian	
Windows	
Generic Java package (.war)	

Before we start Jenkins, we will set the `JENKINS_HOME` environment variable. When we install and configure Jenkins, all the files reside in the `jenkins` directory by default. We often need to change the default location for our convenience. We can do that by setting the `JENKINS_HOME` environment variable. Follow the following steps:

1. To set the `JENKINS_HOME` environment variable in Windows 10, go to **Control Panel | All Control Panel Items | System** and click on **Advanced system settings | Advanced | Environment Variables...**. Please see the manual for other operating systems to set **Environment Variables...**:



2. Click on **New**. Enter the variable name and location and click **OK**:



3. Now our **JENKINS_HOME** is set so Jenkins will use that directory to store all configuration files.
4. Open the command prompt or terminal (depending on your operating system) and execute the following command:

```
Java -jar Jenkins.war
```

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All Rights reserved.

F:\jenkinsEssentials\FirstDraft>java -jar Jenkins.war
Running from: F:\jenkinsEssentials\FirstDraft\jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
[23]May 20, 2017 8:07:36 PM Hain delate\winstoneTempContents
WARNING: Failed to delete the temporary file C:\Users\Himesh\AppData\local\Temp\winstone\jenkins.war
[24]May 20, 2017 8:07:36 PM org.eclipse.jetty.util logging initialized
INFO: Logging initialized with level org.eclipse.jetty.util.log.JavaUtilLogging
May 20, 2017 8:07:36 PM winstone.Logger logInternal
INFO: Beginning extraction from war file
[25]May 20, 2017 8:07:44 PM org.eclipse.jetty.server.handler.ContextHandler setContextPath
WARNING: Empty contextPath
[26]May 20, 2017 8:07:49 PM org.eclipse.jetty.server.Server doStart
INFO: jetty-9.4.12-SNAPSHOT
May 20, 2017 8:07:49 PM org.eclipse.jetty.webapp.StandardDescriptorProcessor visitServlet
INFO: Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
May 20, 2017 8:07:49 PM org.eclipse.jetty.server.session.DefaultSessionIdManager doStart
INFO: DefaultSessionIdManager workerName=none
May 20, 2017 8:07:49 PM org.eclipse.jetty.server.session.DefaultSessionIdManager doStart
INFO: No SessionScavenger set, using defaults
May 20, 2017 8:07:50 PM org.eclipse.jetty.server.session.Housekeeper startScavenging
INFO: Scavenging every 600000ms
Jenkins home directory: F:\jenkinsEssentials\FirstDraft\jenkinsHome found at: EnvVars.masterEnvVars.get("JENKINS_HOME")
May 20, 2017 8:07:50 PM org.eclipse.jetty.server.handler.ContextHandler doStart
INFO: Started w:\Webcall256ef\file:///F:/233jenkinsEssentials\FirstDraft\jenkinsHome/war/_AVAILABLE{F:\jenkinsEssentials\FirstDraft\jenkinsHome.war}
May 20, 2017 8:07:50 PM org.eclipse.jetty.server.AbstractConnector doStart
INFO: Started serverConnector@8080[HTTP/1.1][0.0.0.0:8080]
May 20, 2017 8:07:50 PM org.eclipse.jetty.server.Server doStart
INFO: Started WebApp
May 20, 2017 8:07:45 PM winstone.Logger logInternal
INFO: Winstone Servlet Engine v4.0 running: controlPort=disabled
May 20, 2017 8:07:46 PM jenkins.InitReactorRunner$1 onAttained
INFO: Started Initialization
May 20, 2017 8:07:46 PM jenkins.InitReactorRunner$1 unattained
INFO: Listed all 46 plugins
May 20, 2017 8:07:47 PM jenkins.InitReactorRunner$1 unattained
INFO: Prepared all 46 plugins
May 20, 2017 8:07:47 PM jenkins.InitReactorRunner$1 onAttained
```

5. This is a fresh installation of Jenkins, so initial setup is required. Note the password:

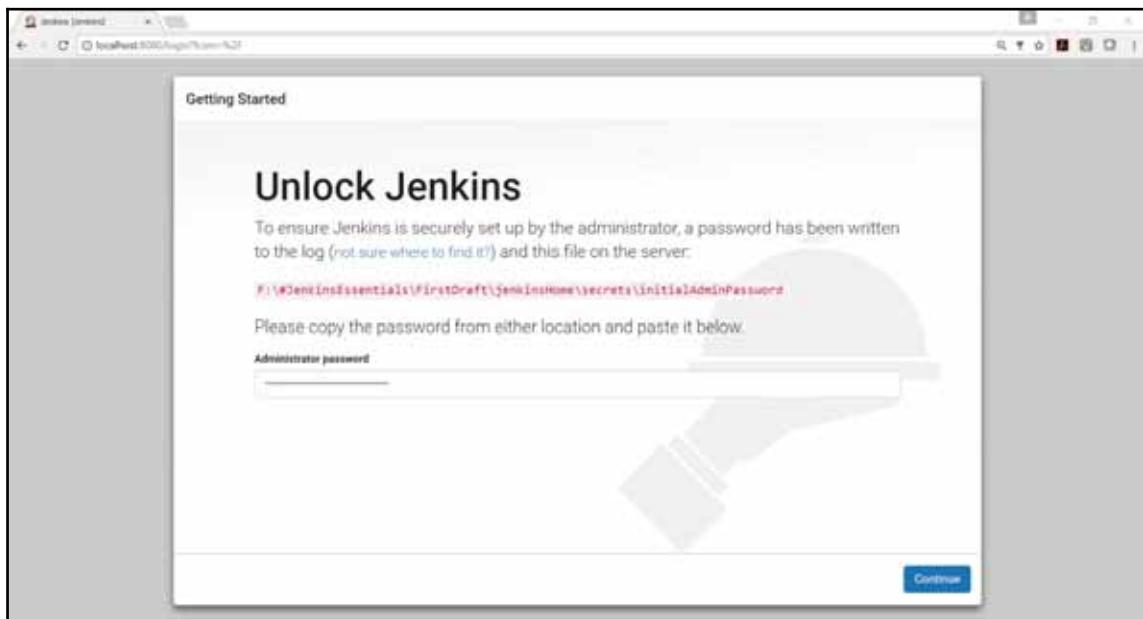
```
dot of factory hierarchy
May 20, 2017 8:07:56 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.web.context.support.StaticWebApplicationContext@18078200: display name [Root WebApplicationContext]; startup date [Sat May 20, 2017 8:07:56 PM org.springframework.context.support.AbstractApplicationContext obtainFreshBeanFactory
INFO: Bean factory for application context [org.springframework.web.context.support.StaticWebApplicationContext@18078200]: org.springframework.beans.factory.support.DefaultListableBeanFactory@6f2fc47
May 20, 2017 8:07:56 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@6f2fc47; defining beans [filter,legacy]; root of factory hierarchy
May 20, 2017 8:07:56 PM jenkins.install.SetupWizard init
INFO

*****
***** Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
***** 75495e17d897498bbdbb53d77122#008s
***** This may also be found at: F:\Jenkins\jessentials\FirstDraft\JenkinsHome\secrets\initialAdminPassword
*****



***** May 20, 2017 8:08:05 PM Hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
May 20, 2017 8:08:09 PM hudson.WebAppMain$3 run
INFO: Jenkins is fully up and running
May 20, 2017 8:08:10 PM Hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
May 20, 2017 8:08:12 PM Hudson.model.DownloadServices$Downloadable load
INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
INFO: Obtained the updated data file for hudson.tools.JDKInstaller
May 20, 2017 8:08:15 PM Hudson.model.DownloadServices$Downloadable load
INFO: Obtained the updated data file for hudson.tools.AsyncPeriodicWork$1 run
May 20, 2017 8:08:15 PM Hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata. 26,342 ms
```

- Once Jenkins is fully up and running, visit `http://localhost:8080` and it will open the **Getting Started** page to **Unlock Jenkins**. Give the password we copied from the terminal or go to the location given in the dialog box. Copy the password from there and click on **Continue**:



7. Click on **Install** to see the suggested plugins. If we are behind the proxy, then another dialog box will pop up before this page to provide proxy details:



8. Wait while all the plugins are installed properly:

Getting Started

Getting Started

✓ Folders Plugin	✓ OWASP Markup Formatter Plugin	⌚ build timeout plugin	⌚ Credentials Binding Plugin	** boonycastle API Plugin Folders Plugin ** Structs Plugin ** JUnit Plugin OWASP Markup Formatter Plugin PAM Authentication plugin ** windows Slaves Plugin ** Display URL API Jenkins Mailer Plugin LDAP Plugin ** Token Macro Plugin ** External Monitor Job Type Plugin ** Icon Shim Plugin Matrix Authorization Strategy Plugin ** Script Security Plugin ** Matrix Project Plugin Jenkins build timeout plugin
⌚ Timestamper	⌚ Workspace Cleanup Plugin	⌚ Ant Plugin	⌚ Gradle Plugin	
⌚ Pipeline	⌚ GitHub Branch Source Plugin	⌚ Pipeline: GitHub Groovy Libraries	⌚ Pipeline: Stage View Plugin	
⌚ Git plugin	⌚ Subversion Plug-in	⌚ SSH Slaves plugin	✓ Matrix Authorization Strategy Plugin	
✓ PAM Authentication plugin	✓ LDAP Plugin	⌚ Email Extension Plugin	✓ Mailer Plugin	
** - required dependency				
Jenkins 2.61				

9. Verify the green tick boxes for all the plugins that have been installed successfully:

Getting Started

Getting Started

✓ Folders Plugin	✓ OWASP Markup Formatter Plugin	✓ build timeout plugin	✓ Credentials Binding Plugin	** Credentials Plugin ** Pipeline: Step API ** Plain Credentials Plugin Credentials Binding Plugin Timestamper ** SCM API Plugin ** Pipeline: API ** Pipeline: Supporting APIs ** Durable Task Plugin ** Pipeline: Nodes and Processes ** Resource Disposer Plugin Jenkins Workspace Cleanup Plugin Ant Plugin Gradle Plugin ** Pipeline: Milestone Step ** JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI) plugin ** Pipeline: Input Step ** JavaScript GUI Lib: ACE Editor bundle plugin ** Pipeline: SCM Step ** Pipeline: Groovy ** . required dependency
✓ Timestamper	✓ Workspace Cleanup Plugin	✓ Ant Plugin	✓ Gradle Plugin	
✗ Pipeline	✗ GitHub Branch Source Plugin	✗ Pipeline: GitHub Groovy Libraries	✗ Pipeline: Stage View Plugin	
✗ Git plugin	✗ Subversion Plug-in	✗ SSH Slaves plugin	✓ Matrix Authorization Strategy Plugin	
✓ PAM Authentication plugin	✓ LDAP Plugin	✗ Email Extension Plugin	✓ Mailer Plugin	

Jenkins 2.61

10. Once all plugins are installed successfully, create the first admin user and click on **Save and Finish**:

Getting Started

Create First Admin User

Username:

Password:

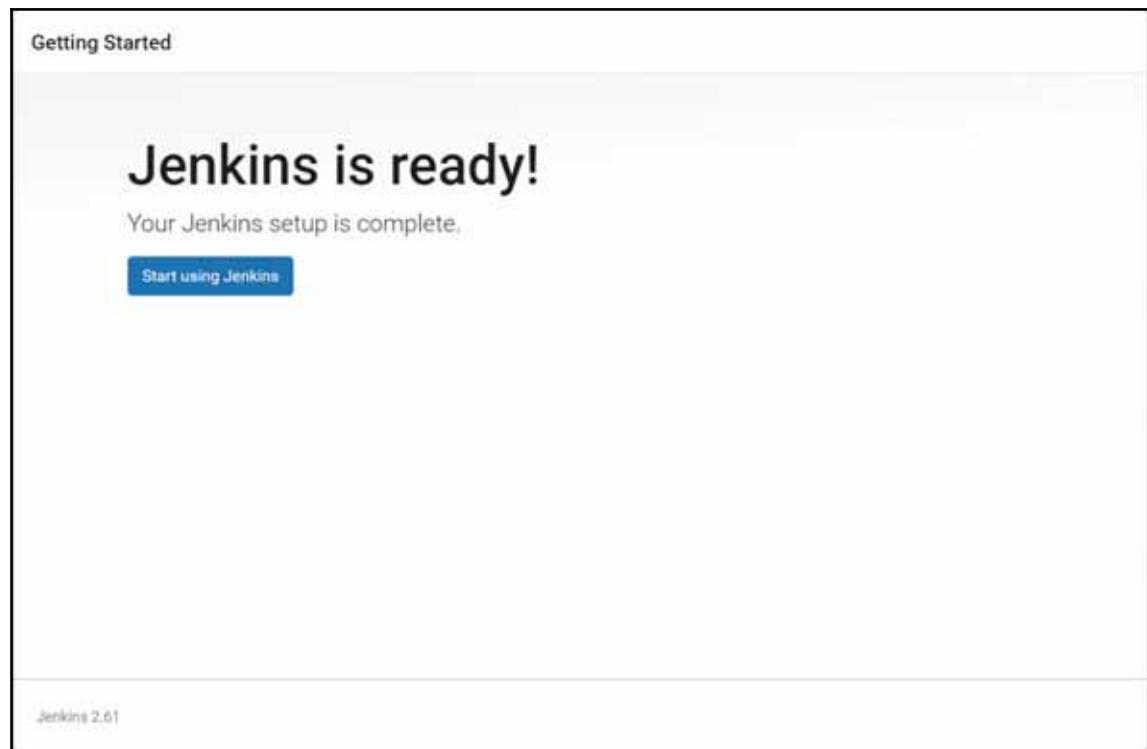
Confirm password:

Full name:

E-mail address:

Jenkins 2.61 Continue as admin **Save and Finish**

11. Click on **Start using Jenkins**:



In the next section, we will see details of the Jenkins dashboard.

Jumpstart tour of the Jenkins dashboard

The Jenkins dashboard is the place where all the operations related to CI and CD can be managed:



Click on the **Manage Jenkins** link on the Jenkins dashboard. Here, we can configure Jenkins, Security, Global Tools, Plugins, Users, and more:



Click on **Manage Plugins**. You will see the following tabs:

- The **Updates** tab provides details on updates available on the installed plugins.
- The **Available** tab provides a list of plugins that are not installed yet.
- The **Installed** tab provides a list of plugins that are already installed.
- The **Advanced** tab contains sections to configure proxies so we can download plugins even after we are behind the proxy. It also provides sections to upload HPI files for plugins in case we have already downloaded the plugin from the internet:

The screenshot shows the Jenkins Plugin Manager interface. At the top, there are tabs for 'Updates', 'Available', 'Installed' (which is selected), and 'Advanced'. Below the tabs is a table with columns: Enabled, Name, Version, Previously Installed version, and Uninstall. The table lists several installed plugins, each with a brief description and an 'Uninstall' button. The plugins listed are: Ant Plugin, Authentication Token API Plugin, bouncycastle API Plugin, Branch API Plugin, build timeout plugin, Credentials Binding Plugin, Credentials Plugin, and Display URLs API.

Enabled	Name	Version	Previously Installed version	Uninstall
*	Ant Plugin Adds Apache Ant support to Jenkins	1.5		Uninstall
*	Authentication Token API Plugin This plugin provides an API for converting credentials into authentication tokens in Jenkins.	1.3		Uninstall
*	bouncycastle API Plugin This plugin provides an stable API in Bouncy Castle related tasks.	2.16.1		Uninstall
*	Branch API Plugin This plugin provides an API for multiple branch based projects.	2.0.9		Uninstall
*	build timeout plugin This plugin allows builds to be automatically terminated after the specified amount of time has elapsed.	1.10		Uninstall
*	Credentials Binding Plugin Allows credentials to be bound to environment variables for use from miscellaneous build steps.	1.11		Uninstall
*	Credentials Plugin This plugin allows you to store credentials in Jenkins	2.1.12		Uninstall
	Display URLs API			

In the **Manage Jenkins** section, click on **Manage Nodes**.

By default, the system on which Jenkins is installed is a master node. This is the section that can be utilized to create the master agent architecture that we will cover later in the book:

The screenshot shows the Jenkins dashboard with the 'Nodes' section selected. On the left sidebar, there are links for 'Back to Dashboard', 'Manage Jenkins', 'New Node', and 'Configure'. The main content area displays a table with one row for the 'master' node. The columns are: S, Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The 'master' node has the following details: Name is 'master', Architecture is 'Windows 10 (amd64)', Clock Difference is 'In sync', Free Disk Space is '231.78 GB', Free Swap Space is '10.86 GB', Free Temp Space is '248.38 GB', and Response Time is '0ms'. Below the table, there is a 'Data obtained' status message and a 'Refresh status' button. At the bottom of the dashboard, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 idle, 2 idle).

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
1	master	Windows 10 (amd64)	In sync	231.78 GB	10.86 GB	248.38 GB	0ms

In the next section, we will cover different kinds of configuration available in the **Manage Jenkins** section.

Configuration settings in Jenkins

In **Manage Jenkins**, click on the **Configure System** link.

Here, we can get information about the `JENKINS_HOME` directory, the workspace root directory, and so on. We can set the Jenkins URL as well in this section:

The screenshot shows the Jenkins configuration interface. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', and 'Credentials'. The main area has sections for 'Build Queue' (empty), 'Build Executor Status' (2 idle executors), 'Home directory' (set to '\$JENKINS_HOME/workspace/\$ITEM_FULLNAME'), 'Workspace Root Directory' (set to '\$ITEM_ROOTDIR/builds'), 'Labels' (empty), 'Usage' (set to 'Use this node as much as possible'), 'Quiet period' (set to 5), and 'SCM checkout retry count' (empty). At the bottom are 'Save' and 'Apply' buttons.

In **Manage Jenkins**, click on **Configure Global Security** to see the security settings available in Jenkins. We will cover role-based access, matrix-based project security, and other features in later parts of this book:

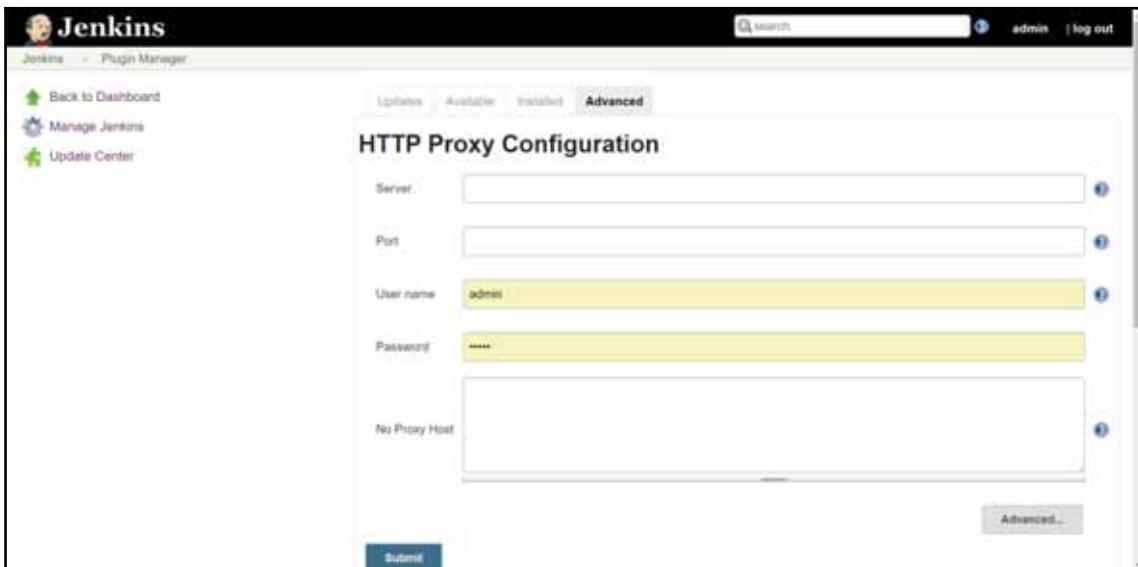
The screenshot shows the 'Configure Global Security' page. It includes fields for enabling security, choosing a TCP port for JNLP agents (fixed at 50000), and a 'Security Realm' section where 'Jenkins' own user database' is selected. Other options include 'Delegate to server container', 'Allow users to sign up', and 'LDAP'. An 'Authorization' section allows selecting 'Anyone can do anything', 'Legacy mode', or 'Logged-in users can do anything'. At the bottom are 'Save' and 'Apply' buttons.

In **Manage Jenkins**, click on the **Global Tool Configuration** link to provide details related to all tools available on the system that can be utilized to perform certain tasks. Depending on plugins related to specific tools, that section will appear on this page.

Another important thing to mention is that we can configure multiple versions of the same tool. For example, we can configure Java 6, Java 7, and Java 8. It is highly likely that different projects require different versions of Java for their execution. In such cases, we can configure multiple JDKs here and utilize specific JDKs in specific build jobs:

The screenshot shows the Jenkins Global Tool Configuration interface. At the top, there's a navigation bar with links for 'Back to Dashboard' and 'Manage Jenkins'. The main title is 'Global Tool Configuration'. Under 'Maven Configuration', there are dropdowns for 'Default settings provider' (set to 'Use default maven settings') and 'Default global settings provider' (set to 'Use default maven global settings'). Below that is a 'JDK' section with a 'JDK installations' dropdown and an 'Add JDK' button. A list of existing JDK installations is shown, with one entry for 'git' having its name set to 'Default' and the path to its executable set to 'git.exe'. At the bottom are 'Save' and 'Apply' buttons.

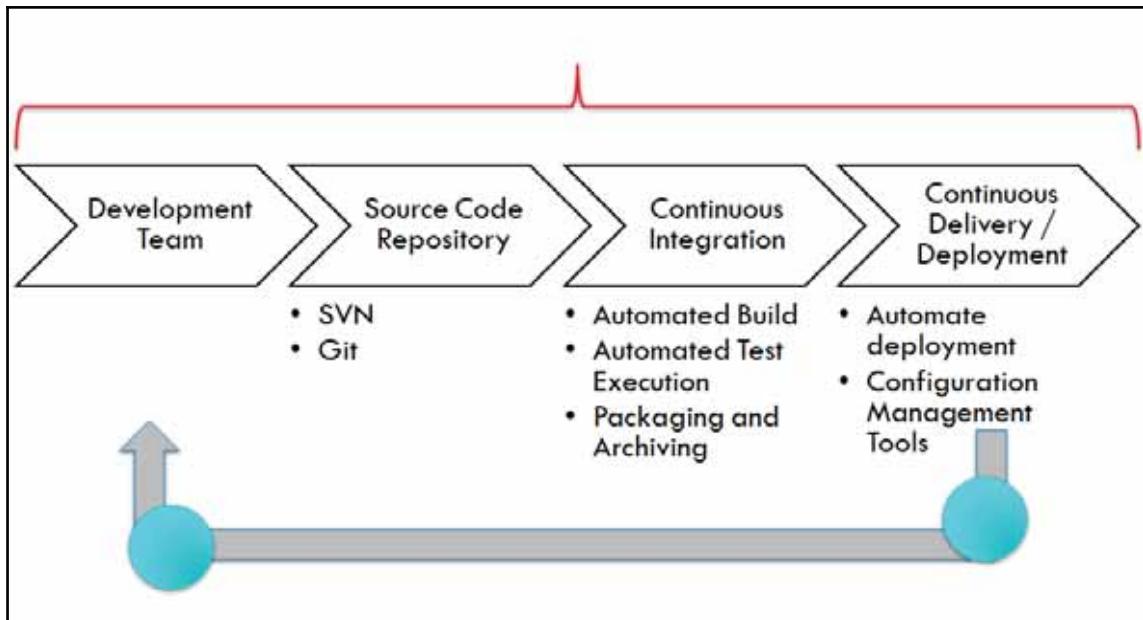
As discussed earlier in this chapter, the **Manage Plugins** section has an **Advanced** tab that allows us to configure proxy details.



This was all about the basic installation and configuration of Jenkins 2.x, and now we will cover what we are going to achieve in this book.

Overview of the CI/CD pipeline

The application development life cycle is traditionally a lengthy manual process. In addition, it requires effective collaboration between development and operations teams. The CI/CD pipeline is a demonstration of automation involved in the application development lifecycle that contains automated build execution, automated test execution, notifications to stakeholders, and deployment in different runtime environments. Effectively, a deployment pipeline is a combination of continuous integration and continuous delivery, and hence a part of DevOps practices. The following figure depicts the pipeline process. Depending on the culture of organization and the available tools, the flow and tools may differ:



Members of the development team check code into a source code repository. Continuous integration products such as Jenkins are configured to poll changes from the code repository. Changes in the repository are downloaded to the local workspace and Jenkins triggers a build process that is assisted by Ant or Maven or Gradle or any build script. Automated test execution, unit testing, static code analysis, reporting, and notification of successful or failed build processes are also parts of the Continuous Integration process.

Once the build is successful, it can be deployed to different runtime environments such as testing, pre-production, production, and so on. Deploying a WAR file in terms of a JEE application is normally the final stage in the deployment pipeline. However, after the deployment of this package into a pre-production environment, functional and security testing can be performed.

One of the biggest benefits of the pipeline is a faster feedback cycle. Identification of issues in the application in early stages and no dependency on manual effort make this entire end-to-end process more effective.

In the following chapters, we will see how Jenkins can be used to implement CI and CD practices in modernizing the culture of an organization.

Summary

Congratulations! We have reached the end of this chapter. So far, we have covered the basics of CI and have introduced Jenkins and its features. We have also completed the installation of Jenkins using generic package files. We also completed a quick tour of features available in the Jenkins dashboard. In addition to this, we have discussed the CI/CD pipeline and its importance in cultivating DevOps culture.

Now that we are able to use our automation server, Jenkins, we can begin creating a job and verify how Jenkins works. Before that, we will see how different configurations can be done in Jenkins in the next chapter.

2

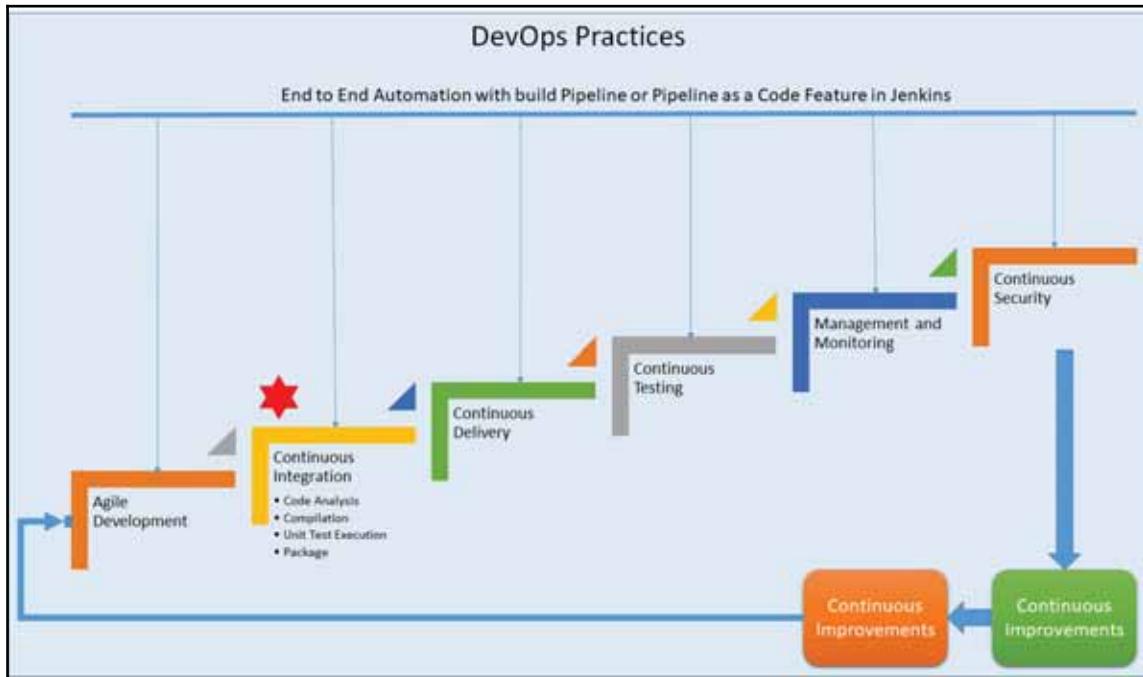
Installation and Configuration of Code Repository and Build Tools

We have seen the CI/CD pipeline in the previous chapter, where source code repositories and automated build were discussed in detail. SVN, Git, CVS, and StarTeam are some of the popular code repositories that manage changes to code, artifacts, or documents while Ant and Maven are popular build automation tools for Java applications.

This chapter describes in detail how to prepare an environment for application life cycle management and configure it with Jenkins—an open source **Continuous Integration (CI)** tool. It will cover how to integrate Eclipse and Jenkins so builds can be run from Eclipse as well. These are the major points that we will cover in this chapter:

- Overview of Jenkins
- Installing Java and configuring environment variables
- Installing and configuring Ant
- Installing Maven
- Configuring Ant, Maven, and JDK in Jenkins
- Overview of GitHub
- Creating a new build job in Jenkins with GitHub
- Eclipse and Jenkins integration

In this chapter, we will cover the configuration in Jenkins as part of CI best practice and as a part of our DevOps journey:



At the end of this chapter, we will know how to configure Jenkins and how to integrate different tools in Jenkins.

Overview of Jenkins

We have seen in Chapter 1, *Exploring Jenkins*, that the **Manage Jenkins** link on the dashboard is used to configure systems. Click on the **Global Tool Configuration** link to configure Java, Ant, Maven, and other third-party products' related information.

In this book, we will try to make things general and not operating system-specific. We have used Windows 10 for most of the sections in this book for CICD implementation, but it can be implemented on any operating system. We will specify some operating system-specific requirements if needed. Normally, path style changes and installation procedure changes, but the rest are the same irrespective of OS.

Installing Java and configuring the environment variables

In this section, we will cover installing Jenkins on both Windows and CentOS operating systems.

Installing Java on Windows 10

Go to

<http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html> and download the Java installer for either the 32-bit or 64-bit operating system and follow the wizard to install it.

Execute the `java` and `javac` commands from the command window to verify the installation.

Installing Java on CentOS

If Java is not already installed on the system, then you can install it as follows:

1. Find Java-related packages in the CentOS repository and locate the appropriate package to install by using the following code:

```
[root@localhost ~]# yum search java
Loaded plugins: fastestmirror, refresh-packagekit, security
.
.
.
ant-javamail.x86_64 : Optional javamail tasks for ant
eclipse-mylyn-java.x86_64 : Mylyn Bridge: Java Development
.
.
.
java-1.5.0-gcj.x86_64 : JPackage runtime compatibility layer for
GCJ
```

```
java-1.5.0-gcj-devel.x86_64 : JPackage development compatibility
layer for GCJ
java-1.5.0-gcj-javadoc.x86_64 : API documentation for libgcj
java-1.6.0-openjdk.x86_64 : OpenJDK Runtime Environment
java-1.6.0-openjdk-devel.x86_64 : OpenJDK Development Environment
java-1.6.0-openjdk-javadoc.x86_64 : OpenJDK API Documentation
java-1.7.0-openjdk.x86_64 : OpenJDK Runtime Environment
jcommon-serializer.x86_64 : JFree Java General Serialization
Framework
.
.
```

2. Now install the Java package in the local repositories by executing the `yum install` command as follows:

```
[root@localhost ~]# yum install java-1.7.0-openjdk.x86_64
Loaded plugins: fastestmirror, refresh-packagekit, security
Loading mirror speeds from cached hostfile
Setting up Install Process
Resolving Dependencies
--> Running transaction check
--> Package java-1.7.0-openjdk.x86_64 1:1.7.0.3-2.1.el6.7 will be
installed
--> Finished Dependency Resolution

Dependencies Resolved
.

.

Install           1 Package(s)

Total download size: 25 M
Installed size: 89 M
Is this ok [y/N]: y
Downloading Packages:
java-1.7.0-openjdk-1.7.0.3-2.1.el6.7.x86_64.rpm
| 25 MB     00:00
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : 1:java-1.7.0-openjdk-1.7.0.3-2.1.el6.7.x86_64
  1/1
  Verifying   : 1:java-1.7.0-openjdk-1.7.0.3-2.1.el6.7.x86_64
  1/1

Installed:
  java-1.7.0-openjdk.x86_64 1:1.7.0.3-2.1.el6.7

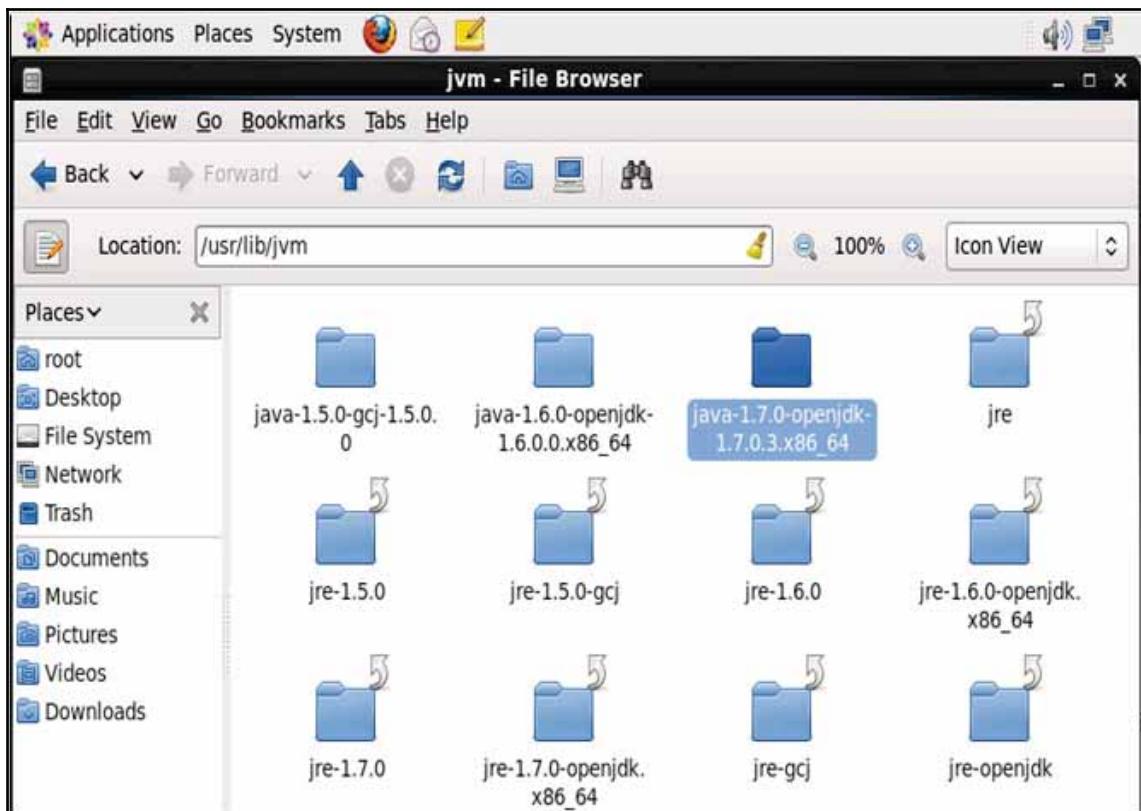
Complete!
```

Java is now installed successfully from the local repository.

Configuring environment variables

The following are the steps to configure environment variables:

1. Set the `JAVA_HOME` and `JRE_HOME` variables.
2. Go to `/root`.
3. Press `Ctrl + H` to list hidden files.
4. Find `.bash_profile` and edit it by appending the Java path, as shown in the following screenshot:



Installing and configuring Ant

Ant is a build tool. Download Ant from <https://ant.apache.org/bindownload.cgi> and unzip it.

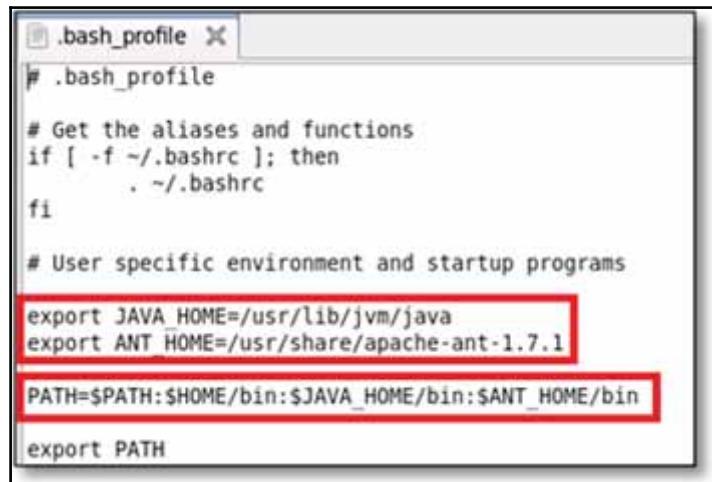
Configuring Ant in Windows

The following are the steps to configure Ant in CentOS:

1. Go to **Control Panel | All Control Panel Items | System** and click on **Advanced system settings | Advanced | Environment Variables....**
2. Click on **New**.
3. Set the variable name as **ANT_HOME** and location and click **OK**.

Configuring Ant in CentOS

Set the **ANT_HOME** and **JAVA_HOME** environment variables:



```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

export JAVA_HOME=/usr/lib/jvm/java
export ANT_HOME=/usr/share/apache-ant-1.7.1

PATH=$PATH:$HOME/bin:$JAVA_HOME/bin:$ANT_HOME/bin

export PATH
```

There is an option available in Jenkins to install Ant or Maven automatically. We will see this in the Jenkins configuration section.

Installing Maven

Maven is a build tool. Download the Maven binary zip from <https://maven.apache.org/download.cgi> and extract it to the local system where Jenkins is installed:

The screenshot shows the Apache Maven 3.3.3 download page. The left sidebar has a 'Download' section selected, showing links for 'Releases History', 'Release Notes (3.3.3)', 'Release Notes (3.2.6)', 'Release Notes (3.1.1)', 'Release Notes (3.0.9)', and 'All Release Notes'. Below that are 'Licence', 'Security', 'mirrors', 'Eclipse', 'NetBeans', 'ABOUT MAVEN', 'What is Maven?', 'Features', 'FAQ', 'Support and Training', 'Documentation', 'Maven Plugins', and 'Index (category)'. The main content area is titled 'Download Apache Maven 3.3.3'. It contains instructions about Maven distribution formats and how to verify signatures. It also encourages users to configure a Maven repository mirror. A 'Mirror' section shows the currently selected mirror as 'http://mirror.cogentco.com/pub/apache/'. There is a dropdown for 'Other mirrors' with 'http://mirror.cogentco.com/' selected and a 'Change' button. Below this is a link to the 'complete list of mirrors'. The 'Maven 3.3.3' section states it is the current stable version. A table lists two download options: 'Maven 3.3.3 (Binary tar.gz)' and 'Maven 3.3.3 (Binary zip)'. The table columns are 'Link', 'Checksum', and 'Signature'. The 'Link' column shows the URLs for each file type, and the 'Checksum' and 'Signature' columns show the corresponding MD5 checksums and GPG signatures.

	Link	Checksum	Signature
Maven 3.3.3 (Binary tar.gz)	apache-maven-3.3.3-bin.tar.gz	apache-maven-3.3.3-bin.tar.gz.md5	apache-maven-3.3.3-bin.tar.gz.asc
Maven 3.3.3 (Binary zip)	apache-maven-3.3.3-bin.zip	apache-maven-3.3.3-bin.zip.md5	apache-maven-3.3.3-bin.zip.asc

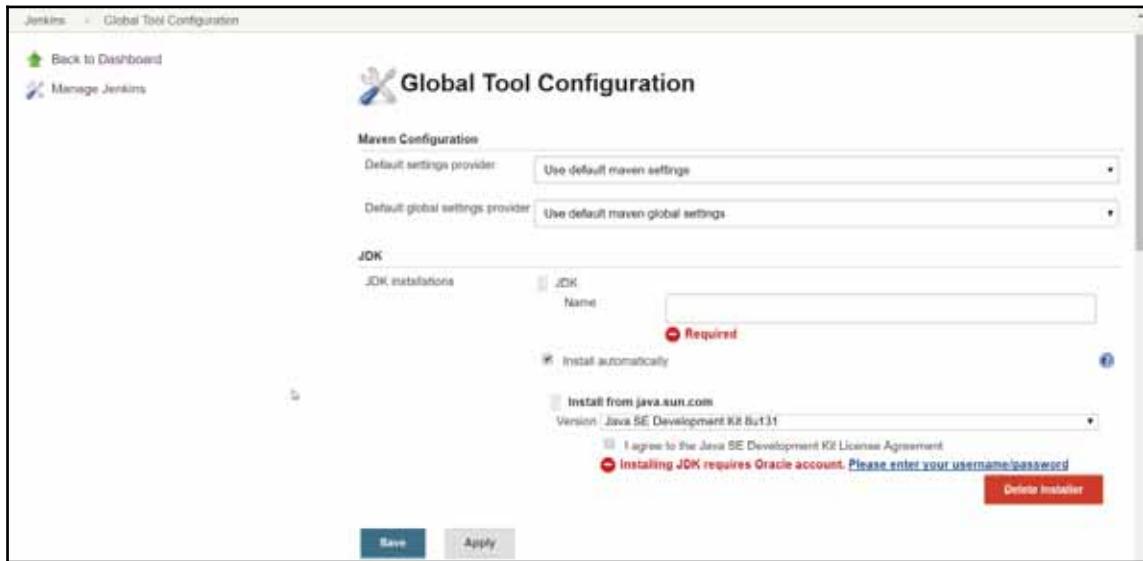
Once we have downloaded Java, Ant, and Maven, our next task is to configure them.

Configuring Ant, Maven, and JDK in Jenkins

The following are the steps to configure Ant, Maven, and JDK in Jenkins:

1. Open the Jenkins dashboard in a browser with the URL http://<ip_address>:8080. Go to the **Manage Jenkins** section and click on **Global Tool Configuration**.

2. Configure Java based on the installation, as shown in the following screenshot.
We can install it automatically:



3. If Java is already installed, then uncheck the checkbox of **Install automatically** and give the JAVA_HOME path. If Jenkins, Ant, Maven, and Java are installed on CentOS, the path style will be different than this:

The screenshot shows the Jenkins Global Tool Configuration page. Under the 'Maven Configuration' section, there are two dropdown menus: 'Default settings provider' set to 'Use default maven settings' and 'Default global settings provider' also set to 'Use default maven global settings'. In the 'JDK' section, there is a table for 'JDK installations'. It lists one entry: 'JDK 8' with 'Name' and 'JAVA_HOME' both set to 'C:\Program Files\Java\jdk1.8.0_111'. There is an unchecked checkbox for 'Install automatically' and a red 'Delete JDK' button. Below the table is a link 'List of JDK installations on this system'. At the bottom of the page are 'Add JDK' and 'Delete JDK' buttons.

4. Download Git installer for Windows and install it on the system. Keep the settings as they are in Jenkins after you click on **Add Git**. If Jenkins, Ant, Maven, and Java are installed on CentOS, the path style will be different than this:

The screenshot shows the Jenkins Global Tool Configuration page with sections for Git, Gradle, Ant, and Maven. The 'Git' section has one entry: 'Git' with 'Name' set to 'Default' and 'Path to Git executable' set to 'git.exe'. The 'Gradle' section has a 'Gradle Installations' table with an 'Add Gradle' button below it. The 'Ant' section has an 'Ant installations' table with an 'Add Ant' button below it. The 'Maven' section has a 'Maven installations' table with an 'Add Maven' button below it. At the bottom are 'Save' and 'Apply' buttons.

5. Click on **Add Ant** and provide a **Name** and **ANT_HOME** location. The value that we give in the **Name** box will be used in build job to identify the Ant version we want to use. This is similar practice for any tool that we will use. If Jenkins, Ant, Maven, and Java are installed on CentOS, the path style will be different than this:

The screenshot shows the Jenkins Global Tool Configuration page for the 'Ant' section. It displays a single entry for 'Ant' with the following details:

- Name:** apache-ant-1.9.4
- ANT_HOME:** C:\apache-ant-1.9.4
- Install automatically:** Unchecked checkbox

At the bottom right of the entry is a red 'Delete Ant' button. Below the entries is a grey 'Add Ant' button.

6. Click on **Add Ant** and provide a **Name** and **MAVEN_HOME** location. If Jenkins, Ant, Maven, and Java are installed on CentOS, the path style will be different than this:

The screenshot shows the Jenkins Global Tool Configuration page for the 'Maven' section. It displays a single entry for 'Maven' with the following details:

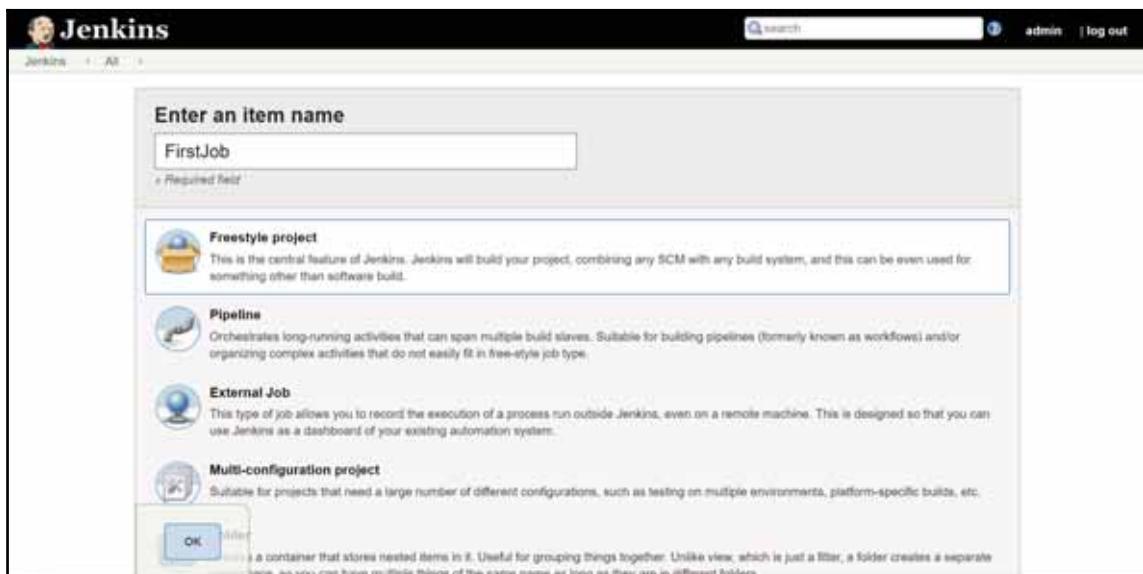
- Name:** apache-maven-3.3.1
- MAVEN_HOME:** C:\apache-maven-3.3.1
- Install automatically:** Unchecked checkbox

At the bottom right of the entry is a red 'Delete Maven' button. Below the entries is a grey 'Add Maven' button.

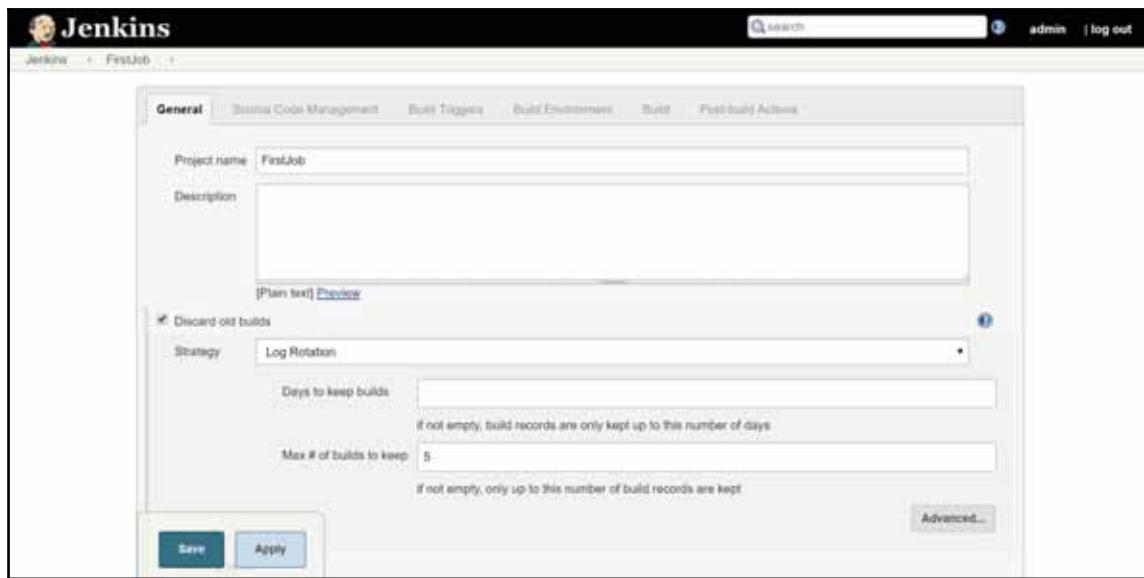
First job in Jenkins

Let's create a job in Jenkins that provides details about IP addresses and other configuration details. Basically, we will execute the `ipconfig` command. If you are using Linux OS, you can execute `ifconfig`. The following are the steps to create your first job in Jenkins:

1. On the Jenkins dashboard, click on **New Item**.
2. Enter the item's name.
3. Select **Freestyle project**.
4. Click **Ok**:



5. Give a **Project name**:

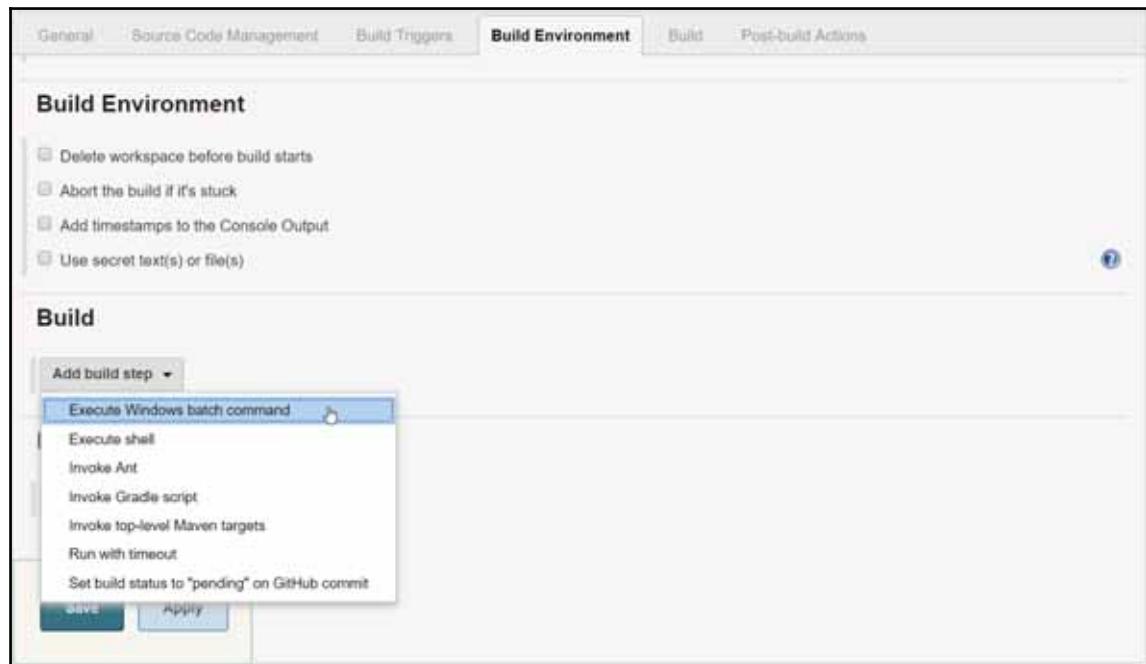


6. Select **None** in **Source Code Management**.

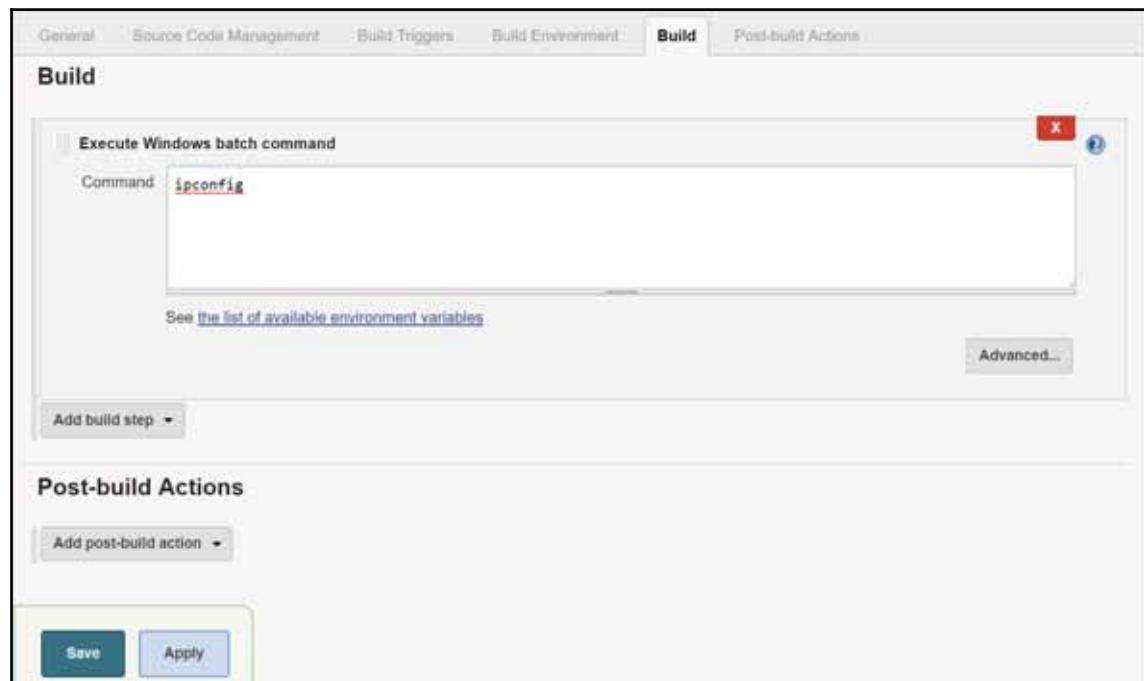
7. In the **Build Triggers** section, select **Build periodically** and give cron syntax in **Schedule**. It will always run at 8:52 AM in the morning:

The screenshot shows the Jenkins configuration interface for a project. The top navigation bar includes tabs for General, Source Code Management, Build Triggers, Build Environment, Build, and Post-build Actions. The Source Code Management tab is selected, showing options for None, Git, and Subversion, with None currently chosen. The Build Triggers tab is also selected, displaying three trigger types: Trigger builds remotely, Build after other projects are built, and Build periodically. The 'Build periodically' option is checked, and its schedule is set to '52 8 * * *'. A warning message below the schedule indicates that the cron syntax is not ideal for load distribution and suggests using 'H 8 * * *' instead. The bottom of the screen shows a 'Save' button and an 'Apply' button.

8. In **Build**, click on **Execute Windows batch command**. To execute on CentOS or other flavours of Linux, select **Execute Shell**, and the command will be ifconfig:



9. In the textbox, write the ipconfig command to get the IP address of the system.
10. Click on **Save**:



11. Click on **Build now:**

Project FirstJob

Back to Dashboard

Status

Changes

Workspace

Build Now

Delete Project

Configure

Workspace

Recent Changes

Build History

trend —

find

#1 May 21, 2017 8:52 AM

RSS for all RSS for failures

12. Observe the **Build History** and click on the blue ball to go to **Console Output**.
Bingo! We have created our first job in Jenkins:



The screenshot shows the Jenkins interface for a job named 'FirstJob' with build number '#1'. On the left, there's a sidebar with links: Back to Project, Status, Changes, Console Output (which is currently selected and highlighted in blue), View as plain text, Edit Build Information, and Delete Build. The main content area is titled 'Console Output' with a blue circular icon. It displays the following log output:

```
Started by timer
Building in workspace F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob
[FIRSTJOB] $ cmd /c call C:\Users\Umesh\AppData\Local\Temp\jenkins83878738311927477.bat

F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

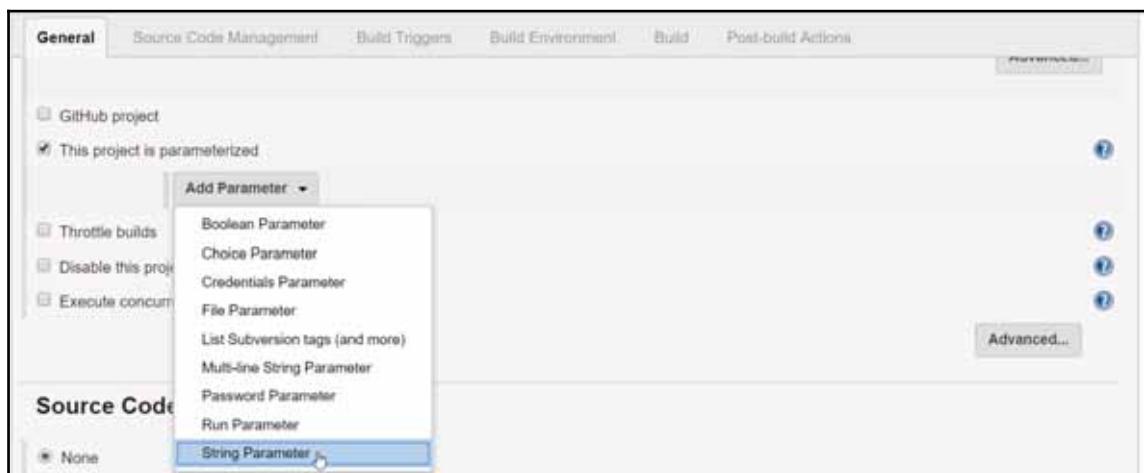
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Ethernet adapter VirtualBox Host-Only Network:

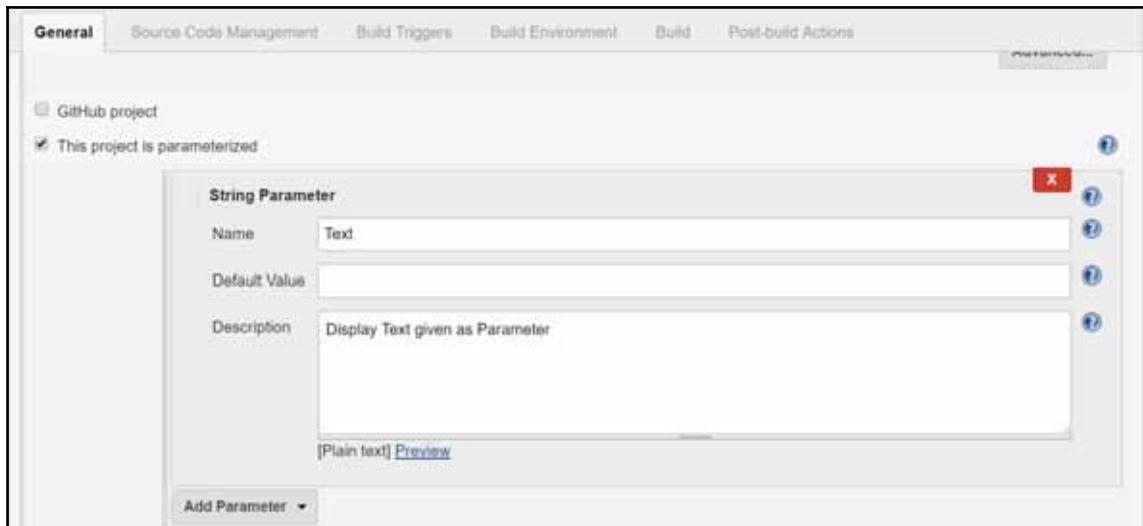
    Connection-specific DNS Suffix . :
    Link-local IPv6 Address . . . . . : fe80::94f3:4e00:7cf8:8855%17
    IPv4 Address . . . . . : 192.168.56.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . :

Ethernet adapter VirtualBox Host-Only Network #2:
```

13. To add a parameter in the Jenkins build, click on **This project is parameterized**. Click on **Add Parameter**. Select **String Parameter**:



14. Provide a name and description. Click on **Save**:



15. In the **Build** step, write the following commands in the **Execute Windows Batch Command** box:

```
echo %JOB_NAME%
echo %Text%
```

16. Click on **Build with Parameters**:

The screenshot shows the Jenkins interface for the project 'FirstJob'. The top navigation bar includes links for 'Jenkins', 'FirstJob', 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build with Parameters', 'Delete Project', and 'Configure'. The main content area is titled 'Project FirstJob'. It features a 'Workspace' icon and a 'Recent Changes' icon. Below this is a 'Build History' section with a search bar containing 'find'. It lists four builds: #4 (May 21, 2017 8:57 AM), #3 (May 21, 2017 8:55 AM), #2 (May 21, 2017 8:55 AM), and #1 (May 21, 2017 8:52 AM). At the bottom of the build history are 'RSS for all' and 'RSS for failures' links. To the right of the build history is a 'Permalinks' section with a list of four links: 'Last build (#4), 1 min 45 sec ago', 'Last stable build (#4), 1 min 45 sec ago', 'Last successful build (#4), 1 min 45 sec ago', and 'Last completed build (#4), 1 min 45 sec ago'.

17. It will ask for the parameter. Provide some text and click on **Build**:

The screenshot shows the Jenkins interface for the 'Project FirstJob'. On the left, there's a sidebar with links like 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build with Parameters' (which is highlighted in red), 'Delete Project', and 'Configure'. The main area is titled 'Project FirstJob' and displays the message 'This build requires parameters:'. Below this, there's a text input field with 'Hello etutorialsworld.com!' and a link 'Display Test given as Parameter'. A large blue 'Build' button is centered. At the bottom, there's a 'Build History' section showing four previous builds (B#4, B#3, B#2, B#1) with their respective dates: May 21, 2017 8:57 AM, May 21, 2017 8:55 AM, May 21, 2017 8:55 AM, and May 21, 2017 8:52 AM. Below the history, there are two RSS feed links: 'RSS for all' and 'RSS for failures'.

18. Verify the **Console Output**:

The screenshot shows the Jenkins 'Console Output' page. It features a blue circular icon with a white terminal symbol. The title 'Console Output' is displayed prominently. The log content is as follows:

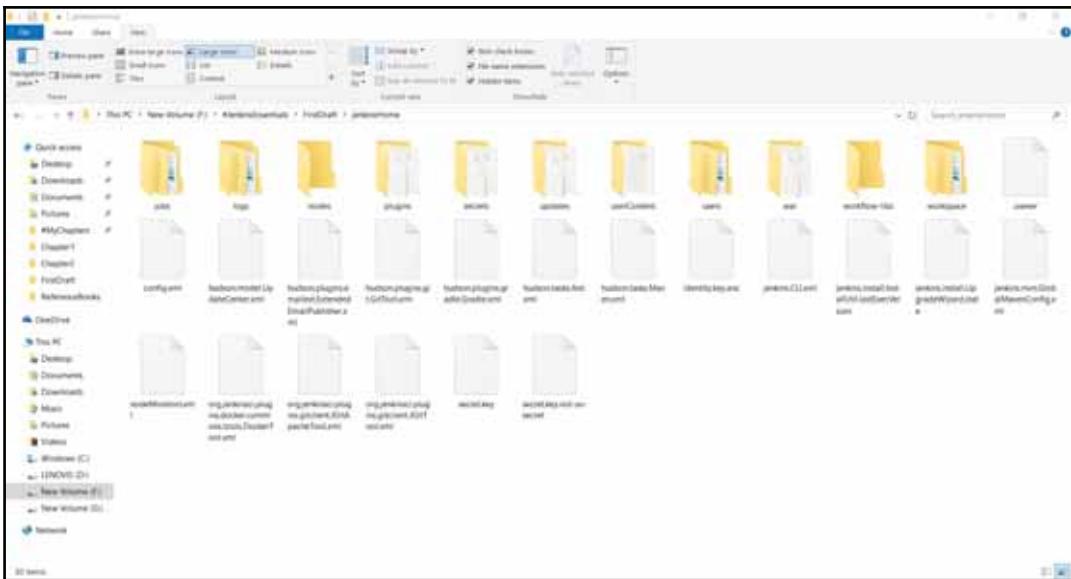
```
Started by user admin
Building in workspace F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob
[FirstJob] $ cmd /c call C:\Users\Mitesh\AppData\Local\Temp\jenkins3630239299526944525.bat

F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob>echo FirstJob
FirstJob

F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob>echo Hello etutorialsworld.com!
Hello etutorialsworld.com!

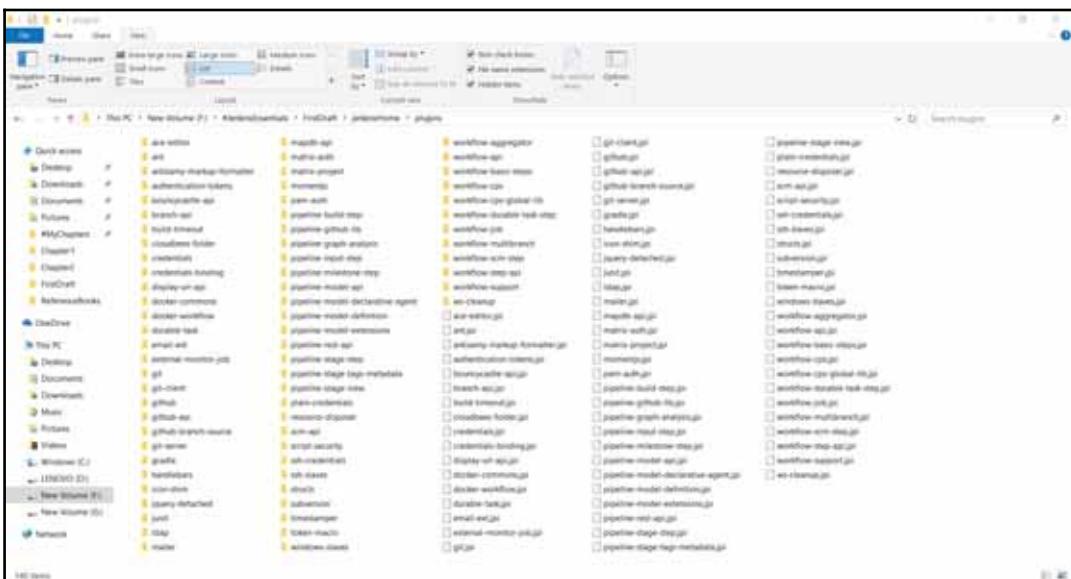
F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob>exit 0
Finished: SUCCESS
```

19. Go to JENKINS_HOME and try to find out what each directory contains:



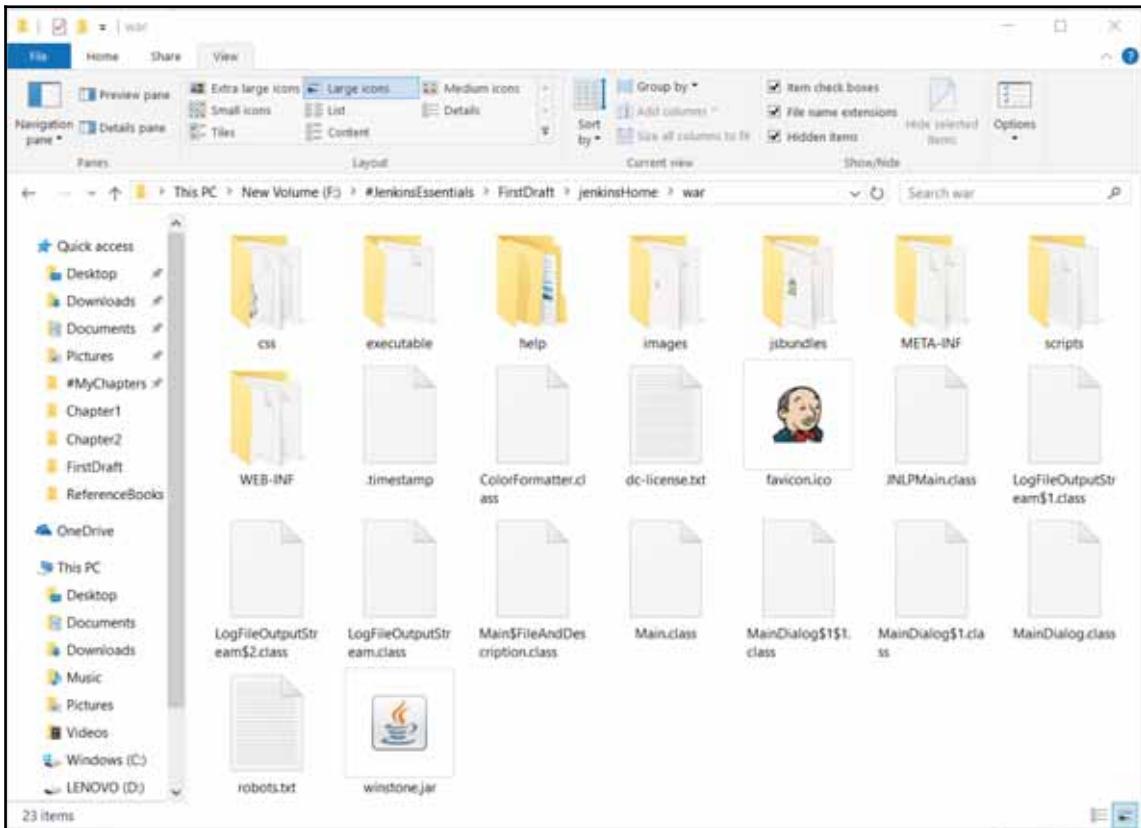
All the Directories available in JENKINS_HOME folder

20. In the `plugins` directory, all installed plugins are available:



All the Plugins available in JENKINS_HOME directory

21. The `war` directory contains the actual files that are used in the Jenkins application:



Installing and configuring the Git repository on CentOS

Git is a free and open source distributed version control system. In this section, we will try to install and configure Git:

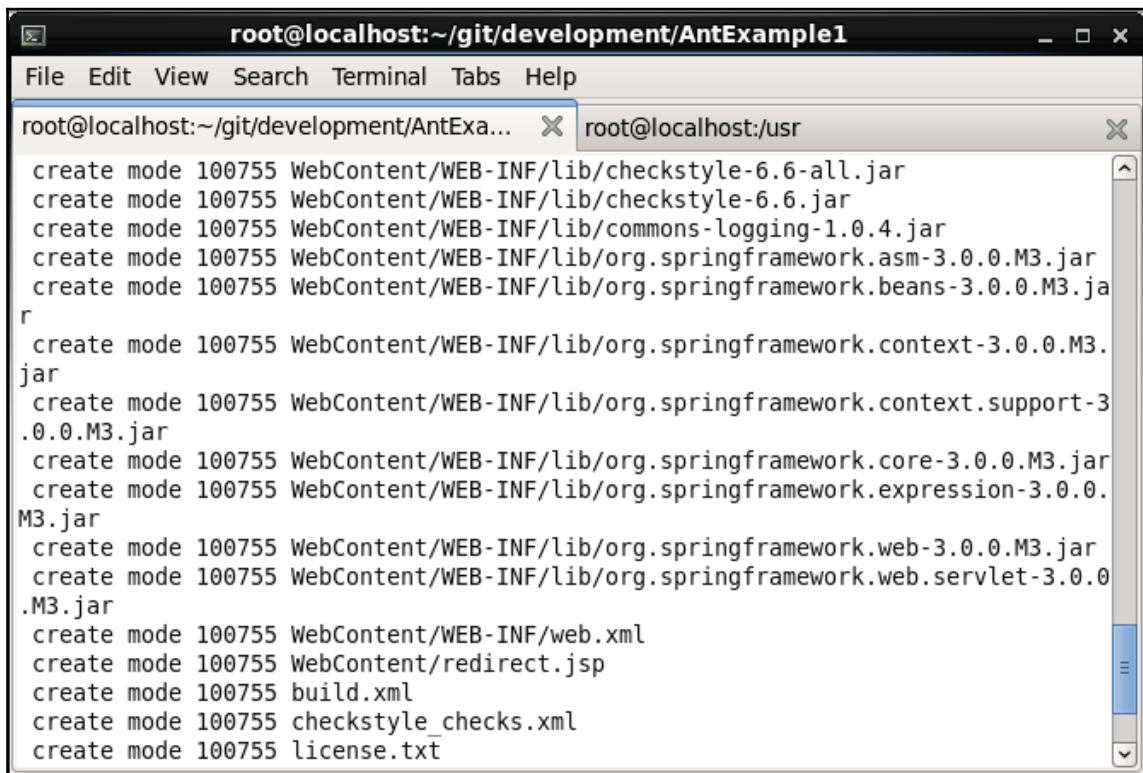
1. Open a terminal in the CentOS-based system and execute the `yum install git` command in the terminal.
2. Once it is successfully installed, verify the version with the `git --version` command.

3. Submit information about the user with the use of the `git config` command so that commit messages will be generated with the correct information attached.
4. Provide the name and email address to embed into commits.
5. To create a workspace environment, create a directory called `git` in the home directory and then create a subdirectory inside of that called `development`.
6. Use `mkdir -p ~/git/development ; cd ~/git/development` in the terminal.
7. Copy the `AntExample1` directory into the development folder.
8. Convert an existing project into a workspace environment by using the `git init` command.
9. Once the repository is initialized, add files and folders:

The screenshot shows a terminal window titled "root@localhost:~/git/development/AntExample1". The window has two tabs: "root@localhost:~/git/development/AntExa..." and "root@localhost:/usr". The terminal content is as follows:

```
[root@localhost Desktop]# git --version
git version 1.7.1
[root@localhost Desktop]# git config --global user.name "Mitesh"
[root@localhost Desktop]# git config --global user.email '[REDACTED]@gmail.co
m'
[root@localhost Desktop]# git config --list
user.name=Mitesh
user.email=[REDACTED]@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
[root@localhost Desktop]# mkdir -p ~/git/development ; cd ~/git/development
[root@localhost development]# cd AntExample1/
[root@localhost AntExample1]# git init
Initialized empty Git repository in /root/git/development/AntExample1/.git/
[root@localhost AntExample1]# git add .
[root@localhost AntExample1]#
```

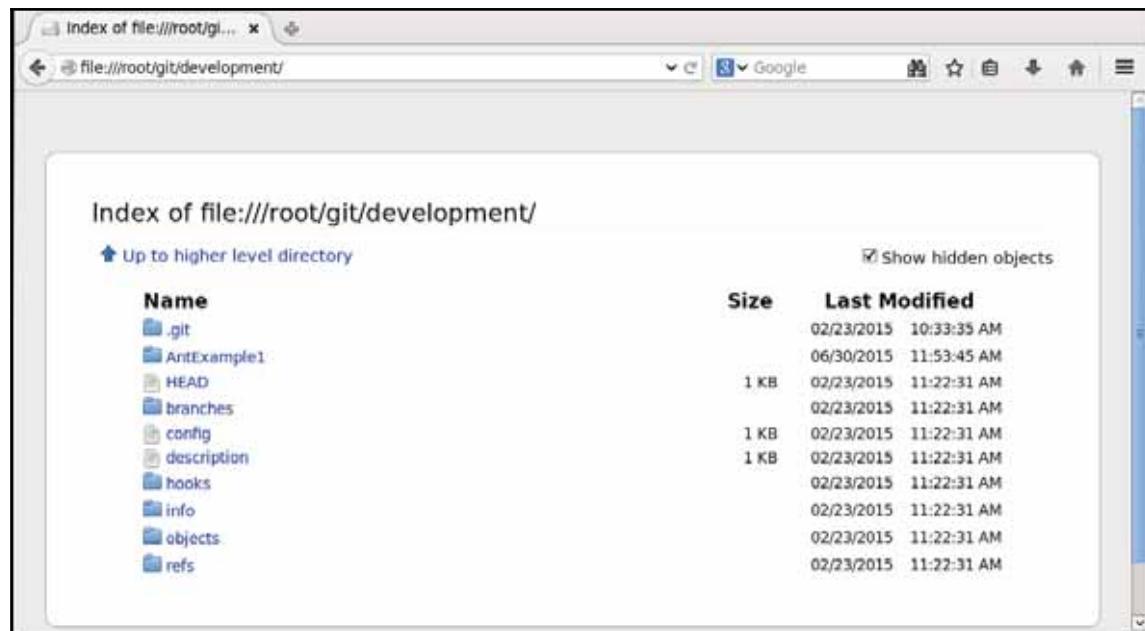
10. Commit by executing git commit -m "Initial Commit" -a:



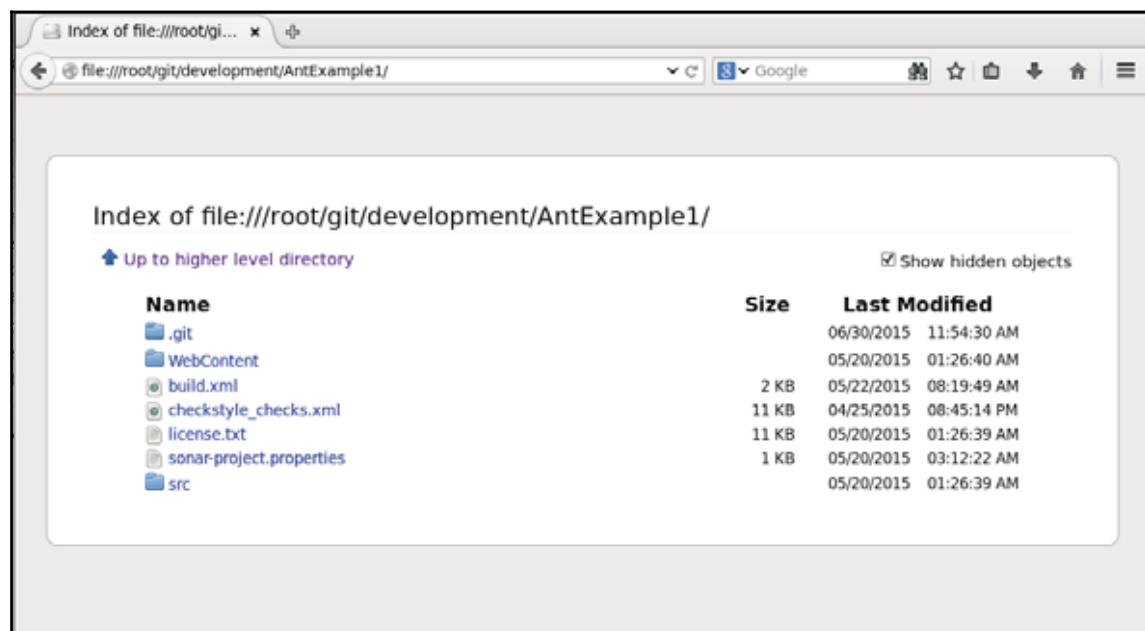
The screenshot shows a terminal window titled "root@localhost:~/git/development/AntExample1". It has two tabs: "root@localhost:~/git/development/AntExa..." and "root@localhost:/usr". The current tab displays a list of file creation commands:

```
create mode 100755 WebContent/WEB-INF/lib/checkstyle-6.6-all.jar
create mode 100755 WebContent/WEB-INF/lib/checkstyle-6.6.jar
create mode 100755 WebContent/WEB-INF/lib/commons-logging-1.0.4.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.asm-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.beans-3.0.0.M3.ja
r
create mode 100755 WebContent/WEB-INF/lib/org.springframework.context-3.0.0.M3.
jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.context.support-3
.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.core-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.expression-3.0.0.
M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.web-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.web.servlet-3.0.0
.M3.jar
create mode 100755 WebContent/WEB-INF/web.xml
create mode 100755 WebContent/redirect.jsp
create mode 100755 build.xml
create mode 100755 checkstyle_checks.xml
create mode 100755 license.txt
```

11. Verify the Git repository:



12. Verify the project in the Git repository:



In the next section, we will cover how to use Git and GitHub to check out the source code and execute the build.

Creating a new build job in Jenkins with Git and GitHub

The following are the steps required to create a new build job in Jenkins with Git and GitHub:

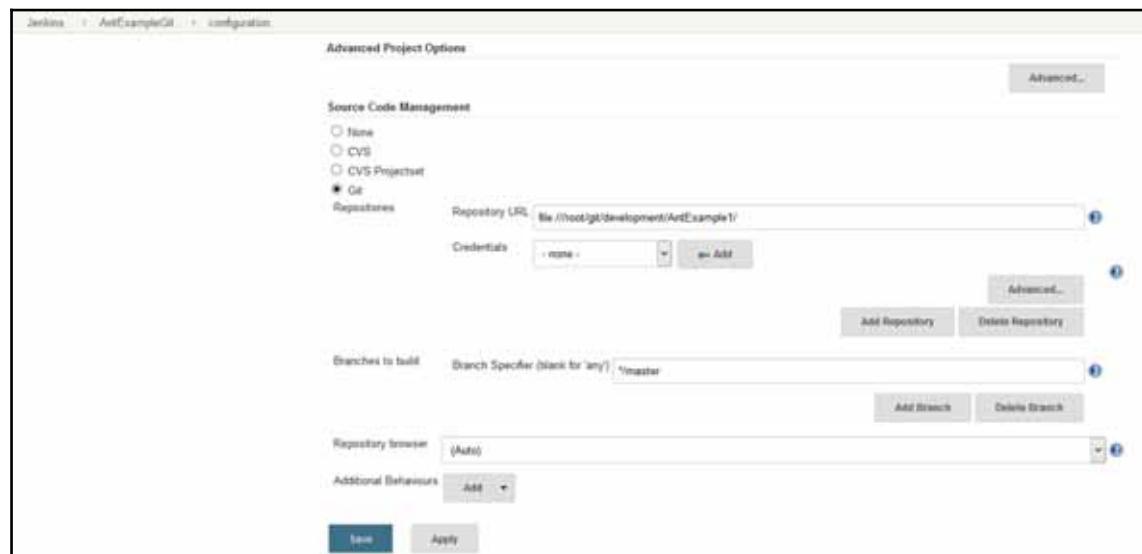
1. On the Jenkins dashboard, click on **Manage Jenkins** and select **Manage Plugins**. Click on the **Available** tab and write **Github Plugin** in the search box.
2. Click the checkbox and click on the **Download now and install after restart** button.
3. Restart Jenkins:



4. Create a new **Freestyle project**. Provide an item name and click on **OK**:



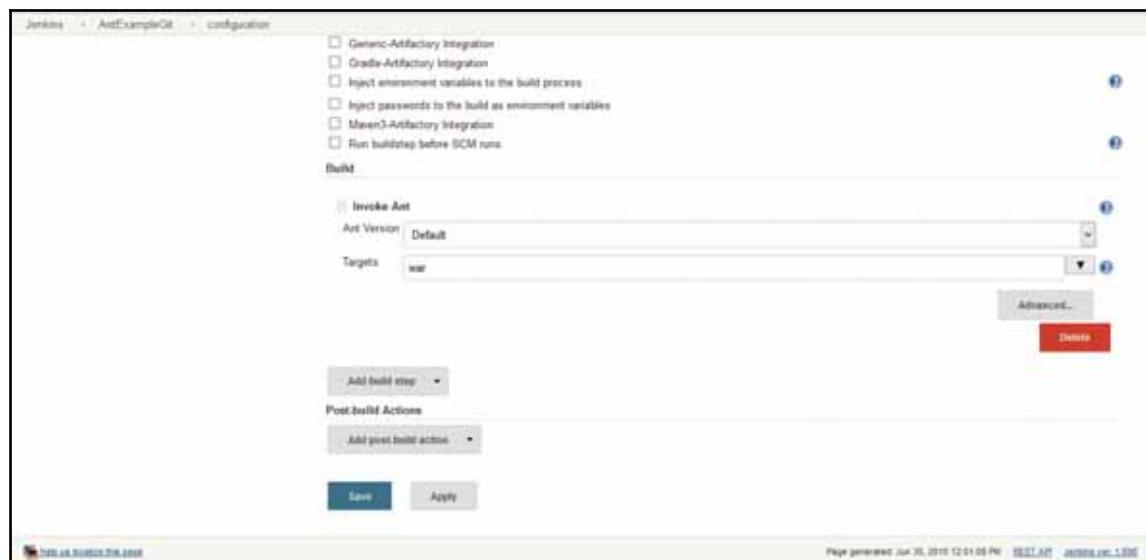
5. Configure Git in the Source Code Management section:



6. In **Repository URL**, we can provide a GitHub URL, which will work fine for publicly accessible projects. We can specify branch too:



7. Add an **Invoke Ant** build step by clicking on **Add build step**:



8. Execute the build:

The screenshot shows the Jenkins interface for a job named 'AntExampleGit'. The main title bar says 'Build #2 (Jun 30, 2015 12:03:05 PM)'. On the right, it shows 'Started 14 sec ago' and 'Took 5.1 sec in master'. Below the title, there's a 'No changes.' message and a 'Started by user Mitesh' entry. It also displays a 'git' icon and revision information: 'Revision: 6a5222023d9cabc5e21ba326d940907607330' and 'refs/remotes/origin/master'. The left sidebar contains links for Back to Project, Status, Changes, Console Output, Edit Build Information, Delete Build, Environment Variables, Git Build Data, No Tags, and Previous Build.

9. Click on the **Console Output** to see the progress of the build:

The screenshot shows the Jenkins interface for the same job. The title bar now says 'Console Output'. The left sidebar includes a link for 'Executed Ant Targets' which lists 'ant', 'compile', and 'war'. The main area displays the build log output. The log shows the Jenkins environment loading, building on the master workspace, cloning the remote Git repository, and performing a git fetch --tags --progress command. It also shows the configuration of remote origins and fetching upstream changes from the master branch. Finally, it shows the execution of Ant targets: 'ant', 'compile', and 'war', followed by the creation of build files.

```
Started by user Mitesh
[EnvInject] - Loading node environment variables.
Building on master in workspace /root/.jenkins/jobs/AntExampleGit/workspace
Cloning the remote Git repository
Cloning repository file:///root/git/development/AntExampleGit/
> git init /root/.jenkins/jobs/AntExampleGit/workspace # timeout=10
Fetching upstream changes from file:///root/git/development/AntExampleGit/
> git fetch --tags --progress file:///root/git/development/AntExampleGit/ >refs/heads/* refs/remotes/origin/*
> git fetch --tags --progress file:///root/git/development/AntExampleGit/ >refs/heads/* refs/remotes/origin/* # timeout=10
> git config --add remote.origin.fetch +refs/heads/* refs/remotes/origin/* # timeout=10
> git config remote.origin.url file:///root/git/development/AntExampleGit/ # timeout=10
Fetching upstream changes from file:///root/git/development/AntExampleGit/
> git fetch --tags --progress file:///root/git/development/AntExampleGit/ >refs/heads/* refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 6a5222023d9cabc5e21ba326d940907607330 into /root/.jenkins/jobs/AntExampleGit/workspace
> git config core.sparsecheckout # timeout=10
> git checkout -f 6a5222023d9cabc5e21ba326d940907607330 # timeout=10
> git rev-list 6a5222023d9cabc5e21ba326d940907607330 # timeout=10
[workspace] $ ant war
Buildfile: build.xml

init:
[delete] Created dir: /root/.jenkins/jobs/AntExampleGit/workspace/build/classes
[delete] Created dir: /root/.jenkins/jobs/AntExampleGit/workspace/dist
```

10. Once the build is successful, verify the workspace in the build job.

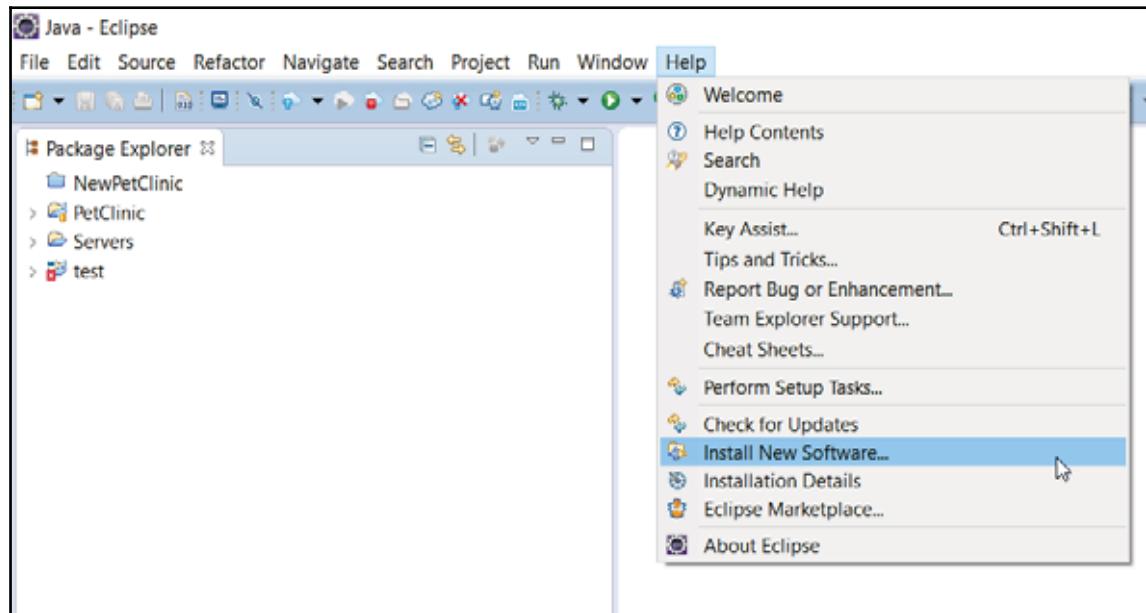
Done!

Eclipse and Jenkins integration

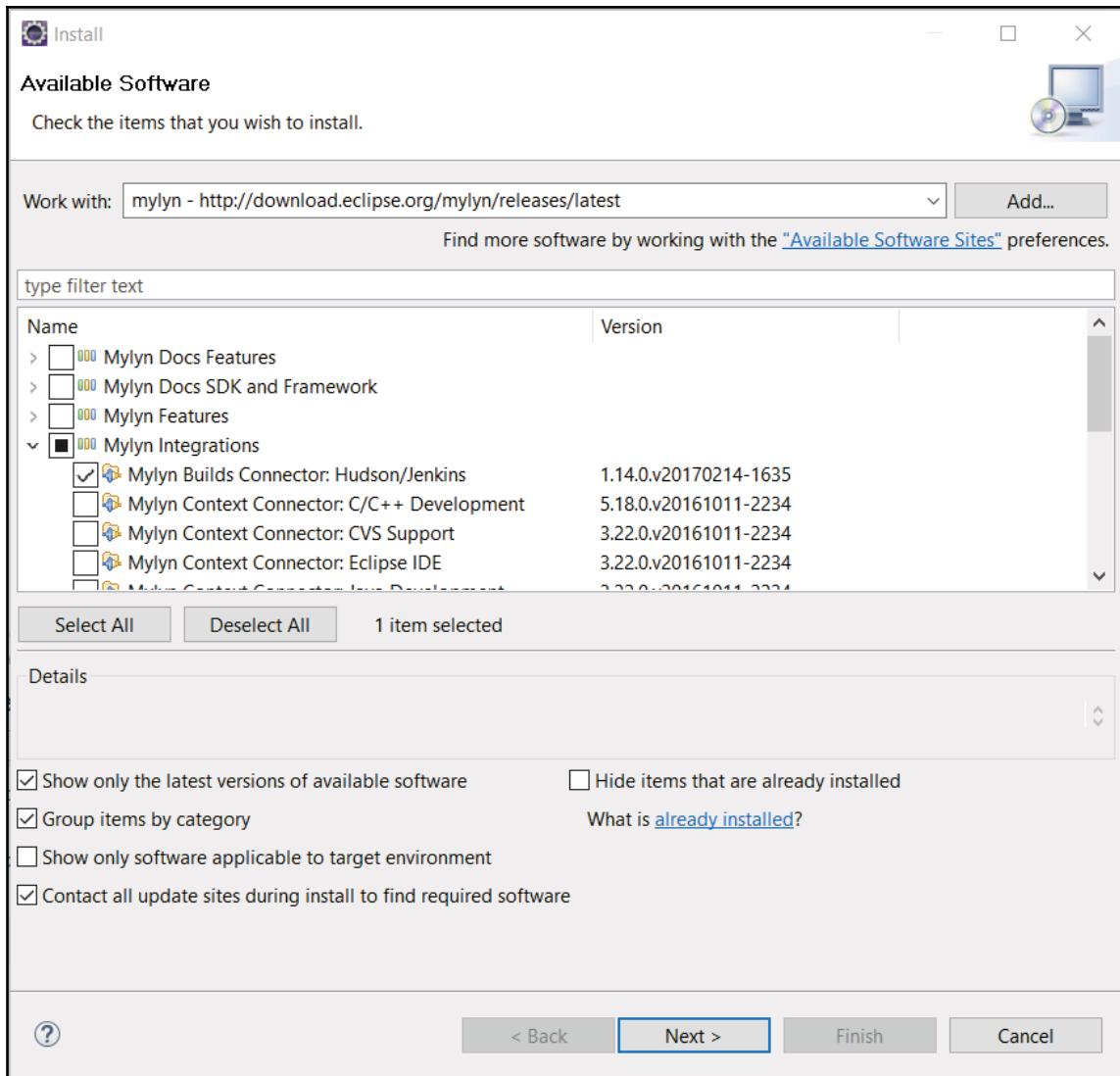
Can we execute a Jenkins job from Eclipse?

Yes, by following these steps:

1. Go to **Help | Install New Software...**:

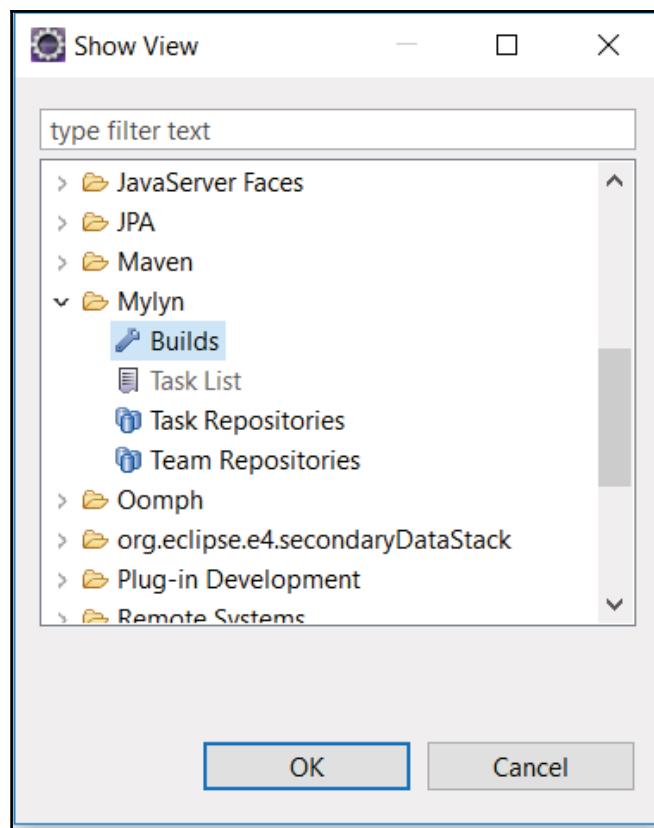


2. Add a site for Mylyn and click on **Next**:

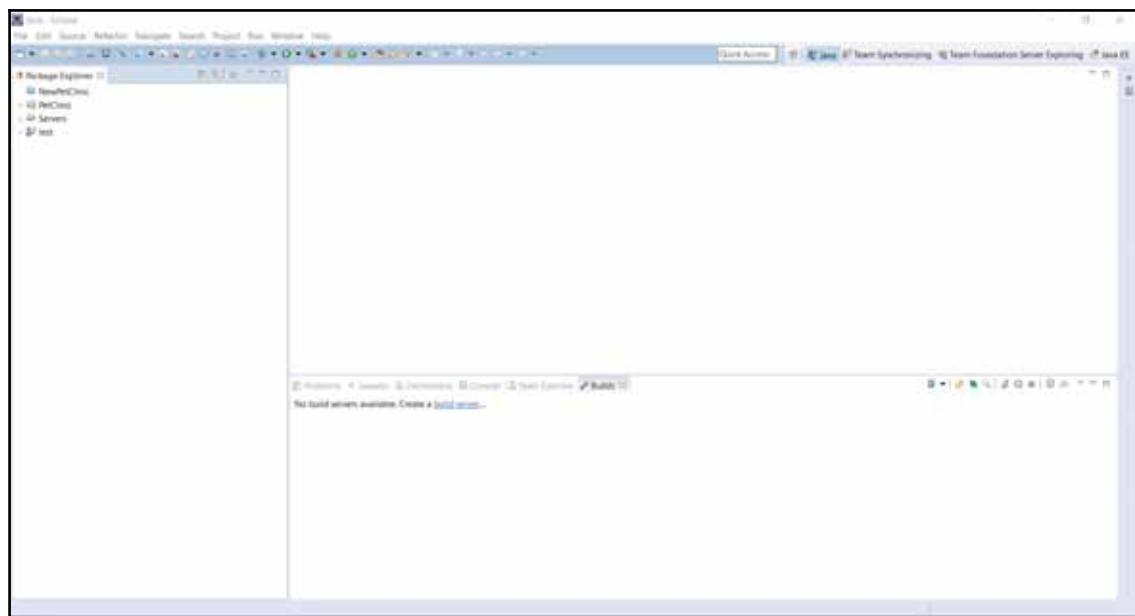


3. Review the items to be installed and click on **Next**.
4. Accept the terms of the license agreement.
5. Click on **Finish**.
6. It will start installing the **Mylyn** package. Once it is finished, restart Eclipse.

7. In the Windows menu, click on **Views**.
8. Select **Mylyn** and click on **Builds**.
9. Click **OK**:

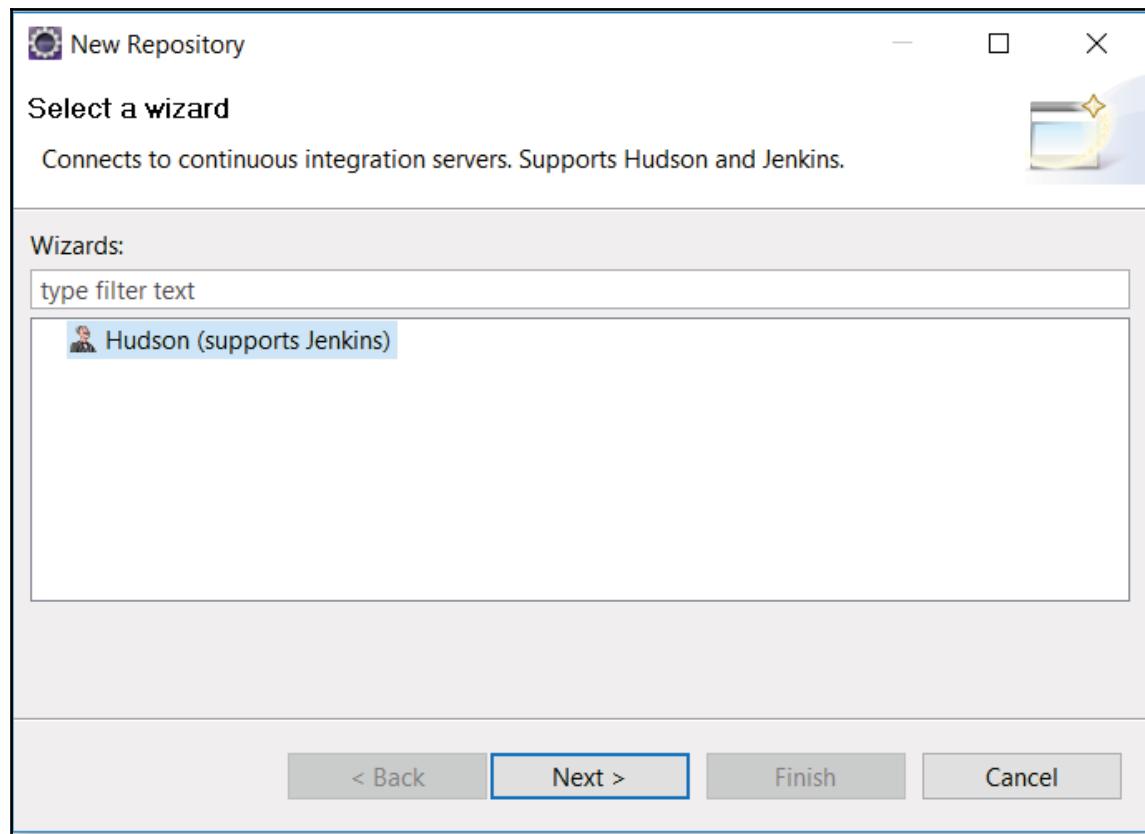


10. In the **Builds** section, click on the **build server** link:

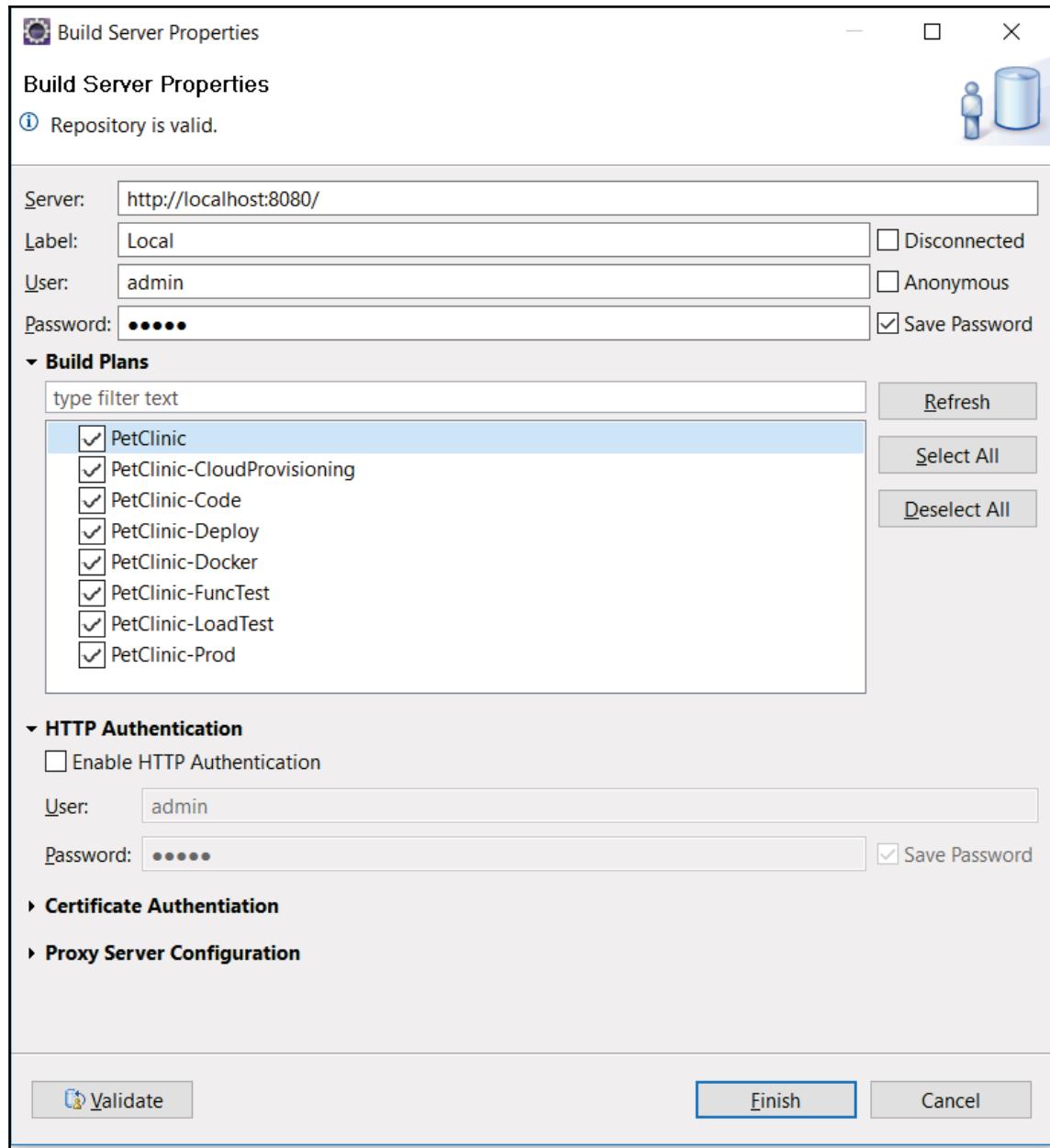


Build Section is Eclipse IDE with No build server configured

11. Select **Hudson (supports Jenkins)** and click on **Next**:

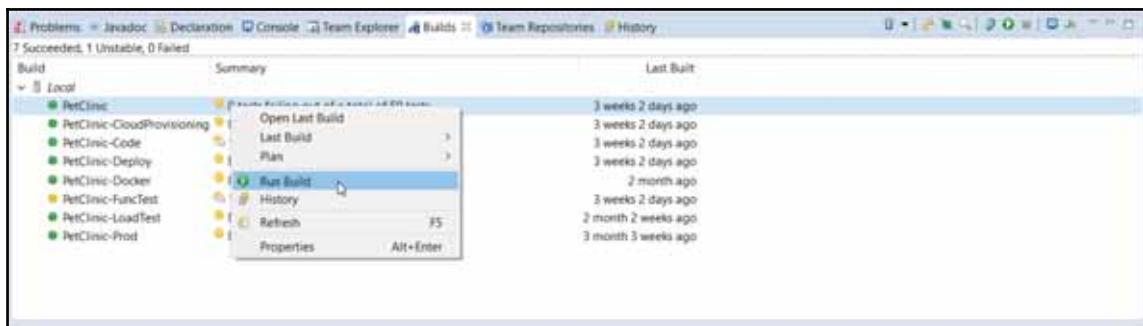


12. Provide Jenkins **Server**, **User**, and **Password** details. Click **Finish**:



13. Find the list of jobs in the **Builds** section.

14. Select any job and click on **Run Build** to execute it from Eclipse:



Try other options as an exercise.

Summary

Hooray! We have reached the end of this chapter. We have covered how to prepare an environment for CI by configuring Java, Ant, and Maven. We have also seen how to configure repositories and build tools in Jenkins. Finally, we have also covered how to integrate Integrated Development Environments with Jenkins so we can execute build jobs from Eclipse itself.

We have also created our first job, installed Git on a local machine, and created a job to access that Git repository in order to access the source code.

In the next chapter, we will configure sample applications for CI.

3

Managing Code Quality and Notifications

So far we have seen how to set up an environment to use Jenkins for Continuous Integration and we have also configured build tools in Jenkins. Integration of Eclipse with Jenkins was also covered and that will help developers to easily execute Jenkins jobs from the IDE.

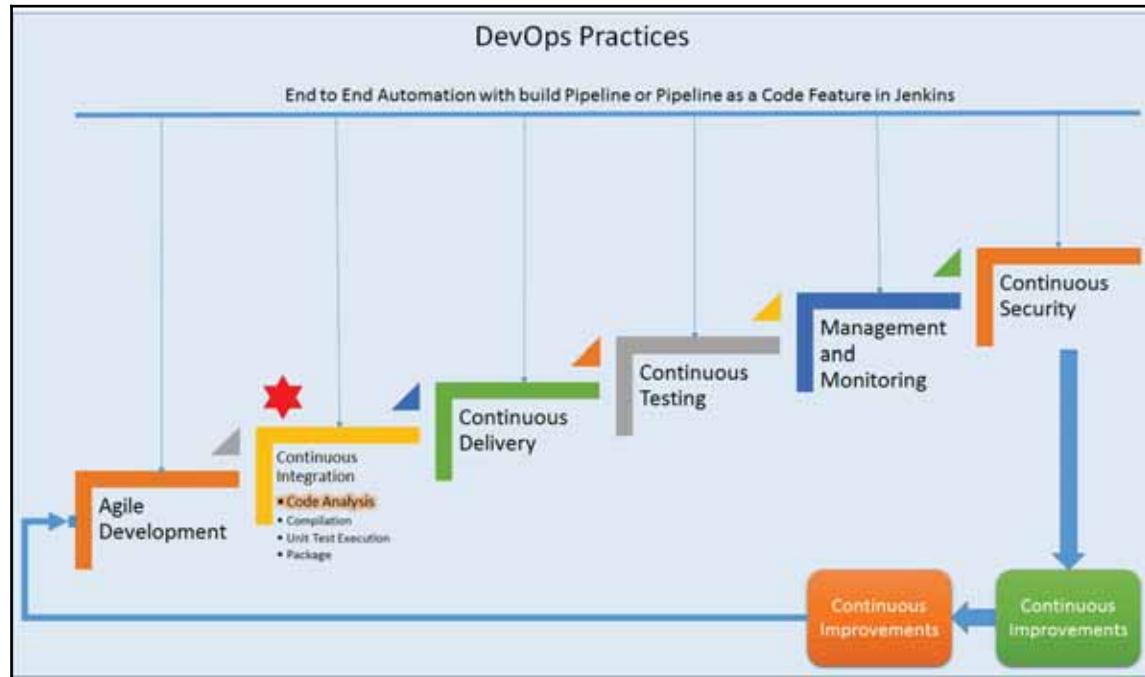
We will start our journey with Continuous Code Quality—static code analysis-- and it will be followed by Continuous Integration, Continuous Delivery, Continuous Testing, Continuous Deployment, Continuous Monitoring, and Continuous Security. For static code analysis, we will use **SonarQube** to analyze a spring-based Java project.

SonarQube is an open source quality management platform for maintaining Continuous Code Quality.

In this chapter, we will cover the following topics:

- Jenkins 2.x integration with Sonar 6.3
- Quality Gate plugin
- Email notifications on build status

In this chapter, we will cover static code analysis as part of Continuous Integration practice as a part of our DevOps journey:

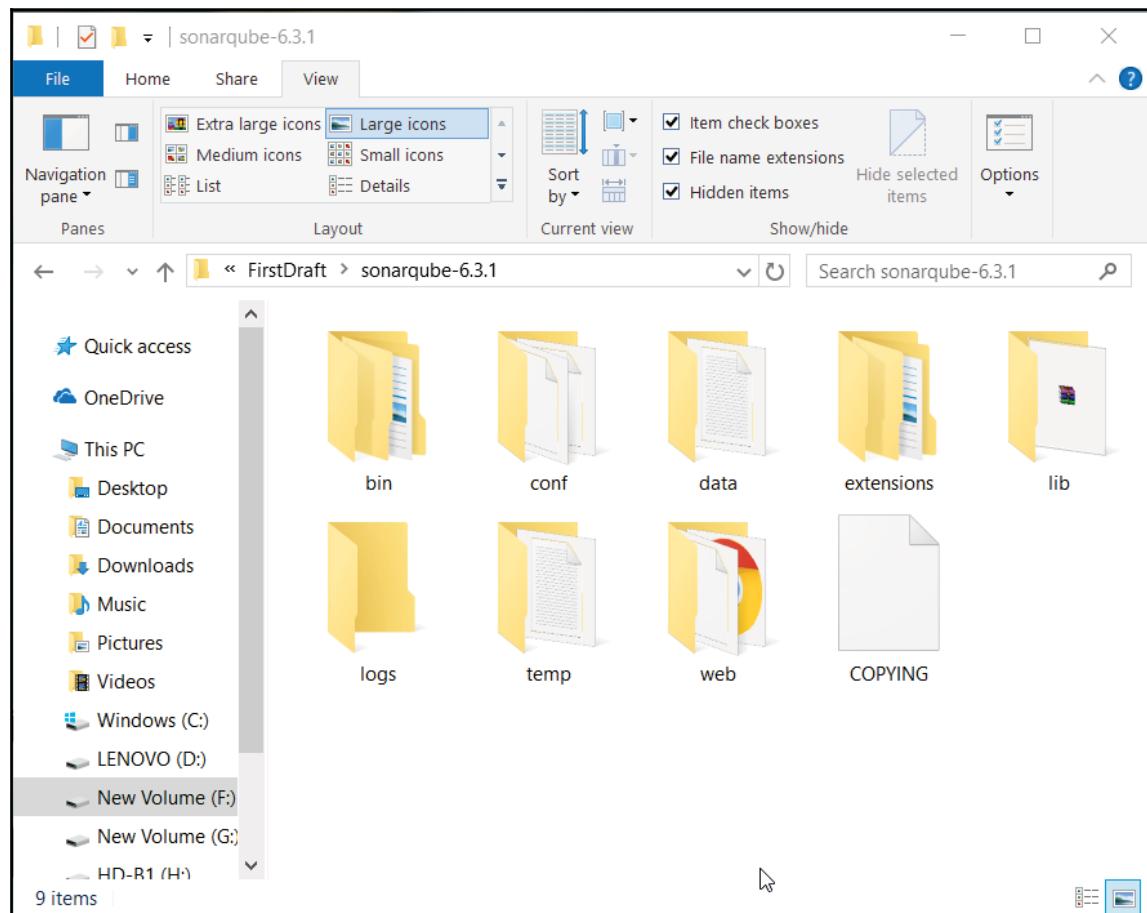


At the end of this chapter, we will know how to configure a SonarQube server with Jenkins and perform static code analysis.

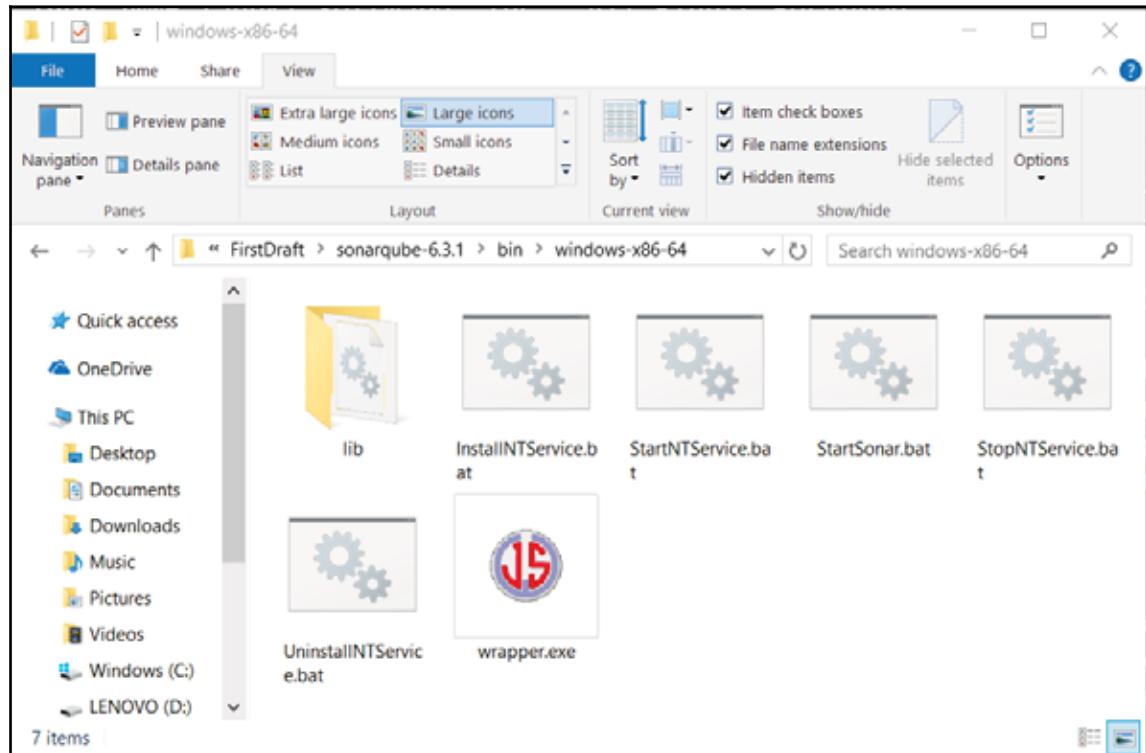
Jenkins 2.x integration with Sonar 6.3

In this chapter, we will use SonarQube 6.3 for static code analysis. Go to <https://www.sonarqube.org/downloads/> and download the latest version available:

1. Extract files:



2. Go to the `bin` directory and, based on the operating system and platform of the operating system, go to a specific directory:

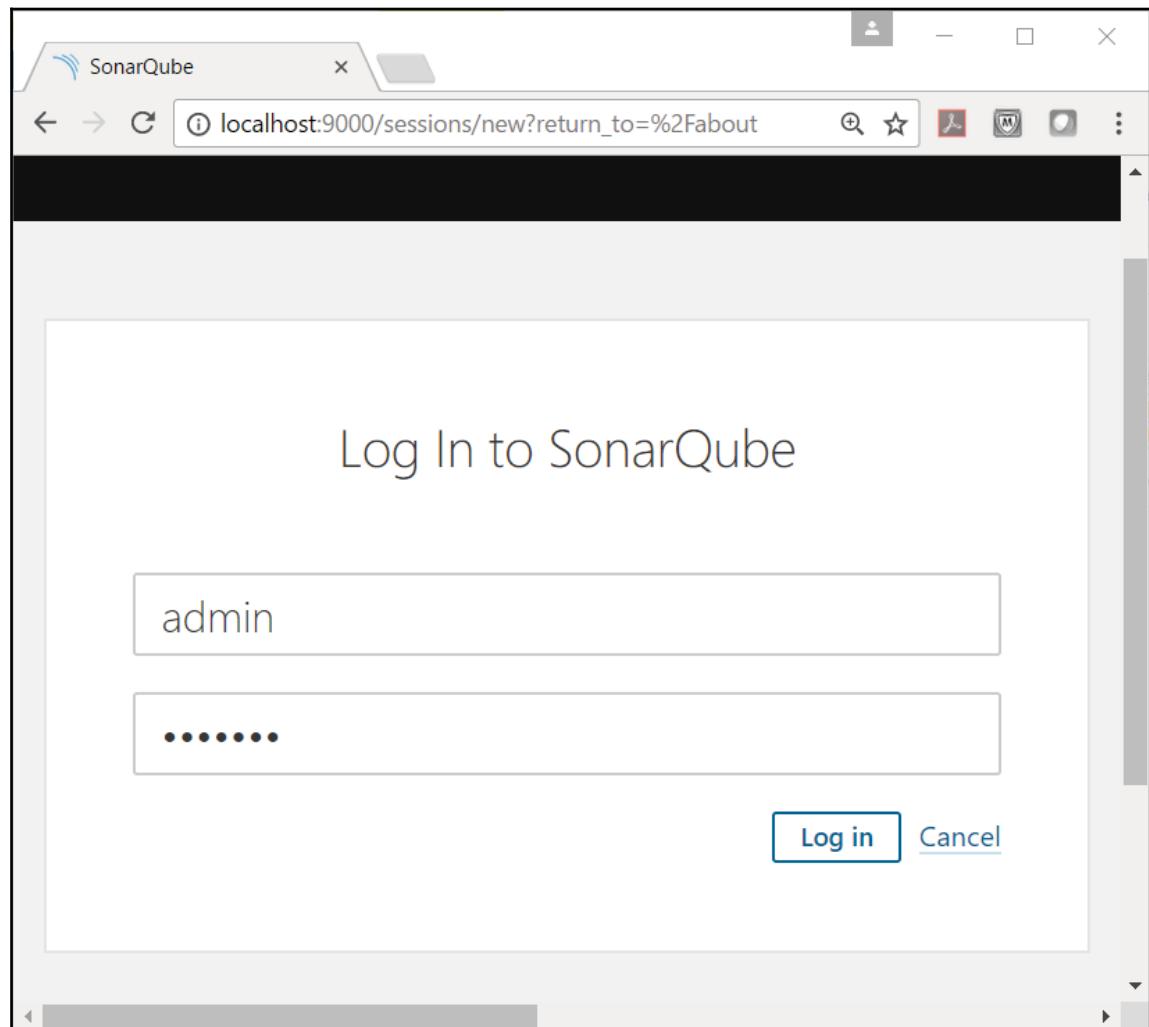


3. Execute `StartSonar.bat` in the command window. On Linux or MacOS execute the `.sh` file in the respective folder.

- Once SonarQube is up and running, open `http://localhost:9000` in a browser to visit the SonarQube dashboard:

The screenshot shows the SonarQube dashboard. At the top, there's a navigation bar with links for 'Projects', 'Issues', 'Rules', 'Quality Profiles', and 'Quality Gains'. On the right side of the header is a 'Log in' button and a search bar. Below the header, the main title 'Continuous Code Quality' is displayed, along with a 'Log in' button and a 'Read documentation' link. To the right, there's a summary section with the number '0' and the text 'Projects Analyzed'. Below this, there are three categories: 'Bugs' (0), 'Vulnerabilities' (0), and 'Code Smells' (0). Further down, there's a 'Multi-Language' section listing supported languages: Java, C/C++, C#, COBOL, ABAP, HTML, RPG, JavaScript, Objective C, XML, VB.NET, PL/SQL, Flex, Python, Groovy, PHP, Swift, Visual Basic, and PL/I. At the bottom, there's a 'Quality Model' section with three items: 'Bugs' (track code that is demonstrably wrong or highly likely to yield unexpected behavior), 'Vulnerabilities' (raise on code that is potentially vulnerable to exploitation by hackers), and 'Code Smells' (will confuse maintainers or give them pause; measured primarily in terms of the time they will take to fix).

5. Click on **Login** and give a default username and password --admin and default to-- log in as an administrator:



6. As of now, there is no project available in the SonarQube dashboard:

The screenshot shows the SonarQube interface with the 'Projects' tab selected. On the left, there are three main filter sections: 'Quality Gate', 'Reliability', and 'Security'. Each section contains several items, each with a circular icon and the text '0'. In the center, a message reads: 'You don't have any favorite projects yet.' followed by a link 'Explore Projects'. At the top right, it says 'Administrator' and '0 projects'.

7. Click on the **Quality Profiles** tab to get details on the default quality profiles available in SonarQube.

8. **Quality Profiles** are the heart of SonarQube; they are nothing but sets of rules specific to a language. If not mentioned explicitly, all the projects are analyzed with default profiles. However, it is ideal to have a profile for each project so specific rules can be set or deactivated. Each language has a default profile named **Sonar way**:

The screenshot shows the SonarQube interface for managing Quality Profiles. The top navigation bar includes 'Projects', 'Issues', 'Rules', 'Quality Profiles' (which is the active tab), 'Quality Gates', and 'Administration'. A 'Create' button is visible in the top right. The main content area is titled 'Quality Profiles' and contains the following information:

Quality Profiles are collections of rules to apply during an analysis. For each language there is a default profile. All projects not explicitly assigned to some other profile will be analyzed with the default.

All Profiles ▾

C#, 1 profile(s)	Projects	Rules	Updated	Used
Sonar way	1	134	Never	Never

Flex, 1 profile(s)	Projects	Rules	Updated	Used
Sonar way	1	60	Never	Never

Java, 1 profile(s)	Projects	Rules	Updated	Used
Sonar way	1	277	Never	Never

JavaScript, 1 profile(s)	Projects	Rules	Updated	Used
Sonar way	1	100	Never	Never

Recently Added Rules

- Skipped unit tests should be either removed or fixed. (OK, not yet activated)
- Failed unit tests should be fixed. (OK, not yet activated)
- Source files should not have any duplicated lines. (OK, not yet activated)
- Source files should have a sufficient density. (OK, not yet activated)
- Skipped unit tests should be either removed or fixed. (INPROGRESS, not yet activated)
- Failed unit tests should be fixed. (INPROGRESS, not yet activated)
- Source files should not have any duplicated lines. (INPROGRESS, activated on 1 profile(s))
- Lines should have sufficient coverage by tests. (INPROGRESS, not yet activated)
- Source files should have a sufficient density. (INPROGRESS, not yet activated)
- Branches should have sufficient coverage by tests. (INPROGRESS, not yet activated)

9. A Quality Gate is used to enforce policy in organization for static code analysis. The SonarQube way is the default **Quality Gate**:

Metric	Over Leak Period	Operator	Warning	Error
Coverage on New Code	Always	is lower than	80	<input type="button" value="Update"/> <input type="button" value="Delete"/>
Maintainability Rating on New Code	Always	is worse than	<input type="button" value="Update"/> <input type="button" value="Delete"/>	<input type="button" value="Update"/> <input type="button" value="Delete"/>
Reliability Rating on New Code	Always	is worse than	<input type="button" value="Update"/> <input type="button" value="Delete"/>	<input type="button" value="Update"/> <input type="button" value="Delete"/>
Security Rating on New Code	Always	is worse than	<input type="button" value="Update"/> <input type="button" value="Delete"/>	<input type="button" value="Update"/> <input type="button" value="Delete"/>

Projects
You must first select specific projects for the default quality gate.

- Click on the **Rules** tab to get more details about the existing rules available in profiles:

Rule	Severity	Category	Details
"equals()" should not be used to test the values of "Atomic" classes	Java	Style, bug	<input type="button" value="Details"/>
"@Deprecated" code should not be used	Java	Code Smell, cert, cost, obsolet	<input type="button" value="Details"/>
"@NonNull" value should not be set to null	Java	Style, bug	<input type="button" value="Details"/>
"@Override" should be used on overriding and implementing methods	Java	Code Smell, style, readability	<input type="button" value="Details"/>
"action" mappings should not have too many "forward" entries	Java	Code Smell, brain-overloaded, style	<input type="button" value="Details"/>
"ArrayList" should be used for primitive arrays	Java	Style, bug, performance	<input type="button" value="Details"/>
"insert" should only be used with boolean variables	Java	Style, bug, cert, suspicious	<input type="button" value="Details"/>
"BigDecimal(double)" should not be used	Java	Style, bug, cert	<input type="button" value="Details"/>
"catch" clauses should do more than rethrow	Java	Code Smell, cert, cleanup, finding, unused	<input type="button" value="Details"/>
"clone" should not be overridden	Java	Code Smell, suspicious	<input type="button" value="Details"/>
"Cloneables" should implement "clone"	Java	Style, bug	<input type="button" value="Details"/>
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	Java	Code Smell, obfuscate, pitfall	<input type="button" value="Details"/>
"compareTo" results should not be checked for specific values	Java	Style, bug, unpredictable	<input type="button" value="Details"/>
"compareTo" should not return "Integer.MIN_VALUE"	Java	Style, bug	<input type="button" value="Details"/>

11. Go to **Quality Profiles** and select the **Sonar way** default profile. Observe total **Active** and **Inactive** rules:

The screenshot shows the SonarQube interface with the 'Quality Profiles' tab selected. The profile 'Sonar way' is displayed, which is the default profile. The left sidebar shows a summary of rules: Total Active (277) and Inactive (120). The right sidebar shows inheritance from 'Sonar way' leading to 277 active rules. A note states that every project not specifically associated with this profile will be associated by default. A warning at the bottom indicates that the embedded database should be used for evaluation purposes only and that it does not support upgrading or migrating data.

Rules	Active	Inactive
Total	277	120
• Bugs	129	29
• Vulnerabilities	18	11
• Code Smells	130	81

Projects: Every project not specifically associated with a quality profile will be associated to this one by default.

Embedded database should be used for evaluation purpose only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA
Version 6.3.1 (build 21392) · LGPL v3 · Community · Documentation · Get Support · Plugins · Web API · About

12. Go to the Jenkins dashboard and click on **Manage Jenkins**. Go to **Manage Plugins** and in the **Available** tab find the SonarQube plugin.

13. Click on **Install without restart**:

The screenshot shows the Jenkins Plugin Manager interface. The browser address bar indicates the URL is `localhost:8080/pluginManager/available`. The Jenkins logo is at the top left, and the user is logged in as 'admin'. The main content area displays the 'Available' tab of the plugin list. A search bar at the top right contains the text 'Sonarqube'. The table lists available plugins:

Install	Name	Version
<input checked="" type="checkbox"/>	SonarQube Scanner for Jenkins	2.6.1
<input type="checkbox"/>	Mashup Portlets	
	Additional Dashboard Portlets: Generic JS Portlet (lets you put in arbitrary content via JS), Recent Changes Portlet (shows the SCM changes for a given job), SonarQube Portlets (show SonarQube statistics directly in Jenkins) and Test Results Portlet (shows the test results for a given job).	1.0.8

At the bottom of the page, there are three buttons: 'Install without restart' (highlighted in blue), 'Download now and install after restart', and 'Check now'. A status message says 'Update information obtained: 3 days 10 hr ago'. The footer of the page includes the text 'Page generated: May 25, 2017 11:46:58 PM IST BEST API Jenkins ver. 2.61'.

14. Verify when installation is successful:

Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

jQuery plugin Success

SonarQube Scanner for Jenkins Success

[Go back to the top page](#)
(you can start using the installed plugins right away)

Restart Jenkins when installation is complete and no jobs are running

15. Go to the Jenkins dashboard and click on **Manage Jenkins**.
16. Click on **Configure system** and find the SonarQube section.
17. Click on **Add SonarQube**.

18. Provide name, URL, and version. It also asks for a **Server Authentication token**. We can get it from the SonarQube server dashboard:

The screenshot shows the Jenkins 'Configure System' page with the 'Projects - SonarQube' tab selected. Under the 'SonarQube servers' section, there is a configuration for 'SonarQube installations'. The configuration includes:

- Name:** Sonarqube6.3
- Server URL:** http://localhost:9000/ (Default is http://localhost:9000)
- Server version:** 5.3 or higher
- Server authentication token:** (Input field)
- SonarQube account login:** (Input field)
- SonarQube account password:** (Input field)

Below the configuration fields, there is an 'Advanced...' button and a red 'Delete SonarQube' button. At the bottom of the page, there are 'Save' and 'Apply' buttons.

19. Click on the **Administration** tab. In the **Security** menu, click on **Users**:

The screenshot shows the SonarQube administration interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. The Administration link is highlighted in orange. Below the navigation is a secondary menu labeled "Administration" which includes Configuration, Security, Projects, and System. Under the Security dropdown, a sub-menu is open with options: General, Edit global, Users (which has a cursor icon pointing to it), Groups, Global Permissions, and Permission Templates. To the right of this menu, the text "instance." is visible.

20. Observe that there is a **0** token for **Administrator**:

The screenshot shows the SonarQube User Management page under the Administration tab. The top navigation bar includes Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. Below the navigation is a secondary menu labeled "Administration" with Configuration, Security, Projects, and System. Under the Security dropdown, the "Users" section is active. It displays a table with columns for SCM Accounts, Groups, and Tokens. One user, "Administrator admin", is listed under SCM Accounts. This user is also listed under Groups as "sonar-administrator" and "sonar-user". Under the "Tokens" column, there is a value of "0" next to a small icon. A "Create User" button is located in the top right corner of the users section.

21. Click on **Tokens**:

The screenshot shows the SonarQube Administration interface under the Security tab. On the left, there's a sidebar with 'Administration' and 'Configuration' sections. The main area has tabs for 'Users', 'SCM Accounts', 'Groups', and 'Tokens'. Under 'Users', there's a search bar and a table with one row: 'Administrator admin'. Under 'Groups', there are two rows: 'sonar-administrators' and 'sonar-users'. Under 'Tokens', there's a table with one row: '0'. A 'Create User' button is visible in the top right corner.

22. Give a name in the **Generate Tokens** section and click on **Generate**:

The screenshot shows the 'Tokens' page. At the top, it says 'Tokens'. Below that is a table with columns 'Name' and 'Created'. There is one entry: 'No tokens'. Below the table is a section titled 'Generate Tokens' with a text input field containing 'JenkinsEssentials' and a blue 'Generate' button. A cursor is hovering over the 'Generate' button. In the bottom right corner of the page, there is a 'Done' link.

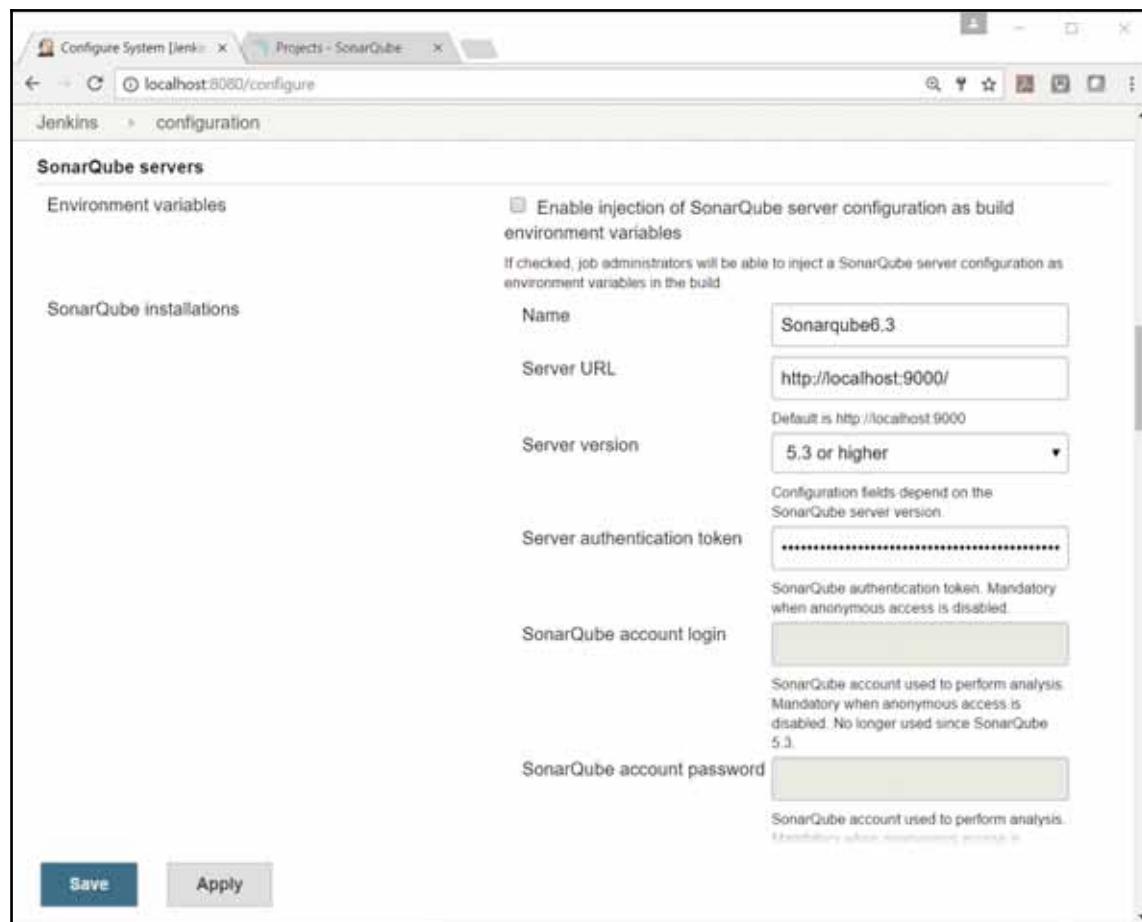
23. Copy the newly created **Token**. Click on **Done**:

The screenshot shows the 'Tokens' section of the SonarQube interface. It lists a single token named 'JenkinsEssentials' created on 'May 25, 2017'. A red-bordered 'Revoke' button is visible next to the creation date. Below this, there's a 'Generate Tokens' section with a text input field 'Enter Token Name' and a 'Generate' button. A success message box displays the text: 'New token "JenkinsEssentials" has been created. Make sure you copy it now, you won't be able to see it again!'. A 'Copy' button with a cursor icon is highlighted, pointing to the generated token value: '4dc45d3a67f0c445845d53dc1d47e1e8279ff2ca'. At the bottom right of the page is a 'Done' button.

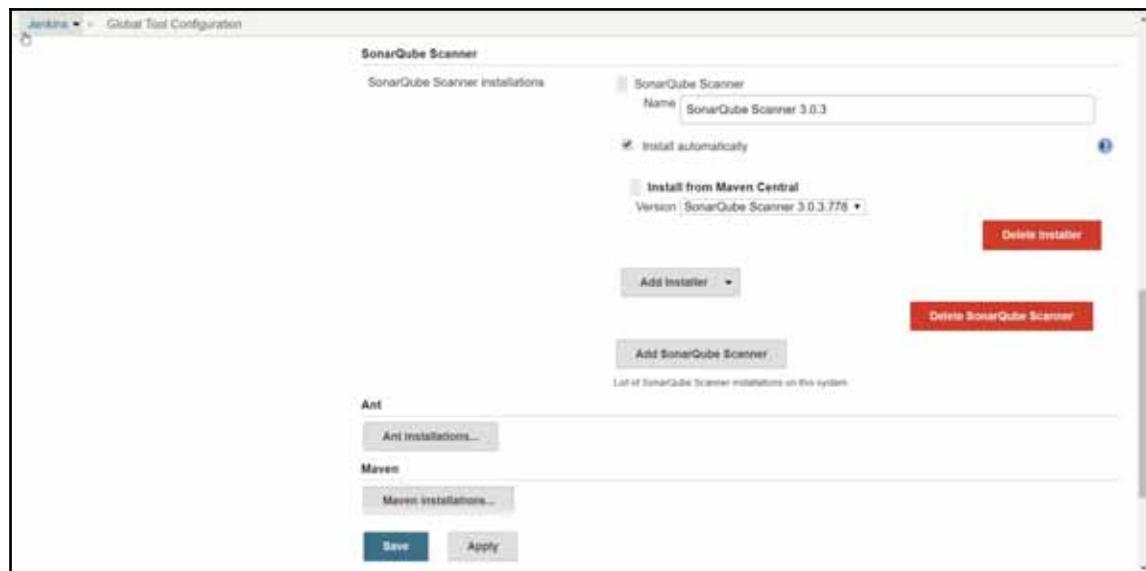
24. Verify the number of **Tokens** for the **Administrator** user:

The screenshot shows the 'Administration - Users' section of the SonarQube interface. It lists the 'Administrator' user with the login 'admin'. Under the 'Tokens' column, there is one token listed: 'sonar-administrators'. The interface includes standard SonarQube navigation links like Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration at the top, and Configuration, Security, Projects, and Systems dropdowns in the header.

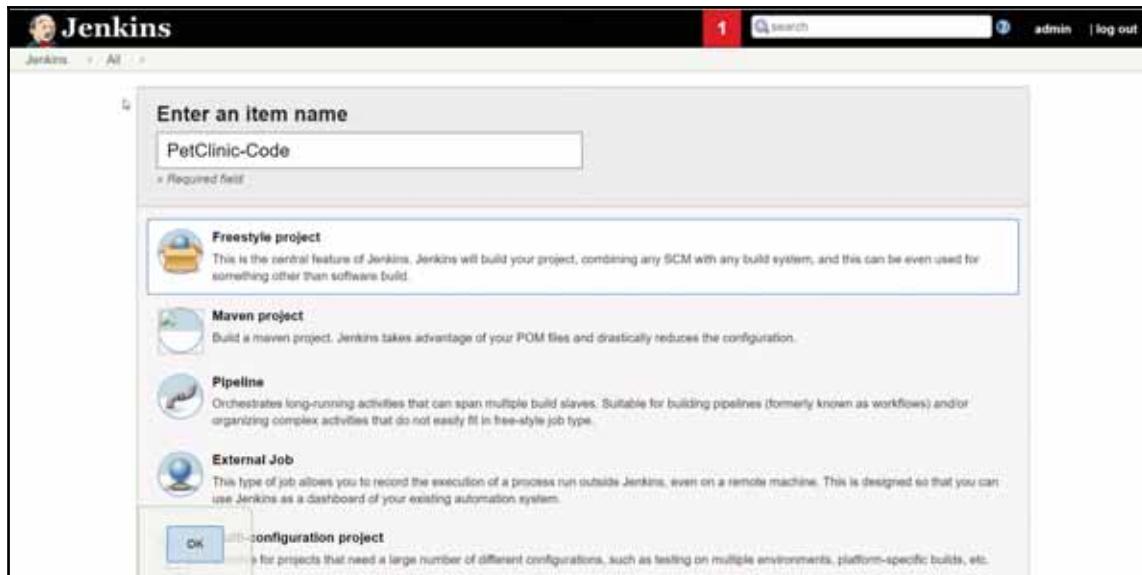
25. Paste the token value in Jenkins and **Save**:



26. Go to **Global Tool Configuration** and configure SonarQube Scanner:



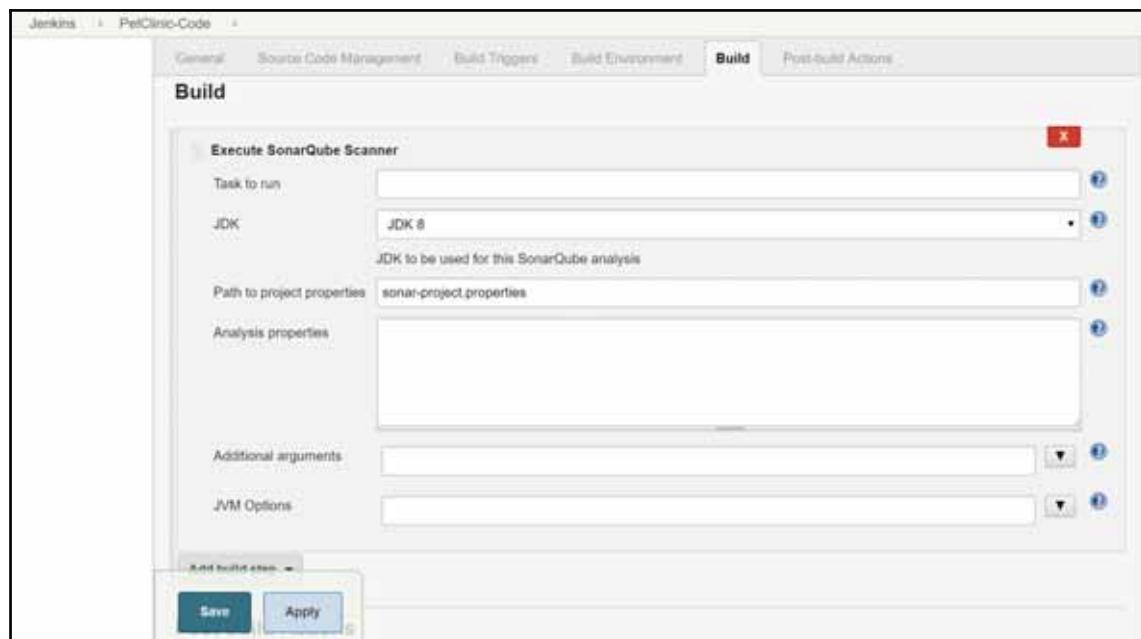
27. Go to the Jenkins dashboard and click on **New Item**.
28. Give a name, PetClinic-Code, and select **Freestyle project**. We are going to perform static code analysis on the sample application using SonarQube here:



29. Provide a repository URL in the **Source Code Management** section.
30. In the **Build**, section select **Execute SonarQube Scanner**:

A screenshot of the Jenkins job configuration for 'PetClinic-Code'. The 'Build Environment' tab is selected. Under the 'Build' section, the 'Add build step' dropdown menu is open, and 'Execute SonarQube Scanner' is highlighted with a blue selection bar. Other options in the dropdown include: Execute Windows batch command, Execute shell, Invoke Ant, Invoke Gradle script, Invoke top-level Maven targets, Run with timeout, Set build status to "pending" on GitHub commit, SonarQube Scanner for MSBuild - Begin Analysis, and SonarQube Scanner for MSBuild - End Analysis.

31. Select **JDK** and provide a **Path to project properties**:



32. `sonar-project.properties` contains the following details. `sonar.source` is the main property for static code analysis. We inform SonarQube which directory needs to be analyzed. We can add same content in the analysis properties to achieve same results and not require `sonar-project.properties`:

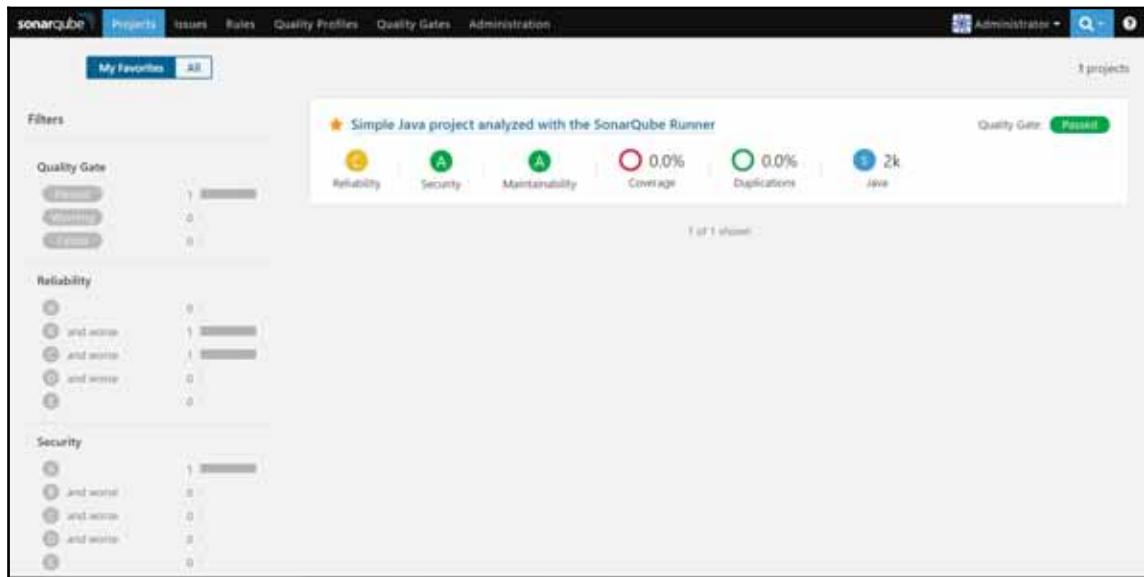
```
# Required metadata
sonar.projectKey=java-sonar-runner-simple
sonar.projectName=Simple Java project analyzed with the SonarQube Runner
sonar.projectVersion=1.0

# Comma-separated paths to directories with sources (required)
sonar.sources=src

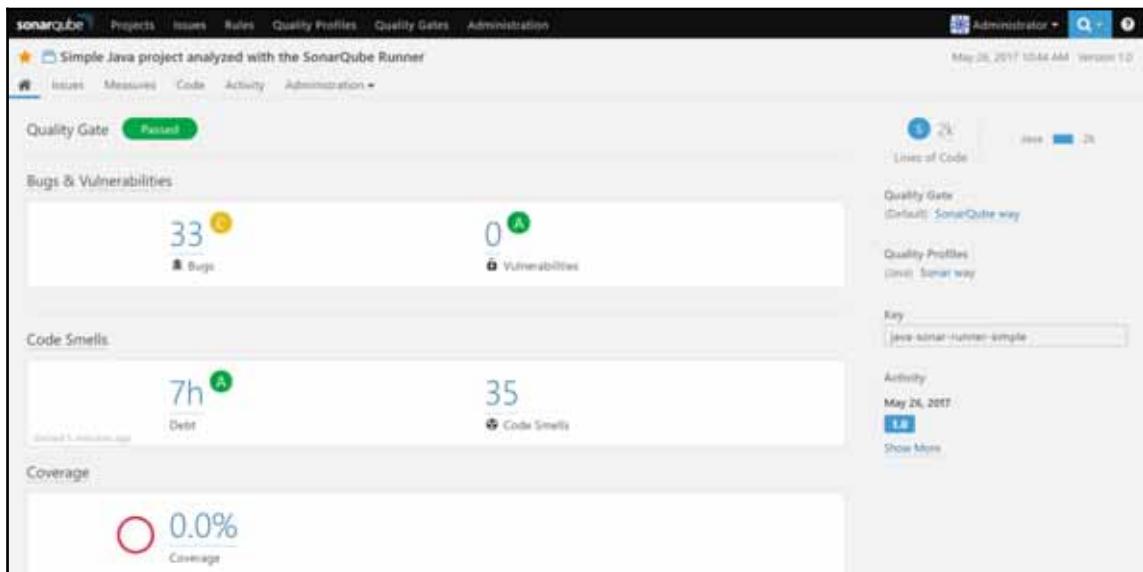
# Language
sonar.language=java

# Encoding of the source files
sonar.sourceEncoding=UTF-8
```

33. If language property is not mentioned, then SonarQube is intelligent enough to detect the language available in the source files. The same properties can be given in the **Analysis Properties** textbox.
34. Click on **Save** and then click on **Build Now**.
35. Once the Jenkins job is executed successfully, go to SonarQube and verify.
36. Click on the **Project**:



37. It gives details on **Bugs**, **Vulnerability**, and **Code Smells**. Verify each tab and content available in it as a self-exercise:



So we have successfully done static code analysis of a sample application using Jenkins.

Now let's create a new **Quality Profile** and assign the project so every time static code analysis is performed, a default profile is not used, but a custom profile is utilized:

1. Go to **Quality Profiles**; in the Java section, copy the default profile:

The screenshot shows the SonarQube interface for managing Quality Profiles. There are five profiles listed:

- C#, 1 profile(s): Sonar way (Projects: Default, Rules: 134, Updated: Never, Used: Never)
- Flex, 1 profile(s): Sonar way (Projects: Default, Rules: 60, Updated: Never, Used: Never)
- Java, 1 profile(s): Sonar way (Projects: Default, Rules: 277, Updated: 6 minutes ago, Used: Never). A context menu is open over this entry with the following options: Activate More Rules, Back up, Compare, Copy, and Rename. The "Copy" option is highlighted.
- JavaScript, 1 profile(s): Sonar way (Projects: Default, Rules: 111, Updated: Never, Used: Never)
- PHP, 3 profile(s): Sonar way (Projects: Default, Rules: 202, Updated: Never, Used: Never)

A sidebar on the right lists "Recently Added Rules" with several items:

- Skipped unit tests should be either removed or fixed, not yet activated
- Failed unit tests should be fixed, not yet activated
- Source files should not have any duplicated lines, not yet activated
- Source files should have a sufficient density of comments, not yet activated
- Skipped unit tests should be either removed or fixed, PHP, not yet activated
- Failed unit tests should be fixed, PHP, not yet activated
- Source files should not have any duplicated lines, PHP, activated on 1 profile(s)
- Lines should have sufficient coverage by tests, PHP, not yet activated
- Source files should have a sufficient density of comments, PHP, not yet activated
- Branches should have sufficient coverage by tests, PHP, not yet activated

2. Give a specific name to it and click on **Copy**:

The dialog box is titled "Copy Profile Sonar way - Java". It contains a single input field labeled "New name*" with the value "PetClinic". At the bottom right are two buttons: "Copy" (highlighted in blue) and "Cancel".

3. We can specify projects for a specific quality profile by clicking on **Change Projects** as well:

The screenshot shows the SonarQube interface for managing Quality Profiles. The top navigation bar includes 'Projects', 'Issues', 'Rules', 'Quality Profiles' (which is the active tab), 'Quality Gates', and 'Administration'. Below the navigation is a sub-header 'Quality Profiles / Java'. A table on the left displays rule statistics: Total (Active: 277, Inactive: 120), Bugs (Active: 129, Inactive: 20), Vulnerabilities (Active: 18, Inactive: 13), and Code Smells (Active: 185, Inactive: 87). A button 'Activate More' is at the bottom of this section. To the right, under 'Inheritance', it shows 'PetClinic' with '277 active rules'. Under 'Projects', it states 'No projects are explicitly associated to the profile.' A 'Change Projects' button is located here. At the top right, there are buttons for 'Changelog' and 'Actions'.

4. We can also specify **Quality Profile** by clicking a specific project. Go to **Administration**, select **Quality Profiles**, and select the custom profile created for Java:

The screenshot shows the SonarQube Administration Quality Profiles page. The top navigation bar includes 'Issues', 'Measures', 'Code', 'Activity', 'Administration' (which is the active tab), and 'Administration'. The sub-header 'Quality Profiles' is displayed. A note says: 'Choose which profile is associated with this project on a language-by-language basis. (Note that you will only need to select profiles for multiple languages for multi-language projects.)' Below this, a table maps languages to quality profiles: C# (Default: Sonar way), F# (Default: Sonar way), Java (Default: Sonar way, PetClinic is selected), JavaScript (Default: Sonar way), PHP (Default: Sonar way), and Python (Default: Sonar way).

5. Just for troubleshooting, if we come across a Jenkins job failure due to SCM blame, then we need to fix that by configuring it in SonarQube.

6. Go to SonaQube and disable the SCM sensor:

The screenshot shows the SonarQube administration interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. The Administration tab is selected. Below the navigation, there are tabs for Configuration, Security, Projects, and System. The main content area is titled "General Settings" with the sub-instruction "Edit global settings for this SonarQube instance." On the left, a sidebar lists Analysis Scope options: C#, Flex, General, Java, PHP, Python, Scanner for MSBuild, SCM, Security, and SonarJS. The "SCM" option is currently selected. The right panel displays the "SCM" configuration section. It contains a heading "Disable the SCM Sensor" with a descriptive text: "Disable the retrieval of blame information from Source Control Manager" and a key: "sonar.scm.disabled". A toggle switch is shown in the "on" position. At the bottom of this section are "Save" and "Cancel" buttons. Below this, the "SVN" section is partially visible, showing fields for "Passphrase" (with a "Set" button) and "Username" (with a text input field). The overall background is light gray, and the active section is highlighted with a yellow background.

In the next section, we will cover the Quality Gate plugin.

Quality Gate plugin

The Quality Gate plugin is useful if we want to fail the Jenkins job based on the result of Quality Gate:

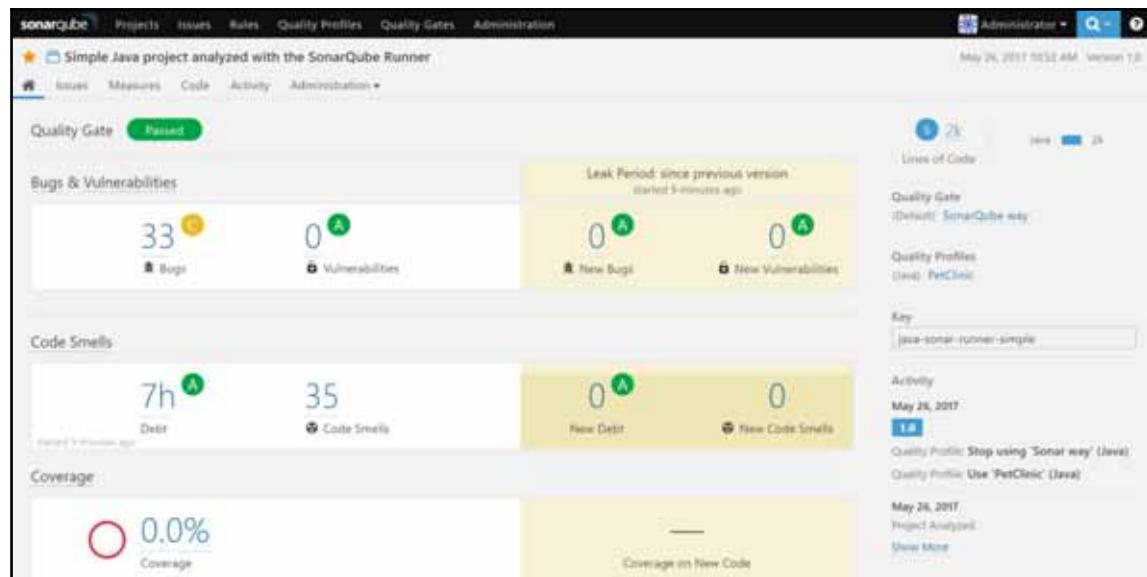
1. Install the **Sonar Quality Gates Plugin** in Jenkins:

The screenshot shows the Jenkins Plugins page with the 'Available' tab selected. A list of available plugins is displayed, with the 'Sonar Quality Gates Plugin' checked for installation. The table includes columns for Name, Version, and a brief description. At the bottom, there are buttons for 'Install without restart' and 'Download now and install after restart', along with a 'Check now' button and an update status message.

Updates	Available	Installed	Advanced
Install ↓			
	<input type="checkbox"/> CodeSonar Plugin		2.0.5
	<input checked="" type="checkbox"/> SonarQube Scanner for Jenkins		2.6.1
	<input type="checkbox"/> Sonargraph Integration Jenkins Plugin		2.0.2
	<input type="checkbox"/> Sonargraph Plugin		1.6.4
	<input type="checkbox"/> Mashup Portlets		
	<input type="checkbox"/> Additional Dashboard Portlets: Generic JS Portlet (lets you pull in arbitrary content via JS), Recent Changes Portlet (shows the SCM changes for a given job), SonarQube Portlets (show SonarQube statistics directly in Jenkins) and Test Results Portlet (shows the test results for a given job)		1.0.8
	<input type="checkbox"/> Sonar Gerrit Plugin		2.0
	<input type="checkbox"/> Quality Gates Plugin		2.5
	<input type="checkbox"/> Sonar Quality Gates Plugin	Fails the build whenever the Quality Gates criteria in the Sonar analysis aren't met (the project Quality Gates status is different than "Passed")	1.0.4
	<input checked="" type="checkbox"/> Sonar Quality Gates Plugin	Fails the build whenever the Quality Gates criteria in the Sonar 5.6+ analysis aren't met (the project Quality Gates status is different than "Passed")	

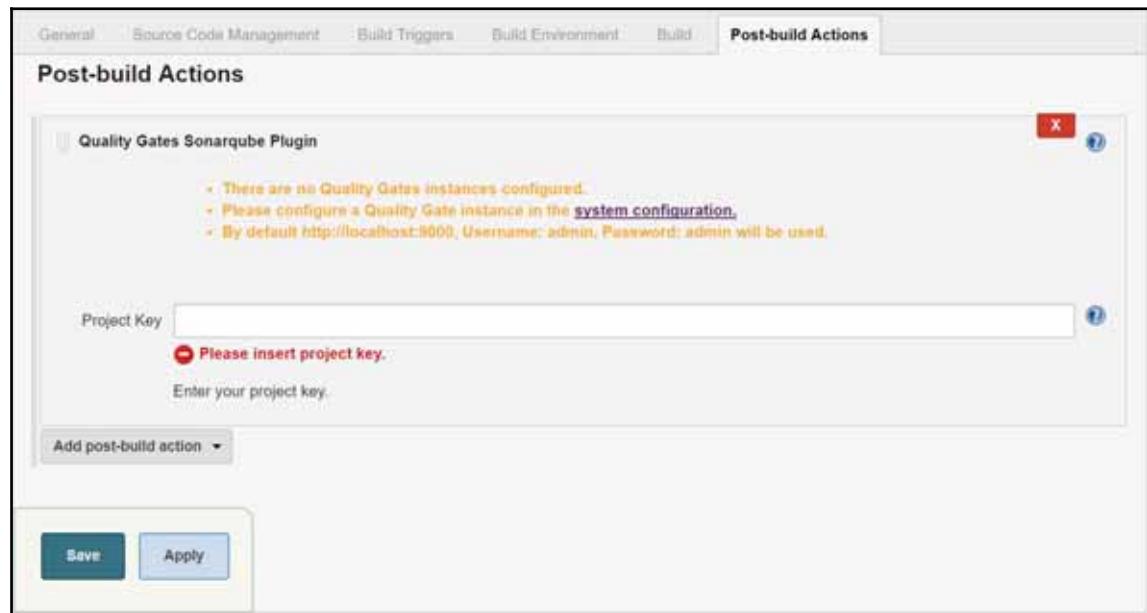
Install without restart Download now and install after restart Update information obtained: 6 min 29 sec ago Check now

2. As of now, Quality Gate is passed for our sample application:



3. Go to the **Quality Gates** tab and add a condition where if issues are greater than 10, then it should give an error.

4. In the same **PetClinic-Code** build job, as a **Quality Gates SonarQube** plugin action from **Add post-build action**. It asks for the Quality Gates configuration in the Jenkins configuration:



5. Go to **Manage Jenkins**, click on **Configure system**, and configure Sonar instance for **Quality Gates**:

The screenshot shows the Jenkins configuration interface at localhost:8080/configure. The left sidebar shows 'Jenkins' and 'configuration'. The main content area is titled 'Quality Gates - Sonarqube'. It contains the following fields:

Name	Value	Description
SonarQube Server URL	http://localhost:9000	Default value is 'http://localhost:9000'
SonarQube account login	admin	Default value is 'admin'
SonarQube account password	*****	Default value is 'admin'
Time to wait next check (milliseconds)	10000	Default value is '10000'

Buttons at the bottom include 'Delete' (red) and 'Add Sonar instance'.

6. We already have `sonar-project.properties` in the application. Note the project key:

```
# Required metadata
sonar.projectKey=java-sonar-runner-simple
sonar.projectName=Simple Java project analyzed with the SonarQube Runner
sonar.projectVersion=1.0

# Comma-separated paths to directories with sources (required)
sonar.sources=src

# Language
sonar.language=java

# Encoding of the source files
sonar.sourceEncoding=UTF-8
```

7. In the Jenkins job, enter the same **Project Key** and click **Save**:

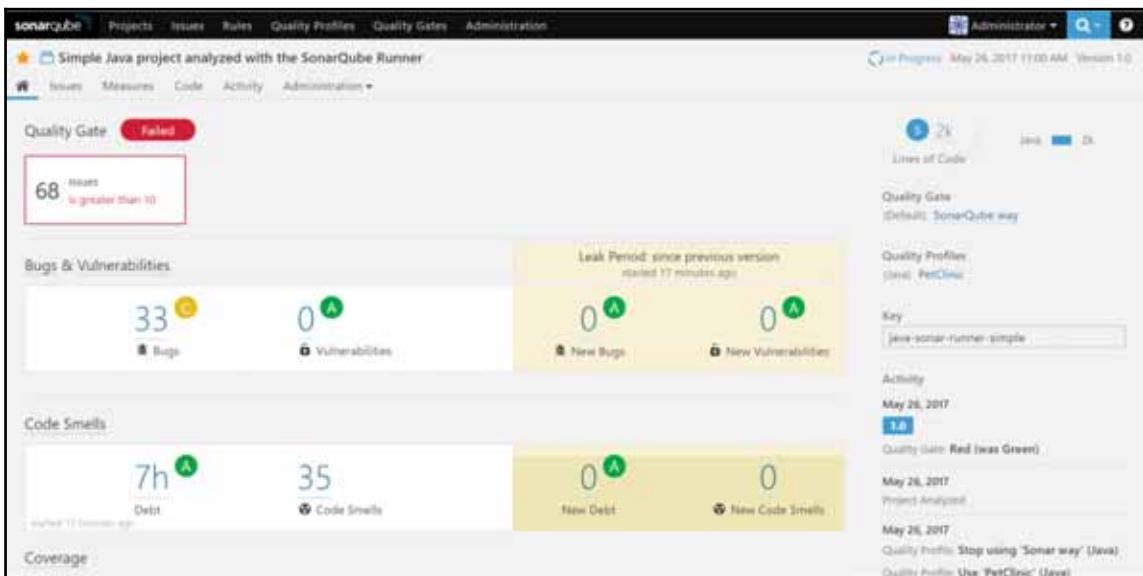


8. Click on **Build now** to execute a Jenkins build job.

9. The Jenkins job has failed. Go through the console output and the reason will be Quality Gate failure:

```
INFO: ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard/index/java-sonar-runner-simple
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted
analysis report
INFO: More about the report processing at http://localhost:9000/api/ce/task?id=AVxDQYIS-pm0HP4V18a0
INFO: Task total time: 43.474 s
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
INFO: Total time: 1:01.431s
INFO: Final Memory: 46M/134M
INFO: -----
Has build IN_PROGRESS with id: AVxDQYIS-pm0HP4V18a0 - waiting 10000 to execute next check.
Has build IN_PROGRESS with id: AVxDQYIS-pm0HP4V18a0 - waiting 10000 to execute next check.
Has build IN_PROGRESS with id: AVxDQYIS-pm0HP4V18a0 - waiting 10000 to execute next check.
ERROR: Build step failed with exception
org.quality.gates.sonar.api.MaxExecutionTimeException: Max time to wait sonar job!
    at
org.quality.gates.sonar.api.QualityGatesProvider.getAPIResultsForQualityGates(QualityGatesProvider.java:82)
    at org.quality.gates.jenkins.plugin.BuildDecision.getStatus(BuildDecision.java:22)
    at org.quality.gates.jenkins.plugin.QGPublisher.perform(QGPublisher.java:84)
    at hudson.tasks.BuildStepMonitor$1.perform(BuildStepMonitor.java:20)
    at hudson.model.AbstractBuild$AbstractBuildExecution.perform(AbstractBuild.java:735)
    at hudson.model.AbstractBuild$AbstractBuildExecution.performAllBuildSteps(AbstractBuild.java:676)
    at hudson.model.Build$BuildExecution.post2(Build.java:186)
    at hudson.model.AbstractBuild$AbstractBuildExecution.post(AbstractBuild.java:621)
    at hudson.model.Run.execute(Run.java:1760)
    at hudson.model.FreeStyleBuild.run(FreeStyleBuild.java:43)
    at hudson.model.ResourceController.execute(ResourceController.java:97)
    at hudson.model.Executor.run(Executor.java:405)
Build step 'Quality Gates Sonarqube Plugin' marked build as failure
Finished: FAILURE
```

10. Go to the SonarQube dashboard and verify the reason for **Failure** and **Quality Profile** too:



This is how static code analysis can be configured using Jenkins and SonarQube integration.

Email notifications on build status

What if we want to send a notification of the build status from Jenkins? We can configure mail notifications in such scenarios.

If a Gmail account is configured with two-factor authentication, then the following process can help to setup notification systems:

1. Go to **Less secure apps** in the Google account.
2. Click on **Learn More** in the second paragraph where two-factor authentication is mentioned:

The screenshot shows a web browser window with the URL <https://myaccount.google.com/lesssecureapps?pli=1>. The page is titled "Less secure apps". It contains a message about less secure sign-in technology and a note that this setting is unavailable for accounts with 2-Step Verification enabled.

Some apps and devices use less secure sign-in technology, which makes your account more vulnerable. You can turn off access for these apps, which we recommend, or turn on access if you want to use them despite the risks. [Learn more](#)

This setting is not available for accounts with 2-Step Verification enabled. Such accounts require an application-specific password for less secure apps access. [Learn more](#)

3. Click on the App Password page:

The screenshot shows the "Sign in using App Passwords" section of the Google Account Help page. It provides instructions for generating an App password and links to other signing-in options like 2-Step Verification and Backup codes.

Sign in using App Passwords

An App password is a 16-digit password that gives an app or device permission to access your Google Account. If you use 2-Step Verification and are seeing a "password incorrect" error when trying to access your Google Account, an App password may solve the problem. Most of the time, you'll only have to enter an App password since per app or device, so don't worry about memorizing it.

Note: If you have iOS 8.3 or greater on your iPhone or OS X 10.10.3 on your Mac, you will no longer have to use App passwords to use 2-Step Verification when using the Gmail or any Google branded Apps from iTunes. Using the Google option on the native iOS mail client also does not require App passwords.

Why you may need an App password

- Visit your app password [\[1\]](#) page. You may be asked to sign in to your Google Account.
- At the bottom, click **Select app** and choose the app you're using.
- Click **Select device** and choose the device you're using.
- Select **Generate**.
- Follow the instructions to enter the App password (the 16 character code in the yellow bar) on your device.
- Select **Done**.

Once you are finished, you won't see that App password code again. However, you will see a list of apps and

Sigining in with 2-Step Verification

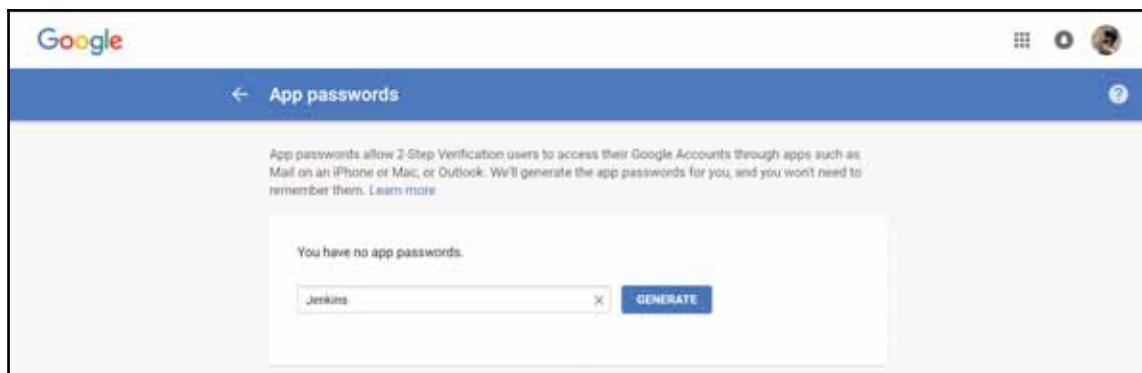
If you sign in with 2-Step Verification:

- Sign in faster with 2-Step Verification phone prompts
- Sign in using App Passwords**

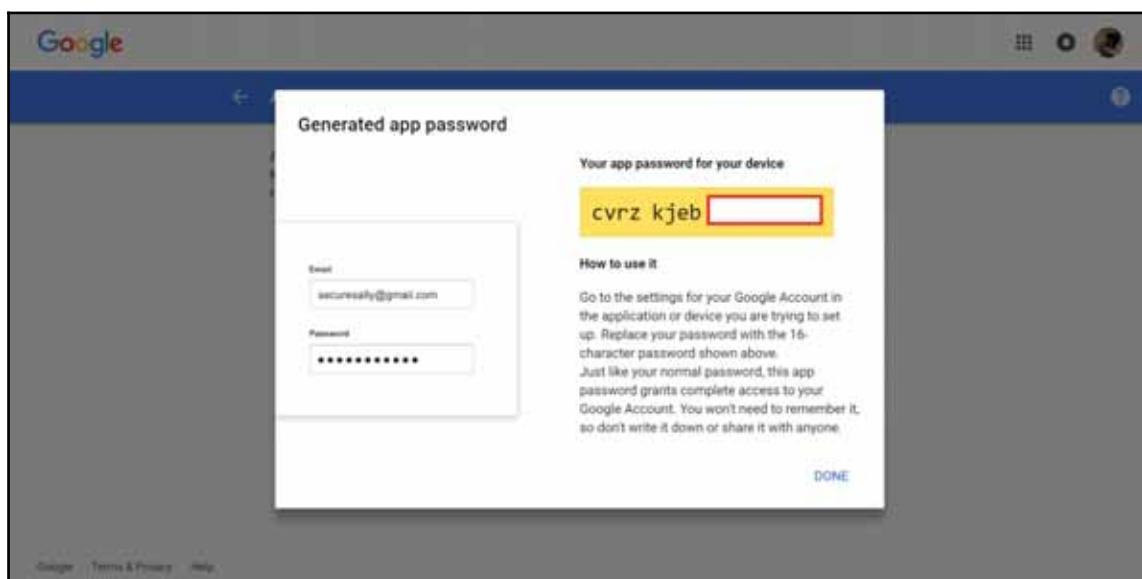
Sign in using Backup code

Sign in with your backup phone

4. Provide a name and click on **Generate**:



5. Copy the password:



6. Go to **Manage Jenkins** and click on **Configure system**.

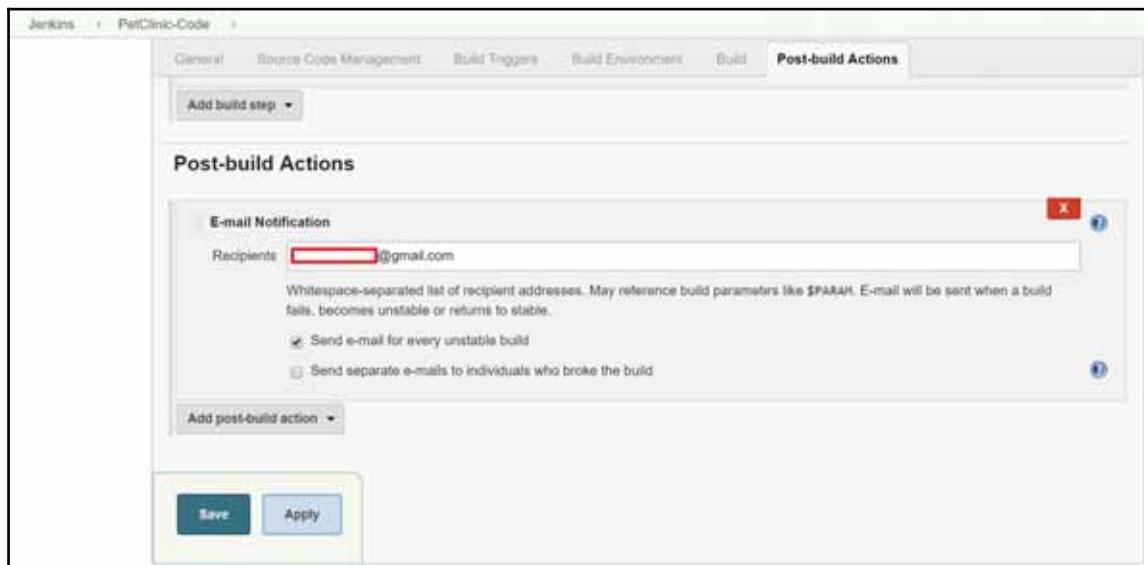
7. Go to the **Email notification** section.

8. Provide all required details and in the password field, copy the recently generated password for the Jenkins app.
9. Click on **Test configuration**:

E-mail Notification

SMTP server	smtp.gmail.com	(?)
Default user e-mail suffix		(?)
<input checked="" type="checkbox"/> Use SMTP Authentication		(?)
User Name	[REDACTED]@gmail.com	
Password	*****	
Use SSL	<input checked="" type="checkbox"/>	(?)
SMTP Port	465	(?)
Reply-To Address	[REDACTED]@gmail.com	
Charset	UTF-8	
<input checked="" type="checkbox"/> Test configuration by sending test e-mail		
Test e-mail recipient	[REDACTED]@gmail.com	
Email was successfully sent		Test configuration
Save	Apply	

10. Go to the **PetClinic-Code** job and select **E-mail** notification from the **Post build action**:



11. Execute the Jenkins job and if it fails, go to the console output and verify the output:

```
ERROR:  
ERROR: Re-run SonarQube Scanner using the -X switch to enable full debug logging.  
ERROR: SonarQube scanner exited with non-zero code: 1  
Sending e-mails to: [REDACTED]@gmail.com  
Finished: FAILURE
```

12. Go to your inbox and verify the mail sent by Jenkins:

The screenshot shows an email from Jenkins titled "Build failed in Jenkins: PetClinic-Code #20". The recipient is "address not configured yet - [REDACTED]@gmail.com" with "to me". The email was sent at "10:14 PM (1 minute ago)". The message body contains the Jenkins log output:

```
Started by user admin
Building in workspace <http://localhost:8080/job/PetClinic-Code/ws/>
FSSCM checkout F:\Jenkins\JenkinsEssentials\FirstDraft\PetClinic to <http://localhost:8080/job/PetClinic-Code/ws/>
FSSCM check completed in 600 seconds
[PetClinic-Code] $ F:\Jenkins\JenkinsEssentials\FirstDraft\jenkinsHome\tools\hudson.plugins.sonar.SonarRunnerInstallation\SonarQube_Scanner_3.0.3\bin\sonar-scanner.bat -e -
-Dsonar.host.url=http://localhost:9000/ -Dproject.settings=http://localhost:8080/job/PetClinic-Code/ws/sonar-project.properties -Dsonar.projectBaseDir=http://localhost:8080/job/PetClinic-Code/ws/
INFO: Option -e/-errors is no longer supported and will be ignored
INFO: Scanner configuration file: F:\Jenkins\JenkinsEssentials\FirstDraft\jenkinsHome\tools\hudson.plugins.sonar.SonarRunnerInstallation\SonarQube_Scanner_3.0.3\bin\sonar-scanner.properties
INFO: Project root configuration file: <http://localhost:8080/job/PetClinic-Code/ws/sonar-project.properties>
INFO: SonarQube Scanner 3.0.3.78
INFO: Java 1.8.0_111 Oracle Corporation (64-bit)
INFO: Windows 10 10.0 amd64
INFO: User cache: C:\Users\Itesh\sonar\cache
ERROR: SonarQube server [http://localhost:9000] can not be reached
INFO:
INFO: EXECUTION FAILURE
INFO:
INFO: Total time: 3.498s
INFO: Final Memory: 3M/61M
INFO:
ERROR: Error during SonarQube Scanner execution
org.sonarsource.scanner.api.internal.ScannerException: Unable to execute SonarQube
    at org.sonarsource.scanner.api.internal.IsolatedLauncherFactory$1.run(IsolatedLauncherFactory.java:84)
    at org.sonarsource.scanner.api.internal.IsolatedLauncherFactory$1.run(IsolatedLauncherFactory.java:71)
    at java.security.AccessController.doPrivileged(Native Method)
    at org.sonarsource.scanner.api.internal.IsolatedLauncherFactory.createLauncher(IsolatedLauncherFactory.java:71)
    at org.sonarsource.scanner.api.internal.IsolatedLauncherFactory.createLauncher(IsolatedLauncherFactory.java:67)
    at org.sonarsource.scanner.api.EmbeddedScanner.doStart(EmbeddedScanner.java:218)
    at org.sonarsource.scanner.api.EmbeddedScanner.start(EmbeddedScanner.java:156)
    at org.sonarsource.scannercli.Main.execute(Main.java:74)
    at org.sonarsource.scannercli.Main.main(Main.java:61)
Caused by: java.lang.IllegalStateException: Fail to get bootstrap index from server
```

Done!

Summary

Here we are again at the section of a chapter that gives us a sense of achievement about knowing something more. In this chapter, we have covered how to configure SonarQube for static code analysis using Jenkins. We have also created a custom profile in SonarQube.

We used the Quality Gate plugin to reflect the state of quality gate of SonarQube in Jenkins.

In the last section, we configured email notifications for an unstable build where Gmail accounts use two-factor authentication.

In the next chapter, we will cover how to perform Continuous Integration using Jenkins.

4

Continuous Integration with Jenkins

Continuous Integration is one of the most important DevOps practices and serves as a base to implement DevOps culture in any organization. It is all about committing code into shared repositories such as Git or SVN multiple times, based on feature completion or bug fixes, and then verifying it with static code analysis using SonarQube, automated builds (using Ant, Maven, or Gradle), executing unit test cases, and creating a package.

There are many tools that can be utilized for Continuous Integration. Jenkins is one of the most popular open source tools that can be utilized for multiple programming languages in which applications can be built. VSTS and Atlassian Bamboo are some other tools or services that can be utilized for Continuous Integration.

This chapter describes in detail how to create and configure build jobs for Java, and how to run build jobs and unit test cases using Ant and Maven build tools. It covers all aspects of running a build to create a distribution file or war file for deployment. We will focus on the following topics in this chapter :

- The Dashboard View plugin
- Creating and configuring a build job for a Java application with Ant
- Creating and configuring a build job for a Java application with Maven
- Build execution with test cases

In this chapter, we will cover the main parts of Continuous Integration practices as a part of our DevOps journey.



At the end of this chapter, we will know how to configure Ant, and Java-based projects in Jenkins so we can compile source files, execute unit test cases, and create a package file.

Dashboard View plugin

Before creating and configuring build jobs for Java applications, we will install the Dashboard View plugin for better management of builds and to display results of builds and tests.

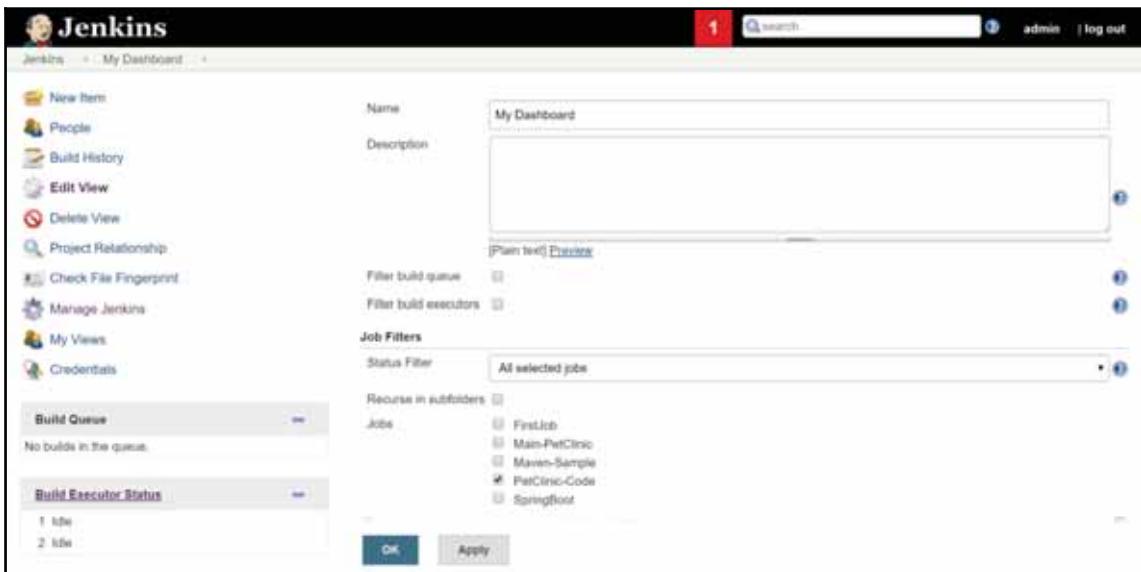
This plugin provides a portal-like view for Jenkins build jobs. Download it from <https://wiki.jenkins-ci.org/display/JENKINS/Dashboard+View>. It will be beneficial in showing results and trends. In addition, it also allows users to arrange display items in an effective manner. On the Jenkins dashboard, go to the **Manage Jenkins** link, click on **Manage Plugins**, and install the **Dashboard View** plugin. Verify the successful installation by clicking on the **Installed** tab.

Now go to the Jenkins dashboard and click on the plus sign available on the tab.

Provide a **View name**, select **Dashboard**, and click on **OK**:



Once the **Dashboard** view is created, we can configure it by selecting **Jobs** and customizing it as shown in the following screenshot:



This is what our dashboard looks like. We can configure multiple projects and multiple portlets can be set in different sections available on dashboard:

The screenshot shows the Jenkins 'My Dashboard' page. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Edit View', 'Delete View', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', and 'Credentials'. The main area has a title 'My Dashboard' with a dropdown menu. Below it is a table with one row for the job 'PetClinicCode'. The table columns are 'S' (Status), 'W' (Workflow), 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The job status is red, indicating failure. The 'Last Success' is '11 hr - 818', 'Last Failure' is '19 min - 820', and 'Last Duration' is '1 min 34 sec'. At the bottom of the table, there are links for 'Logfile', 'RSS for all', 'RSS for failures', and 'RSS for last failed builds'. There are also two small green icons on the right.

Often the **Dashboard View** plugin is used to organize and arrange jobs with specific details, which is essential to know for an administrator or users in the context of automation results.

Configuring different portlets related to test results in the **Dashboard View**.

In Chapter 2, *Installation and Configuration of Code Repository and Build Tools*, we installed and configured Java and Ant. We will now integrate a sample Ant project with Jenkins and understand how things work.

Creating and configuring a build job for a Java application with Ant

We always say that tools are not important, but it is always a good idea to have some understanding of tools so we can perform operations and troubleshoot in an easy manner.

Ant uses the `build.xml` file to execute different tasks that lead to the creation of a package file.

We will use a sample project for Ant that is available at <https://github.com/mitesh51/AntExample>.

Its build.xml file contains the following details:

```
<?xml version="1.0" ?>
<project name="AntExample1" default="war">

<path id="compile.classpath">
<fileset dir="WebContent/WEB-INF/lib">
<include name="*.jar"/>
</fileset>
</path>

<target name="init">
<mkdir dir="build/classes"/>
<mkdir dir="dist" />
</target>

<target name="compile" depends="init">
<javac destdir="build/classes" debug="true" srcdir="src">
<classpath refid="compile.classpath"/>
</javac>
</target>

<target name="war" depends="compile">
<war destfile="dist/AntExample.war"
 webxml="WebContent/WEB-INF/web.xml">
<fileset dir="WebContent"/>
<lib dir="WebContent/WEB-INF/lib"/>
<classes dir="build/classes"/>
</war>
</target>

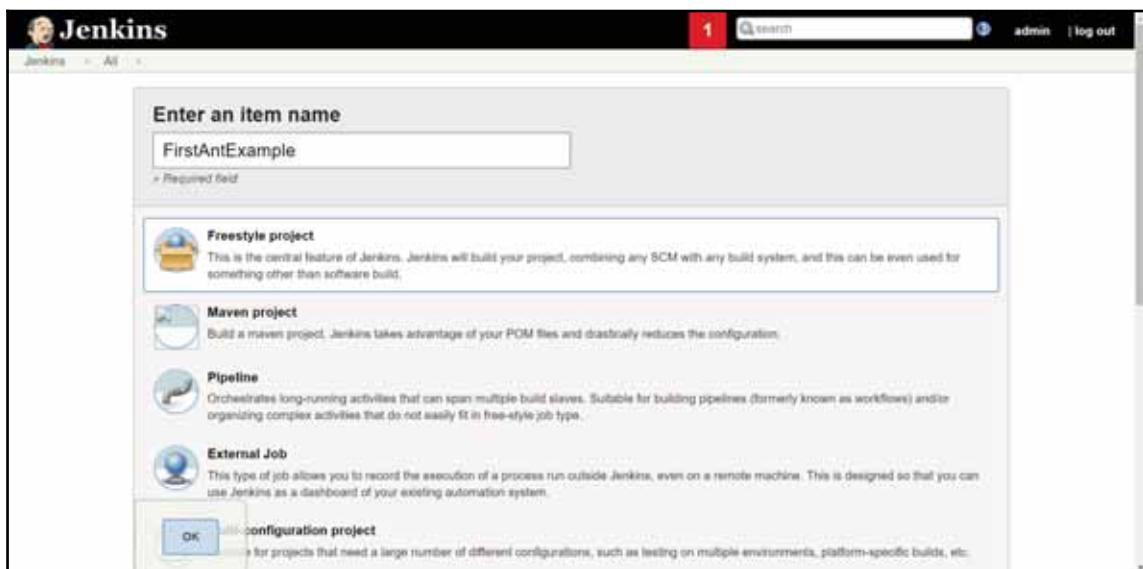
<target name="clean">
<delete dir="dist" />
<delete dir="build" />
</target>

</project>
```

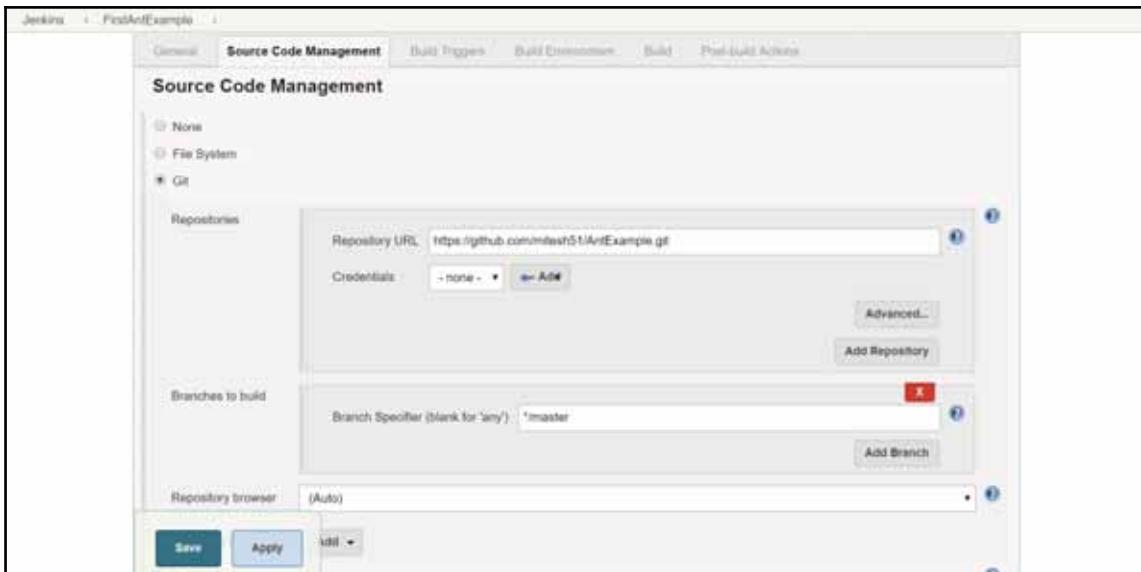
We will use a war target from this build file to create a package that we can deploy in a Local or Remote Tomcat server.

We will now create a **Freestyle project** in Jenkins for our first Ant project integration in Jenkins:

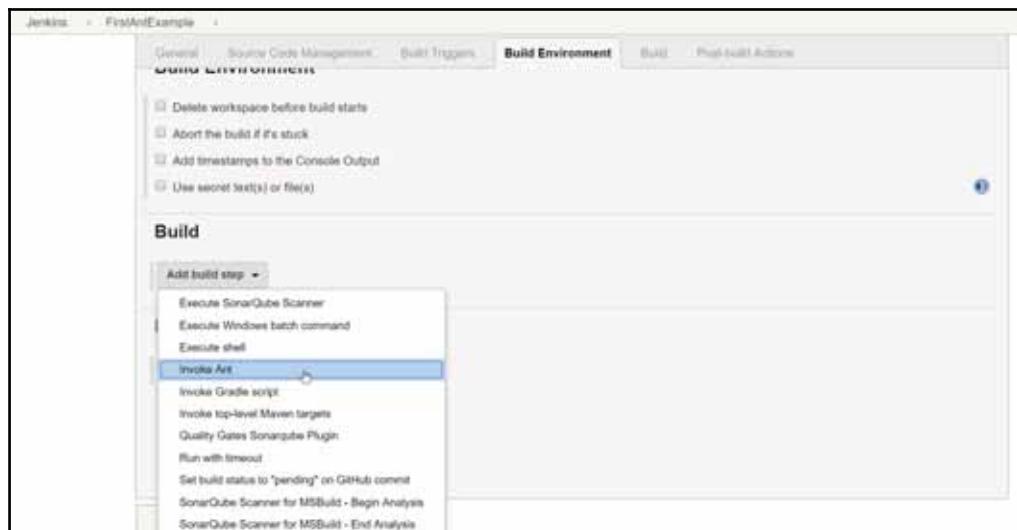
1. Go to the **Jenkins** dashboard and click on **New item**.
2. Enter an item name and select **Freestyle** project.
3. Click **OK**.



4. Once a project is created in Jenkins, it will open it in edit mode. Here we can configure all things related to automation that we want to perform in different sections.
5. Go to the **Source Code Management** section and select **Git**.
6. Provide a **Repository URL**. In our case, it is available on GitHub.

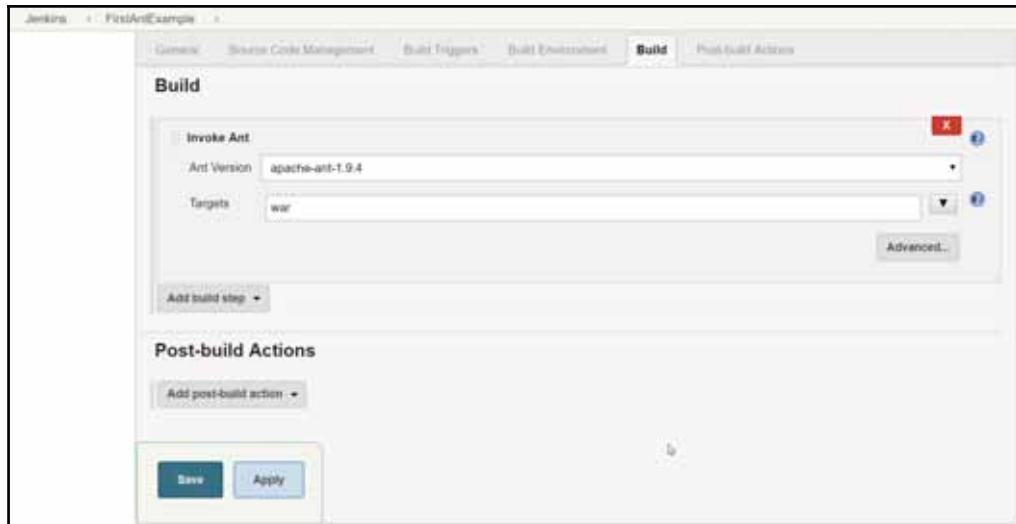


7. Click on the **Build Environment** section.
8. Click on **Add build step** and select **Invoke Ant**.



9. We have configured **Apache Ant** in Global Tool Configuration.
10. Select **Ant configured** in Jenkins from the list box.

11. Select the target we want to execute from the build.xml file and click on **Save**.



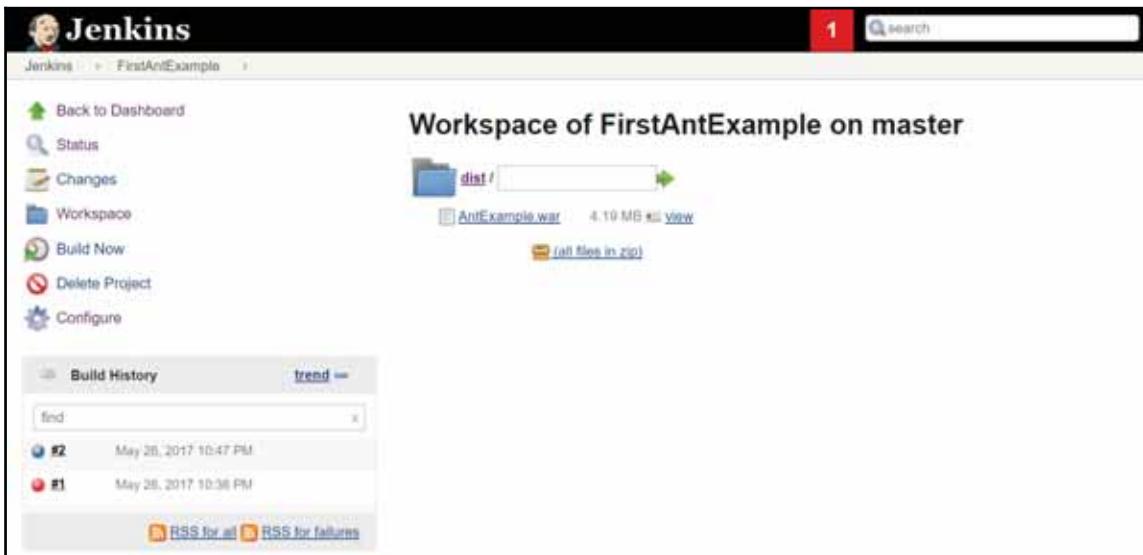
12. Click on **Build now**.
13. Go to the Console output of a currently executed build.

```
> git.exe config remote.origin.url https://github.com/mitesh51/AntExample.git # timeout=10
Fetching upstream changes from https://github.com/mitesh51/AntExample.git
> git.exe --version # timeout=10
> git.exe fetch --tags --progress https://github.com/mitesh51/AntExample.git +refs/heads/*:refs/remotes/origin/*
> git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
> git.exe rev-parse "refs/remotes/origin/origin/master^{commit}" # timeout=10
Checking out Revision b88fd734base7a009d404e4781718716297ec84 (refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f b88fd734base7a009d404e4781718716297ec84
> git.exe rev-list b88fd734base7a009d404e4781718716297ec84 # timeout=10
[FirstAntExample] 3 cmd.exe /C "C:\apache-ant-1.9.4\bin\ant.bat war && exit %ERRORLEVEL%"
Buildfile: F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstAntExample\build.xml
init:
    [mkdir] Created dir: F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstAntExample\build\classes
    [mkdir] Created dir: F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstAntExample\dist

compile:
    [javac] F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstAntExample\build.xml:17: warning:
      'includeantruntime' was not set, defaulting to build.sysclasspathlast; set to false for repeatable builds
    [javac] Compiling 4 source files to
    F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstAntExample\build\classes
    [javac] Note:
    F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstAntExample\src\com\vaannila\web\userController.java
    uses or overrides a deprecated API.
        [javac] Note: Recompile with -Xlint:deprecation for details.

WAR:
    [war] Building war:
    F:\JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstAntExample\dist\AntExample.war
BUILD SUCCESSFUL
Total time: 6 seconds
Finished: SUCCESS
```

14. Observe the log. All targets available in `build.xml` will be executed based on dependencies mentioned in it.
15. Verify that the `.war` file has been created.
16. Go to **Workspace** and find the `.war` file in the `dist` directory:



So we have now seen how to configure an Ant-based Java project in Jenkins and create a package file.

Creating and configuring a build job for a Java application with Maven

Apache Maven is a build automation tool specifically used for Java-based projects to automate the creation of an application build by compiling source code, running automated unit tests, and packaging binary code.

It is based on the **Project Object Model (POM)**. We will use a PetClinic Maven-based project available at <https://github.com/mitesh51/spring-petclinic>.

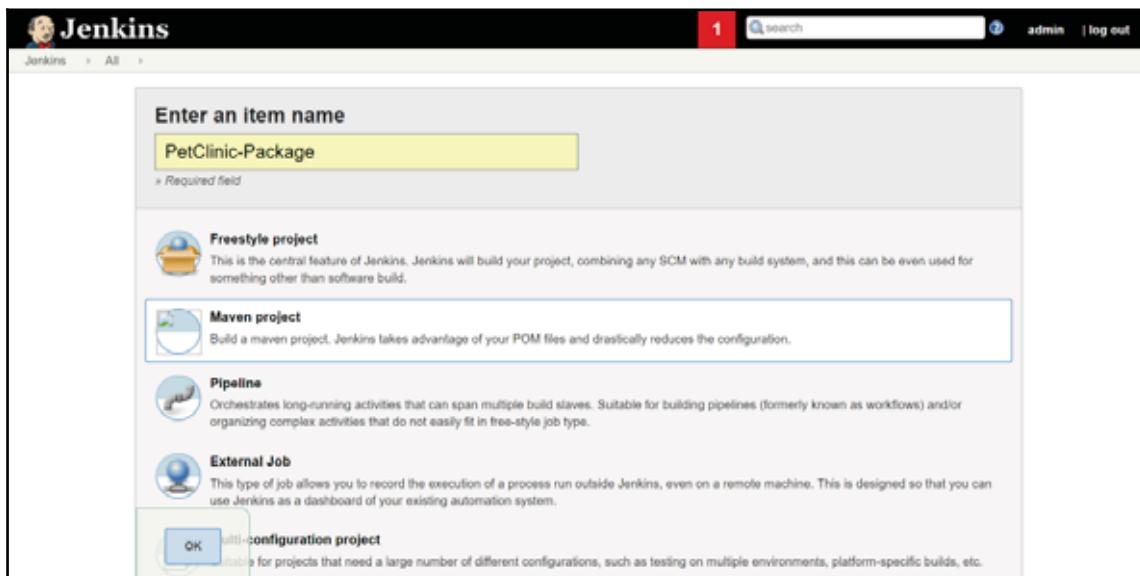
It has a pom.xml that is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://maven.apache.org/POM/4.0.0"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.springframework.samples</groupId>
<artifactId>spring-petclinic</artifactId>
<version>4.2.5-SNAPSHOT</version>
<name>petclinic</name>
<packaging>war</packaging>
<properties>
  <!-- Generic properties -->
  <java.version>1.7</java.version>
  <project.build.sourceEncoding>UTF-
  8</project.build.sourceEncoding><project.reporting.outputEncoding>UTF-
  8</project.reporting.outputEncoding>
  <!-- Spring -->
  <spring-io-platform.version>2.0.3.RELEASE</spring-io-
  platform.version><spring-data-jdbc.version>1.1.0.RELEASE</spring-data-
  jdbc.version>
  <!-- Java EE / Java SE dependencies -->
  <tomcat.version>7.0.59</tomcat.version>
  <!-- Test -->
  <assertj.version>2.2.0</assertj.version>
  <!-- Dates -->
  <jodatime-hibernate.version>1.3</jodatime-
  hibernate.version>
  <jodatime-jsptags.version>1.1.1</jodatime-jsptags.version>
  <jadira-usertype-core.version>3.2.0.GA</jadira-usertype-
  core.version>
  <!-- Others -->
  <mysql-driver.version>5.1.36</mysql-driver.version>
  <!-- Web dependencies -->
  <dandelion.version>1.1.1</dandelion.version>
  <dandelion.datatables.version>1.1.0
    </dandelion.datatables.version>
  <cobertura.version>2.7</cobertura.version>
</properties>
<dependencyManagement>
  <!-- Import the maven Spring IO Platform Bill Of Materials
  (BOM) -->
  <dependencies>
    <dependency>
      <groupId>io.spring.platform</groupId>
      <artifactId>platform-bom</artifactId>
```

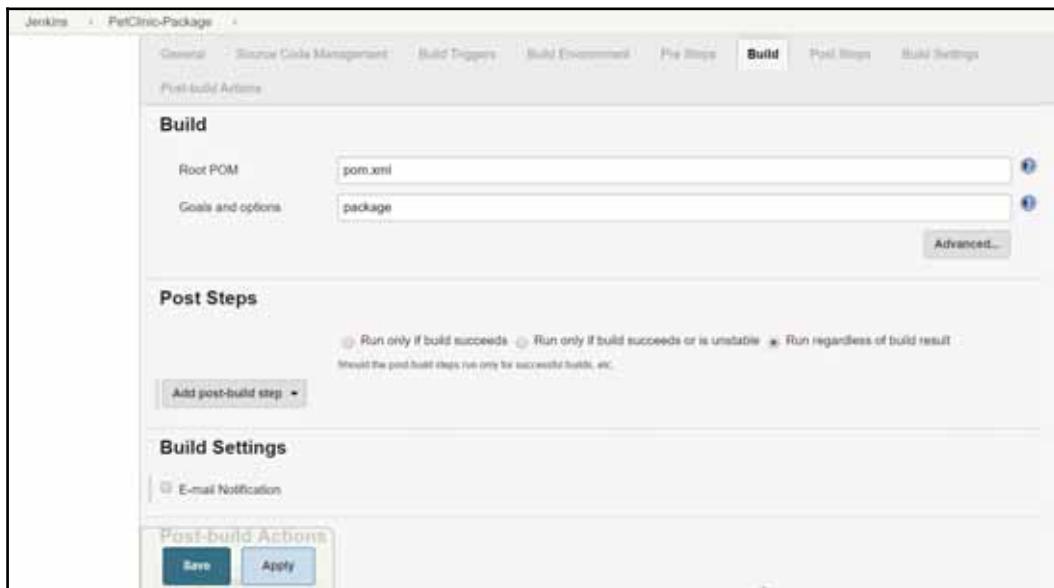
```
<version>${spring-io-platform.version}</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>
<url>demopetclinic</url>
</project>
```

In Chapter 2, *Installation and Configuration of Code Repository and Build Tools*, we have already configured Java and Maven. We need to utilize both in our Maven project in Jenkins:

1. Go to the **Jenkins** dashboard and click on **New item**. Enter an item name and select **Maven project**. Click **OK**:



2. In the **Build** section, give the location of **Root POM**. There are defined goals for Maven-based projects. Give package as a **Goal**:



3. Click on **Save**.
4. If we are behind the proxy then we need to provide proxy details in Maven so it can use that to download dependencies.
5. Go to the `MAVEN_HOME` directory.
6. Go to the `conf` directory and open `settings.xml`.

Uncomment the following block:

```
<proxies>
<proxy>
<id>optional</id>
<active>true</active>
<protocol>http</protocol>
<username>testuser</username>
<password>sdbjsdhbhjw</password>
<host>proxy.***.com</host>
<port>8080</port>
<nonProxyHosts>localhost*</nonProxyHosts>
</proxy>
</proxies>
```

7. Click on **Build Now**.
8. Go to **Console Output** to verify the log. As it is a Maven project, it will download dependencies mentioned in the pom.xml file from the internet.
9. Verify the test results in the log and also verify that the .war file has been created.

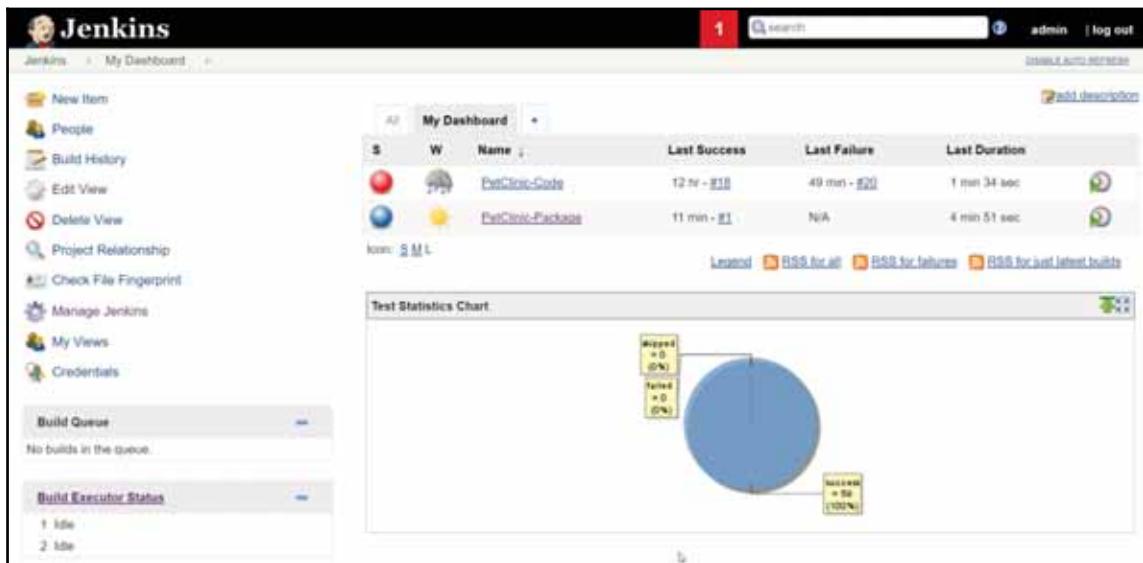


The screenshot shows the Jenkins console output for a build named "PetClinic-Package" (build #1). The output is as follows:

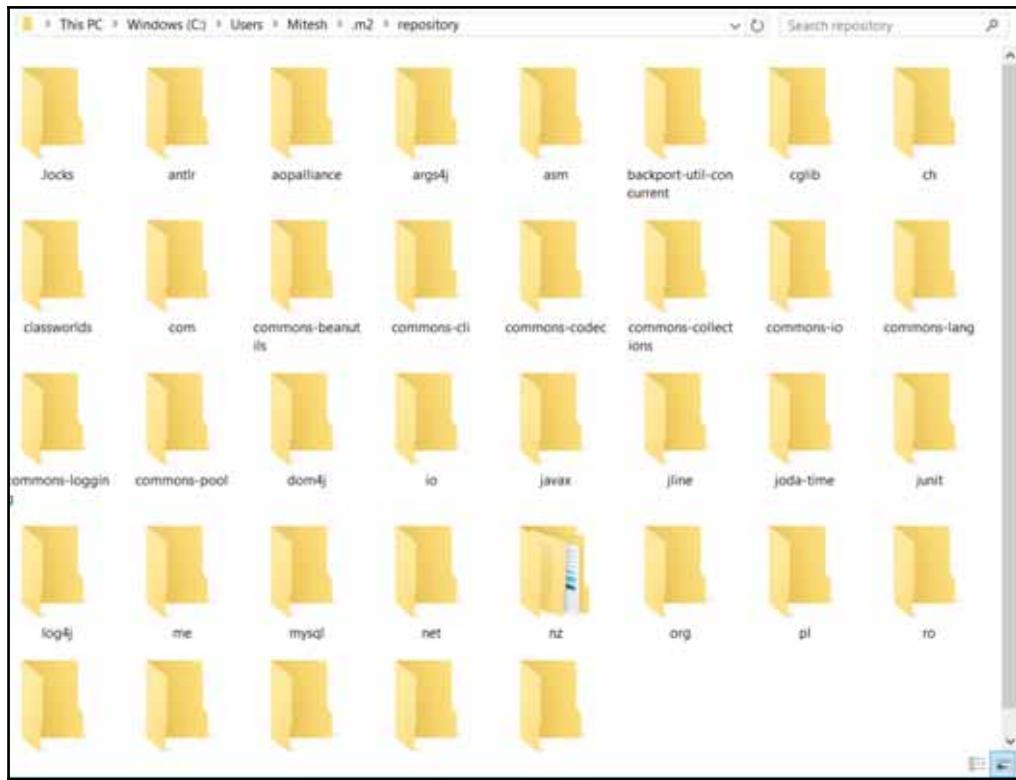
```
Results : Tests run: 59, Failures: 0, Errors: 0, Skipped: 0

[JENKINS] Recording test results
[INFO]
[INFO] --- maven-war-plugin:2.3:war (default-war) @ spring-petclinic ---
[INFO] Packaging webapp
[INFO] Assembling webapp [spring-petclinic] in [F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-Package\target\spring-petclinic-4.2.5-SNAPSHOT]
[INFO] Processing war project
[INFO] Copying webapp resources [F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-Package\src\main\webapp]
[INFO] Webapp assembled in [3618 msec]
[INFO] Building war: F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-Package\target\petclinic.war
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 02:19 min
[INFO] Finished at: 2017-05-26T22:56:23+05:30
[INFO] Final Memory: 30M/102M
[INFO]
[INFO] -----
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-Package\pom.xml to org.springframework.samples/spring-petclinic/4.2.5-SNAPSHOT/spring-petclinic-4.2.5-SNAPSHOT.pom
[JENKINS] Archiving F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-Package\target\petclinic.war to org.springframework.samples/spring-petclinic/4.2.5-SNAPSHOT/spring-petclinic-4.2.5-SNAPSHOT.war
channel stopped
Finished: SUCCESS
```

10. Go to the customized dashboard that we have created and verify whether any results have come in or not:



11. To verify the JAR files that are downloaded from the Maven repository, go to the `Users` directory and select the specific user that we have logged in as. Find the `.m2` directory and all the JAR files will be available there in the case of Windows operating systems:



12. To check the package file created with Jenkins and Maven integration, go to the `JENKINS_HOME/workspace/jobname/target` directory.
13. We have successfully created a `.war` file that can be deployed on the server. It can be a web server or an application server. We will use the Tomcat server for application deployment.

Summary

As we promised in the beginning of the chapter, we have covered how to integrate Ant and Maven based applications in Jenkins in detail. We have also provided details on the **Dashboard View** plugin.

In the next chapter, we will cover deployment of the `.war` file that we have created using Continuous Integration. We will deploy application packages in different public clouds such as AWS and Microsoft Azure in Infrastructure as a Service and Platform as a Service.

5

Continuous Delivery - Implementing Automated Deployment

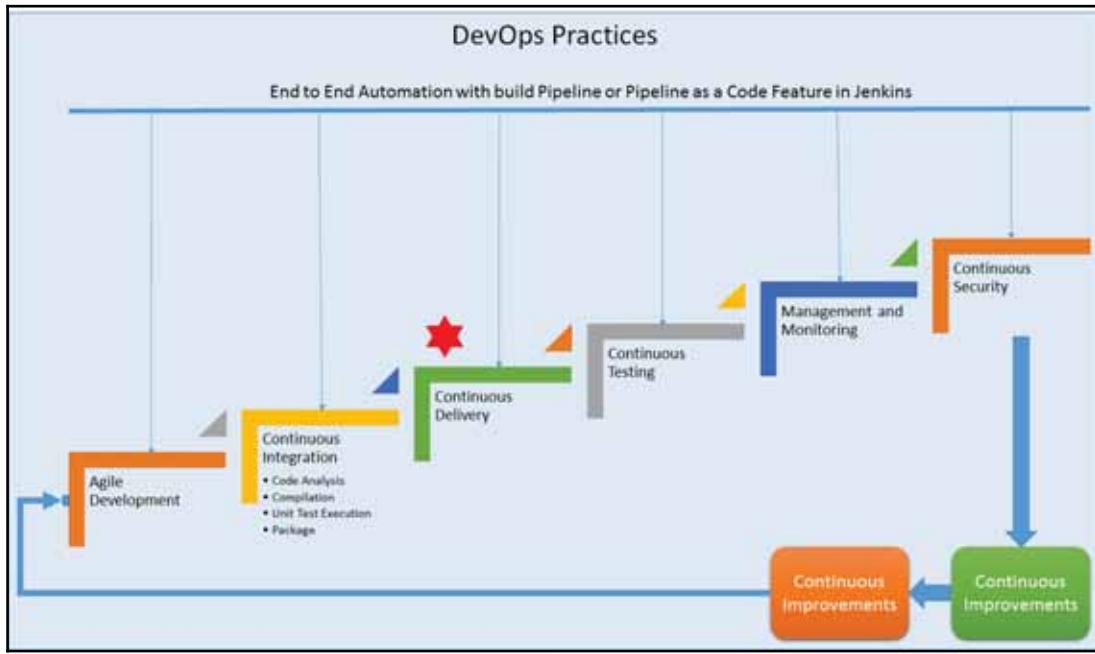
Once we have a package ready for deployment after Continuous Integration, our next step should be deployment of that package in to a web or application server.

We can deploy WAR files manually, or with commands (batch file or shell script), or with Jenkins plugins, or any third-party tool that can be integrated with Jenkins. In our case, we will use Jenkins plugins for application deployment into runtime environments, which can be local or remote.

This chapter will take one step forward in the DevOps pipeline by deploying artifacts in local or remote application servers. It will give insight into automated deployment and continuous delivery processes and it will also cover how to deploy applications on public cloud platforms using Jenkins. In this chapter, we will cover following topics:

- An overview of Continuous Delivery and Continuous Deployment
- Installing Tomcat
- Deploying a war file from Jenkins to Tomcat
- Deploying a war file from Jenkins to AWS Elastic Beanstalk
- Deploying a war file from Jenkins to Microsoft Azure App Services

In this chapter, we will cover the main parts of Continuous Delivery practice as a part of our DevOps journey:



At the end of this chapter, we will know how to deploy an application to web or application servers in AWS and Microsoft Azure.

An overview of Continuous Delivery and Continuous Deployment

Continuous Delivery (CD) is a DevOps practice that is used to deploy an application quickly to a high quality, with an automated approach, in non-production environments. In Continuous Delivery, an application package is always production ready.

Continuous Deployment is a DevOps practice that is used to deploy an application quickly to a high quality, with an automated approach, in a production environment.

Automated approaches to deploying application packages in production and non-production won't change. The approval process may be set up in cases of the deployment of application packages in the production environment though.

In the following sections, we will deploy war files into different environments using different approaches.

Installing Tomcat

Apache Tomcat is an open source server that can be utilized to deploy Java-based web applications. Apache Tomcat implements several Java EE specifications including Java Servlet, JavaServer Pages, Java EL, and WebSocket.

Tomcat installation is very simple. Download it from <http://tomcat.apache.org/>, based on the requirement.

Download the installable files and extract them. Go to the folder and find the bin directory. Based on the operating system, run `startup.bat` or `startup.sh` in the command window or Terminal.

Deploying a war file from Jenkins to Tomcat

For application deployment, we can utilize multiple ways to deploy an application in a web server or application server. We can use batch script or shell script to copy the package file created after a Continuous Integration process, or we can use a Jenkins plugin to deploy an application:

1. Go to **Manage Jenkins | Manage Plugins** and install **Deploy to container Plugin**:

The screenshot shows the Jenkins Manage Plugins interface. At the top, there are tabs for 'Updates' (disabled), 'Available' (selected), 'Installed', and 'Advanced'. A search bar at the top right contains the placeholder 'Deploy to'. Below the tabs, there's a table with two rows. The first row has a checkbox icon, the name 'Deploy to container Plugin', and the version '1.10'. The second row has a checkbox icon, the name 'Deploy to WebSphere container Plugin', and the version '1.0'. At the bottom of the table are three buttons: 'Install without restart' (disabled), 'Download now and install after restart', and 'Check now'. A status message 'Update information obtained: 1 day 2 hr ago' is displayed below the table.

Install ↓	Name	Version
<input type="checkbox"/>	Deploy to container Plugin	1.10
<input type="checkbox"/>	Deploy to WebSphere container Plugin	1.0

Install without restart Download now and install after restart Check now

Update information obtained: 1 day 2 hr ago

2. Wait until the plugin is installed successfully:

Installing Plugins/Upgrades

<p>Preparation</p>	<ul style="list-style-type: none">• Checking internet connectivity• Checking update center connectivity• Success
<p>Dashboard View</p>	Success
<p>Deploy to container Plugin</p>	Success

[Go back to the top page](#)
(you can start using the installed plugins right away)

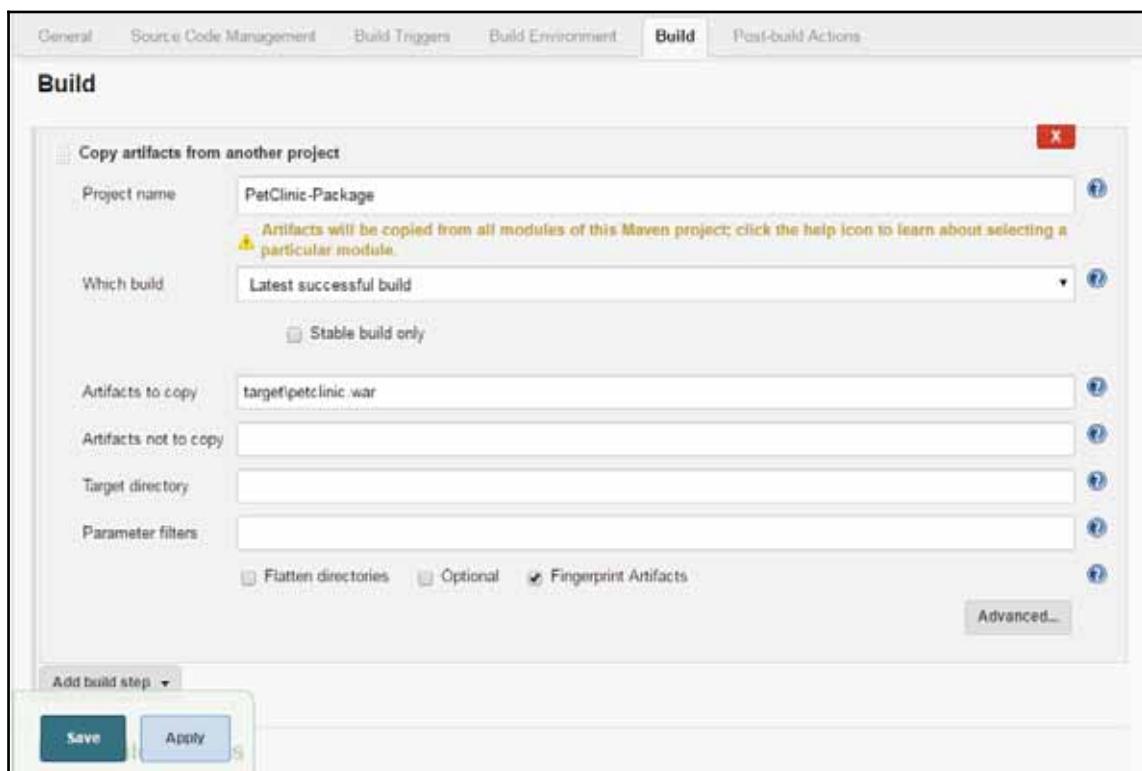
Restart Jenkins when installation is complete and no jobs are running

3. To allow deployment using the Jenkins plugin, go to the Tomcat installation directory and open `conf\tomcat-users.xml`.
4. Create a new role and new user as follows:

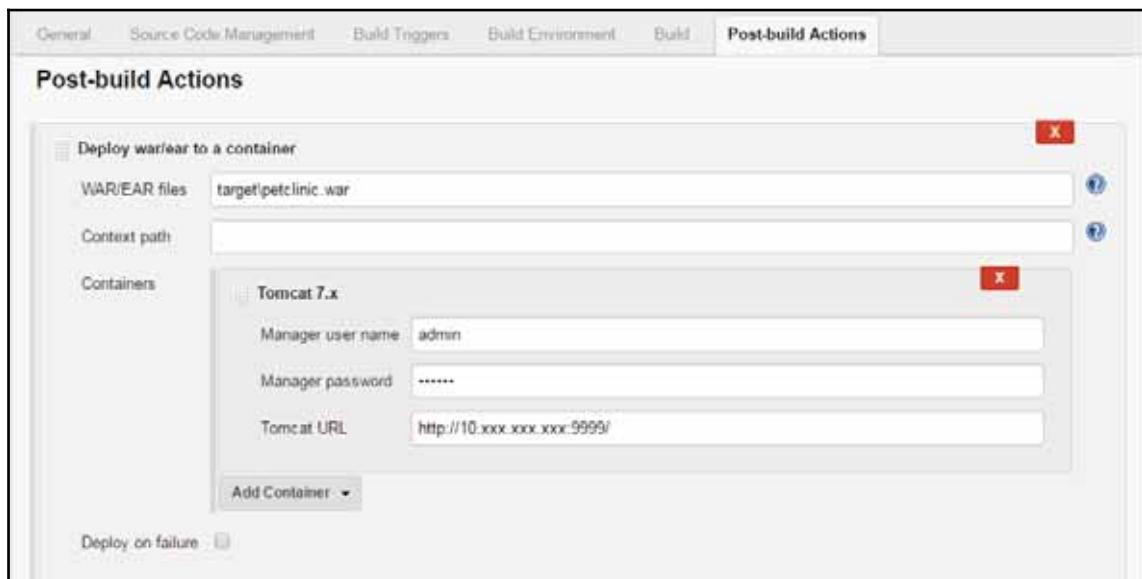
```
<?xml version='1.0' encoding='utf-8'?>
<!--
<tomcat-users>
<!--
<!--
NOTE: The sample user and role entries below are intended for use with the
examples web application. They are wrapped in a comment and thus are ignored
when reading this file. If you wish to configure these users for use with the
examples web application, do not forget to remove the <!... ...> that surrounds
them. You will also need to set the passwords to something appropriate.
-->
<role rolename="manager-script"/>
<user username="admin" password="admin@123" roles="manager-script"/>
</tomcat-users>
```

5. Restart Tomcat.
6. Create a new **Freestyle build** in Jenkins named `PetClinic-Deploy`.

7. What we will do here is copy the artifact created from the PetClinic-Package job and deploy it in Tomcat. Install the Copy Artifact plugin to perform this action. Give the project a name and path from which we need to copy the WAR file:



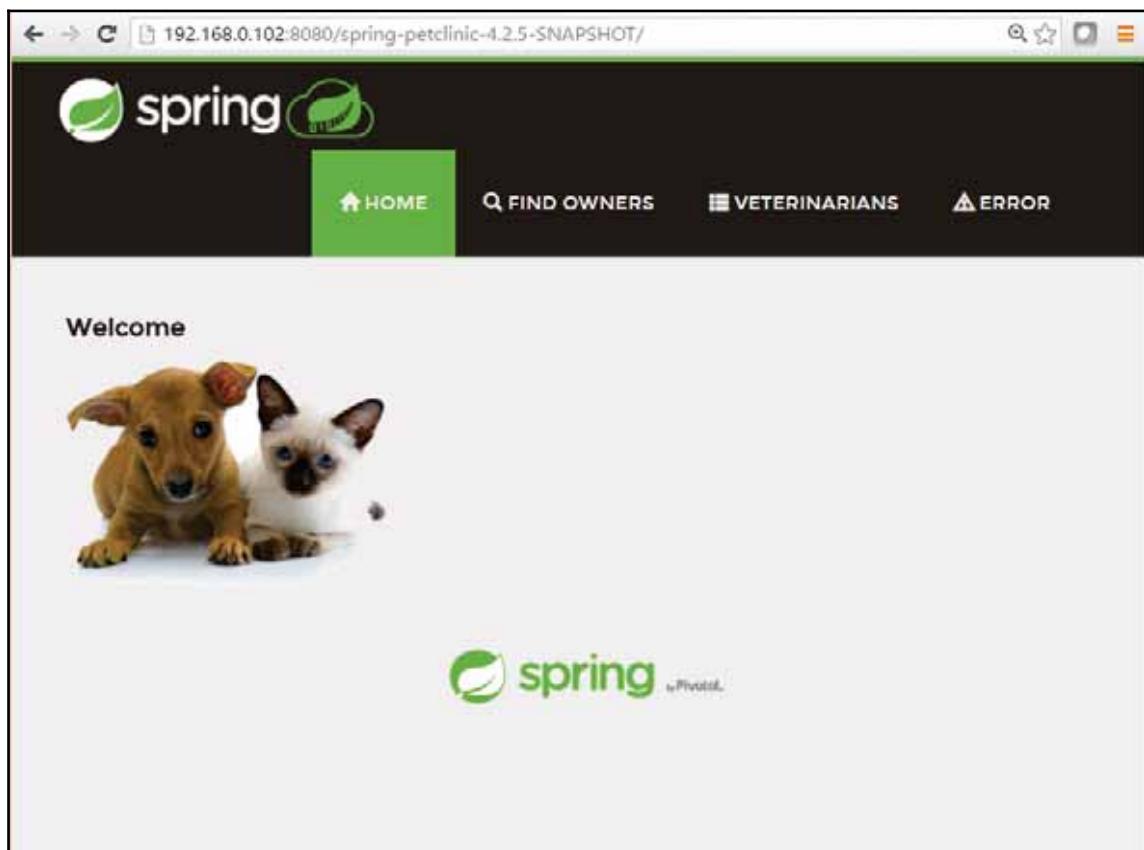
8. Give a path to the WAR file for deployment using the Jenkins plugin. Select **Deploy war/ear to a container** from Post build actions. Click on **Add Container** and select the latest version of Tomcat. Give a Tomcat URL. Give the username and password we defined in `tomcat-users.xml`:



9. Execute the build by clicking on **Build now**. Verify the logs for application deployment:

```
Results :  
  
Tests run: 59, Failures: 0, Errors: 0, Skipped: 0  
  
[INFO]  
[INFO] --- maven-war-plugin:2.3:war (default-war) @ spring-petclinic ---  
[INFO] Packaging webapp  
[INFO] Assembling webapp [spring-petclinic] in [d:\jenkins\workspace\PetClinic-Test\target\spring-petclinic-4.2.5-SNAPSHOT]  
[INFO] Processing war project  
[INFO] Copying webapp resources [d:\jenkins\workspace\PetClinic-Test\src\main\webapp]  
[INFO] Webapp assembled in [1669 msecs]  
[INFO] Building war: d:\jenkins\workspace\PetClinic-Test\target\spring-petclinic-4.2.5-SNAPSHOT.war  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 28.772 s  
[INFO] Finished at: 2016-07-06T22:59:37+05:30  
[INFO] Final Memory: 29M/261M  
[INFO] -----  
Deploying d:\jenkins\workspace\PetClinic-Test\target\spring-petclinic-4.2.5-SNAPSHOT.war to container  
Tomcat 7.x Remote  
[d:\jenkins\workspace\PetClinic-Test\target\spring-petclinic-4.2.5-SNAPSHOT.war] is not deployed.  
Doing a fresh deployment.  
Deploying [d:\jenkins\workspace\PetClinic-Test\target\spring-petclinic-4.2.5-SNAPSHOT.war]  
Finished: SUCCESS
```

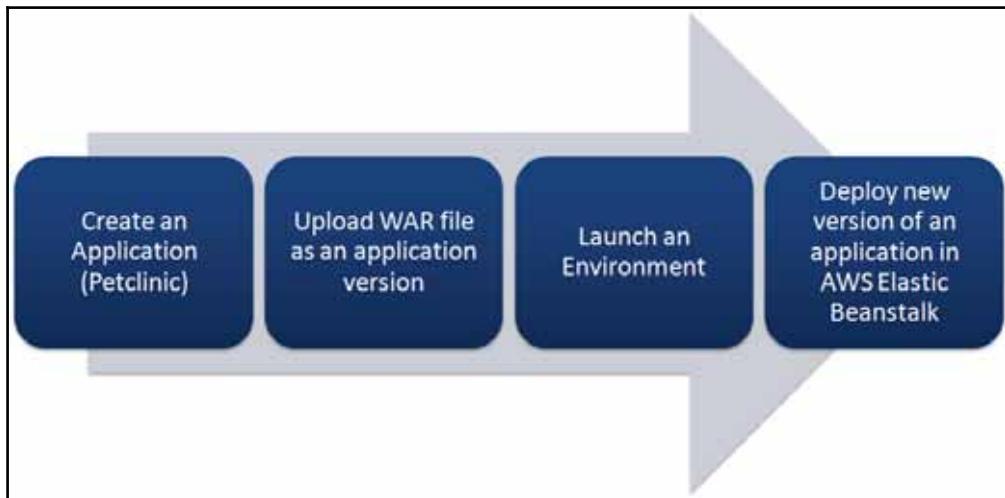
10. Go to a browser and visit the application with the Tomcat URL and the context of an application:



In the next section, we will deploy the PetClinic application in Tomcat that resides in the AWS EC2 instance.

Deploying a WAR file from Jenkins to AWS Elastic Beanstalk

AWS Elastic Beanstalk is a **Platform as a Service (PaaS)**. We will use it to deploy the PetClinic application. These are the steps to deploy an application on AWS Elastic Beanstalk:



Let's create a sample application in Elastic Beanstalk to understand how Elastic Beanstalk works and then we will use the Jenkins plugin to deploy an application into it:

The screenshot shows the AWS VPC Dashboard. At the top, there are navigation links for AWS, Services, and Edit. Below the navigation is a sidebar titled "VPC Dashboard" containing links for Virtual Private Cloud, Your VPCs, Subnets, Route Tables, Internet Gateways, DHCP Options Sets, Elastic IPs, Endpoints, NAT Gateways, Peering Connections, Security, and Network ACLs. A dropdown menu under "Filter by VPC" is set to "None". The main content area is titled "Resources" and includes a "Start VPC Wizard" button and a "Launch EC2 Instances" button. A note states: "Note: Your Instances will launch in the US East (N. Virginia) region." Below this, it says: "You are using the following Amazon VPC resources in the US East (N. Virginia) region:" followed by a table of resource counts. The table data is as follows:

1 VPC	1 Internet Gateway
4 Subnets	1 Route Table
1 Network ACL	0 Elastic IPs
0 VPC Peering Connections	0 Endpoints
0 Nat Gateways	1 Security Group
0 Running Instances	0 VPN Connections
0 Virtual Private Gateways	0 Customer Gateways

VPN Connections

Amazon VPC enables you to use your own isolated resources within the AWS cloud, and then connect those resources directly to your own datacenter using industry-standard encrypted IPsec VPN connections.

1. Click on **Services** in the AWS management console and select **AWS Elastic Beanstalk**. Create a new application named `petclinic`. Select **Tomcat** as the **Platform** and select the **Sample application** radio button:

The screenshot shows the 'Create a web app' wizard in the AWS Elastic Beanstalk console. The application name is set to 'petclinic'. The platform is 'Tomcat'. Under 'App code', the 'Sample application' option is selected, which includes instructions for configuration and mentions the ability to upload new source code later. The 'Upload your own code' option is also available.

Elastic Beanstalk petclinic Create New Application

Create a web app

Choose a name and a platform for your app. You can start with a sample app or upload your own code. Then, you can configure more options before you deploy your app. By creating an app, you allow AWS Elastic Beanstalk to administer AWS resources and necessary permissions on your behalf. [Learn more](#).

Application name petclinic

Maximum length of 100 characters, not including forward slash (/)

Platform Tomcat

Tomcat 8 Java 8 (this can be updated after initial setup)

App code Sample application

Comes with instructions on how to configure your application. You can upload a new source code for this application later.

Upload your own code

You can upload a file or provide a URL to your app code in Amazon S3.

- Verify the sequence of events for the creation of a sample application:

The screenshot shows the 'All Applications' page for the 'petclinic' application. A progress message indicates 'Creating petclinic' and notes it will take a few minutes. A log window displays the creation process, including the creation of EC2 instances, an EIP, a security group, and an S3 bucket. A 'Learn More' sidebar provides links to get started, modify the code, and connect to a database.

Elastic Beanstalk petclinic Create New Application

All Applications > petclinic > petclinic (Environment ID: e-y2mww0n, URL:) Actions ▾

Creating petclinic
This will take a few minutes

11:04pm Waiting for EC2 instances to launch. This may take a few minutes.
11:02pm Created EIP: 52.73.142.147
11:02pm Environment health has transitioned to Pending. Initialization in progress (running for 8 seconds). There are no instances.
11:02pm Created security group named awseb-e-y2mww0n-stack-AWSEBSecurityGroup-E1E762FFSL6Q
11:01pm Using elasticbeanstalk-us-east-1-68523027657 as Amazon S3 storage bucket for environment data
11:01pm createEnvironment is starting.

Learn More

[Get started using Elastic Beanstalk](#)
[Modify the code](#)
[Create and connect to a database](#)
[Add a custom domain](#)

Command Line Interface (v3)

[Installing the AWS EB CLI](#)
[EB CLI Command Reference](#)

3. It will take some time, and once the environment has been created, it will be highlighted in green:

The screenshot shows the AWS Elastic Beanstalk console. In the top navigation bar, 'AWS' is selected under 'Services'. The application dropdown shows 'petclinic'. On the right, there are user details 'Mitesh', location 'N. Virginia', and a 'Support' link. A blue button 'Create New Application' is visible. Below the navigation, the breadcrumb path 'All Applications > petclinic' is shown. To the right of the breadcrumb is an 'Actions' dropdown. On the left, a sidebar menu lists 'Environments', 'Application Versions', and 'Saved Configurations'. The 'Environments' section contains a single entry for 'petclinic', which is highlighted with a green background. A tooltip for this entry provides detailed information: 'Environment tier: Web Server', 'Running versions: Sample Application', 'Last modified: 2016-07-07 23:06:19 UTC+0530', and 'URL: petclinic.mjczu0cvp.us-east-1.elasticbeanstalk.com'.

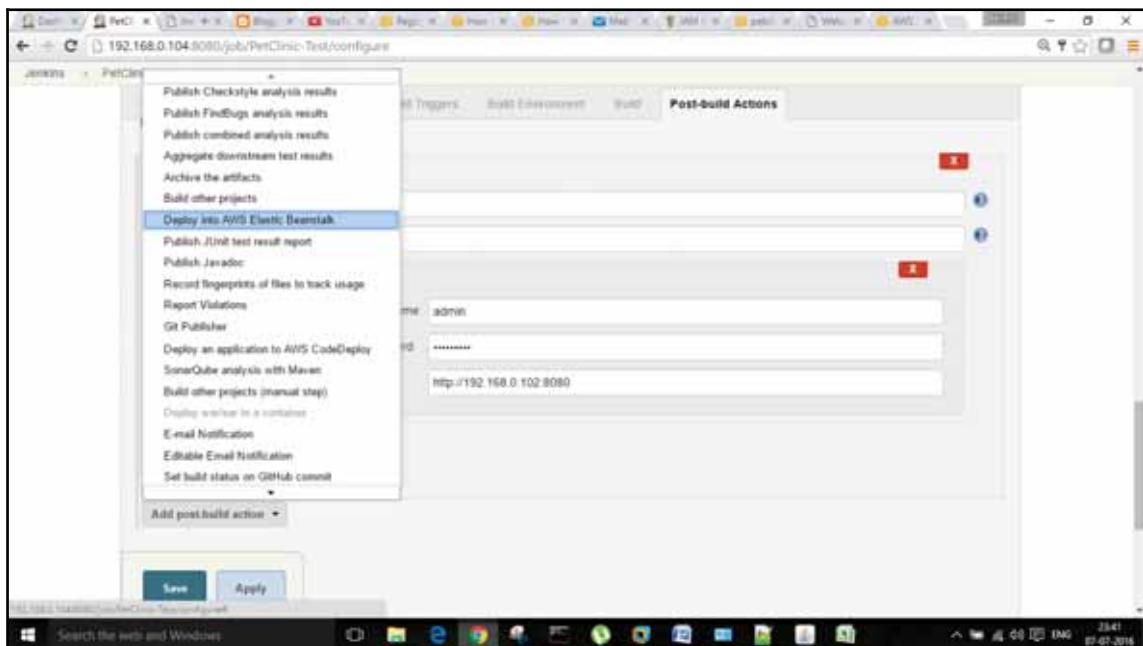
4. Click on the **petclinic** environment and verify that you can see **Health** and **Running Version** in the dashboard:

The screenshot shows the AWS Elastic Beanstalk environment dashboard for the 'petclinic' application. The top navigation bar and application selection are identical to the previous screenshot. The breadcrumb path now includes 'petclinic > petclinic'. The main dashboard features a sidebar with links like 'Dashboard', 'Configuration', 'Logs', 'Health', 'Monitoring', 'Alarms', 'Managed Updates', and 'Events'. The central area has tabs for 'Overview' (selected) and 'Actions'. Under 'Overview', there is a large green circle with a white checkmark icon labeled 'Health OK'. Below it is a 'Causes' button. To the right, there is a 'Running Version' section showing 'Sample Application' and a 'Upload and Deploy' button. Further right is a 'Configuration' section featuring a yellow cat icon, the text '64bit Amazon Linux 2016.03 v2.1.3 running Tomcat 8 Java 8', and a 'Change' button. At the bottom right of the dashboard is a 'Refresh' button.

5. Verify that you can see the environment ID and URL. Click on the URL and verify that you can see the default page:

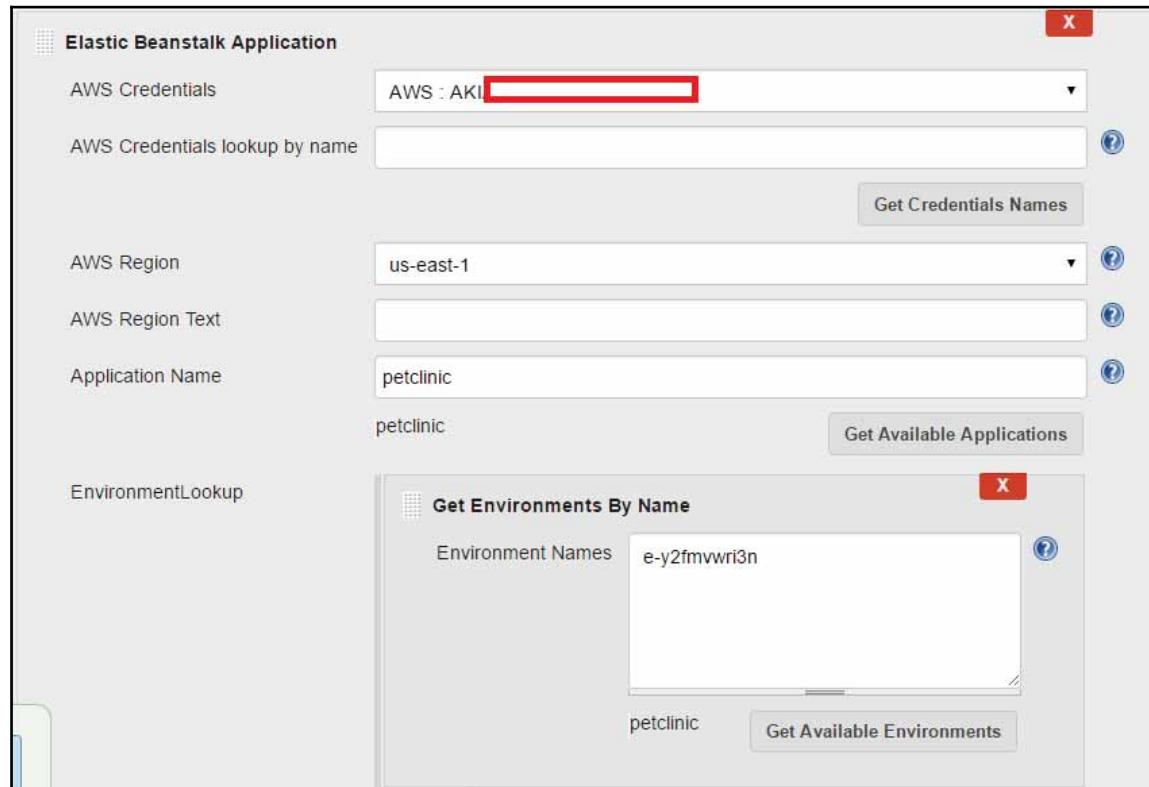


6. Install the AWS Elastic Beanstalk Publisher plugin. For more details, visit <https://wiki.jenkins-ci.org/display/JENKINS/AWS+Beanstalk+Publisher+Plugin>:



7. A new section will come up in **Post-build Actions** for Elastic Beanstalk.
8. Click on the Jenkins dashboard and select **Credentials**; add your AWS credentials.
9. Go to your Jenkins build and select AWS Credential, which is set in the global configuration.
10. Select **AWS Region** from the list and click on **Get Available Applications**. As we have created a sample application, it will show up like this.

11. In **Environment Lookup**, provide an environment ID in the **Get Environments By Name** box and click on **Get Available Environments**:



12. Save the configuration and click on **Build now**.

Now let's verify the AWS management console to check whether the WAR file is being copied in Amazon S3 or not:

1. Go to S3 Services and check the available buckets:

The screenshot shows the AWS Management Console interface for the S3 service. At the top, there are navigation tabs for 'AWS' and 'Services'. A blue 'Create Bucket' button is prominently displayed. To its right is an 'Actions' dropdown menu. Below this, a section titled 'All Buckets (1)' lists a single bucket. The table has two columns: 'Name' and 'Actions'. The bucket name is 'elasticbeanstalk-us-east-1-685239287657', and there is a small icon next to it.

2. Since the WAR file is large, it will take a while to upload to Amazon S3. Once it is uploaded, it will be available in the Amazon S3 bucket.
3. Verify the build job's execution status in Jenkins. Some sections of the expected output are that:
 - The test case execution and WAR file creation are successful
 - The build is successful
4. Now check the AWS management console:

The screenshot shows the AWS Management Console interface for the S3 service, specifically viewing the contents of a bucket. The top navigation bar includes 'AWS', 'Services', and 'Edit' dropdowns, along with user information 'Mitesh', 'Global', and 'Support'. Below the navigation is a toolbar with 'Upload', 'Create Folder', 'Actions', and search/filter options. The main area displays a table of 'All Buckets / elasticbeanstalk-us-east-1-685239287657' contents. The table has columns for 'Name', 'Storage Class', 'Size', and 'Last Modified'. Three objects are listed:

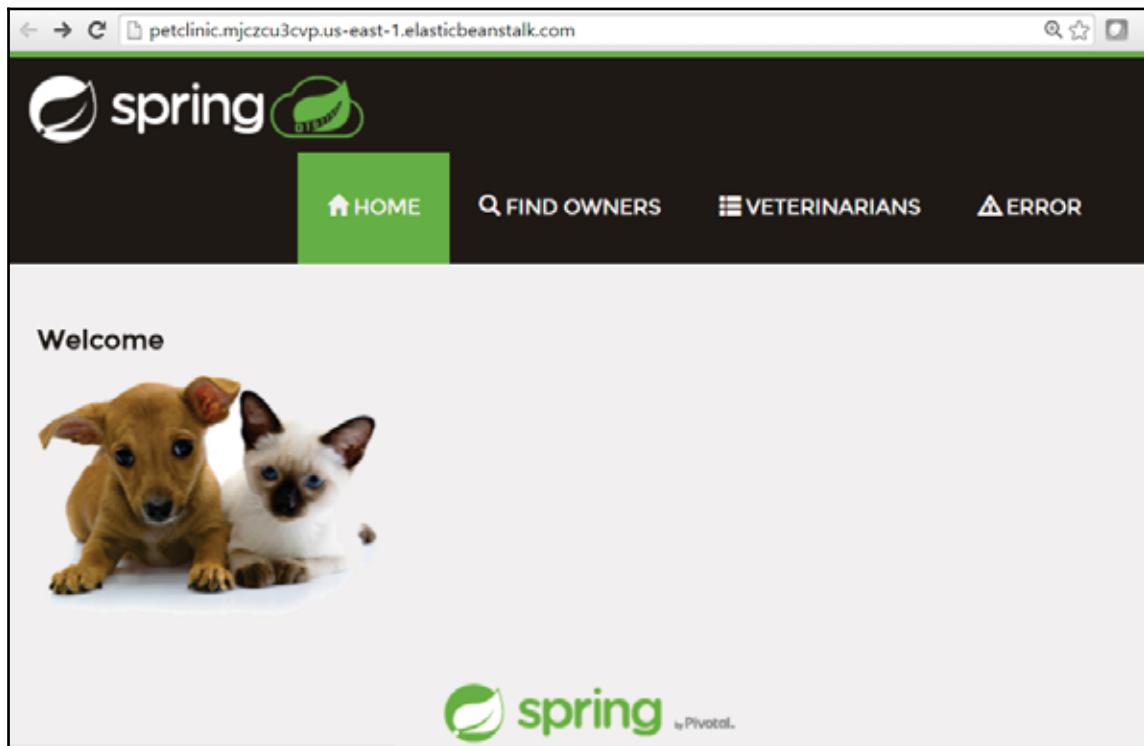
- '.'.elasticbeanstalk': Standard storage class, 0 bytes, last modified Thu Jul 07 23:01:35 GMT+5.
- 'petclinic-jenkins-PetClinic-Test-39.zip': Standard storage class, 39.9 MB, last modified Fri Jul 06 00:52:04 GMT+5.
- 'resources': Standard storage class, size is listed as '-'.

5. Go to **Services**, click on **AWS Elastic Beanstalk**, and verify the environment. The previous version was **Sample Application**. Now, the version is updated as given in **Version Label Format** in the Jenkins build job configuration:

The screenshot shows the AWS Elastic Beanstalk console. At the top, there are navigation links for AWS, Services, Edit, and user information (Mitesh, N. Virginia). Below this, the Elastic Beanstalk service icon and the application name 'petclinic' are displayed, along with a 'Create New Application' button. To the left, a sidebar titled 'Learn More' contains links for getting started with Elastic Beanstalk, modifying code, connecting to a database, adding a custom domain, and using the Command Line Interface (v3). It also includes links for installing the AWS EB CLI and viewing the EB CLI Command Reference. The main area is titled 'All Applications' and shows a single entry for 'petclinic'. A green box highlights this entry, containing detailed information: Environment tier: Web Server; Running versions: Jenkins-PetClinic-Test-39; Last modified: 2016-07-08 01:04:41 UTC+0530; URL: petclinic.mjczcu3c1v.us-east-1.elasticbeanstalk... There is also an 'Actions' dropdown menu next to the application name.

6. Go to the dashboard and verify **Health** and **Running Version** again.

7. Once everything has been verified, click on the URL for the environment, and our PetClinic application should now be live:

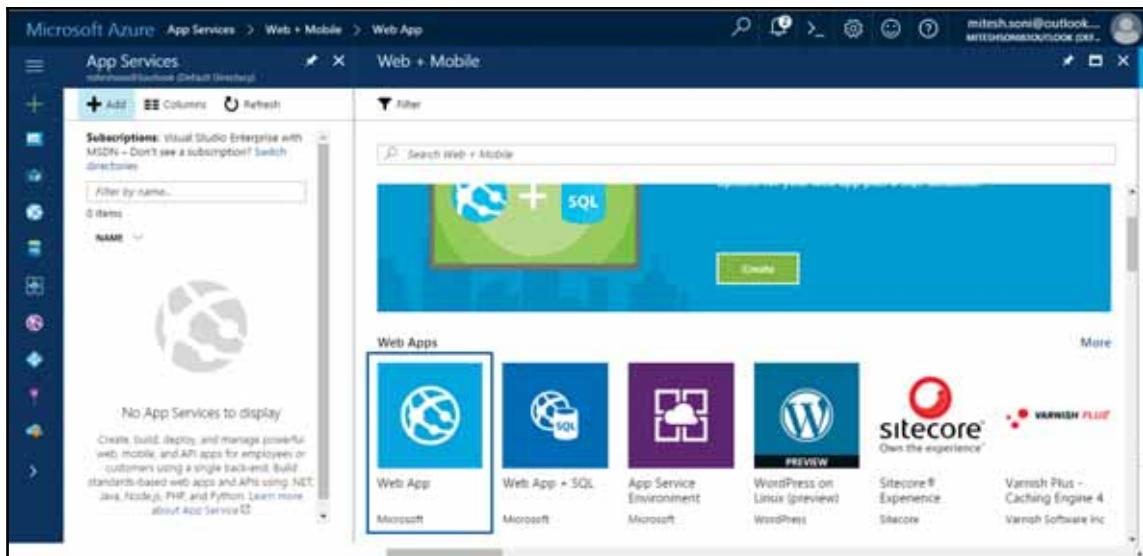


Once the application deployment is successful, terminate the environment. We have thus successfully deployed our application on Elastic Beanstalk.

Deploying a war file from Jenkins to Microsoft Azure App Services

Microsoft Azure App Services is a PaaS. In this section, we will look at the Azure Web App and how we can deploy our PetClinic application:

1. We need to have a Microsoft Azure subscription. Go to App Services and click on **Add**:



2. Click on **Create**:

Microsoft Azure App Services > Web + Mobile > Web App

Web App

Create and deploy web sites in seconds, as powerful as you need them

Leverage your existing tools to create and deploy applications without the hassle of managing infrastructure. Microsoft Azure Web Sites offers secure and flexible development, deployment, and scaling options for any sized web application. Use frameworks and templates to create web sites in seconds. Choose from source control options like TFS, GitHub, and BitBucket. Use any tool or OS to develop your site with .NET, PHP, Node.js or Python.

- Fastest way to build for the cloud
- Provision and deploy fast
- Secure platform that scales automatically
- Great experience for Visual Studio developers
- Open and flexible for everyone
- Monitor, alert, and auto scale (preview)

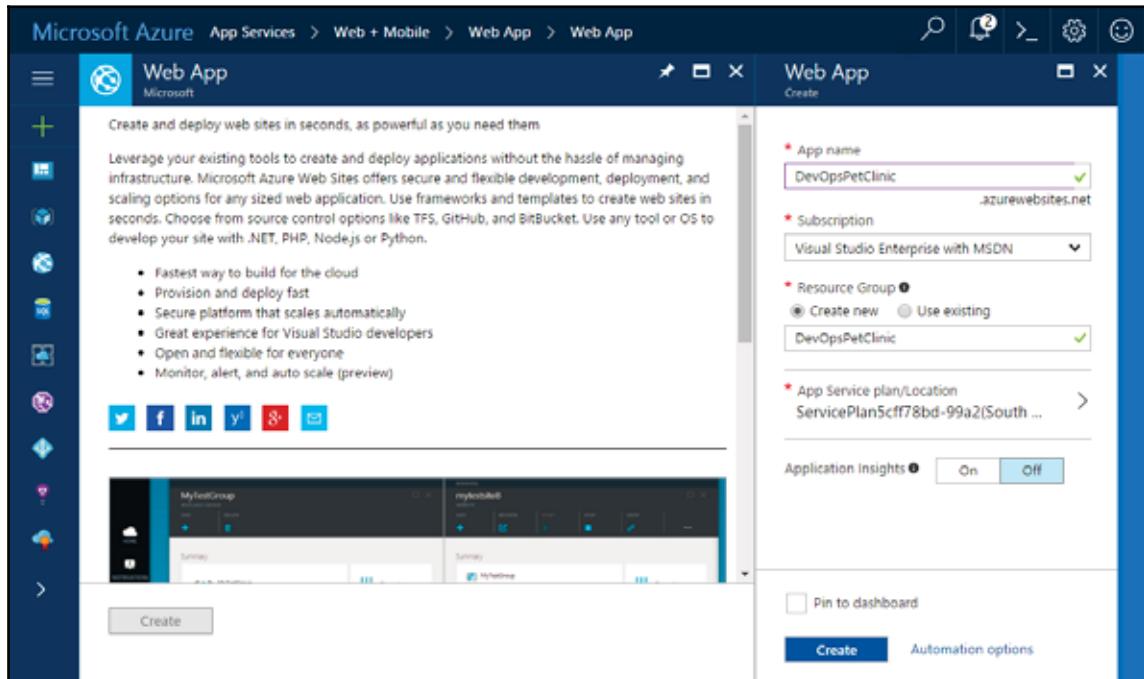
[Twitter](#) [Facebook](#) [LinkedIn](#) [YouTube](#) [Google+](#) [Email](#)

MyTestGroup
Resource Group

mystestsuite8
Website

Create

3. Go to the Microsoft Azure portal at <https://portal.azure.com>. Click on **App Services** and then on **Add**. Provide values for **App name**, **Subscription**, **Resource Group**, and **App Service Plan/Location**. Click on **Create**:



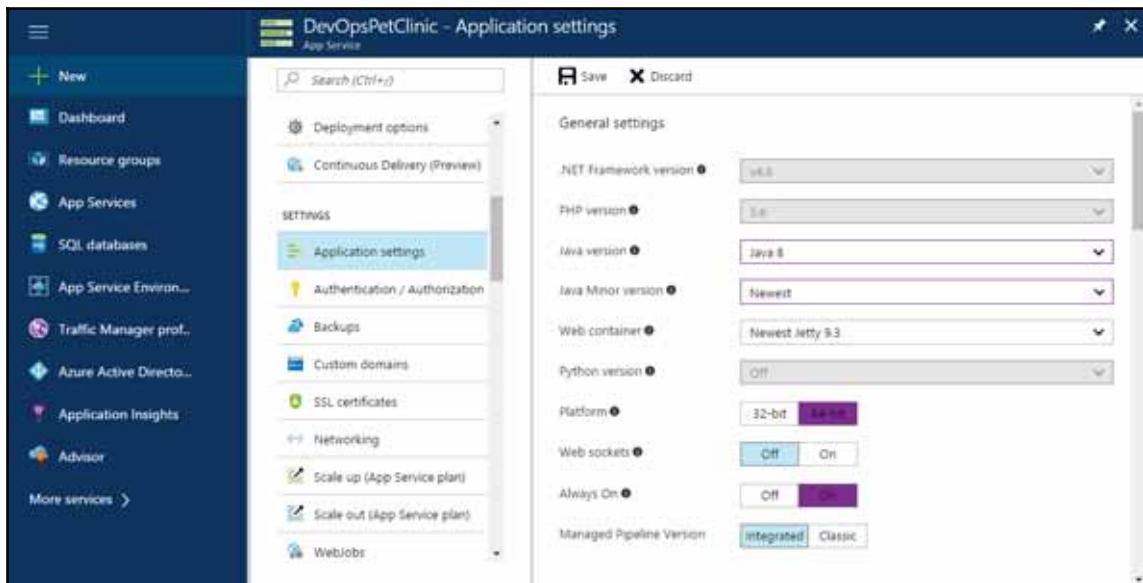
- Once the Azure Web App is created, see whether it shows up in the Azure portal.
Click on **DevOpsPetClinic** for details related to the **URL**, **Status**, **Location**, and so on:

The screenshot shows the Azure portal interface for the 'DevOpsPetClinic' app service. On the left, there's a sidebar with various service icons like New, Dashboard, Resource groups, etc. The main area has a title bar 'DevOpsPetClinic' and 'App Service'. Below the title bar, there's a search bar and a navigation menu with 'Overview' selected. The 'Overview' section contains tabs for Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Under the 'Essentials' heading, detailed information is provided: Resource group (DevOpsPetClinic), Status (Running), Location (South Central US), Subscription name (Visual Studio Enterprise with MSDN), Subscription ID (b88f447-ad0e-44d4-a662-2eb5c950f091), URL (http://devopspetclinic.azurewebsites.net), App Service plan/pricing tier (ServicePlan5cf78bd-99a2 (Standard: 1 Sm...)), FTP/Deployment username (DevOpsPetClinic|m12539666), FTP hostname (ftp://waws-prod-snt-123.ftp.azurewebsite...), and FTPS hostname (https://waws-prod-snt-123.ftp.azurewebsite...). The 'Monitoring' section shows Requests and errors with values 100, 80, and 60.

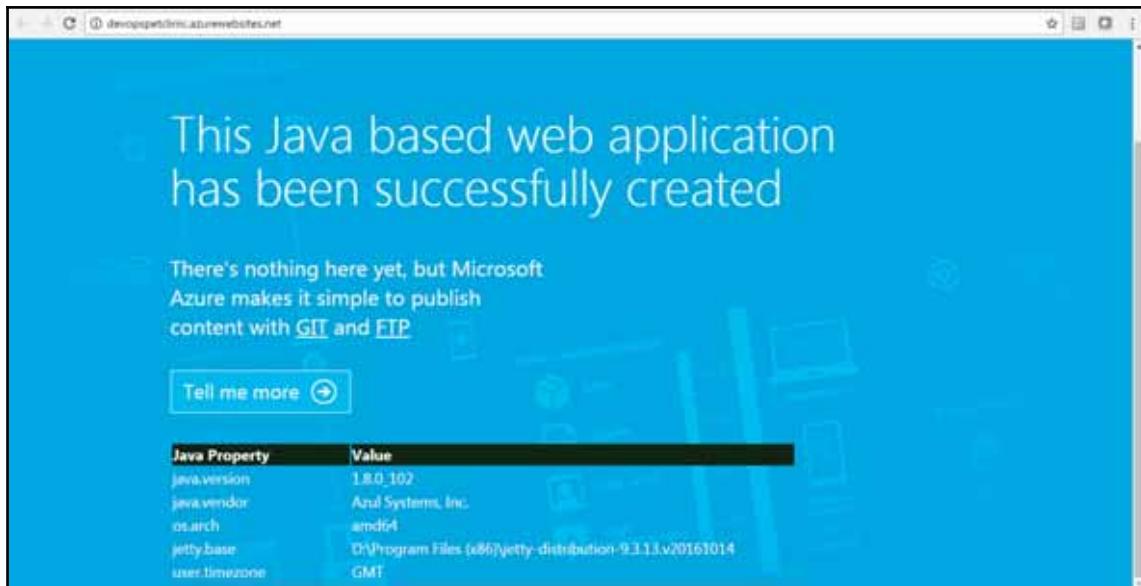
- Verify you can see this in the **App Services** section too:

The screenshot shows the 'App Services' blade in the Microsoft Azure portal. The sidebar on the left includes 'New', 'Dashboard', 'Resource groups', 'App Services' (which is selected), 'SQL databases', 'App Service Environ...', 'Traffic Manager prof...', 'Azure Active Directo...', 'Application Insights', and 'Advisor'. The main area is titled 'App Services' and shows a table of applications. The table has columns: NAME, STATUS, LOCATION, APP TYPE, and APP SERVICE PLAN. One item is listed: 'DevOpsPetClinic' with STATUS 'Running', LOCATION 'South Central US', APP TYPE 'Web app', and APP SERVICE PLAN 'ServicePlan5cf78bd-99a2 (Standard: 1 Sm...)'. There are also buttons for '+ Add', 'Columns', and 'Refresh'.

6. Click on **All Settings**, go to the **GENERAL** section, and click on **Application settings** to configure the Azure Web App for Java web application hosting. Select the **Java version**, **Java Minor version**, **Web container**, and **Platform**, and then click on **Always On**:

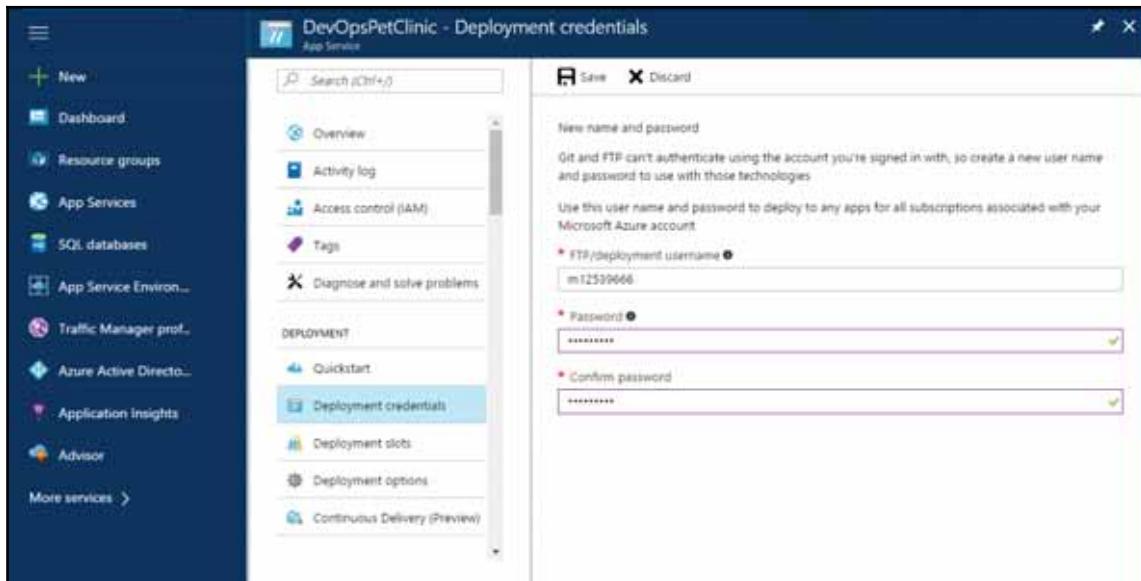


7. Visit the URL of an Azure Web App from your browser and verify that it is ready to host our sample Spring application, PetClinic:



Java Property	Value
java.version	1.8.0_102
java.vendor	Azul Systems, Inc.
osarch	amd64
jetty.base	D:\Program Files (x86)\jetty-distribution-9.3.13.v20161014
user.timezone	GMT

8. Click on **All Settings** and go to **Deployment credentials** in the **PUBLISHING** section. Provide a username and password, and save your changes:



Let's install the Publish Over FTP plugin in Jenkins. We will use the Azure Web App's FTP details to publish the PetClinic WAR file. Let's go to the Jenkins dashboard:

1. Click on **New Item** and select **Freestyle project**.
2. In Jenkins, go to **Manage Jenkins** and click on **Configure | Configure FTP settings**. Provide a **Hostname**, **Username**, and **Password**, which are available in the Azure portal.
3. Go to www.devopspetclinic.scm.azurewebsites.net and download the Kudu console. Navigate to the different options and find the site directory and `webapps` directory.
4. Click on **Test Configuration** and, once you get a **Success** message, you are ready to deploy the PetClinic application:

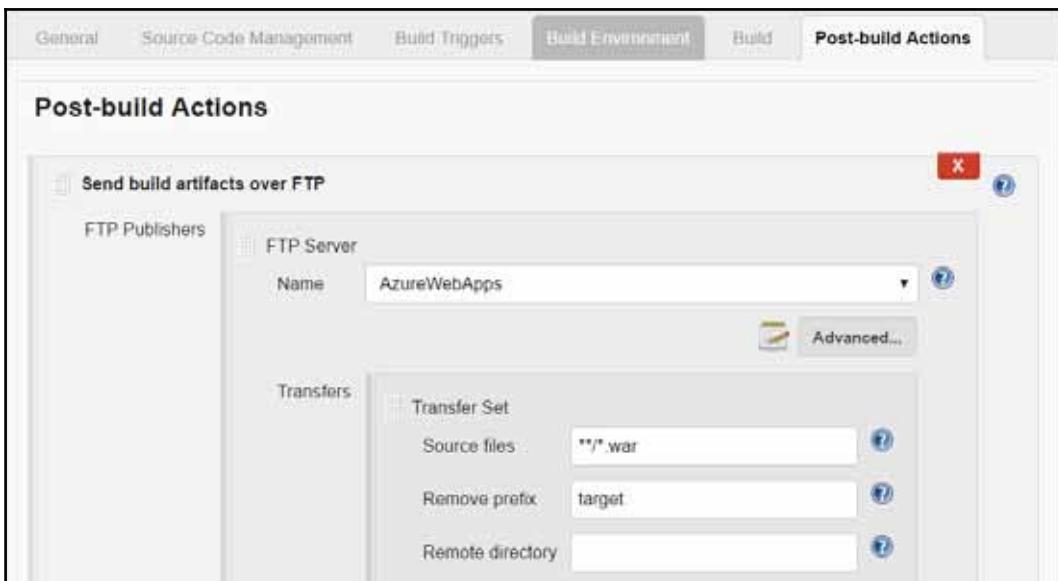
The screenshot shows the Jenkins 'Publish over FTP' configuration page. It displays a table with five rows for an 'FTP Server' named 'AzureWebApps'. The columns are 'Name', 'Hostname', 'Username', 'Password', and 'Remote Directory'. The 'Name' field contains 'AzureWebApps'. The 'Hostname' field contains 'waws-prod-sn1-039.ftp.azurewebsites.net'. The 'Username' field contains 'DevOpsPetClinic\m12539666'. The 'Password' field is obscured by dots. The 'Remote Directory' field contains '\site\wwwroot\webapps'. Below the table are buttons for 'Advanced...', 'Success' (highlighted in blue), 'Test Configuration' (highlighted in blue), 'Delete' (in red), and 'Add'.

FTP Servers	
	FTP Server
Name	AzureWebApps
Hostname	waws-prod-sn1-039.ftp.azurewebsites.net
Username	DevOpsPetClinic\m12539666
Password
Remote Directory	\site\wwwroot\webapps

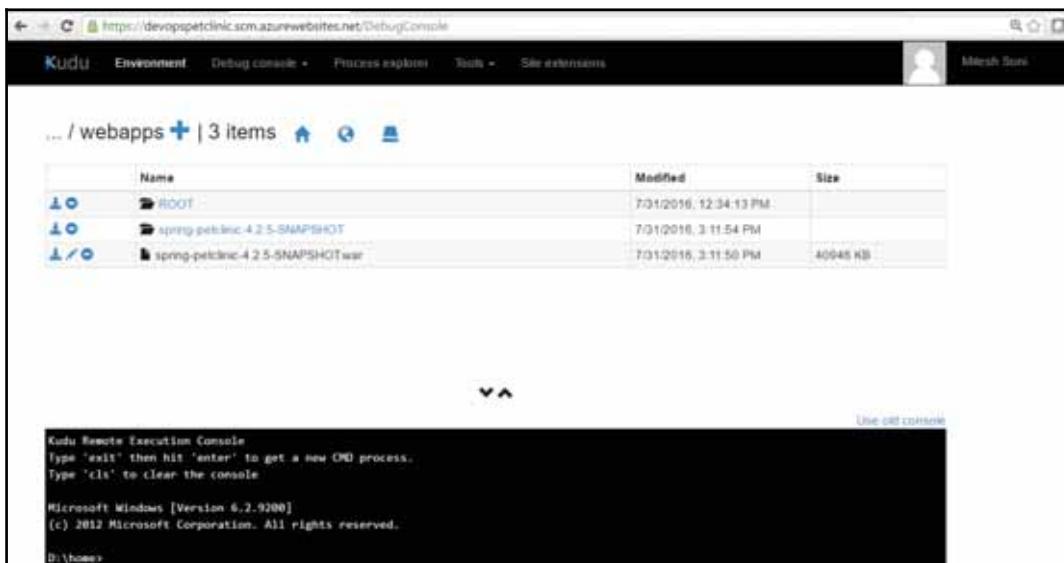
Buttons: Advanced..., Success, Test Configuration, Delete, Add

5. In the build job we created, go to the **Build** section and configure **Copy artifacts from another project**. We will copy the WAR file to a specific location on a virtual machine.
6. In **Post-build Actions**, click on **Send build artifacts over FTP**. Select the **FTP Server Name** configured in Jenkins. Configure **Source files** and the **Remove prefix** accordingly for deployment of an Azure Web App.

7. Select **Verbose output** in the console:



8. Click on **Build** now and see what happens behind the scenes.
9. Go to the Kudu console, click on **DebugConsole**, and go to **Powershell**. Go to **site | wwwroot | webapps**. Check whether the WAR file has been copied:



Now we have an application deployed on Azure Web Apps.

Summary

In this chapter, we have covered definitions of Continuous Delivery and Continuous Deployment. We have seen different approaches to application package deployment, such as application deployment in a local Tomcat server, and a Tomcat server available in Infrastructure as a Service (Amazon EC2), and Platform as a Service (AWS Elastic Beanstalk, Microsoft Azure App Services).

In the next chapter, we will discuss in detail how to perform different types of testing, such as functional testing using Selenium and load testing with Apache JMeter, to implement continuous testing.

6

Continuous Testing - Functional and Load Testing with Jenkins

Continuous Testing is one of the most important DevOps practices available for the End to End Automation of application lifecycle management.

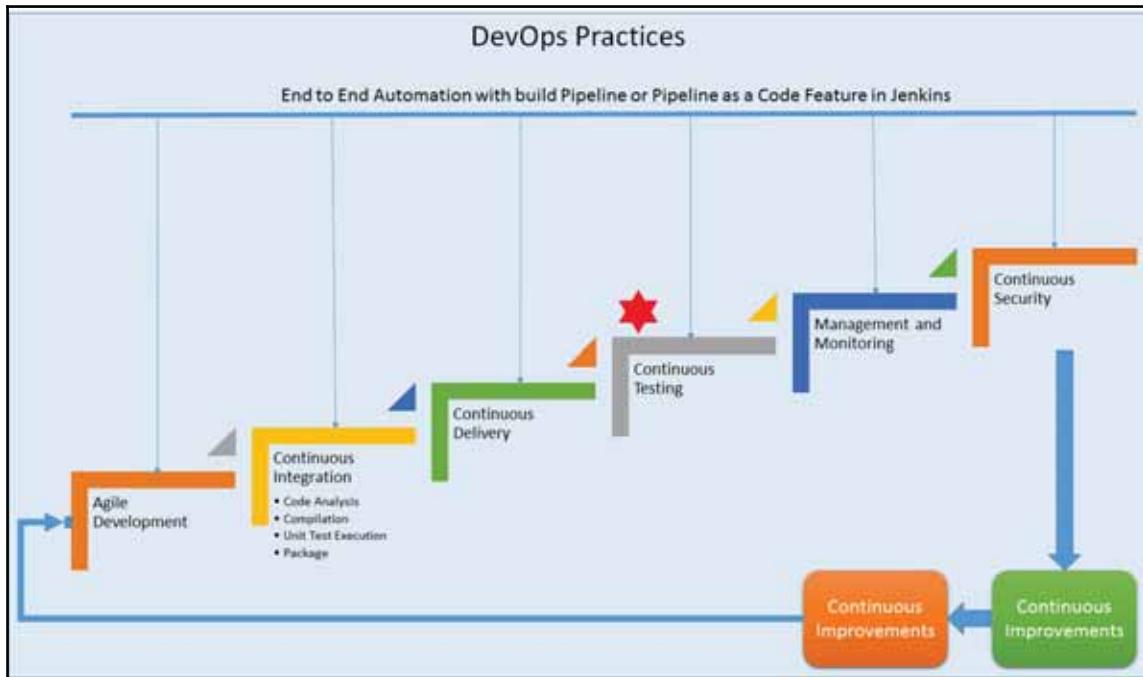
It not only considers automation, but it also includes aspects such as culture change and tools. It is essential to integrate automated tests into application lifecycle management early, to test quickly and in a timely manner, and to repeat the test execution process efficiently.

This chapter will give insights into how functional testing and load testing can be performed and how they can be integrated with Jenkins to adopt Continuous Testing practices as part of a DevOps culture.

This is not the whole picture of Continuous Testing, but it will certainly give a glimpse of how to use Continuous Testing DevOps practices to change the existing culture using automated tests. In this chapter, we will cover following topics:

- Functional testing with Selenium
- Jenkins and Selenium integration
- Load testing with Apache JMeter
- Jenkins and Apache JMeter integration

In this chapter, we will cover Continuous Testing practices as a part of our DevOps journey:



At the end of this chapter, we will know how to perform functional testing and load testing on the deployed application.

Functional testing with Selenium

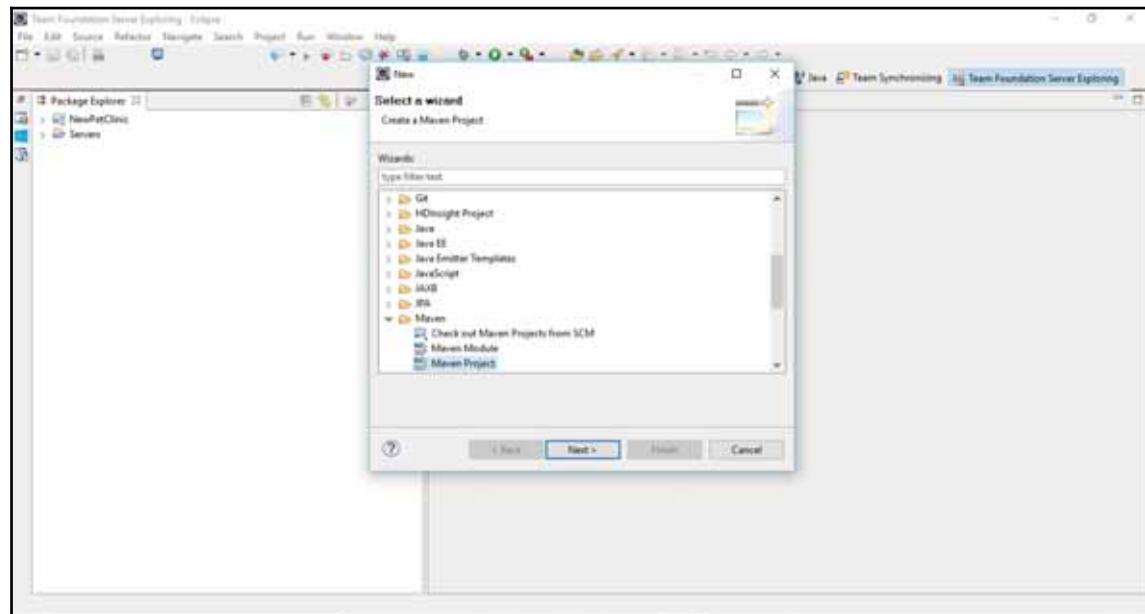
In this chapter, we will use Selenium and Eclipse for a functional test case execution. We have already deployed the application in the Tomcat, so we can perform functional tests and load tests on that deployment.

Let's go step by step through creating a sample functional test case and then executing it using Jenkins.

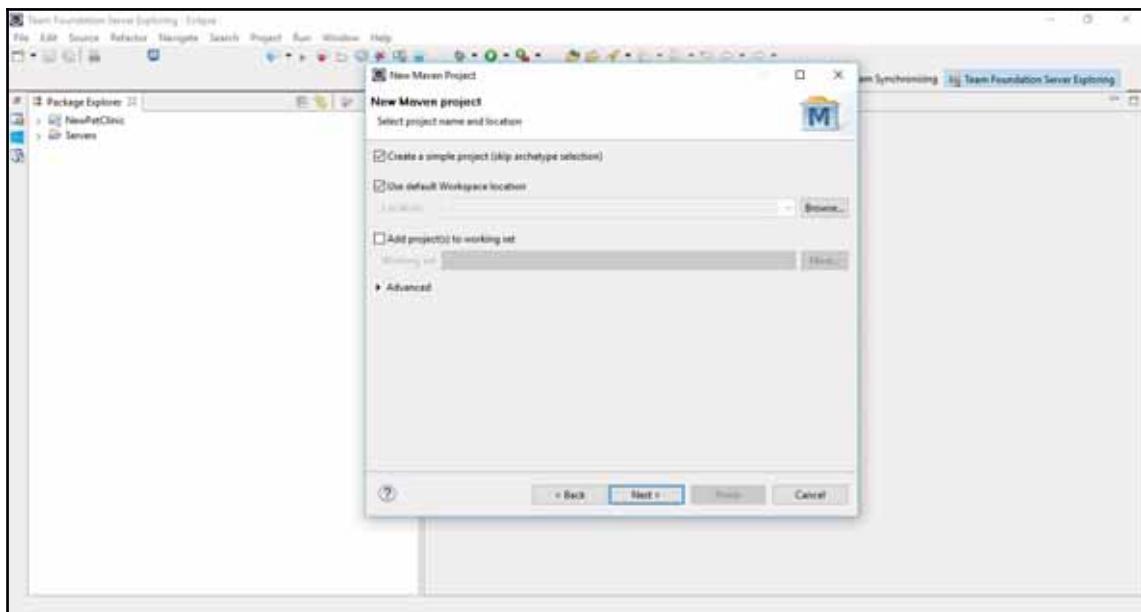
The PetClinic project is a Maven-based Spring application, and we will create a test case using Eclipse and Maven. We will utilize the `m2eclipse` plugin in Eclipse.

We have installed Eclipse Java EE IDE for Web Developers, Version: Mars.2 Release (4.5.2), Build ID: 20160218-0600, so lets start!

1. Go to the Eclipse marketplace and install the **Maven Integration for Eclipse** plugin.
2. Create a **Maven project** using a wizard in the Eclipse IDE:



3. Select **Create a simple project (skip archetype selection)** and click on **Next**:

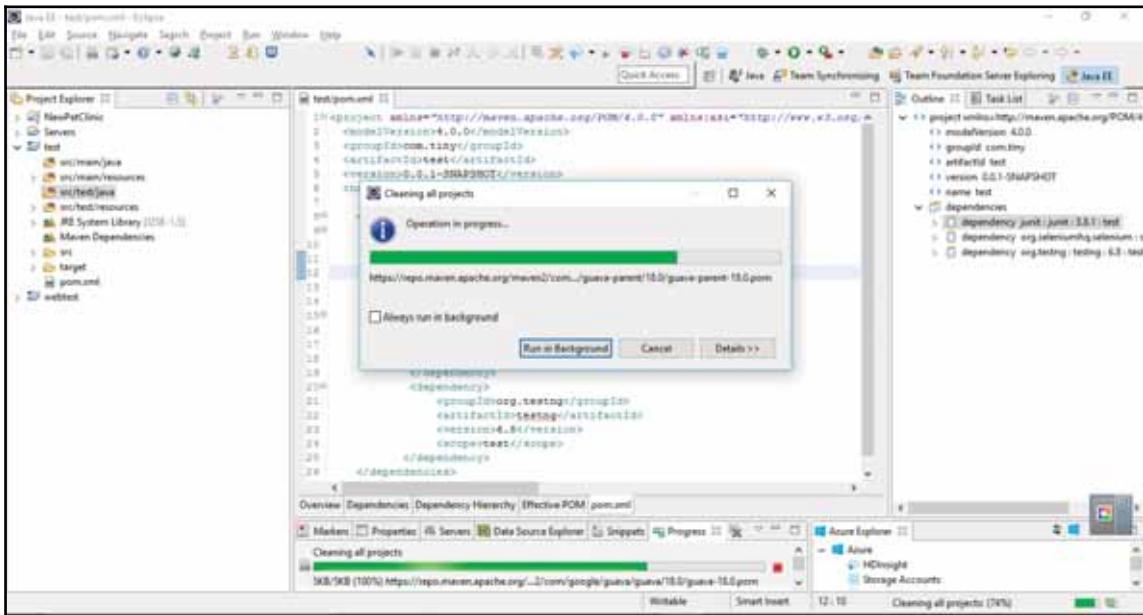


4. Go through the wizard and create a project. It will take some time to create a project in Eclipse. Provide **Artifact**, **Version**, **Packaging**, **Name**, and **Description**. Click on **Finish**.
5. Wait until the Maven project is created and configured. Make sure that Maven is installed and configured properly. If Maven is behind a proxy, configure the proxy details in `conf.xml`, available in the Maven directory.
6. In `Pom.xml`, we need to add Maven, Selenium, TestNG, and JUnit dependencies in the `<project>` node. The following is a modified `Pom.xml`:

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tiny</groupId>
  <artifactId>test</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>test</name>
  <build>
    <plugins>
```

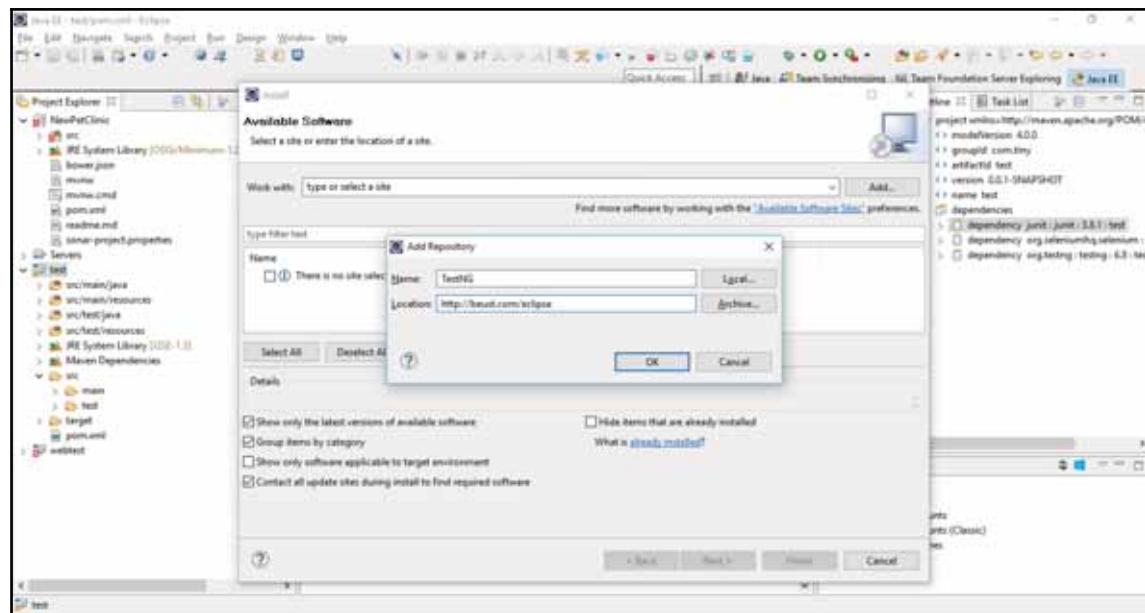
```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.6.1</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.19.1</version>
<configuration>
<suiteXmlFiles>
<suiteXmlFile>testng.xml</suiteXmlFile>
</suiteXmlFiles>
</configuration>
</plugin>
</plugins>
</build>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>3.0.1</version>
</dependency>
<dependency>
<groupId>org.testng</groupId>
<artifactId>testng</artifactId>
<version>6.8</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

7. Save pom.xml after adding these changes and build the project again from the Project menu. It will download new dependencies:

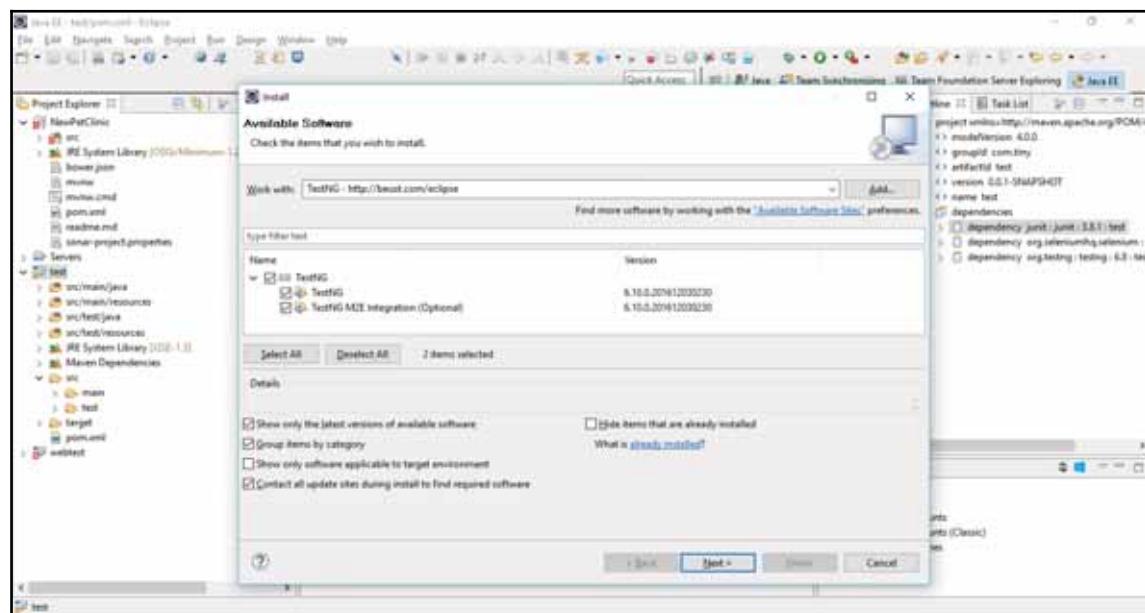


Create a Maven Project in Eclipse IDE

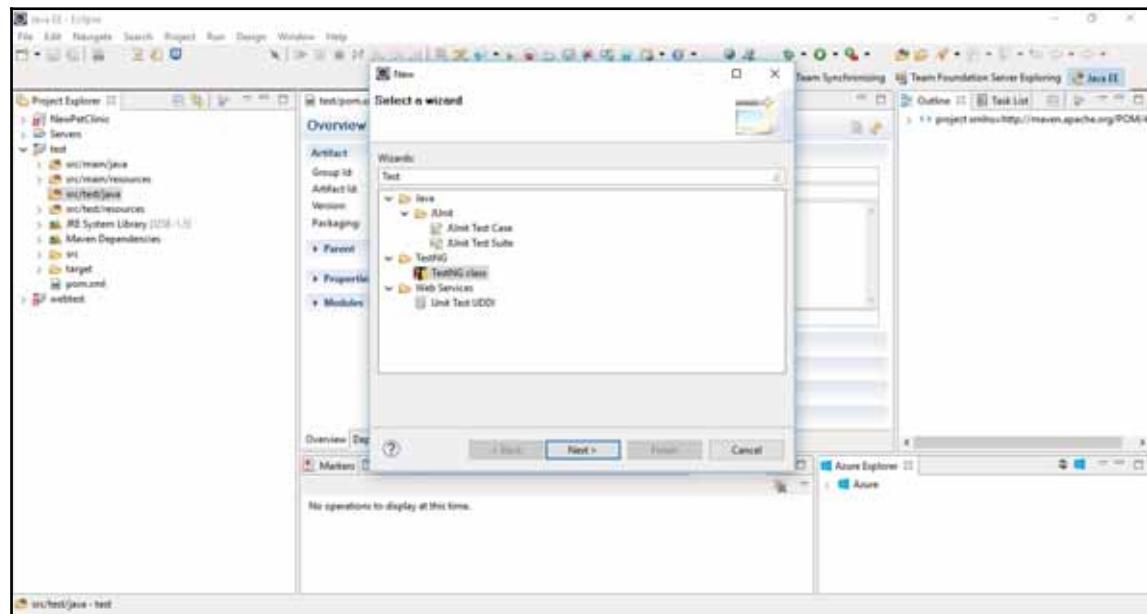
8. Click on the **Details** button of the dialog box to verify the operation in progress.
9. The next task is to write the TestNG class. Install the TestNG plugin. Go to **Help** and click on **Install New Software** and **Add Repository**:



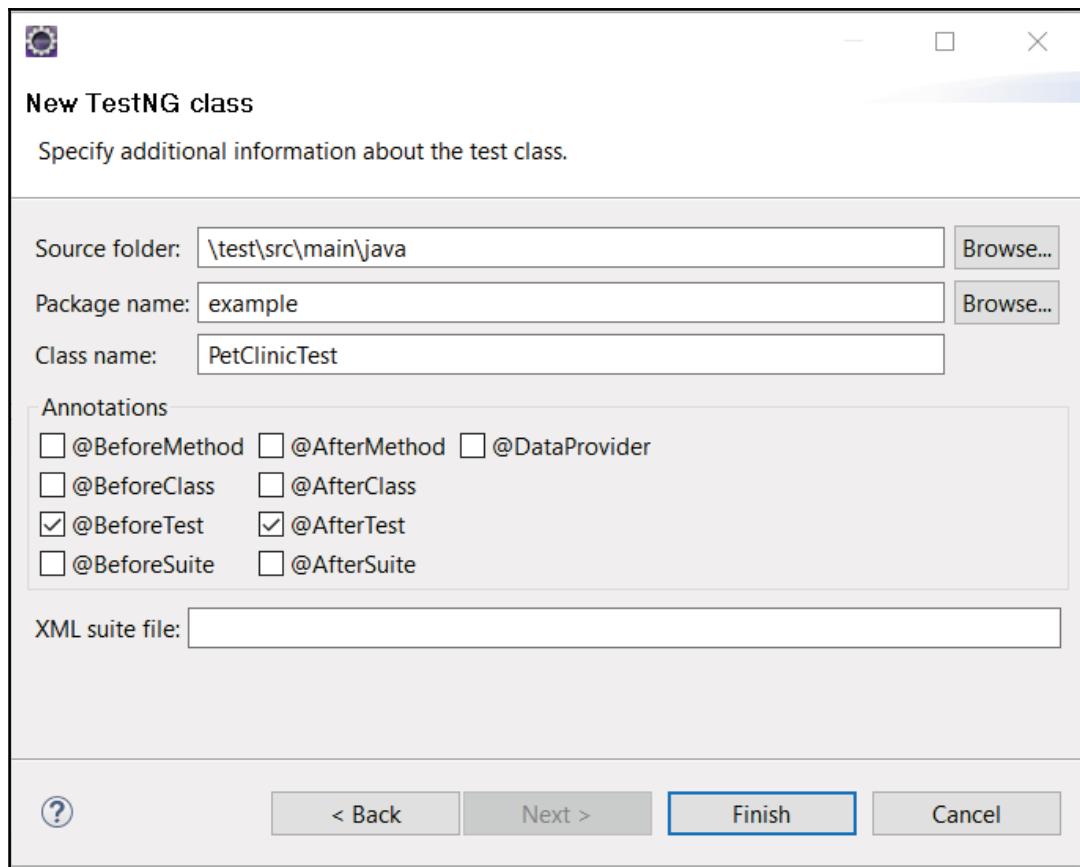
10. Select the items we need to install:



11. Review all the items that need to be installed and click on **Next**.
12. Accept the license and click on **Finish**.
13. Verify the installation progress in Eclipse.
14. Now let's create a **TestNG class**:



15. Provide a class name:



16. Give a package name and click on **Finish**.

17. The newly created class will look like the following screenshot:

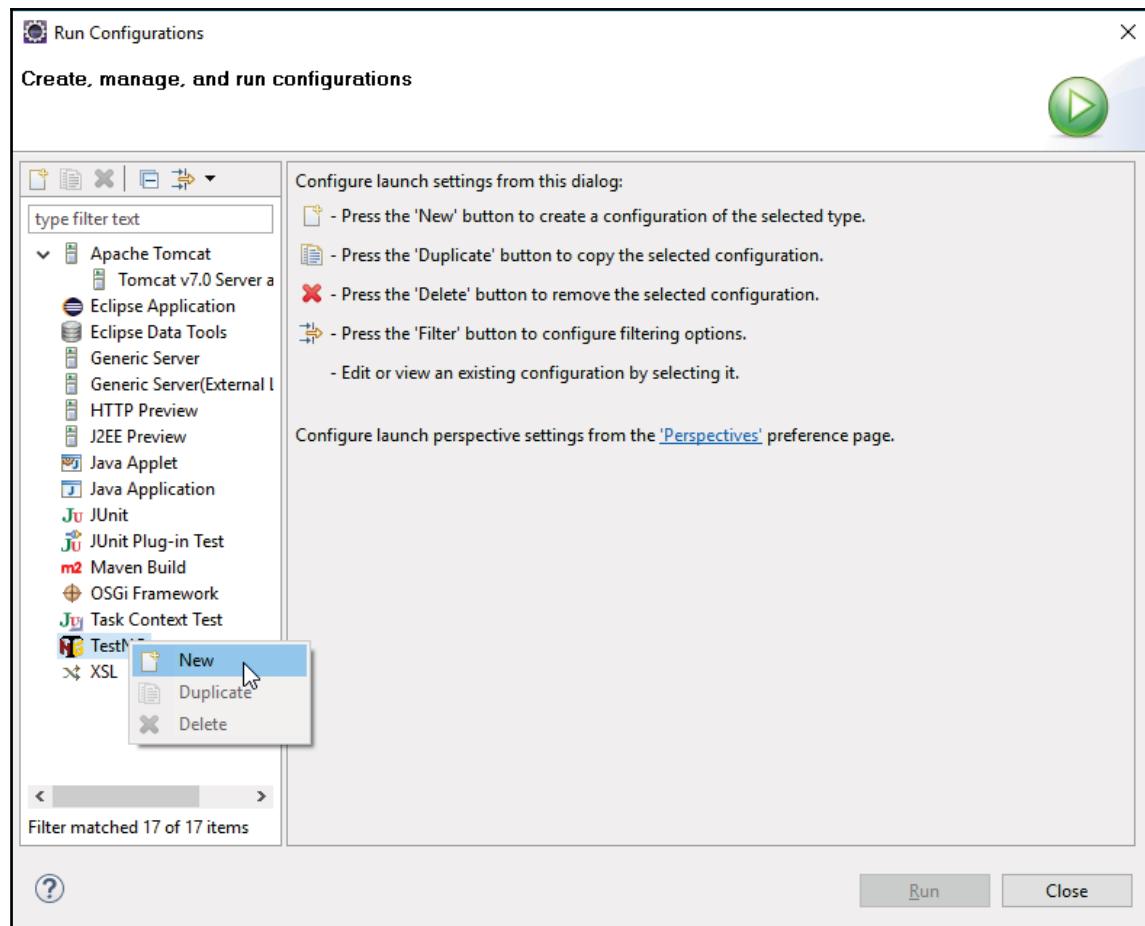
```
package example;

import org.junit.Test;
import org.junit.rules.BeforeTest;
import org.junit.rules.AfterTest;
```

```
public class TestOneTest {
    @Test
    public void testOne() {
        // Test logic
    }
    @BeforeTest
    public void beforeTest() {
    }
    @AfterTest
    public void afterTest() {
    }
}
```

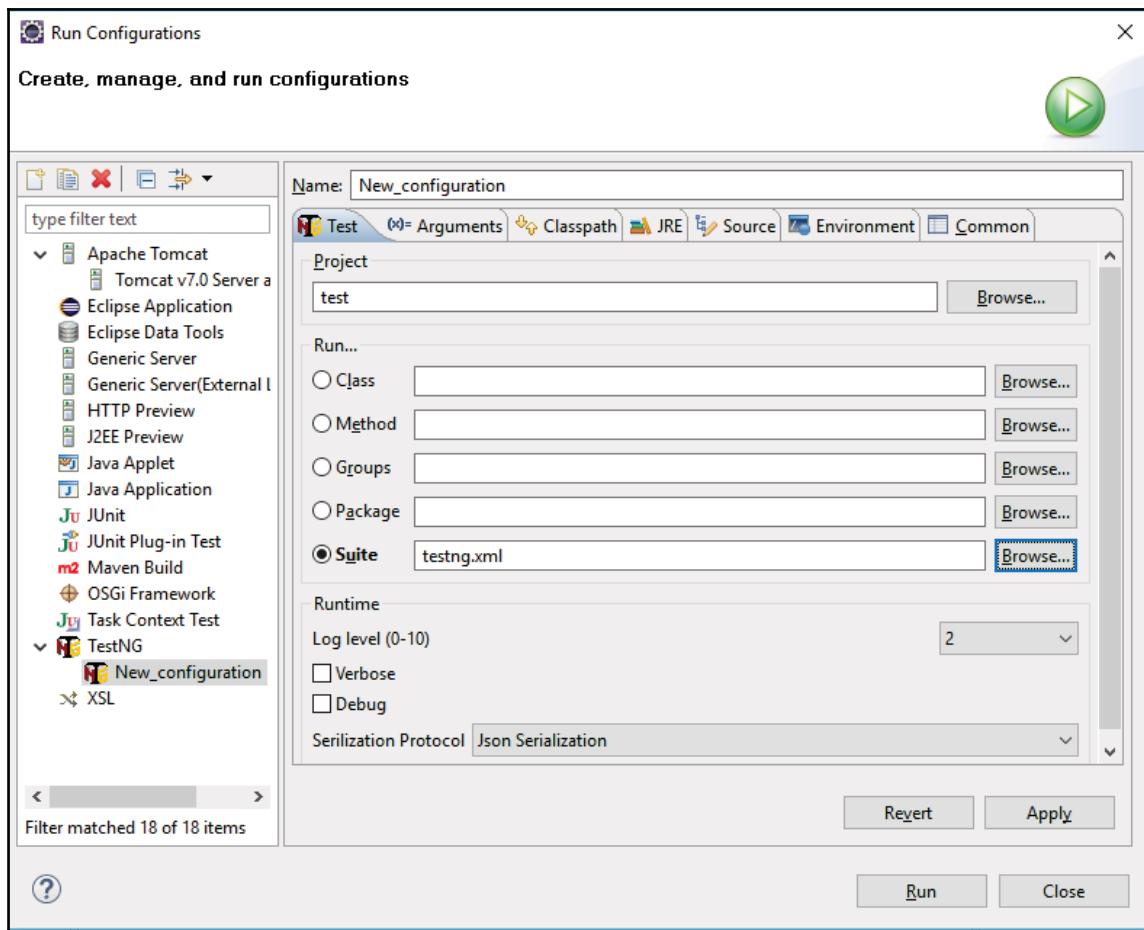
The screenshot shows the Eclipse IDE environment. The Project Explorer on the left lists several Java files and configurations. The Java Editor in the center displays the code for the `TestOneTest` class. Below the editor, the Eclipse Console window shows the output of a test run, indicating a successful execution. To the right of the editor, there is a "Results of running suite" view showing a summary of the test results.

18. Right-click on the test file and click on **TestNG**, convert to TestNG.
19. This will create a `testing.xml` file that has details about the test suite.
20. Right-click on project and click on **Run Configurations**.
21. Right-click on **TestNG** and click on **New**:



22. Provide the project name and select `testing.xml` in the suite.
23. Click **OK** and **Apply**.

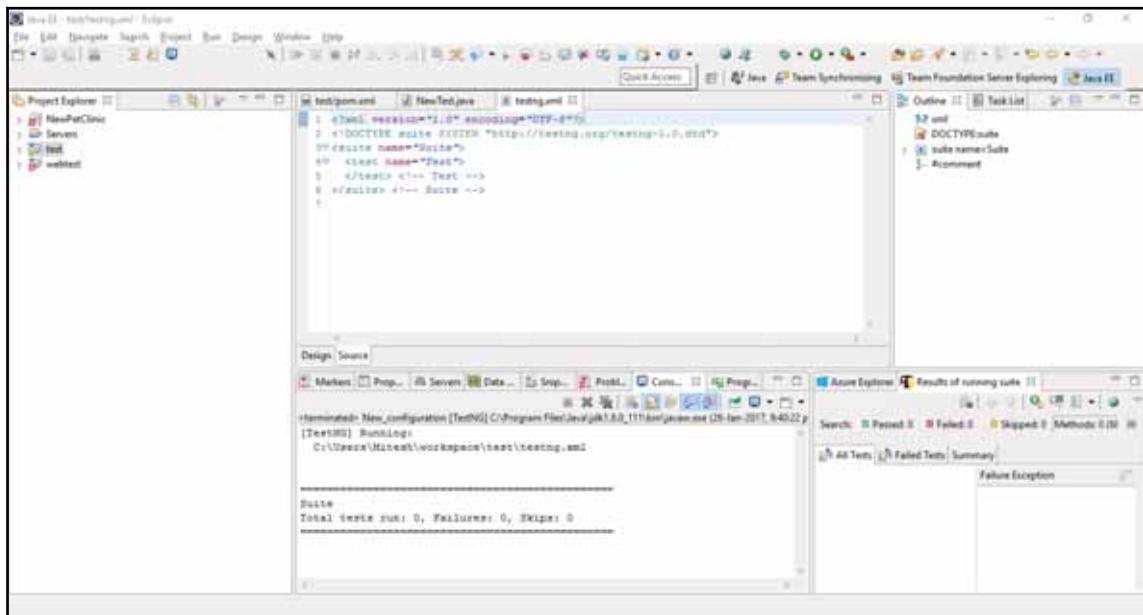
24. Click on Run:



25. If Windows Firewall blocks it, then click on **Allow Access**.
26. There is no configuration available in `testng.xml` for execution, so even if Maven execution runs successfully, no suite will be executed.

27. Generate the TestNG class under the test folder. Select location, suite name, and class name:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
<test name="Test">
<classes>
<class name="example.PetClinicTest"/>
</classes>
</test><!-- Test -->
</suite><!-- Suite -->
```

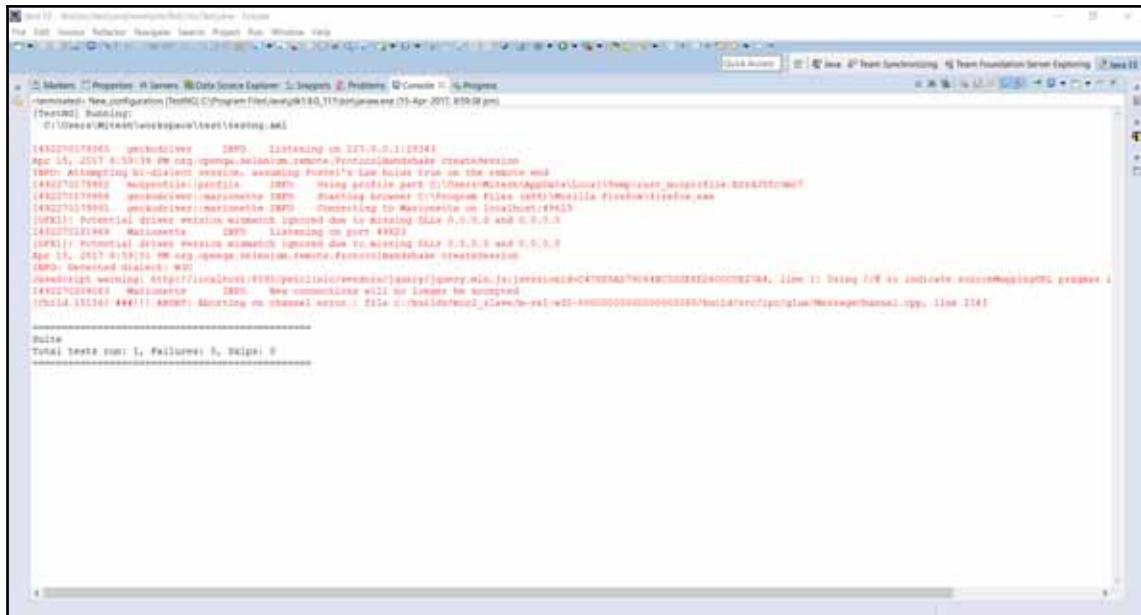


28. Go to <https://github.com/mozilla/geckodriver/releases> and download a version.
29. Extract the file available in the downloaded ZIP file, based on the system configuration you have. In our case, we have downloaded geckodriver-v0.13.0-win64.
30. Click on it and verify the driver details.

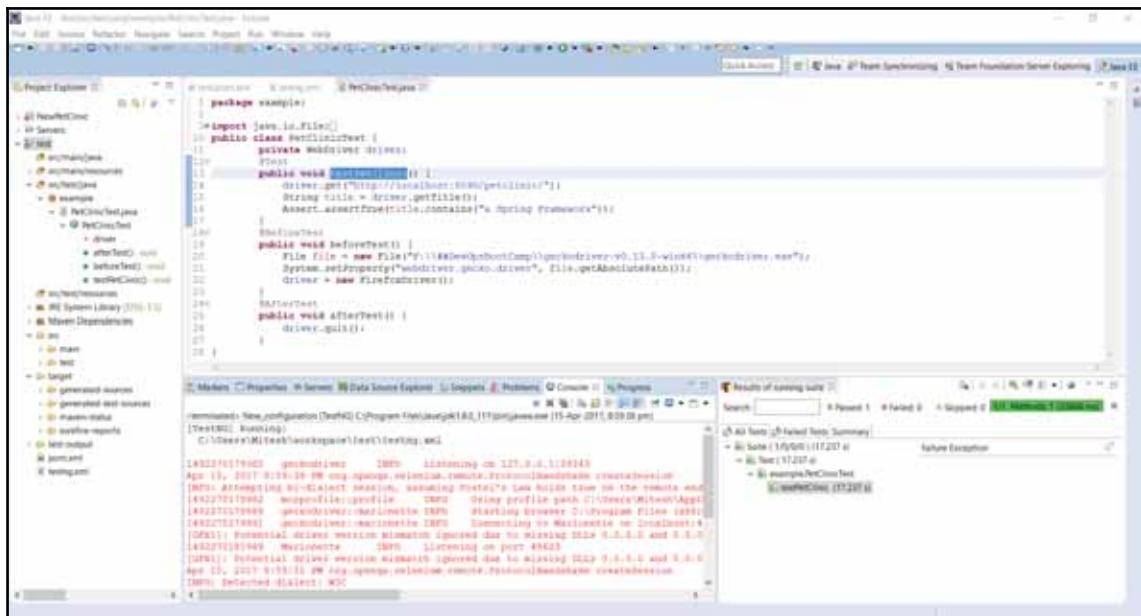
31. Let's write some code as well. It will check whether the title of the web page contains a specific string or not. The result or the outcome of the following code is based on the title of the page. If it contains a given string, then the test case will pass; otherwise, it will fail. Here is an example package:

```
import java.io.File;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;
public class PetClinicTest {
    private WebDriver driver;
    @Test
    public void testPetClinic() {
        //Change the URL based on the location where Tomcat is installed
        //and application is deployed
        driver.get("http://localhost:8090/petclinic/");
        String title = driver.getTitle();
        Assert.assertTrue(title.contains("a Spring
Frameworkk"));
    }
    @BeforeTest
    public void beforeTest() {
        File file = new
        // We have used Firefox for testing; change this driver based on
        requirements and location too
        File("F:\\##JenkinsEssentials\\geckodriver-v0.13.0-
        win64\\geckodriver.exe");
        System.setProperty("webdriver.gecko.driver",
        file.getAbsolutePath());
        driver = new FirefoxDriver();
    }
    @AfterTest
    public void afterTest() {
        driver.quit();
    }
}
```

32. Let's run the Maven test again from Eclipse.
33. The following is the output when the test case is executed successfully:

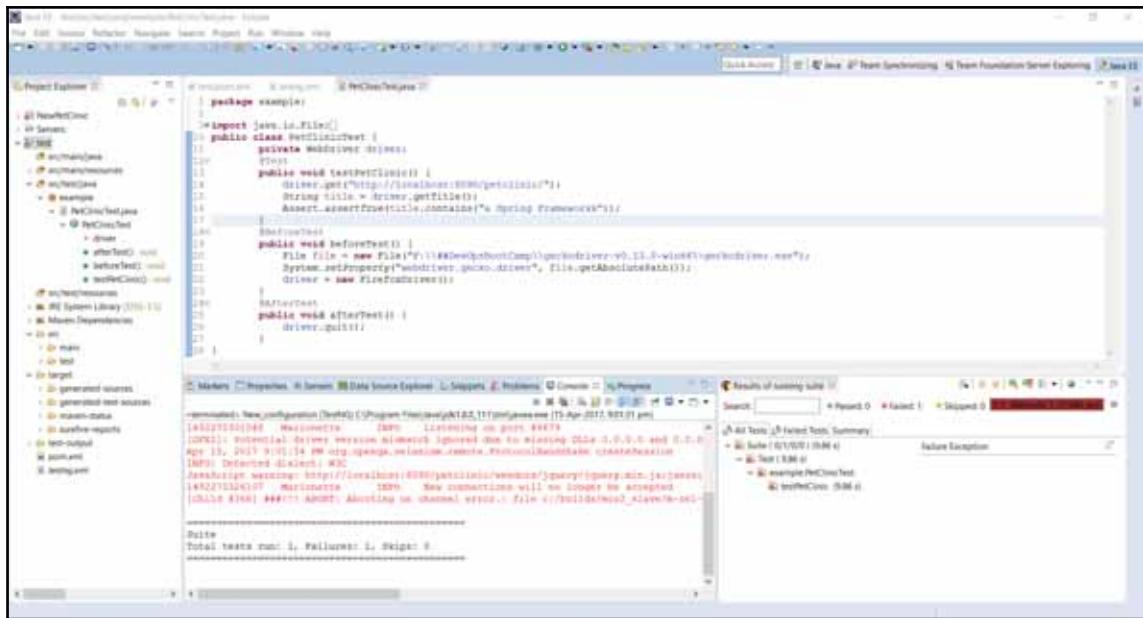


34. Check the **All Tests** tab in the **Results** of the running suite section in Eclipse. We can see successful execution here:



35. Check the **Failed Tests** tab in the **Results** of the running suite section in Eclipse.
 36. Check the **Summary** tab in the **Results** of the running suite section in Eclipse in the successful scenario.
 37. In the code, change the text available for title comparison so the test case fails.
 38. Verify the output in Console:

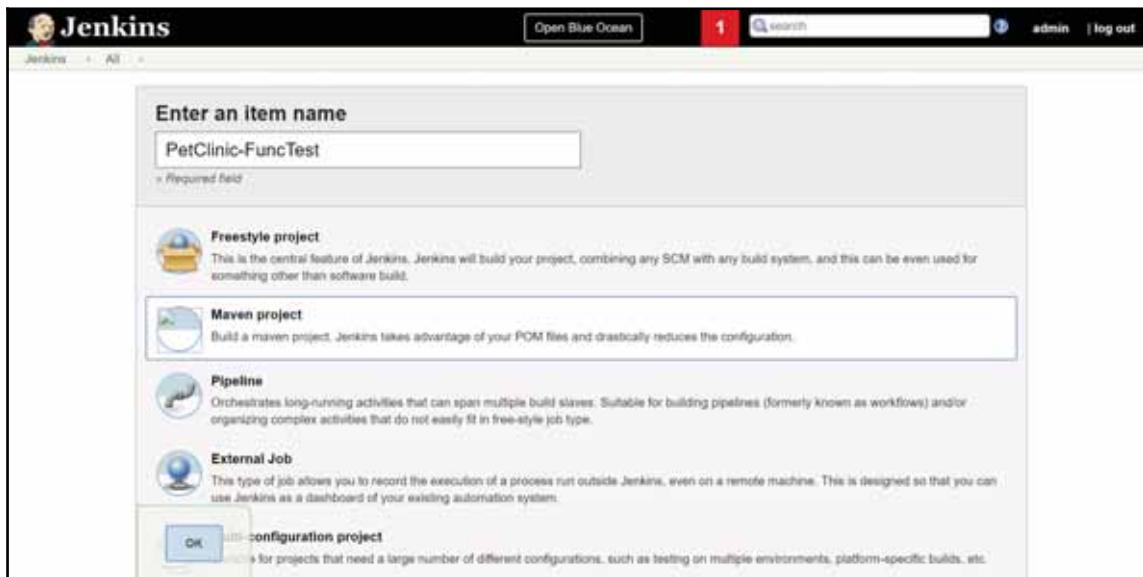
39. Check the **All Tests** tab in the **Results** of running suite section in Eclipse and notice the failure icon.



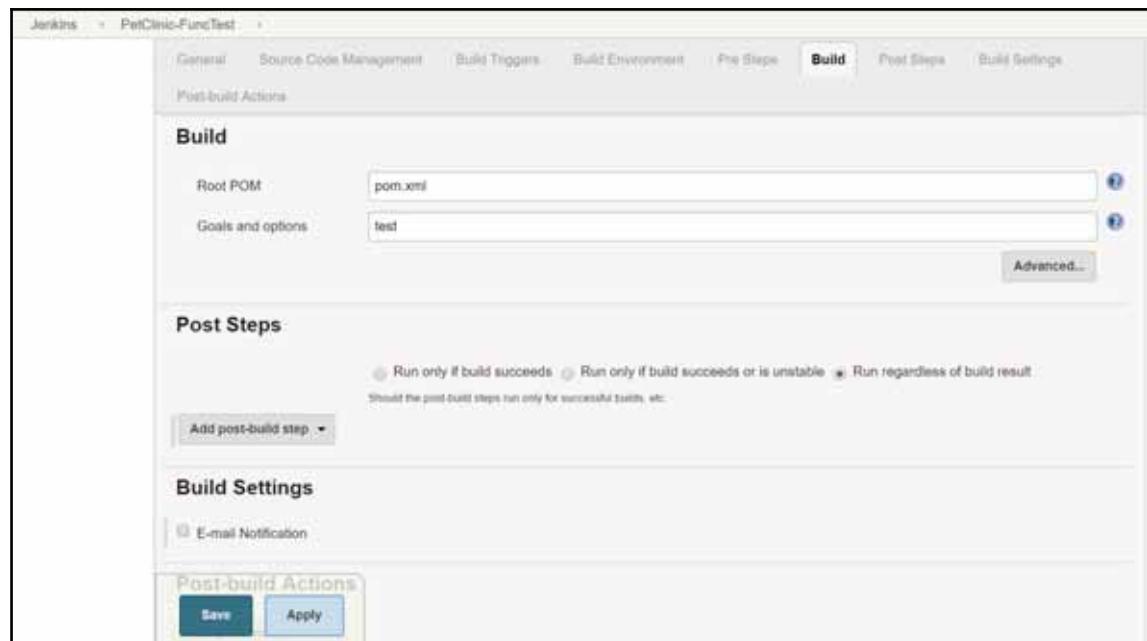
40. Observe the **Failed Tests** tab in the **Results** of the running suite section in Eclipse.
41. Click on `testPetClinic` and verify the **Failure Exception**.
42. Check the **Summary** tab in the **Results** of the running suite section in Eclipse.

So, we have created a sample test case based on Selenium to verify the title of the PetClinic home page.

Now let's try to execute the same thing from Jenkins:



1. Check in the **Test Project in Repository**. Create a PetClinic-FuncTest freestyle job in Jenkins.
2. In the **Build** section, provide the **Root POM** location and **Goals and options** to execute:

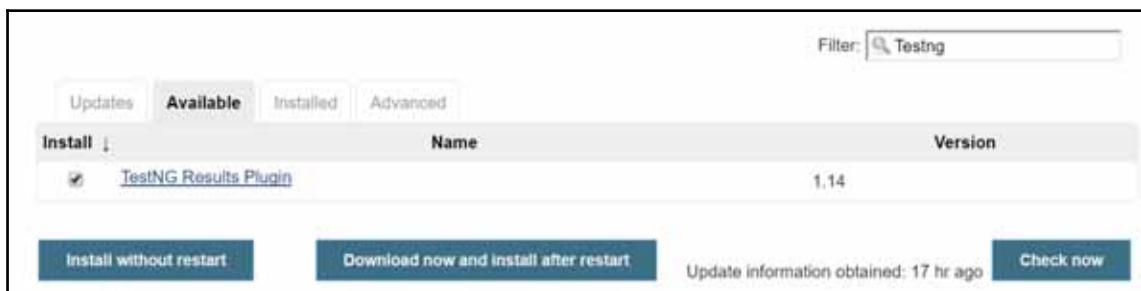


3. Save the build job and click on **Build now**.
4. Verify the execution of the build job in the Console output.

5. This will open a Mozilla Firefox window and open the URL that is given in the code. This requires our PetClinic application to be deployed on a web server and be running without any issues:

```
-----  
T E S T S  
-----  
Running TestSuite  
1496574254522 geckodriver INFO Listening on 127.0.0.1:6486  
Jun 04, 2017 4:34:14 PM org.openqa.selenium.remote.ProtocolHandshake createSession  
INFO: Attempting bi-dialect session, assuming Postel's Law holds true on the remote end  
1496574255134 mozprofile::profile INFO Using profile path  
C:\Users\Mitesh\AppData\Local\Temp\rust_mozprofile.t3u0Siy560nn  
1496574255138 geckodriver::marionette INFO Starting browser C:\Program Files (x86)\Mozilla Firefox\firefox.exe  
1496574255175 geckodriver::marionette INFO Connecting to Marionette on localhost:60430  
[GFX1]: Potential driver version mismatch ignored due to missing DLLs 0.0.0 and 0.0.0  
[GFX1]: Potential driver version mismatch ignored due to missing DLLs 0.0.0 and 0.0.0  
1496574288578 Marionette INFO Listening on port 60430  
Jun 04, 2017 4:34:49 PM org.openqa.selenium.remote.ProtocolHandshake createSession  
INFO: Detected dialect: W3C  
JavaScript warning:  
http://localhost:8090/petclinic/vendors/jquery/jquery.min.js; sessionId=884F4251137D820A5723530BAA688915, line 1:  
Using //@ to indicate sourceMappingURL pragmas is deprecated. Use //# instead  
>>>>PetClinic :: a Spring Framework demonstration  
1496574305189 Marionette INFO New connections will no longer be accepted  
Jun 04, 2017 4:35:12 PM org.openqa.selenium.os.UnixProcess destroy  
SEVERE: Unable to kill process with PID 10376  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 69.499 sec - in TestSuite  
Results :  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

6. Install TestNG Results Plugin:



The screenshot shows the Jenkins plugin manager interface. At the top, there is a search bar labeled "Filter: Testng". Below it, there are tabs for "Updates", "Available" (which is selected), "Installed", and "Advanced". A table lists the "TestNG Results Plugin" with the following details:

Install	Name	Version
<input checked="" type="checkbox"/> TestNG Results Plugin		1.14

At the bottom of the table are three buttons: "Install without restart" (highlighted in blue), "Download now and install after restart", and "Check now". To the right of the table, a message says "Update information obtained: 17 hr ago".

7. Go to Post build Actions and select Publish TestNG Results:

The screenshot shows the Jenkins configuration page for the 'PetClinic-FuncTest' job. The 'Build' tab is selected. In the 'Post-build Actions' section, the 'Publish Testing Results' option is highlighted and selected. A tooltip for 'Publish Testing Results' indicates it runs only if the build succeeds or is unstable. Below the list of actions are 'Save' and 'Apply' buttons.

8. Provide **TestNG XML report pattern**:

The screenshot shows the Jenkins configuration page for the 'PetClinic-FuncTest' job. The 'Post-build Actions' section contains a single action: 'Publish TestNG Results'. The 'TestNG XML report pattern' field is set to '**/testng-results.xml'. Below the list of actions are 'Save' and 'Apply' buttons.

9. Click on **Build now:**

The screenshot shows the Jenkins interface for the 'PetClinic-FuncTest' project. On the left, there's a sidebar with links like 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Maven project', 'Configure', 'Modules', 'Favorite', and 'TestNG Results'. Below that is a 'Build History' section with two entries: #2 (Jun 4, 2017 @ 48 PM) and #1 (Jun 4, 2017 @ 32 PM). At the bottom are 'RSS for all' and 'RSS for failures' links. The main area is titled 'Maven project PetClinic-FuncTest' and features a 'Test Result Trend' graph. The graph has three stacked areas: blue at the bottom, yellow in the middle, and red at the top. A legend on the right side of the graph identifies these colors. To the right of the graph are buttons for 'Add description' and 'Create Project'. Below the graph, there are links for 'TestNG Results', 'Workspace', 'Recent Changes', 'Latest Test Result (1 failure / +1)', 'Disk Usage', 'Job', 'All builds', 'Locked builds', 'All workspaces', and 'Slave workspaces'.

10. Go to the **Project** dashboard and verify the graphs for TestNG results:

This screenshot shows the 'TestNG Results Trends' dashboard for build #2. It displays a message: 'Need at least 2 builds with results to show trend graph'. Below this, it says 'Latest Test Results (build #2)' and lists the following test results:

- Total Tests: 1 (+1)
- Failed Tests: 0 (±0)
- Skipped Tests: 1 (+1)
 - example.PetClinicTest.testPetClinic
- Failed Configurations: 1 (+1)
 - example.PetClinicTest.beforeTest
- Skipped Configurations: 1 (+1)
 - example.PetClinicTest.afterTest

We have seen how to execute Selenium-based test cases in Jenkins. In the next section, we will see how to execute a load test using Jenkins.

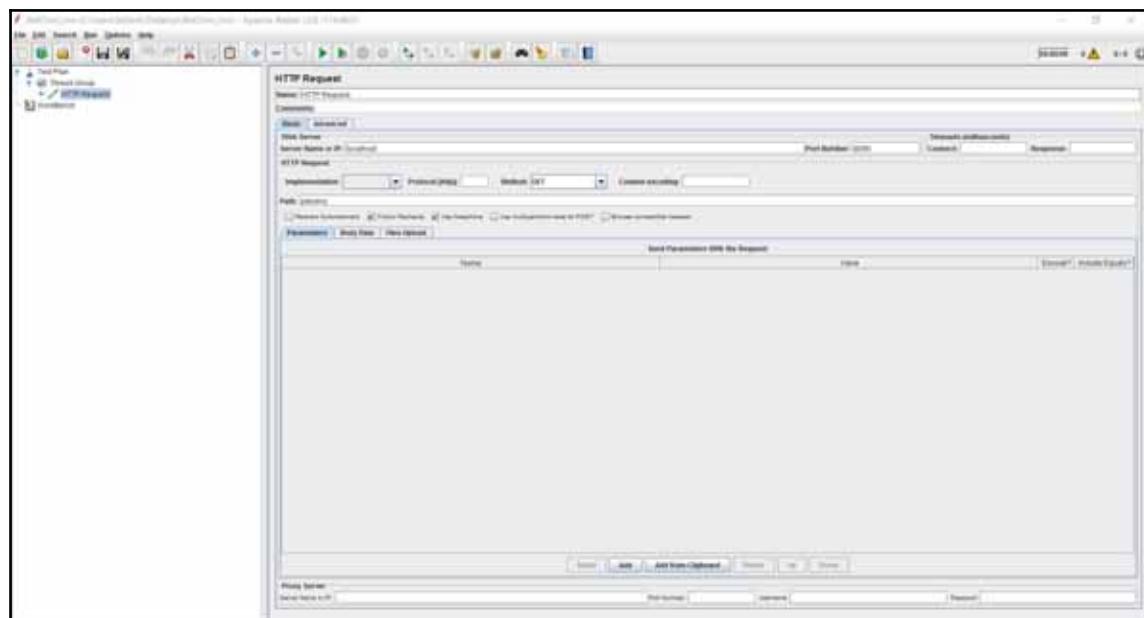
Load testing with Apache JMeter

Apache JMeter is an open source Apache project. It is a pure Java application. Apache JMeter is used to load test, in order to analyze and measure the performance of services.

Download Apache JMeter from http://jmeter.apache.org/download_jmeter.cgi. Extract the files and go to the bin directory. Execute `jmeter.bat` or `jmeter.sh`.

1. Open the Apache JMeter console. **Create a Test Plan**.
2. Right-click on the **Test Plan** and click on **Add**; select **Threads (Users)**.
3. Select **Thread Group**.
4. Provide **Thread Group name**.
5. In **Thread Group** properties, provide **Number of Threads**, **Ramp-up Period**, and **Loop Count**.
6. Right-click on **Thread Group**. Click on **Add**. Click on **Sampler**. Click on **HTTP Request**.
7. In HTTP Request, provide **Server Name or IP**. In our case, it will be localhost or an IP address.
8. Give the **Port Number** where your web server is running.

9. Select the **Get method** and provide a path to the load test:



Create an HTTPRequest in APache JMeter to configure Server Name, Port number, and method details

10. Save the .jmx file.

Now let's create a Jenkins job:

1. Create a freestyle job in Jenkins:

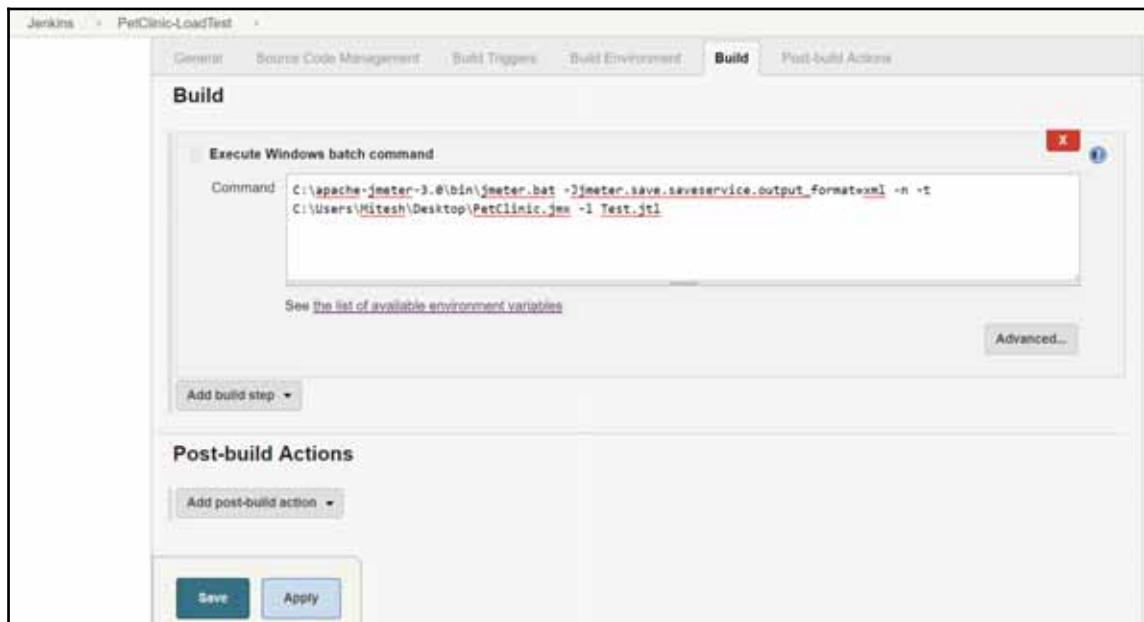
The screenshot shows the Jenkins interface for creating a new job. The top navigation bar includes 'Open Blue Ocean', a notification badge with the number '1', a search bar, and user account links for 'admin' and 'log out'. The main content area is titled 'Enter an item name' with a required field 'PetClinic-LoadTest'. Below this, there is a list of project types with descriptions:

- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software builds.
- Maven project**: Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- Pipeline**: Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- External Job**: This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.
- Multi-configuration project**: For projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

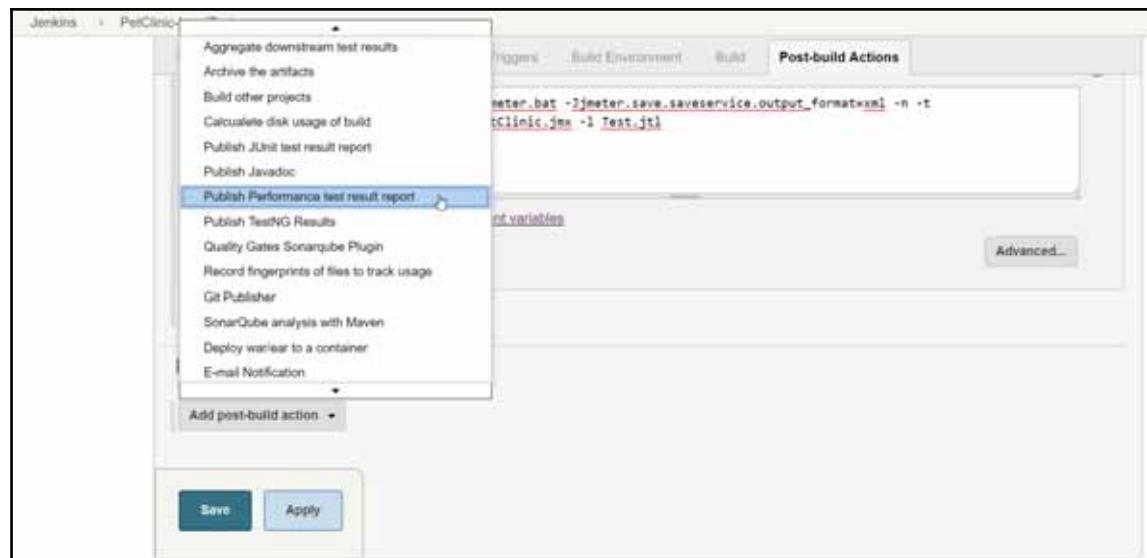
A blue 'OK' button is located at the bottom left of the form.

2. Add the Build step Execute Windows batch command. Add the following command. Replace the location of jmeter.bat based on the installation directory, and the location of the .jmx file too:

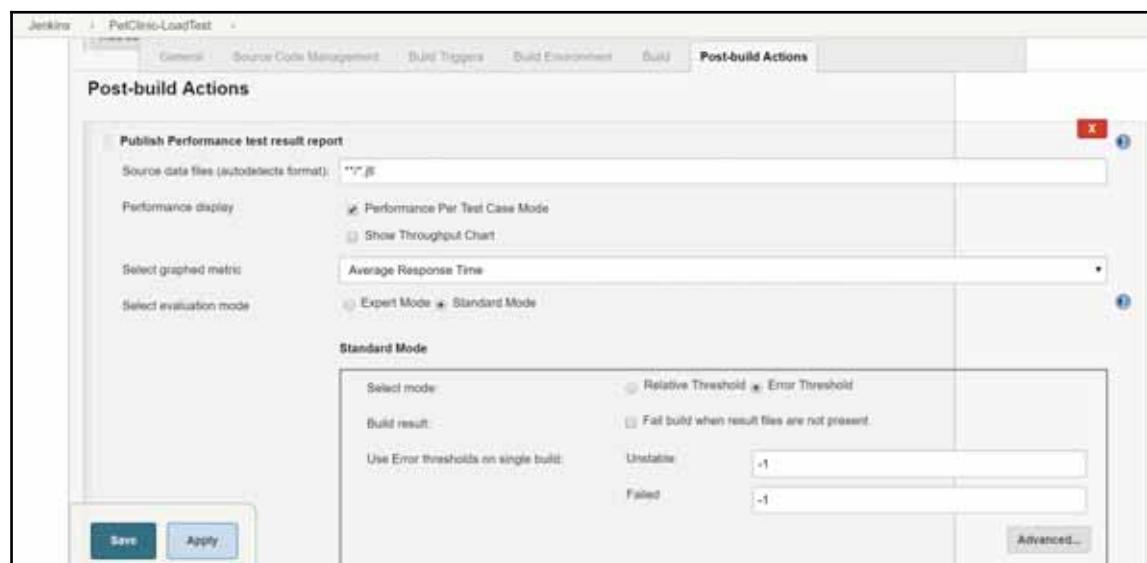
```
C:\apache-jmeter-3.0\bin\jmeter.bat -Jjmeter.save.saveservice.output_format=xml -n -t  
C:\Users\Mitesh\Desktop\PetClinic.jmx -l Test.jtl
```



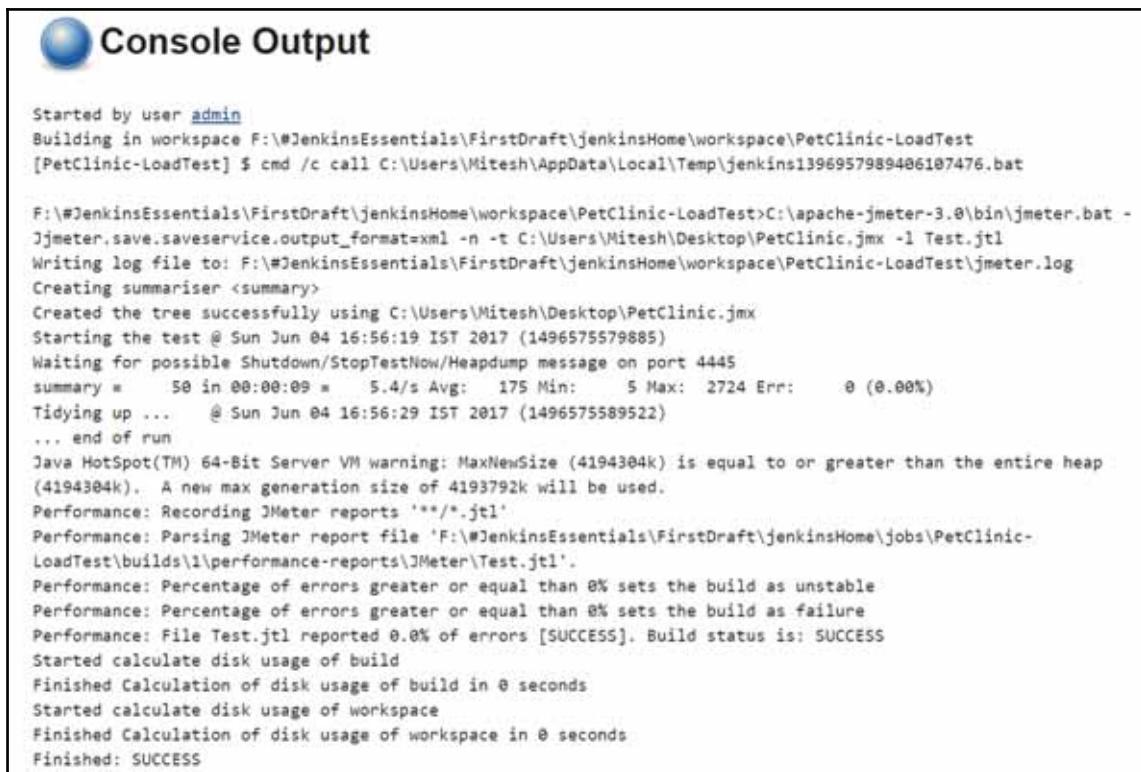
3. Add Post-build Actions:



4. Publish Performance test result report add `**/*.jtl` file.



5. Click on **Build now:**



The screenshot shows the Jenkins 'Console Output' page. At the top left is a blue circular icon with a white dot. To its right, the text 'Console Output' is displayed in a large, bold, black font. Below this, the Jenkins build log for the 'PetClinic-LoadTest' job is shown in a monospaced black font. The log details the execution of JMeter, including the creation of a summary report, the calculation of disk usage, and the final success status.

```
Started by user admin
Building in workspace F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-LoadTest
[PetClinic-LoadTest] $ cmd /c call C:\Users\Mitesh\AppData\Local\Temp\jenkins1396957989486107476.bat

F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-LoadTest>C:\apache-jmeter-3.0\bin\jmeter.bat -Jjmeter.save.saveservice.output_format=xml -n -t C:\Users\Mitesh\Desktop\PetClinic.jmx -l Test.jtl
Writing log file to: F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-LoadTest\jmeter.log
Creating summariser <summary>
Created the tree successfully using C:\Users\Mitesh\Desktop\PetClinic.jmx
Starting the test @ Sun Jun 04 16:56:19 IST 2017 (1496575579885)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary =      50 in 00:00:09 =    5.4/s Avg:   175 Min:     5 Max:  2724 Err:     0 (0.00%)
Tidying up ...  @ Sun Jun 04 16:56:29 IST 2017 (1496575589522)
... end of run
Java HotSpot(TM) 64-Bit Server VM warning: MaxNewSize (4194304k) is equal to or greater than the entire heap (4194304k). A new max generation size of 4193792k will be used.
Performance: Recording JMeter reports '*/*.jtl'
Performance: Parsing JMeter report file 'F:\#JenkinsEssentials\FirstDraft\jenkinsHome\jobs\PetClinic-LoadTest\builds\1\performance-reports\JMeter\Test.jtl'.
Performance: Percentage of errors greater or equal than 0% sets the build as unstable
Performance: Percentage of errors greater or equal than 0% sets the build as failure
Performance: File Test.jtl reported 0.0% of errors [SUCCESS]. Build status is: SUCCESS
Started calculate disk usage of build
Finished Calculation of disk usage of build in 0 seconds
Started calculate disk usage of workspace
Finished Calculation of disk usage of workspace in 0 seconds
Finished: SUCCESS
```

6. Verify **Performance Trend** on the Project dashboard by clicking on the Test results graph.

7. Click on **Performance Trend:**

The screenshot shows the Jenkins Project PetClinic-LoadTest dashboard. On the left, there's a sidebar with links like Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Project, Configure, Favorites, and Performance Trend. Below that is a Build History section with three builds listed: #3 (Jun 4, 2017 4:58 PM), #2 (Jun 4, 2017 4:57 PM), and #1 (Jun 4, 2017 4:56 PM). At the bottom of the sidebar are RSS links for all and failures.

The main content area has a title "Project PetClinic-LoadTest". It includes sections for "Workspace" (Last Successful Artifact: dashboard Test.xml, size 453 kB), "Recent Changes", and "Disk Usage" (Job: 178 kB, All builds: 178 kB, Locked builds: -, All workspaces: 32 kB, Slave workspaces: 32 kB, Non-slave workspaces: 32 kB).

On the right, there are two performance trend graphs: "Response time" and "Percentage of errors". The "Response time" graph shows a red line for 90% time, a blue line for average, and a green line for median. The "Percentage of errors" graph shows a single red line for errors.

8. Verify performance breakdown for **Response Time** and **Percentage of errors**:

This screenshot shows the Jenkins Performance Trend page for the Test file: Test.jtl. The left sidebar is identical to the previous dashboard, including the Performance Trend link. The main area is titled "Performance Trend" and "Test file: Test.jtl". It features two line graphs: "Response time" (red line) and "Percentage of errors" (red line). The "Response time" graph shows a peak around the third build. The "Percentage of errors" graph shows a constant value near zero.

9. Click on **Last Report** and get more details on the load test results:



Done!

Summary

Finally, we are at the end of the chapter. We have performed functional testing using Selenium and then integrated it with Jenkins. We have also performed load testing using Apache JMeter and then integrated it with Jenkins.

This is useful when we want to achieve automated testing in the pipeline for functional testing and load testing. We can set notifications and other configurations based on the culture of an organization too.

In the next chapter, we will cover how to orchestrate all the build jobs we have created up to now, to create a pipeline. We will create a pipeline using the Build Pipeline plugin and the Jenkins 2 Pipeline feature.

7

Build Pipeline and Pipeline as a Code

Up to now, we have covered all specific tasks that are individual and can work as a stepping stone to performing other steps, such as static code analysis, Continuous Integration, Continuous Delivery, and Continuous Testing.

What if we need to fix the sequence of execution of all such tasks with or without manual intervention?

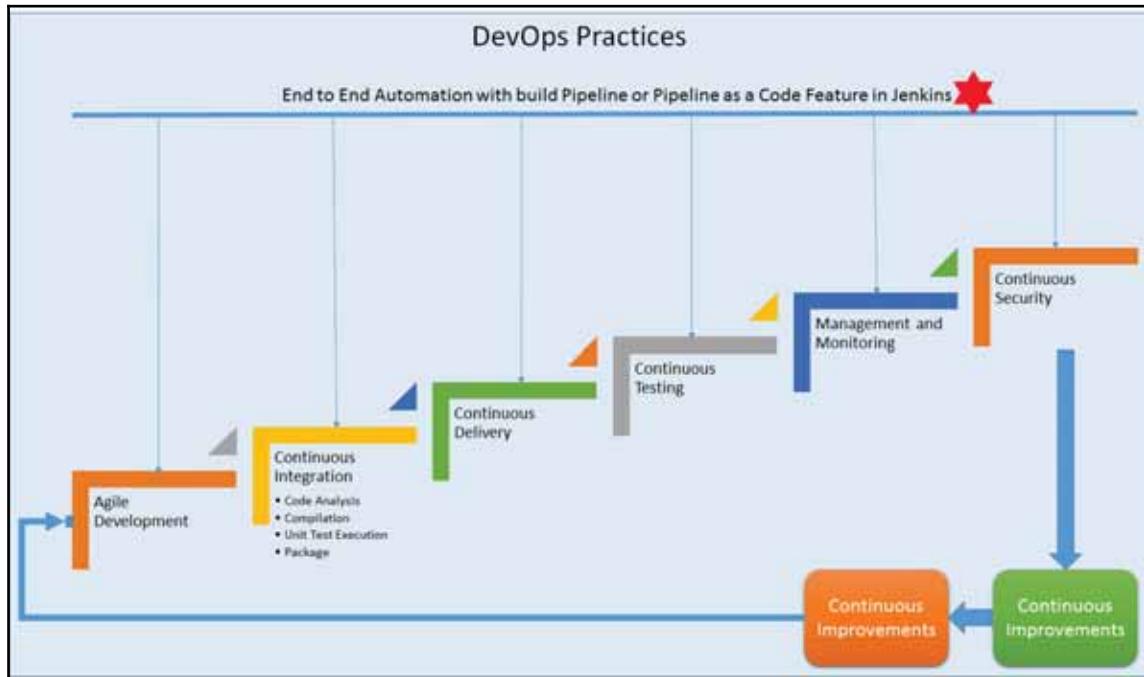
What if we want to create a pipeline where one successful execution of Job can lead to another execution of Job?

This is where we will utilize the Build Pipeline plugin and the Pipeline as a Code feature available in Jenkins 2 for the orchestration of the end-to-end automation of Application Life Cycle Management. We have executed a pipeline and all build jobs on a Windows system. Based on your operating system, there might be some changes that you may need to do and we have mentioned these as comments at specific places in the script or code.

This chapter will cover how to orchestrate a build job to execute it in a specific sequence along with other build jobs. We will cover the Build Pipeline plugin and the Pipeline as a Code feature that is available in Jenkins 2 and later. The following are the major topics that we will cover in this chapter:

- Build Pipeline
- Upstream and downstream jobs
- Overview of Pipeline as a Code
- Pipeline as a Code: implementation
- Promoted builds

In this chapter, we will cover end-to-end automation with the Build Pipeline plugin and the Pipeline as a Code feature as a part of our DevOps journey:



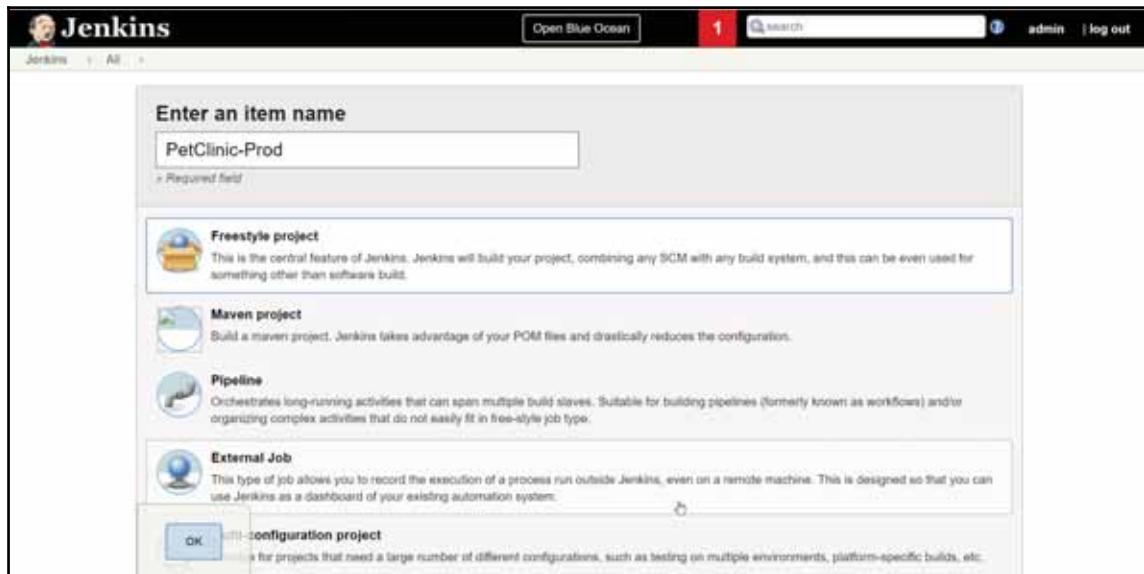
At the end of this chapter, we will know how to create and configure a pipeline using the plugin, and also using Groovy syntax.

Build Pipeline

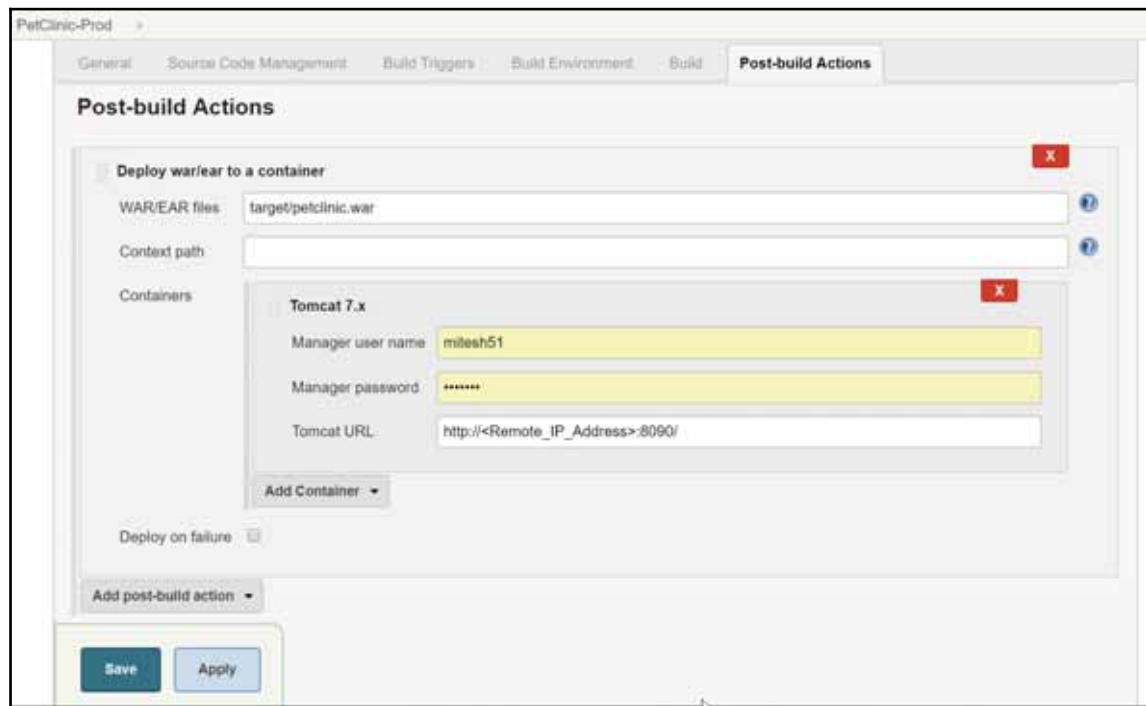
Continuous Integration and Continuous Delivery have become popular practices for application development. The Build Pipeline plugin provides a pipeline view of upstream and downstream connected jobs that typically form a build pipeline, with the ability to define manual triggers or an approval process. We can create a chain of jobs by orchestrating version promotion through different quality gates, before we deploy it in production.

Before starting with the Build Pipeline plugin, let's create a job to deploy into a production environment. We will use the Deploy to Container plugin for application deployment in remote Tomcat.

1. Click on **New Item** in the Jenkins Dashboard and give it a name. Click on **OK**:



2. Configure a post-build action similar to what we configured in the **PetClinic-Deploy** job:



3. Click on **Save** and **Apply**.

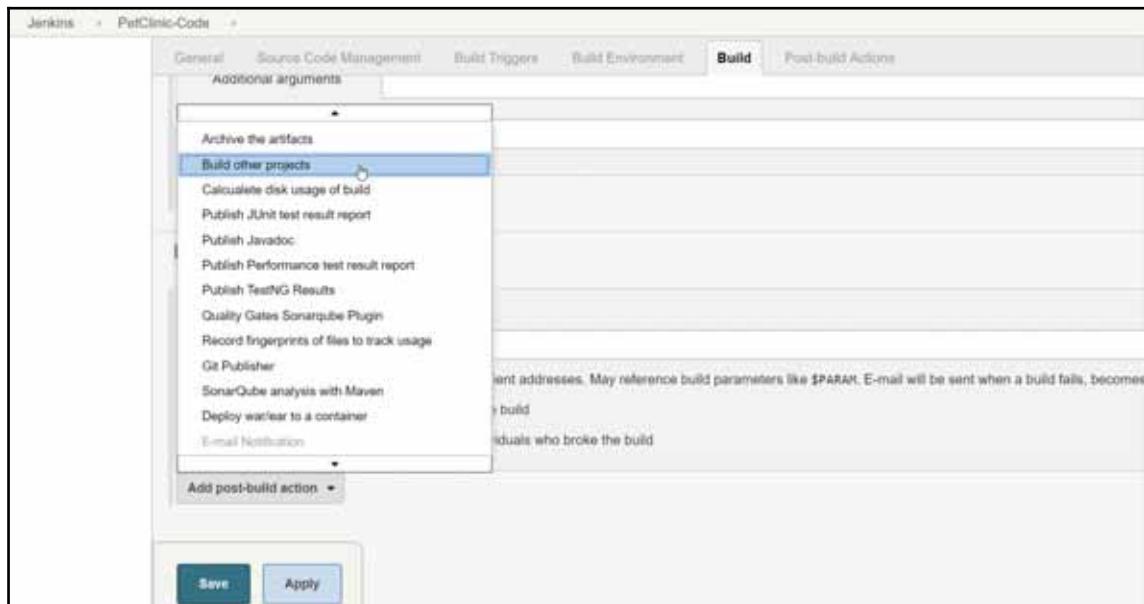
In the next section, we will configure upstream and downstream relationships between all the jobs we have created so far.

Upstream and downstream jobs

An upstream job is a configured project that triggers a project as part of its execution. A downstream job is a configured project that is triggered as part of the execution of the pipeline.

Let's start with the first job we created for static code analysis. Go to the **Post-build Action** section in the configuration of the **PetClinic-Code** build job.

1. Select **Build other projects** from the available options:

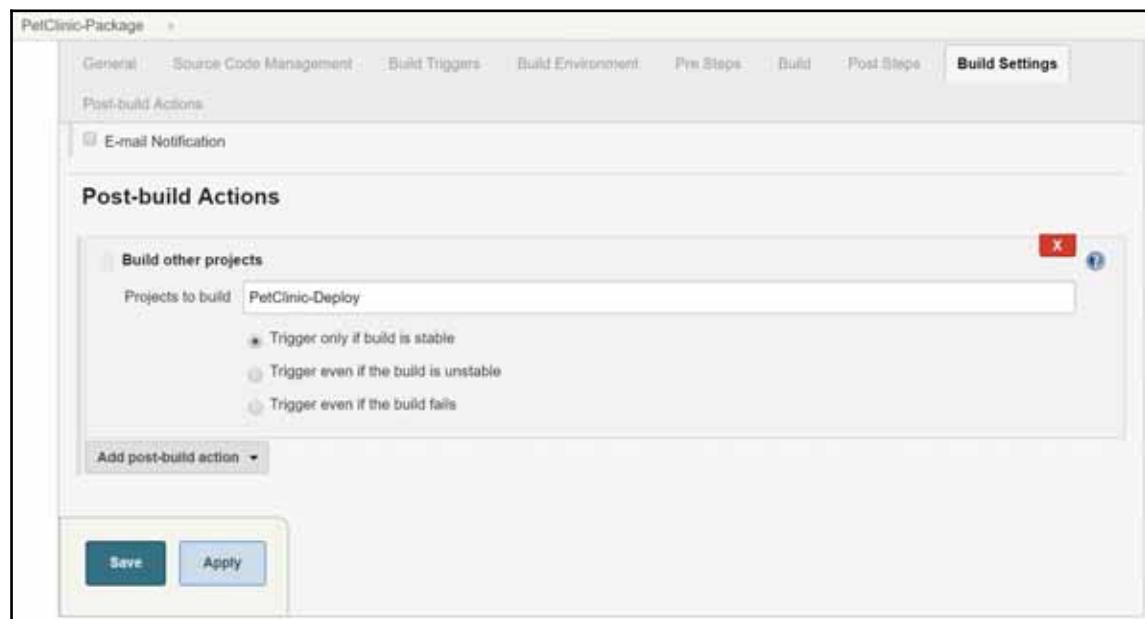


2. We would like to create a package after static code analysis is done, so we will select **PetClinic-Package** where CI is configured for the compilation of source code, unit test execution, and package file creation.
3. Click **Save**.

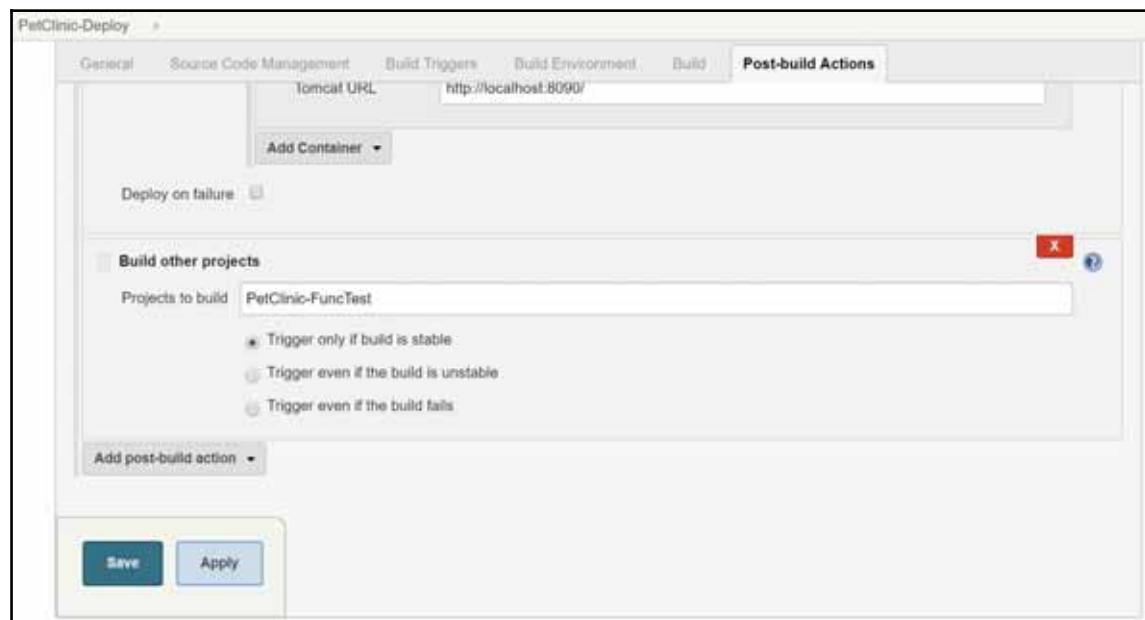
4. For **PetClinic-Code**, **PetClinic-Package** is a downstream job, while for **PetClinic-Package**, **PetClinic-Code** is an upstream job:

The screenshot shows the Jenkins configuration page for the 'PetClinic-Code' job. The top navigation bar includes tabs for General, Source Code Management, Build Triggers, Build Environment, Build, and Post-build Actions. The 'Post-build Actions' tab is selected. The configuration is divided into sections: 'E-mail Notification' (with a recipient redacted to [REDACTED]@gmail.com), 'Build other projects' (with 'PetClinic-Package' selected and a red box highlighting it), and an 'Add post-build action' dropdown menu. At the bottom are 'Save' and 'Apply' buttons.

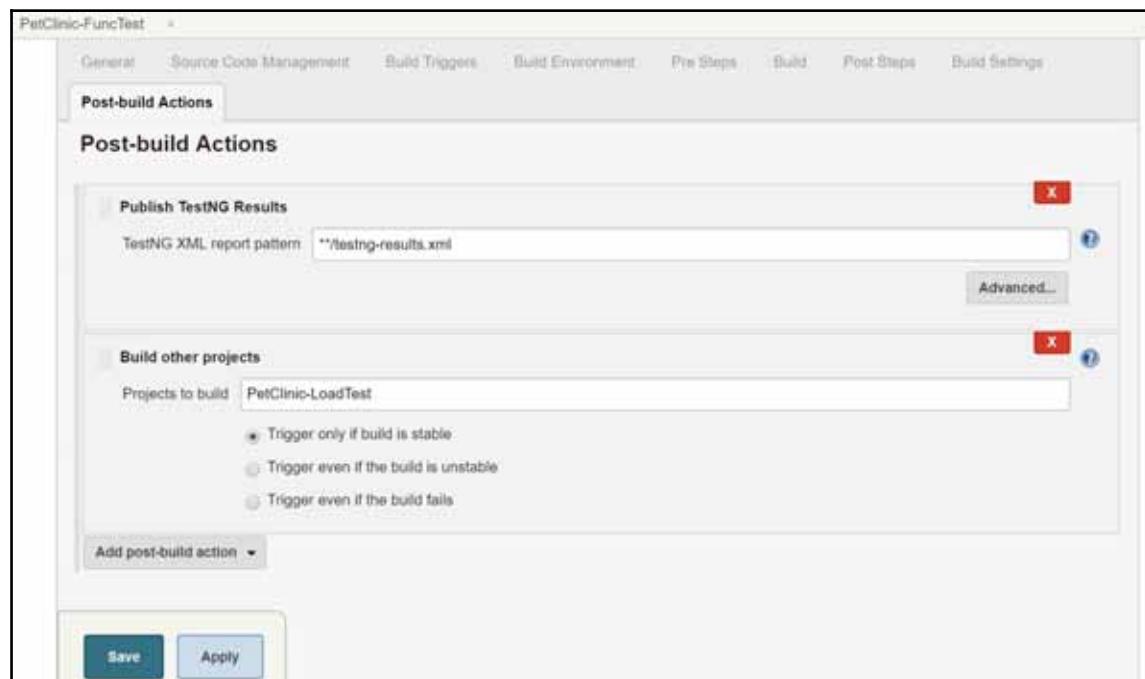
- Once our package is ready, we would like to deploy it, so from the **PetClinic-Package** job, we will configure **PetClinic-Deploy** as a downstream job in **Post-build Actions**:



- Once our application is deployed to the server, we would like to perform functional test cases, so from the PetClinic-Deploy job, we will configure **PetClinic-FuncTest** as a downstream job in **Post-build Actions**:



7. Once the functional test cases are executed successfully, we would like to perform load testing so from the PetClinic-FuncTest job, we will configure **PetClinic-LoadTest** as a downstream job in **Post-build Actions**:



- Once load testing is completed, we would like to deploy the application in the prod environment, so from the **PetClinic-LoadTest** job, we will configure **PetClinic-Prod** as a downstream job in **Post-build Actions**:

The screenshot shows the Jenkins configuration interface for the 'PetClinic-LoadTest' job. The 'Post-build Actions' tab is active. In the 'Build other projects' section, 'PetClinic-Prod' is listed under 'Projects to build'. The 'Trigger only if build is stable' radio button is selected. There are three other options: 'Trigger even if the build is unstable' and 'Trigger even if the build fails', both of which are unselected. At the bottom of the configuration panel are two buttons: 'Save' and 'Apply'.

- Install the plugin from **Manage Jenkins | Manage Plugins**:

The screenshot shows the Jenkins 'Manage Plugins' page. The 'Available' tab is selected. A search bar at the top right contains the text 'Build Pip'. Below the search bar, a table lists the 'Build Pipeline Plugin'. The table has three columns: 'Name', 'Version', and 'Description'. The 'Name' column shows 'Build Pipeline Plugin', the 'Version' column shows '1.5.6', and the 'Description' column contains a brief description of the plugin's functionality. At the bottom of the page are several buttons: 'Install without restart', 'Download now and install after restart', 'Check now', and a status message 'Update information obtained: 17 hr ago'.

10. Verify the successful installation of the Build Pipeline plugin:

Installing Plugins/Upgrades

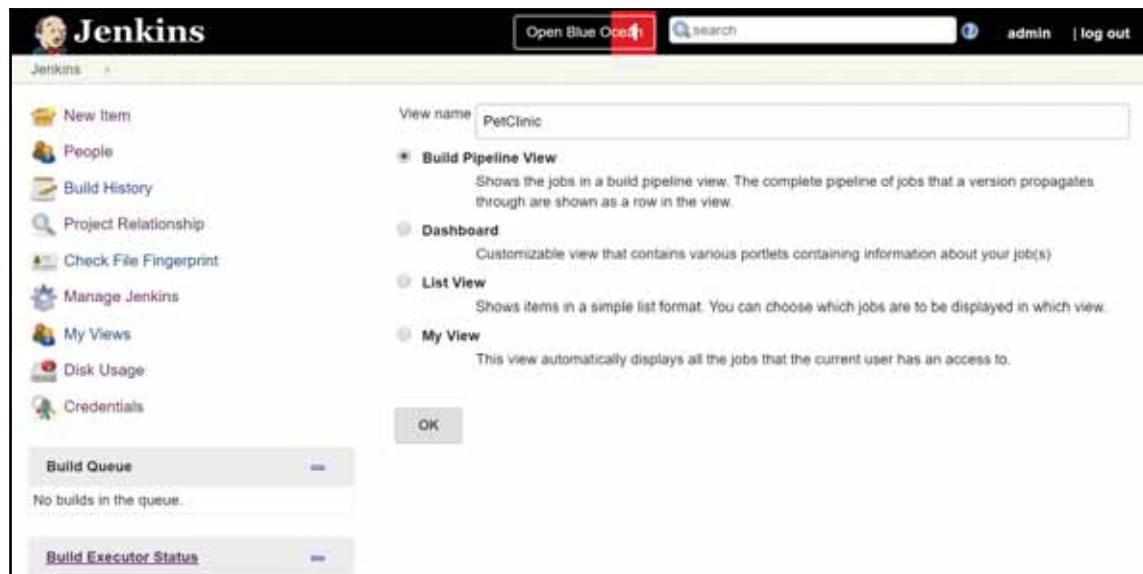
<p>Preparation</p> <ul style="list-style-type: none">• Checking internet connectivity• Checking update center connectivity• Success	<p>TestNG Results Plugin Success</p> <p>promoted builds plugin Success</p> <p>Run Condition Plugin Success</p> <p>Conditional BuildStep Success</p> <p>Parameterized Trigger plugin Success</p> <p>Build Pipeline Plugin Success</p>
---	--

[Go back to the top page](#)
(you can start using the installed plugins right away)

Restart Jenkins when installation is complete and no jobs are running

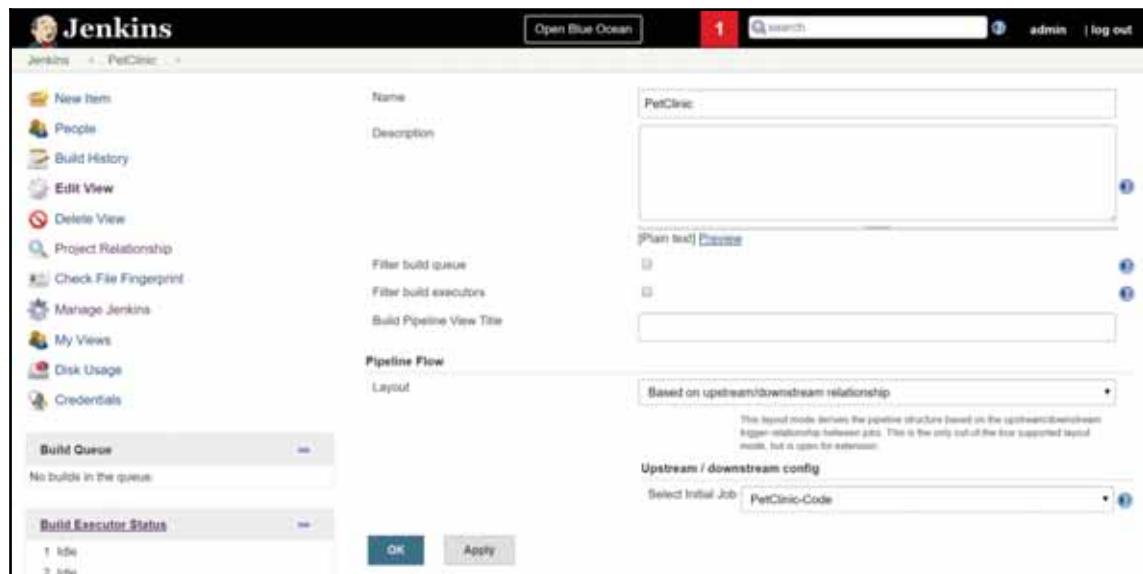
11. Go to the Jenkins dashboard and click on the plus sign on the tabs available.
12. Provide **View name** and select **Build Pipeline View**.

13. Click on **Save**:

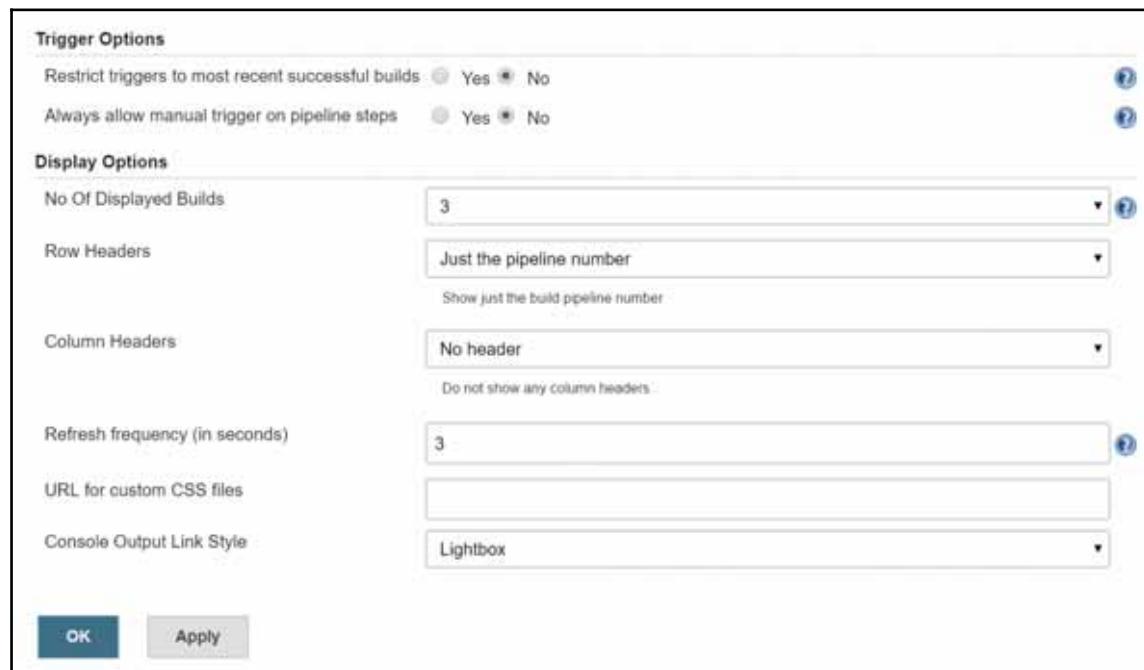


14. Verify the layout is configured as **Based on upstream/downstream relationship**.

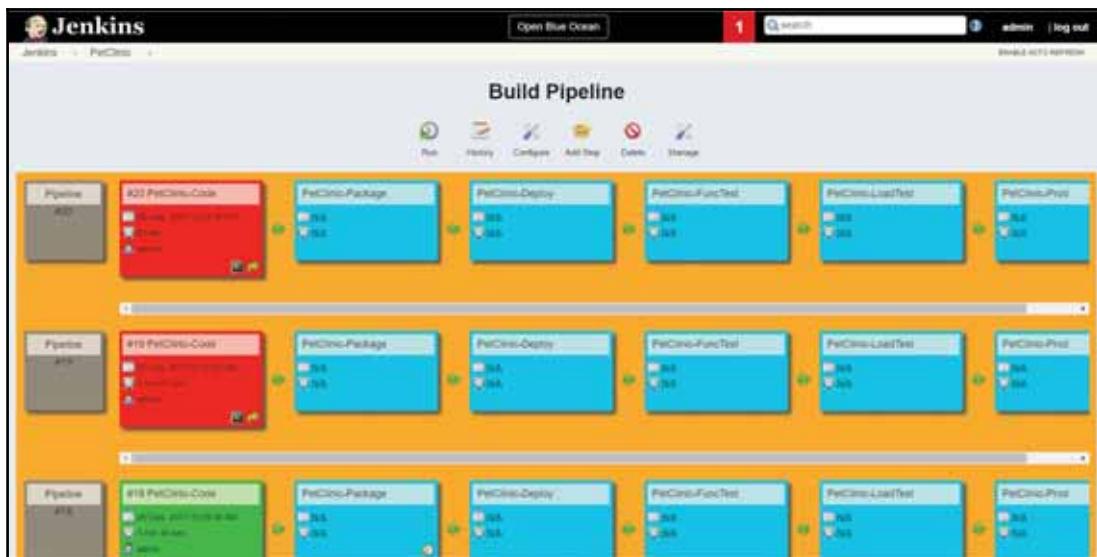
15. We want to execute **PetClinic-Code** as a first job, so select it in **Select Initial Job**:



16. Select the rest of the configuration as per the requirement.
17. Change the number of displayed builds from **1** to **3** so it will display the last three build pipelines executed.
18. Click on **OK**:

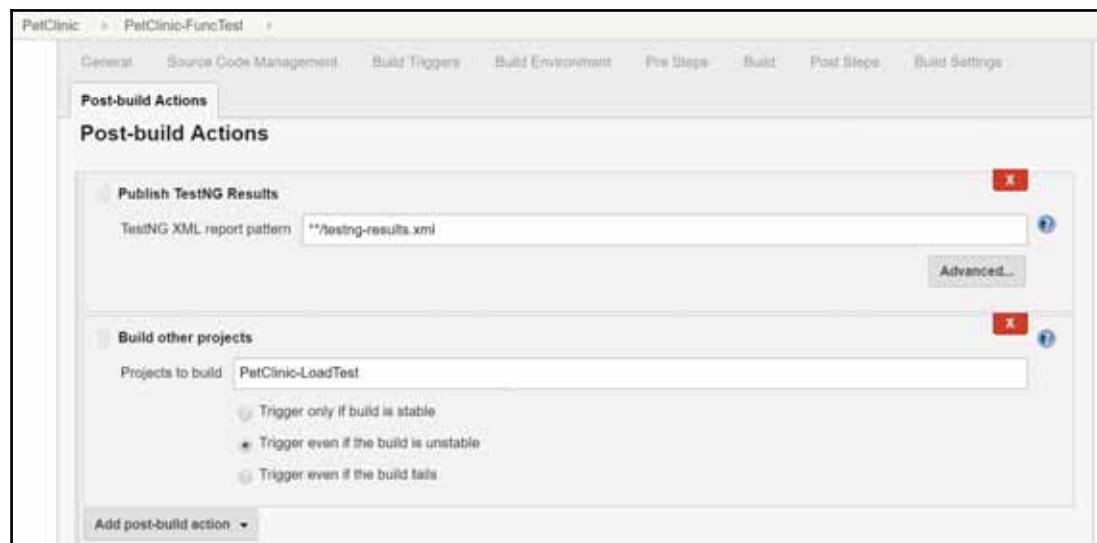


19. Check the **Build Pipeline** view in the Jenkins dashboard:

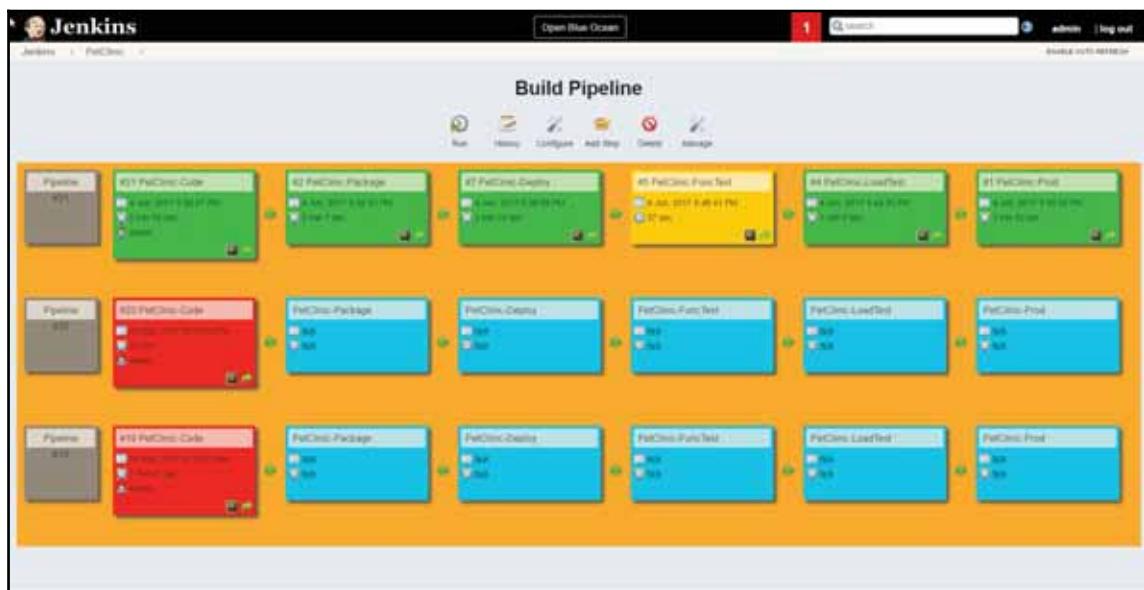


Build Pipeline with configured upstream and downstream job

20. What if we want to continue execution of the pipeline even if a build is unstable? In such a case, we need to select the option **Trigger even if the build is unstable** in **Post-build Actions**, as shown in the following screenshot:



21. Click on **Run** in the **Build Pipeline** view. Wait until the complete pipeline has been executed:



Successful Execution of Build Pipeline with configured upstream and downstream job

Done. This is how we can create a sequence of execution for different build jobs and achieve end-to-end automation for application lifecycle management. In the next section, we will achieve similar things using the Pipeline as a Code feature.

Overview of pipeline as a code

The Jenkins Pipeline feature supports Continuous Delivery pipelines and Continuous Deployment into Jenkins using Pipeline DSL. In Pipeline, we model all related tasks to decide the sequence of execution. We will perform the same tasks we performed with the Build Pipeline plugin.

Pipeline as a code - implementation

Blue Ocean is a new user interface for Jenkins. The idea behind introducing Blue Ocean is to make Jenkins and Continuous Delivery approachable to all team members. We will use Blue Ocean later in the chapter, but we will install it now:

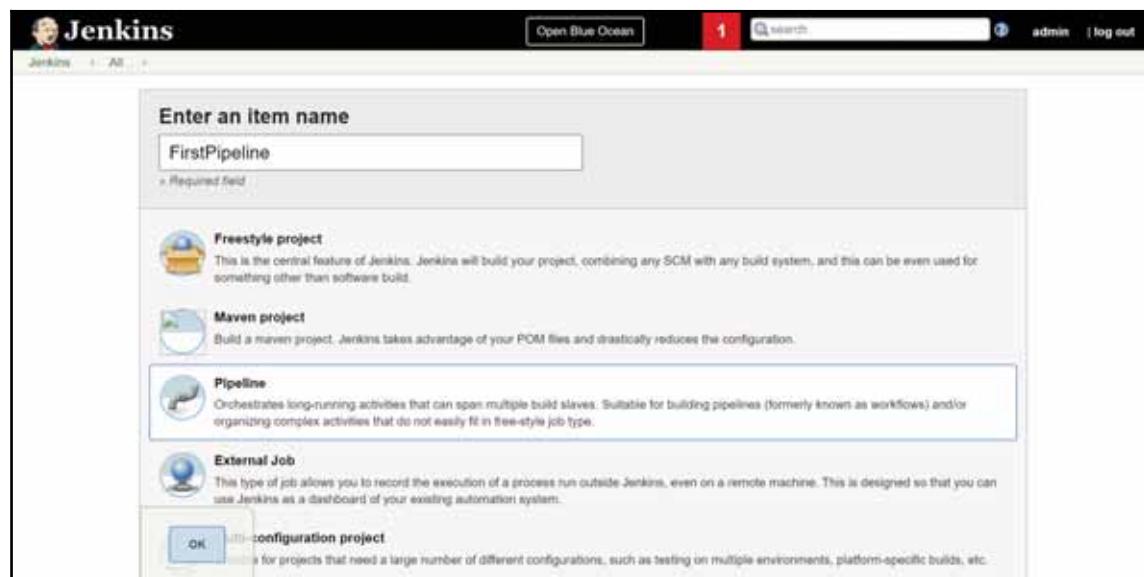
The screenshot shows the Jenkins plugin management interface. The top navigation bar has tabs for 'Updates', 'Available' (which is selected and highlighted in grey), 'Installed', and 'Advanced'. Below the tabs is a search bar with the placeholder 'Install' and a dropdown arrow. A table lists the available plugins:

Install ↓	Name	Version
<input checked="" type="checkbox"/>	Blue Ocean	1.0.1
<input type="checkbox"/>	Common API for Blue Ocean	1.0.1
<input type="checkbox"/>	Config API for Blue Ocean	1.0.1
<input type="checkbox"/>	Dashboard for Blue Ocean	1.0.1
<input type="checkbox"/>	Events API for Blue Ocean	1.0.1
<input type="checkbox"/>	Git Pipeline for Blue Ocean	1.0.1
<input type="checkbox"/>	GitHub Pipeline for Blue Ocean	1.0.1
<input type="checkbox"/>	i18n for Blue Ocean	1.0.1
<input type="checkbox"/>	JWT for Blue Ocean	1.0.1
<input type="checkbox"/>	Personalization for Blue Ocean	1.0.1

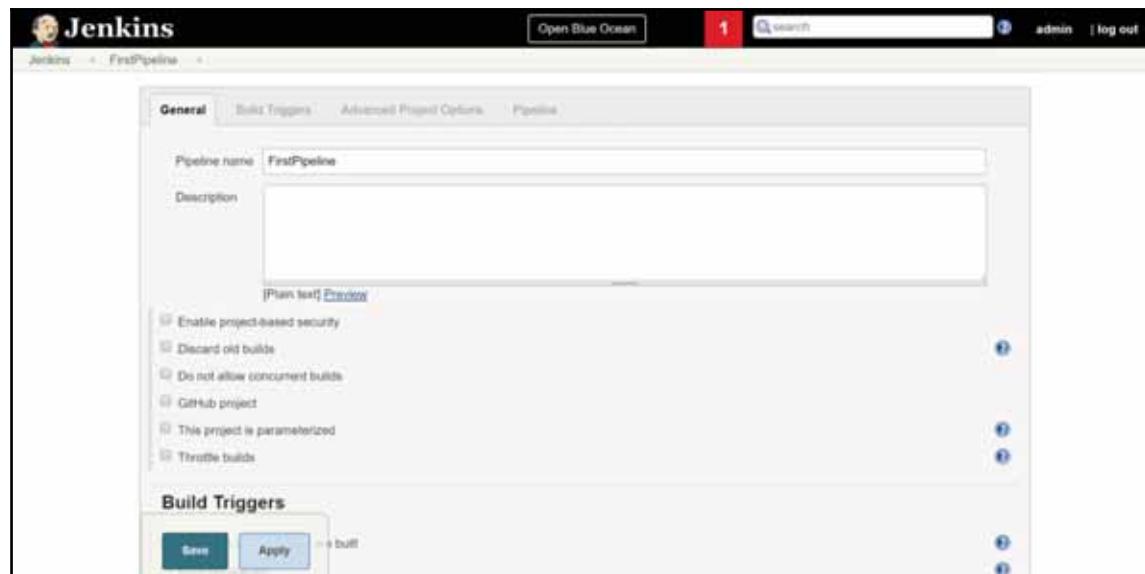
1. Verify the successful installation of the plugin in the Jenkins plugin section:

Common API for Blue Ocean	 Success
Variant Plugin	 Success
Jackson 2 API Plugin	 Success
Metrics Plugin	 Success
Web for Blue Ocean	 Success
REST API for Blue Ocean	 Success
JWT for Blue Ocean	 Success
Favorite	 Success
REST Implementation for Blue Ocean	 Success
Pipeline REST API for Blue Ocean	 Success
Pub-Sub "light" Bus	 Success
GitHub Pipeline for Blue Ocean	 Success
Git Pipeline for Blue Ocean	 Success
Config API for Blue Ocean	 Success
Server Sent Events (SSE) Gateway Plugin	 Success
Events API for Blue Ocean	 Success
Personalization for Blue Ocean	 Success
BlueOcean Display URL plugin	 Success
Blue Ocean Pipeline Editor	 Success
Autofavorite for Blue Ocean	 Success
i18n for Blue Ocean	 Success
Dashboard for Blue Ocean	 Success
Blue Ocean	 Success

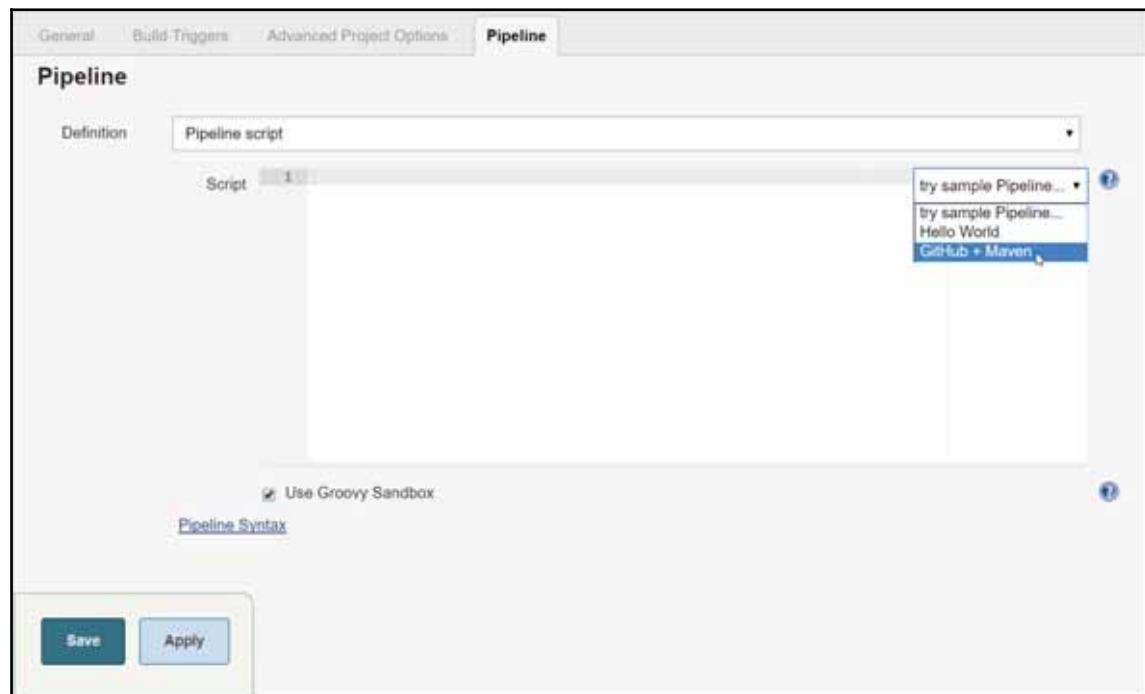
2. Now we will create our first pipeline in Jenkins.
3. Click on **New Item**. Enter an item name and select **Pipeline**.
4. Click **OK**:



5. This will open the configuration of a newly created pipeline job:



6. Go to the **Pipeline** section:



- In the try sample **Pipeline** dropdown, select **GitHub + Maven**. It will automatically generate the syntax for the sample code. Make sure that `mvnHome` has a proper value, as per the path given for Maven in your system:

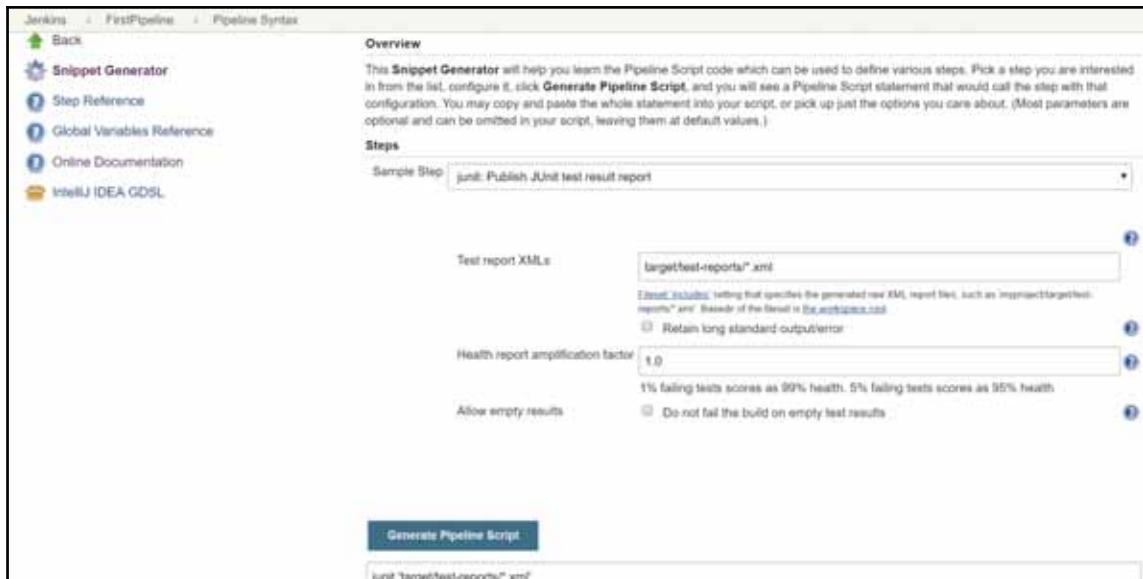
The screenshot shows the Jenkins Pipeline configuration interface. At the top, there are tabs: General, Build Triggers, Advanced Project Options, Pipeline, and Pipeline Syntax. The Pipeline tab is selected. Below it, the Pipeline section is titled "Pipeline". Under "Definition", there is a "Pipeline script" input field containing Groovy code for a GitHub + Maven pipeline. The code defines a node, stages a preparation step to clone a GitHub repository, and then runs a build stage with Maven commands. A "GitHub + Maven" dropdown is shown next to the script input. Below the script, there is a checkbox for "Use Groovy Sandbox" and a link to "Pipeline Syntax". At the bottom, there are "Save" and "Apply" buttons.

```

1+ node {
2+   def mvnHome
3+   stage('Preparation') { // for display purposes
4+     // Get some code from a Github repository
5+     git "https://github.com/jglick/simple-maven-project-with-tests.git"
6+     // Get the Maven tool
7+     // ** NOTE: This 'M3' Maven tool must be configured
8+     // ** in the global configuration.
9+     mvnHome = tool 'M3'
10+  }
11+  stage('Build') {
12+    // Run the maven build
13+    if (isUnix()) {
14+      sh "${mvnHome}/bin/mvn -Dmaven.test.failure.ignore clean package"
15+    } else {
16+      bat("${mvnHome}\bin\mvn" -Dmaven.test.failure.ignore clean package)
17+    }
18+  }
}

```

- Click on **Save** and **Execute** the build to verify it.
- However, we will create our own pipeline with the same sequence we tried with Build Pipeline.
- Click on **Pipeline syntax** to generate the syntax for specific tasks we want to execute.
- We can select the steps and configure the required things, and then click on **Generate Pipeline Script** to get the syntax that we can directly utilize in our pipeline:



12. Before creating our first script for a pipeline, let's understand some important terms:

- **Node** defines the node created in the context of Jenkins' master/agent architecture. It executes the step the moment the executor is available on the node. It creates a workspace or a directory for the pipeline to keep files. The following is the sample syntax:

```
node { // execute the pipeline on Master node } node('windows') {
    // execute the pipeline on node labelled as Windows }
```

- **Stage** is a step that can be considered as a logically separate step such as init, build, test, deploy, and so on.
- **Step or a Build Step** is a task that can be executed to perform some activity such as copy artifact, archiveartifact, check out code from GitHub, and define environment variable.

13. Here is the script for Pipeline:

```
node {  
    def mvnHome  
    stage('Preparation') { // for display purposes  
        // Get PetClinic code from a GitHub repository  
        git 'https://github.com/mitesh51/spring-  
petclinic.git'  
        // Get the Maven tool.  
        // ** NOTE: This 'apache-maven-3.3.1' Maven tool must be  
        // configured in the global configuration.  
        mvnHome = tool 'apache-maven-3.3.1'  
    }  
    stage('SonarQube analysis') {  
        // requires SonarQube Scanner 3.0+  
        def scannerHome = tool 'SonarQube Scanner 3.0.3';  
        // Sonarqube6.3 must be configured in the Jenkins  
        Configuration -> Add SonarWube server  
        withSonarQubeEnv('Sonarqube6.3') {  
            //provide all required properties for Sonar execution  
            bat "${scannerHome}/bin/sonar-scanner -  
Dsonar.host.url=http://localhost:9000/ -  
Dsonar.login=1335c62cbfceab5  
954a5101ab7477cc974f58d56 -  
Dsonar.projectVersion=1.0  
-Dsonar.projectKey=petclinicKey -  
Dsonar.sources=src"  
        }  
    } stage('Build') {  
        // Run the maven build based on the Operating system  
        if (isUnix()) {  
sh "'${mvnHome}/bin/mvn' -  
Dmaven.test.failure.ignore clean package"  
        // Publish JUnit Report  
        junit '**/target/surefire-reports/TEST-*.xml'  
        } else {  
            bat("${mvnHome}\bin\mvn" clean package)  
        // Publish JUnit Report  
        junit '**/target/surefire-reports/TEST-*.xml'  
        }  
    }  
    stage('Deploy') {  
        // Archive the artifact  
        archive 'target/*.war'  
        // Execute the PetClinic-Deploy build to deploy war file into  
        tomcat  
        // Copy Artifact from this Pipeline Project into PetClinic-
```

```
Deploy using Copy Artifact plugin
    build 'PetClinic-Deploy'
}
stage('Functional Test'){
// Checkout the code from Github to execute Functional test
git 'https://github.com/mitesh51/petclinic-func.git'
// Go to GitHub Directory and Fork it ... Change the URL in
petclinic-func/src/test/java/example/NewTest.java
//driver.get("http://localhost:8090/petclinic/");
//In the same file Change location of Gecko driver, we have used
Firefox here on Windows...File file = new
File("C:\\Users\\Mitesh\\Downloads\\geckodriver-v0.13.0-
win64\\geckodriver.exe");
// Run the maven build with test goal to execute functional test
bat("${mvnHome}\\bin\\mvn" test) }
// This stage can be optional based on the requirements stage('Load
Test'){
// Execute command to perform load testing with the use of Apache
JMeter; In our case we are using JMeter that is already installed on
Windows hence the bat file is used. Make sure to change this
location based
on the Apache JMeter installation directory available in your
system.
bat "C:/apache-jmeter-3.0/bin/jmeter.bat -
Jjmeter.save.saveservice.output_format=xml -n -t
C:/Users/Mitesh/Desktop/PetClinic.jmx -l Test.jtl"
// Publish Apache JMeter results
        perfReport errorFailedThreshold: 50,
errorUnstableThreshold: 30, ignoreFailedBuilds:
true, ignoreUnstableBuilds: true,
persistConstraintLog: true, sourceDataFiles:
'Test.jtl'
}
//Done!
}
```

14. Click on **Build Now** for pipeline execution:

Build	Date
#20	Jun 4, 2017 11:58 AM
#19	Jun 4, 2017 11:41 AM
#18	Jun 4, 2017 11:38 AM

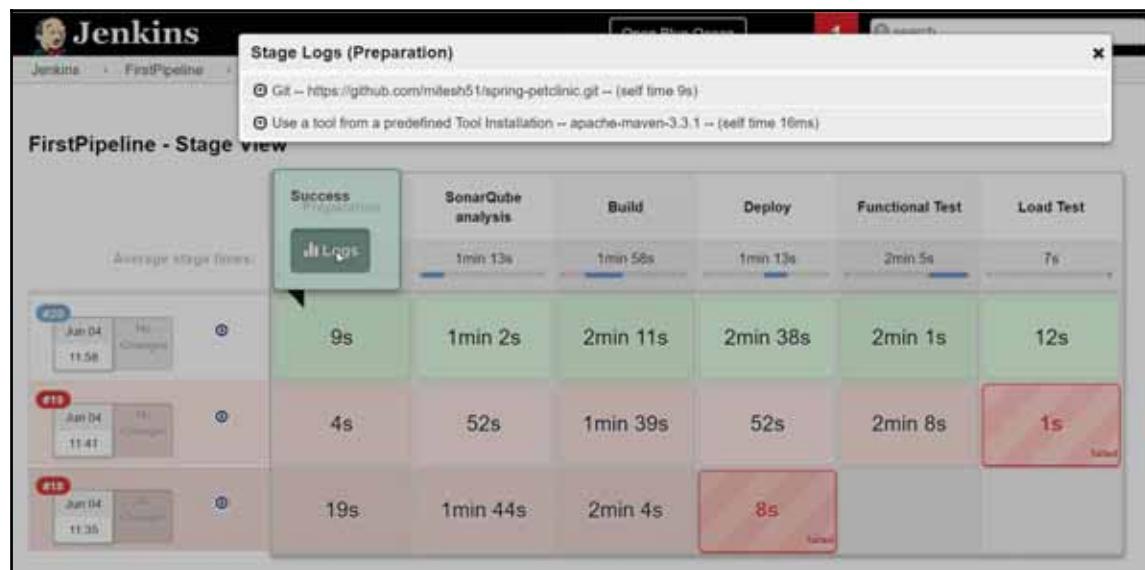
15. Verify the stage view of the pipeline we have created by clicking on **Full Stage View** on the Jenkins dashboard:

	Preparation	SonarQube analysis	Build	Deploy	Functional Test	Load Test
#20 Jun 4 11:58	9s	1min 2s	2min 11s	2min 38s	2min 1s	12s
#19 Jun 4 11:41	4s	52s	1min 39s	52s	2min 8s	1s Failed
#18 Jun 4 11:38	19s	1min 44s	2min 4s	8s Failed		

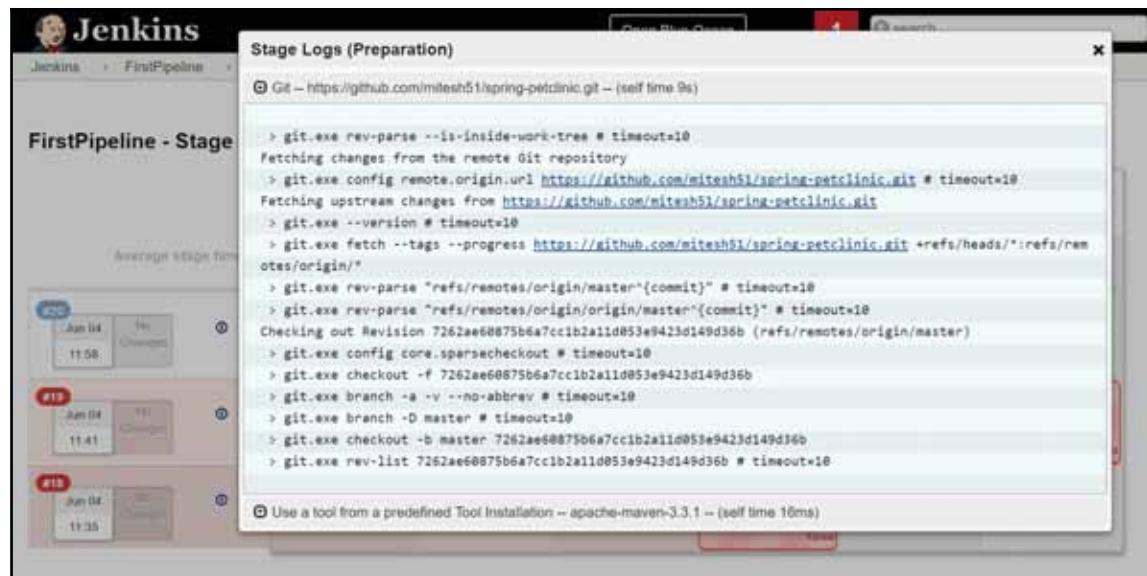
16. Mouse over the specific stage and click on Logs:



17. We can see and verify the stage logs directly from the stage view:



18. Click on the dropdown to get more details on the log:



19. Let's go to individual stage logs in the Console output:

1. Look at the log for **Preparation Stage**:

 **Console Output**

```
Started by user admin
[Pipeline] node
Running on master in F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Preparation)
[Pipeline] git
> git.exe rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git.exe config remote.origin.url https://github.com/mitesh51/spring-petclinic.git # timeout=10
Fetching upstream changes from https://github.com/mitesh51/spring-petclinic.git
> git.exe --version # timeout=10
> git.exe fetch --tags --progress https://github.com/mitesh51/spring-petclinic.git
+refs/heads/*:refs/remotes/origin/*
> git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
> git.exe rev-parse "refs/remotes/origin/origin/master^{commit}" # timeout=10
Checking out Revision 7262ae60875b6a7cc1b2a11d053e9423d149d36b (refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f 7262ae60875b6a7cc1b2a11d053e9423d149d36b
> git.exe branch -a -v --no-abbrev # timeout=10
> git.exe branch -D master # timeout=10
> git.exe checkout -b master 7262ae60875b6a7cc1b2a11d053e9423d149d36b
> git.exe rev-list 7262ae60875b6a7cc1b2a11d053e9423d149d36b # timeout=10
[Pipeline] tool
```

2. Look at the log for Sonarqube analysis stage:

```
[Pipeline] stage
[Pipeline] { (SonarQube analysis)
[Pipeline] tool
[Pipeline] wrap
Injecting SonarQube environment variables using the configuration: Sonarqube6.3
[Pipeline] {
[Pipeline] bat
[FirstPipeline] Running batch script

F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline>F:\#JenkinsEssentials\FirstDraft\jenkinsHome\tools\hudson.plugins.sonar.SonarRunnerInstallation\SonarQube_Scanner_3.0.3\bin\sonar-scanner -
Dsonar.host.url=http://localhost:9000/ -Dsonar.login=***** -Dsonar.projectVersion=1.0 -
Dsonar.projectKey=petclinicKey -Dsonar.sources=src
INFO: Scanner configuration file:
F:\#JenkinsEssentials\FirstDraft\jenkinsHome\tools\hudson.plugins.sonar.SonarRunnerInstallation\SonarQube_Scanner_3.0.3\bin\..\conf\sonar-scanner.properties
INFO: Project root configuration file: F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline\sonar-project.properties
INFO: SonarQube Scanner 3.0.3.778
INFO: Java 1.8.0_111 Oracle Corporation (64-bit)
INFO: Windows 10 10.0 amd64
INFO: User cache: C:\Users\Mitesh\.sonar\cache
INFO: Load global settings
INFO: Load global settings (done) | time=3968ms
INFO: User cache: C:\Users\Mitesh\.sonar\cache
INFO: Load plugins index
INFO: Load plugins index (done) | time=138ms
INFO: SonarQube server 6.3.1
```

3. Look at the log for Build Stage:

```
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] isUnix
[Pipeline] bat
[FirstPipeline] Running batch script

F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline>"C:\apache-maven-3.3.1\bin\mvn" clean package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building petclinic 4.2.5-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ spring-petclinic ---
[INFO] Deleting F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline\target
[INFO]
[INFO] --- cobertura-maven-plugin:2.7:clean (default) @ spring-petclinic ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ spring-petclinic ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 18 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8:compile (default-compile) @ spring-petclinic ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 45 source files to
F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline\target\classes
[parsing started
RegularFileObject[F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline\src\main\java\org\springfram
ework\samples\petclinic\web\PetValidator.java]]
[parsing completed 219ms]
```

4. Look at the log for Deploy and Functional Test Stage:

```
[INFO] Building war: F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline\target\petclinic.war
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 02:03 min
[INFO] Finished at: 2017-06-04T12:01:43+05:30
[INFO] Final Memory: 27M/88M
[INFO]
[INFO] -----
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] step
Recording test results
[Pipeline] archive
[Pipeline] build (Building PetClinic-Deploy)
Scheduling project: PetClinic-Deploy
Starting building: PetClinic-Deploy #6
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Functional Test)
[Pipeline] git
> git.exe rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git.exe config remote.origin.url https://github.com/mitesh51/petclinic-func.git # timeout=10
Fetching upstream changes from https://github.com/mitesh51/petclinic-func.git
> git.exe --version # timeout=10
> git.exe fetch --tags --progress https://github.com/mitesh51/petclinic-func.git
+refs/heads/*:refs/remotes/origin/*
```

5. Look at the log for Load Test Stage:

```
Results :  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 01:42 min  
[INFO] Finished at: 2017-06-04T12:06:24+05:30  
[INFO] Final Memory: 18M/80M  
[INFO] -----  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (Load Test)  
[Pipeline] bat  
[FirstPipeline] Running batch script  
  
F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline>C:/apache-jmeter-3.0/bin/jmeter.bat -Jjmeter.save.saveservice.output_format=xml -n -t C:/Users/Mitesh/Desktop/PetClinic.jmx -l Test.jtl  
Writing log file to: F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline\jmeter.log  
Creating summariser <summary>  
Created the tree successfully using C:/Users/Mitesh/Desktop/PetClinic.jmx  
Starting the test @ Sun Jun 04 12:06:32 IST 2017 (1496558192833)  
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445  
summary + 1 in 00:00:01 = 1.0/s Avg: 270 Min: 270 Max: 270 Err: 0 (0.00%) Active: 1 Started: 1  
Finished: 0  
summary + 49 in 00:00:01 = 61.1/s Avg: 13 Min: 4 Max: 157 Err: 0 (0.00%) Active: 0 Started: 1  
Finished: 1  
summary = 50 in 00:00:02 = 27.0/s Avg: 18 Min: 4 Max: 270 Err: 0 (0.00%)
```

20. On the project dashboard, look at the stage view at the bottom:



21. Now let's see how our pipeline looks in the Blue Ocean User Interface. Go to the **FirstPipeline** pipeline job that we have created. Click on the **Blue Ocean** link in the top bar on the Jenkins dashboard.

22. Click on successful pipeline 20:

The screenshot shows the Jenkins interface for the 'FirstPipeline' job. At the top, there are tabs for 'Pipelines' and 'Administration'. Below the job name 'FirstPipeline' are buttons for 'Run' (highlighted in blue), 'Activity', 'Branches', and 'Pull Requests'. A table below lists the execution history:

Status	Run	Commit	Message	Duration	Completed
✓	20	7262ae6	-	8m 19s	3 hours ago
✗	19	7262ae6	-	5m 44s	4 hours ago
✗	18	7262ae6	-	4m 18s	4 hours ago

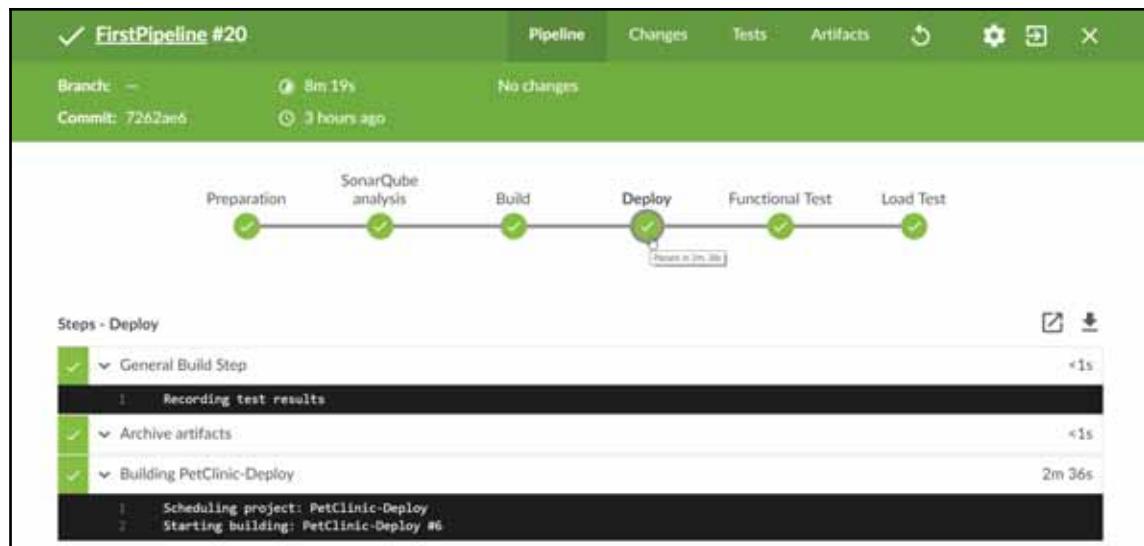
At the bottom of the page, it says '1.0.1 - Core 2.61 - 9b77619 - (no branch) - 12th April 2017 02:47 AM'.

23. It will give details on the execution status of each stage in the **Blue Ocean** dashboard. Logs are available on the same page:

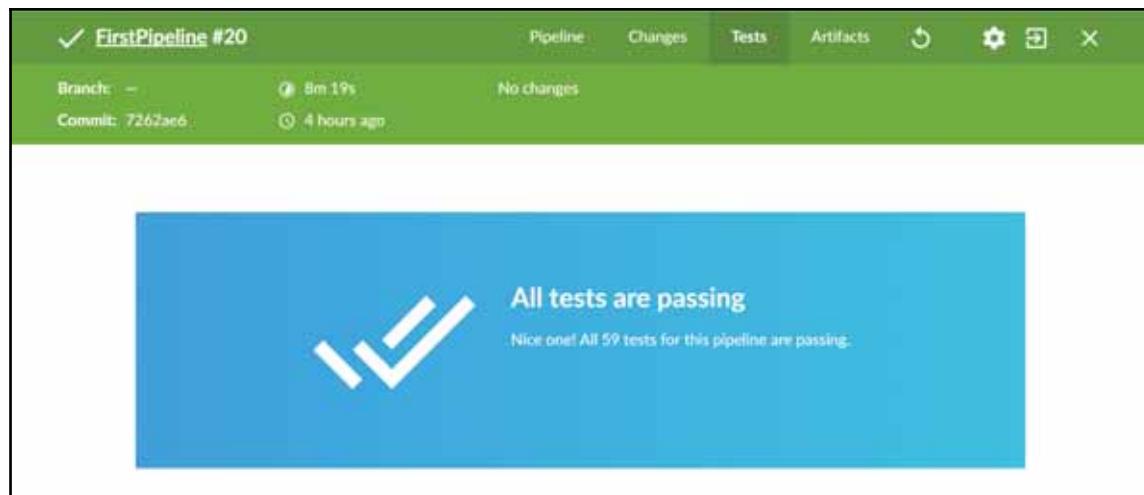
The screenshot shows the Blue Ocean dashboard for the 'FirstPipeline #20' run. The top navigation bar includes 'Pipeline', 'Changes', 'Tests', and 'Artifacts' buttons. Below the timeline, the 'Load Test' step is expanded to show its log output:

```
[FirstPipeline] Running batch script
F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline>C:/apache-jmeter-3.0/bin/jmeter.bat -Jmeter.save.saveservice.output.format=xml -n -t C:/Users/Mitesh/Desktop/PetClinic.jmx -l Test.jtl
Writing log file to: F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstPipeline\jmeter.log
Creating summariser <summary>
Created the tree successfully using C:/Users/Mitesh/Desktop/PetClinic.jmx
Starting the test @ Sun Jun 04 12:06:32 IST 2017 (1496558192833)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +      1 in 00:00:01 =    1.0/s Avg:   270 Min:   270 Max:   270 Err:     0 (0.00%) Active: 1 Started: 1
Finished: 0
```

24. Select any stage and check the logs for the stage on the same page:



25. Click on the **Tests** link on the top bar to verify the status of the Junit test cases executed in the pipeline:



26. Click on the **Artifacts** link on the top bar to verify all the artifacts available in this pipeline:

The screenshot shows the Jenkins Pipeline interface for a build named "FirstPipeline #20". At the top, it displays the branch as "—" and the commit as "7262ae6", with a timestamp of "8m 19s" ago. Below this, under the "Artifacts" tab, there is a table listing three files: "pipeline.log", "dashBoard_Test.xml", and "target/petclinic.war". The "pipeline.log" file is 0 bytes, "dashBoard_Test.xml" is 452 bytes, and "target/petclinic.war" is 40 MB. A "Download All" button is located at the bottom of the artifact list.

In the next section, we will cover one important plugin, the Promoted builds plugin.

Promoted builds

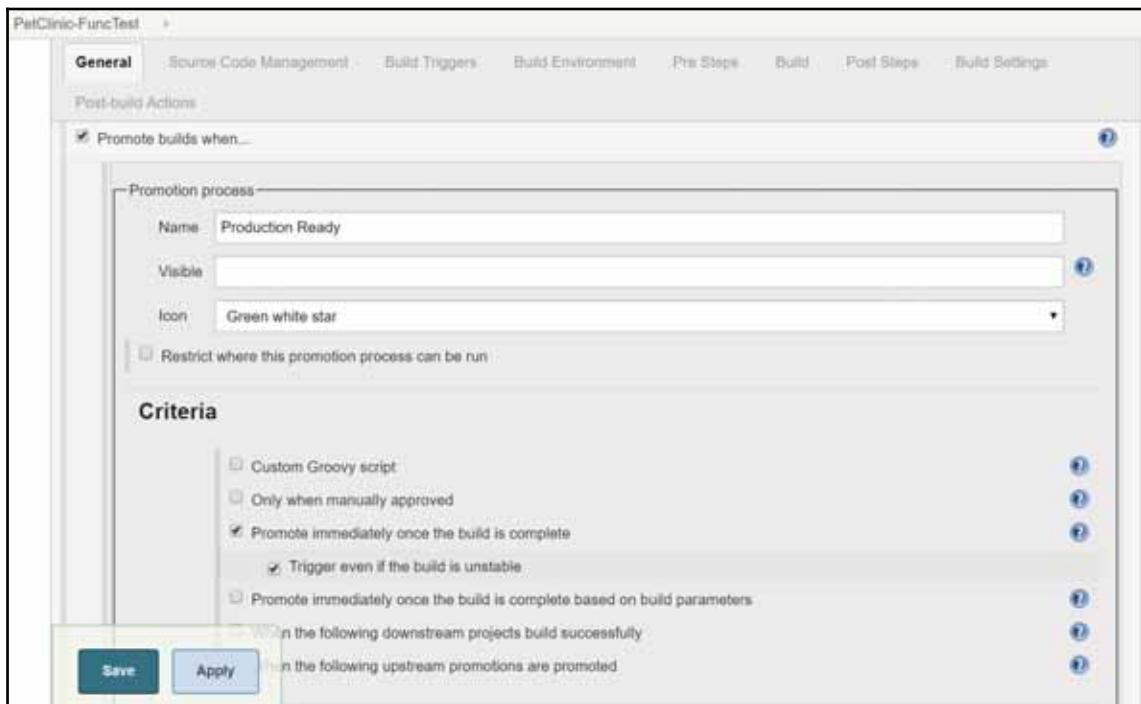
The Promoted builds plugin allows us to tag the builds based on specific stages. This promotion can be manual or automated. We can identify promoted builds based on the star available on the project dashboard or the star available in Build History.

1. Go to **Manage Jenkins** and click on **Manage Plugins**.
2. Select **promoted builds plugin** and click on **Install without restart**:

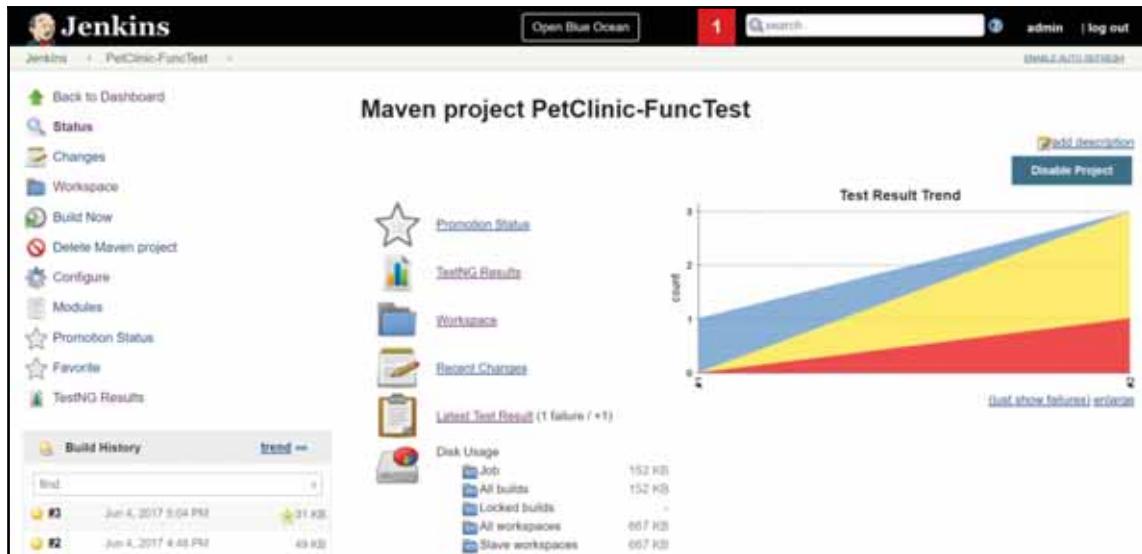
The screenshot shows the Jenkins "Manage Plugins" page. The "Available" tab is selected. A search bar at the top right contains the text "promoted". In the main table, there are two entries: "promoted_builds plugin" (version 2.28.1) and "Promoted Builds (Simple)" (version 1.9). The "promoted_builds plugin" entry has a checked checkbox in the "Install" column. At the bottom of the page, there are three buttons: "Install without restart" (highlighted in blue), "Download now and install after restart", and "Check now". A status message "Update information obtained: 17 hr ago" is displayed between the "Download now" and "Check now" buttons.

3. Go to the **PetClinic-FuncTest** build and open its configuration.
4. In the **General** section, click on **Promote builds when...**

5. Provide a name and select a star you want to associate build if the criteria is passed in the icon list box in the promotion process section.
6. Select **Promote immediately when build is complete**, as shown in following screenshot:



7. We can also select **Only when manually approved** and then we can give the Email ID of the approver.
8. Click on **Build Now** and observe the Jenkins dashboard. Look out for the green star in **Build History** when the build is executed successfully:



The promoted builds feature can be utilized efficiently to assign a quality rating to the outcome of the build, so it can be utilized with confidence.

Summary

We have covered one of the most important concepts in this book; the orchestration of build jobs that performs various important tasks. We have configured end-to-end automation using the Build Pipeline plugin and also using the Pipeline as a Code feature available in Jenkins 2 and later.

We have utilized the Promoted builds plugin to assign quality tags to build jobs as well.

In the next chapter, we will see how to manage and monitor Jenkins and resources efficiently, using features available in Jenkins and also with the use of existing plugins.

8

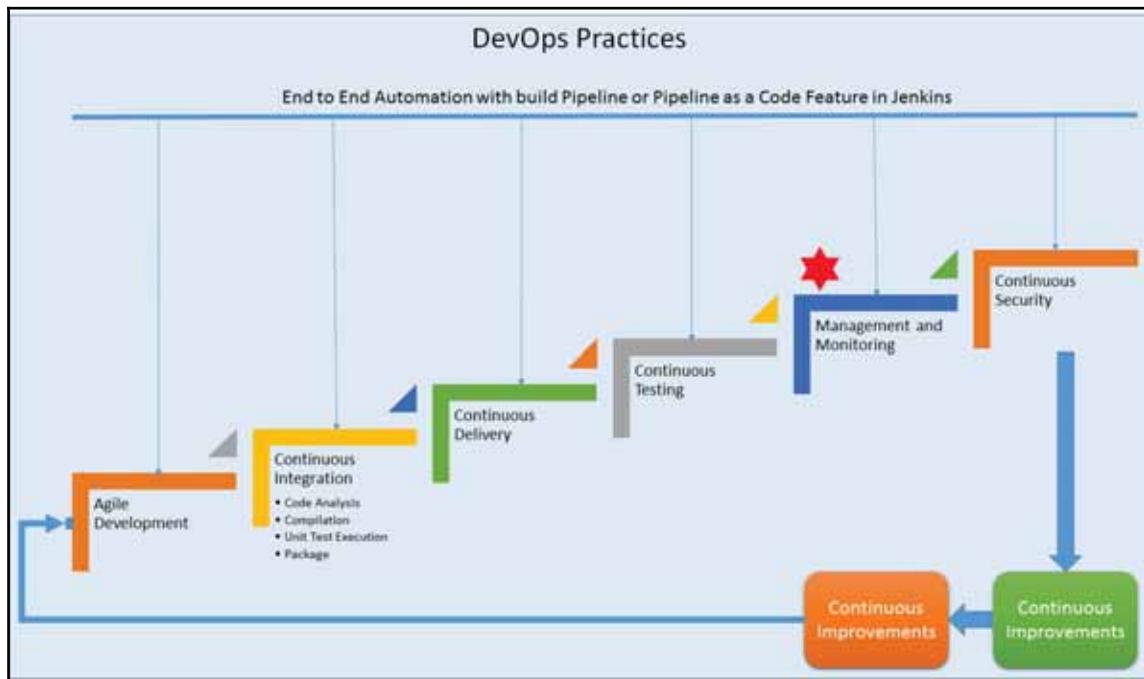
Managing and Monitoring Jenkins

The management and monitoring of Jenkins is essential, as it is at the core of our automation vision. We can utilize existing Jenkins features or plugins to manage and monitor Jenkins and its jobs effectively.

This chapter gives insight into the management of Jenkins nodes and monitoring them with Java melody to provide details on the utilization of resources. It also covers how to monitor build jobs configured for Java or .NET-related applications, and managing those configurations by keeping backups of them. This chapter describes the basic security configuration available in Jenkins in detail, for better access control and authorization. We will cover the following topics in this chapter:

- Managing Jenkins Master and Agent nodes
- Jenkins Monitoring with Java Melody
- Managing job-specific configurations - backup and restore
- Managing disk usage
- Build job-specific monitoring with the Build Monitor plugin
- The Audit Trail plugin- overview and usage
- The Workspace Cleanup plugin
- The pre-scm-build step plugin
- The conditional build step plugin
- The EnvInject plugin

In this chapter, we will cover the management and monitoring of Jenkins as a part of our DevOps journey:



At the end of this chapter, we will know how to configure Agent nodes for distributed architecture and be able to use various other plugins and functionalities to manage and monitor Jenkins effectively.

Managing Jenkins master and slave nodes

Jenkins supports a **Master/Agent** architecture. In a Master/Agent architecture, we can install Jenkins on the master and then utilize other agents for distributing the load.

We can delegate Jenkins jobs to agents for execution. This way we can support multiple executions using different resources.

There are specific scenarios where a Master/Agent architecture is extremely useful, such as the following:

- A Jenkins machine has limited capacity. Even with higher capacity, there will be a time when it can't fulfil all requests. By distributing the load between Agent nodes, we can free system resources where Jenkins is installed.
- Different jobs require different kinds of resources, and they are restricted to specific machines only. In such cases, we can only utilize that machine -- it is not possible to configure it on the Jenkins system -- so it is better to utilize that machine as an agent.
- If different operating systems are required or some tools work only in specific OSes, then we can utilize those tools by making a system agent on which they are installed.
- To avoid a single point of failure caused by installing each and every tool on the Jenkins machine.

The important thing here is we don't install Jenkins on Agent nodes at all. We only install Jenkins on the Master and utilize that Jenkins for orchestrating the Master/Agent architecture.



Just to note, whichever system we install Jenkins on becomes master. Verify that by navigating to **Manage Jenkins | Manage Nodes**.

A screenshot of the Jenkins Manage Nodes page. The left sidebar shows navigation links: Back to Dashboard, Manage Jenkins, New Node, and Configure. Below these are sections for Build Queue (No builds in the queue) and Build Executor Status (1 idle, 2 idle). The main content area displays a table of nodes. The table has columns for \$, Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. One row is shown for the 'master' node, which is running Windows 10 (amd64), is in sync, and has 230.57 GB free disk space, 1.98 GB swap space, 238.77 GB temp space, and a response time of 0ms. A blue 'Rebuild status' button is located at the bottom right of the table.

Navigate to Manage Jenkins | Manage Nodes and click on New Node.

Give the node a name and select Permanent Agent; click OK:

The screenshot shows the Jenkins 'New Node' configuration page. The 'Node name' field contains 'WindowsNode'. The 'Agent Type' dropdown is set to 'Permanent Agent', with a descriptive note below stating: 'Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.' A large 'OK' button is located at the bottom right of the form.

Provide details as required and click on Save:

The screenshot shows the Jenkins 'Edit Node' configuration page for the 'WindowsNode' node. The 'Name' field is filled with 'WindowsNode'. Other configuration fields include 'Description' (empty), '# of executors' (1), 'Remote root directory' (/tmp/Jenkins), 'Labels' (Windows, Test, Android), 'Usage' (Use this node as much as possible), 'Launch method' (Launch agent via execution of command on the master), and 'Launch command' (empty, with a note 'No launch command specified'). The 'Availability' field is set to 'Keep this agent online as much as possible'. At the bottom, there is a 'Node Properties' section with 'Environment variables' and 'Tool Locations' checkboxes, and a prominent blue 'Save' button.

The following table describes all fields available for Agent configuration in detail:

# of executors	Here we can specify the maximum number of concurrent builds that Jenkins may execute on this agent node. Agents must have at least one executor for Job execution, and in case we don't want any execution, then we can configure this setting to 0.
Remote root directory	An agent node requires a directory dedicated to Jenkins. Specify the path to this directory on the agent. Use an absolute path only. All job configurations, build logs, and artifacts are stored on the master. Workspace is available on the Agent Node.
Labels	Labels or tags can be utilized to group multiple Agent Nodes into a logical collection. The best thing is that we can use this mechanism as a pool of resources to execute build jobs in Jenkins. For instance, we have multiple Agents where the test infrastructure is set up. We have two to three projects whose automated testing is done by the QA team. In such a scenario, we can provide the same "Test" label, or tag to all Agent nodes where the test infrastructure is available and then assign the same "Test" label to those projects. It will execute the build jobs on any one of the Test agents with the Test label, but not one without it.
Usage	This setting controls how Jenkins schedules builds on a specific Agent node. Use this node as much as possible: This is the default setting where most of the time this agent node will be utilized for Job execution. Only build jobs with label expressions matching this node: With this setting, Jenkins will execute builds on this agent node when that project (or Jenkins build job) is configured to execute on this node with label expression.
Launch method	This setting controls how Jenkins starts the specific agent node. Launch agent via Java Web Start: This allows an agent to be launched using Java Web Start. Launch the agent via the execution of a command on the master by remotely executing a process on another machine, such as via SSH or RSH. Launch slave agents via SSH by sending commands over a secure SSH connection. Let Jenkins control this Windows slave as a Windows service as it starts a Windows slave by a remote management facility built into Windows.

Availability	This setting controls when Jenkins starts and stops this agent. Keep this agent online as much as possible. Take this agent online and offline at specific times. Take this agent online when in demand, and offline when idle.
Environment variables	We can define Agent node-specific environment variables here.
Tool Locations	We can define Agent node-specific tools locations here.

We can see the newly created agent in the node list as disconnected. Click on it:

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time	
	master	Windows 10 (amd64)	In sync	230.57 GB	1.98 GB	238.77 GB	0ms	
	WindowsNode		N/A	N/A	N/A	N/A	N/A	
	Data obtained		13 ms	2 ms	12 min	12 min	12 min	<button>Refresh status</button>

See the details available on the Agent page. We are not able to start it:



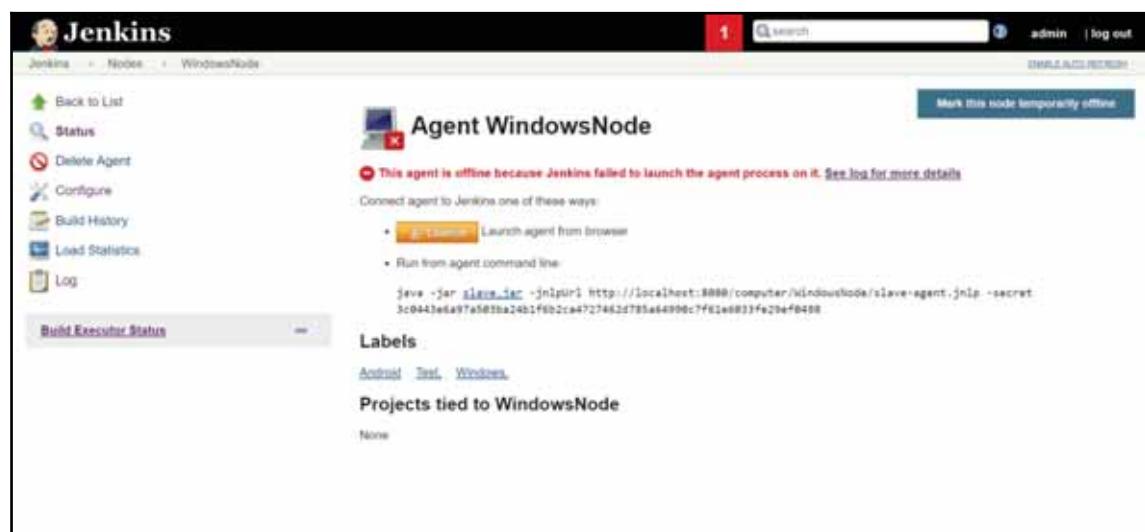
The screenshot shows the Jenkins interface for the 'WindowsNode' agent. On the left, there's a sidebar with links like 'Back to List', 'Status', 'Delete Agent', 'Configure', 'Build History', 'Load Statistics', and 'Log'. The main area is titled 'Agent WindowsNode'. It displays a red error message: 'This agent is offline because Jenkins failed to launch the agent process on it. [See log for more details](#)'. Below this, there's a 'Launch agent' button. To the right, there's a section for 'Labels' with 'Android Test Windows' listed, and another for 'Projects tied to WindowsNode' which says 'None'.

Go to **Manage Jenkins | Configure Global Security | Enable security**. In **JNLP agents setting**, select **Random** and click on **Save**:



The dialog box has a yellow padlock icon and the title "Configure Global Security". It contains a checked checkbox for "Enable security". Below it is a "TCP port for JNLP agents" section with three radio button options: "Fixed" (selected), "Random", and "Disable". A "Agent protocols..." button is below the TCP port settings. At the bottom are "Disable remember me" and a "Save" button.

Go to the Agent node again and execute the given command from the Agent node's terminal or command line; then it should be connected:



The screenshot shows the Jenkins Node Details page for "WindowsNode". The left sidebar includes links for Back to List, Status, Delete Agent, Configure, Build History, Load Statistics, and Log. The main content area displays the "Agent WindowsNode" card, which shows a red error icon and the message "This agent is offline because Jenkins failed to launch the agent process on it. See log for more details". It provides two connection methods: "Launch agent from browser" (with a yellow "Launch" button) and "Run from agent command line" (with a link to the command: `java -jar slave.jar -jnlpUrl http://localhost:8080/computer/windowsnode/slave-agent.jnlp -secret 3c0443e6a97a583ba24b1f6b2ca47274462d785a64990c7f81a883fe29e40488`). Below this are sections for Labels (Android, Test, Windows) and Projects tied to WindowsNode (None).

To configure a job for a specific node, go to **Build job** and click on **Configure**.

In the **General** section, select **Restrict where this project can be run** and provide the label of the Agent node that we have created:

The screenshot shows the Jenkins configuration interface for the 'PetClinic-Code' job. The 'General' tab is active. In the 'Restrict where this project can be run' section, the 'Label Expression' dropdown is set to 'WindowsNode'. A note below states 'Label WindowsNode is serviced by 1 node'. At the bottom, there are 'Save', 'Apply', and 'Advanced...' buttons.

Go to **Manage Nodes** and select the Agent Node to verify the **Build job** associated with it:

The screenshot shows the Jenkins 'Manage Nodes' page. The 'WindowsNode' agent is listed. A message says 'This agent is offline because Jenkins failed to launch the agent process on it.' It provides instructions to 'Launch agent from browser' or 'Run from agent command line' with a provided Java command. Below this, the 'Build Executor Status' table shows one entry for 'PetClinic-Code'. At the bottom, there's a legend for RSS feeds.

S	W	Name	Last Success	Last Failure	Last Duration
●	WindowsNode	PetClinic-Code	1 day 3 hr + 21s	10 hr - 52m	1 min 34 sec

This is how we can create a Master/Agent architecture and distribute the load across agents with only one master available.

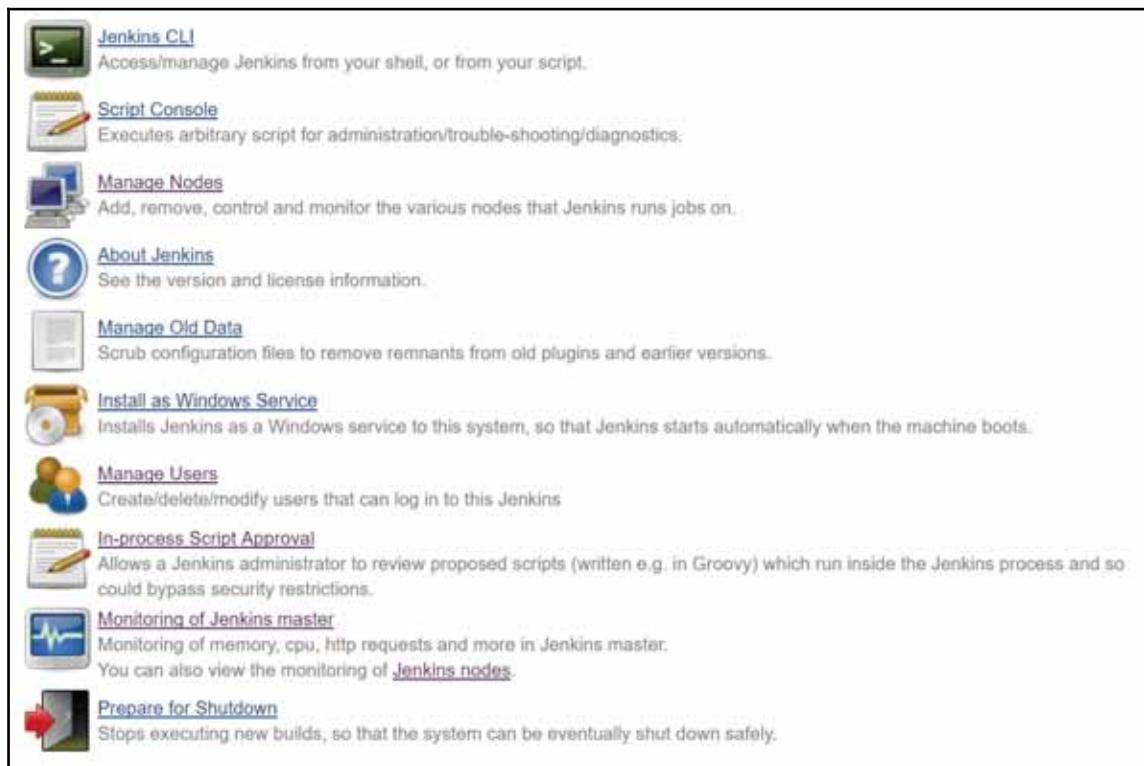
Monitoring Jenkins with JavaMelody

The Monitoring plugin provides monitoring for Jenkins with **JavaMelody**. It provides charts for CPU, memory, system load average, HTTP response time, and so on. It also provides details of HTTP sessions, errors and logs, actions for GC, heap dump, invalidate session(s), and so on. Install the **Monitoring** plugin from the Jenkins dashboard:

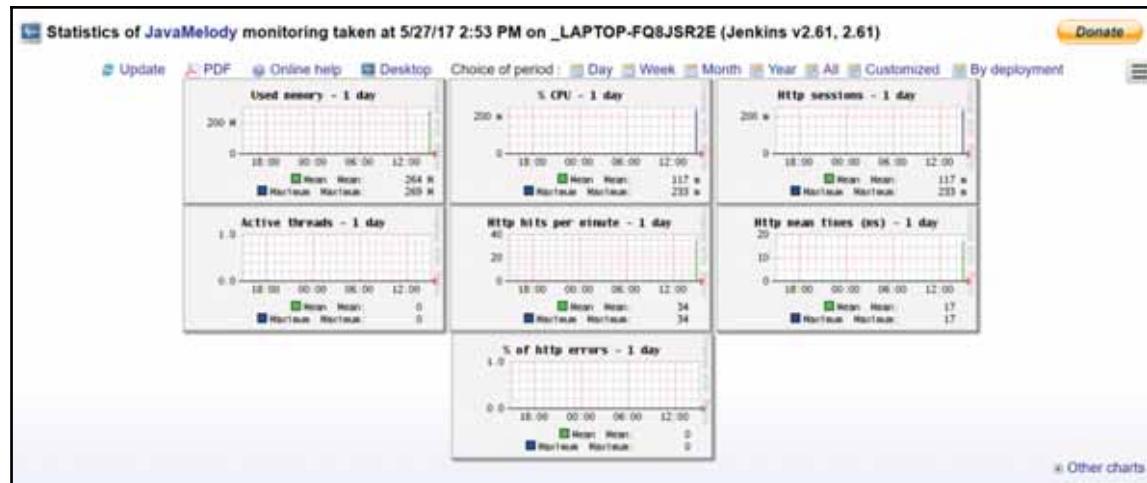
Install	Name	Version
<input type="checkbox"/>	Dynatrace Application Monitoring Plugin	2.0.5
<input checked="" type="checkbox"/>	Monitoring Monitoring of Jenkins	1.67.0
<input type="checkbox"/>	Job/Queue/Staves Monitoring Plugin	1.4

Install without restart Download now and install after restart Update information obtained: 1 day 4 hr ago Check now

1. On the Jenkins dashboard, click on **Manage Jenkins**. Click on **Monitoring of Jenkins master**, as shown in the following screenshot:



2. It will open the statistics for Jenkins instance monitoring, as shown in the following screenshot. Look at all the statistics:



- Click on **Other charts** and see other details related to **Garbage Collector time**, **Threadcount**, and so on:



4. Scroll down the page and find the statistics for the system errors logs. To get more information, click on the **Details** link of any section. HTTP statistics are as shown in the following screenshot:

Statistics http - 1 day

Request	% of cumulative time	Hits	Mean time (ms)	Max time (ms)	Standard deviation	% of cumulative cpu time	Mean cpu time (ms)	% of system error	Mean size (Kb)
http global	100	34	11	126	34	100	7	0.00	1.00
http warning	50	8	20	114	52	24	3	0.00	34
http severe	0	0	1	0	0	0	1	0.00	0

41 hits/min on 18 requests [x Details](#)

Statistics http system errors - 1 day

None

Statistics system errors logs - 1 day

None

Current requests

None

System information

[Execute the garbage collector](#) [Generate a heap dump](#) [View memory histogram](#) [Invalidate http sessions](#) [View http sessions](#)

[View deployment descriptor](#) [MBeans](#) [View OS processes](#) [JNDI tree](#)

Host: LAPTOP-FQ8JSR2E@192.168.99.1
Java memory used: 290 MB / 899 MB
Nb of http sessions: 1
Nb of active threads: 2
(current http requests): 2
% System CPU: 10.07

[x Details](#)

5. Check the details available on **Threads** as well:

Threads

Threads on LAPTOP-FQ8JSR2E@192.168.43.209: Number = 44, Maximum = 51, Total started = 71 [x Details](#)

Thread	Daemon ?	Priority	State	Executed method	Cpu time (ms)	User time (ms)	Kill
Attach Listener	yes	5	RUNNABLE		0	0	red
Computer threadPoolForRemoting [#1]	yes	5	TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	187	109	red
DestroyJavaVM	no	5	RUNNABLE		1,875	1,328	red
FileIoPath.localPool [#1]	yes	5	TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	78	46	red
Finalizer	yes	6	WAITING	java.lang.Object.wait(Native Method)	31	31	red
Handling GET /monitoring from 0:0:0:0:0:1 : RequestHandlerThread[#2]	yes	5	RUNNABLE	java.lang.Thread.dumpThreads(Native Method)	125	109	red
Java2D Disposer	yes	10	WAITING	java.lang.Object.wait(Native Method)	0	0	red
javamelody	yes	5	TIMED_WAITING	java.lang.Object.wait(Native Method)	187	62	red
Jenkins cron thread	no	5	WAITING	java.lang.Object.wait(Native Method)	0	0	red
Jenkins UDP 33848 monitoring thread	no	5	RUNNABLE	java.net.TwoStacksPlainDatagramSocketImpl.receive0(Native Method)	0	0	red
jenkins.util.Timer [#10]	yes	5	WAITING	sun.misc.Unsafe.park(Native Method)	15	15	red
jenkins.util.Timer [#1]	yes	5	WAITING	sun.misc.Unsafe.park(Native Method)	15	15	red
jenkins.util.Timer [#2]	yes	5	TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	15	0	red
jenkins.util.Timer [#3]	yes	5	WAITING	sun.misc.Unsafe.park(Native Method)	31	15	red
jenkins.util.Timer [#4]	yes	5	WAITING	sun.misc.Unsafe.park(Native Method)	0	0	red

In the next section, we will cover details on backing up and restoring JENKINS_HOME.

Managing job-specific configurations - backup and restore

Let's install all the important plugins required for the following sections at once:

Installing Plugins/Upgrades

Preparation

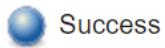
- Checking internet connectivity
- Checking update center connectivity
- Success

Dashboard View



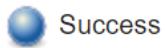
Success

Deploy to container Plugin



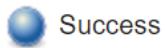
Success

Monitoring



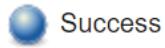
Success

Audit Trail



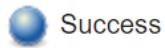
Success

disk-usage plugin



Success

Backup plugin



Success



[Go back to the top page](#)

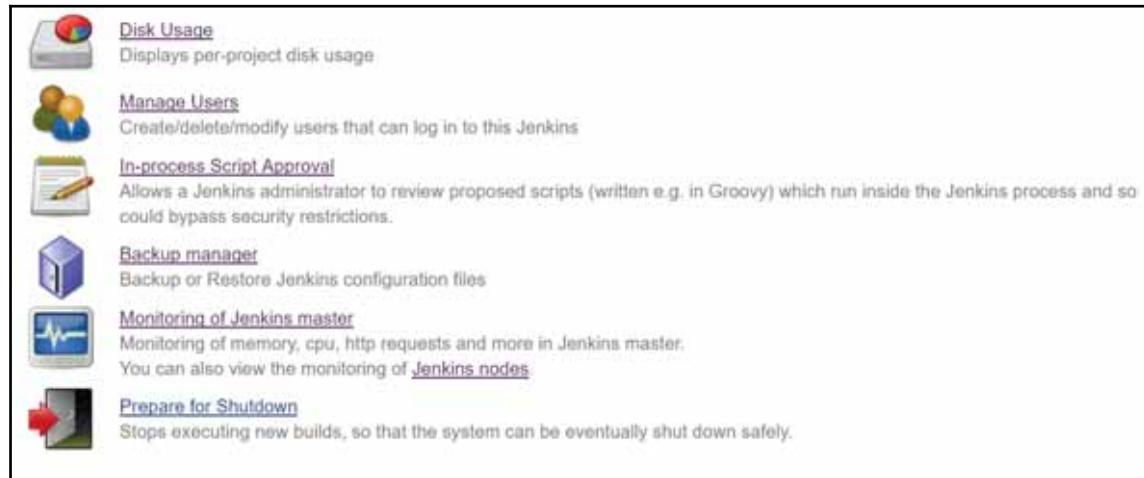
(you can start using the installed plugins right away)



Restart Jenkins when installation is complete and no jobs are running

The Backup plugin allows us to take a backup of JENKINS_HOME and restore it.

1. Go to **Manage Jenkins** and click on **Backup Manager**:



2. Click on **Setup**:

A screenshot of the Jenkins Backup manager setup page. The left sidebar shows navigation links:

- New Item
- People
- Build History
- Project Relationship
- Check File Fingerprint
- Manage Jenkins
- My Views
- Credentials
- Disk Usage

The main content area is titled "Backup manager" and contains the following setup options:

- Setup**: An icon of a wrench and screwdriver.
- Backup Hudson configuration**: An icon of a drive with an arrow pointing outwards.
- Restore Hudson configuration**: An icon of a drive with an arrow pointing inwards.

3. Configure **Backup directory**, **Format**, **File name template**, and so on:

Backup config files

Backup configuration

Hudson root directory F:\#JenkinsEssentials\FirstDraft\jenkinsHome

Backup directory d:\backup [?](#)

Format zip [?](#)

File name template date [?](#)

Custom exclusions [?](#)

Verbose mode [?](#)

Configuration files (.xml) only [?](#)

No shutdown [?](#)

Backup content

Backup job workspace

Includes [?](#)

Excludes [?](#)

Case-sensitive

Backup builds history [?](#)

Backup maven artifacts archives [?](#)

4. Click on **Backup Hudson configuration:**



5. Once the backup has been completed **successfully**, verify the logs:



6. Go to **Backup Manager** and click on **Restore Hudson** configuration:

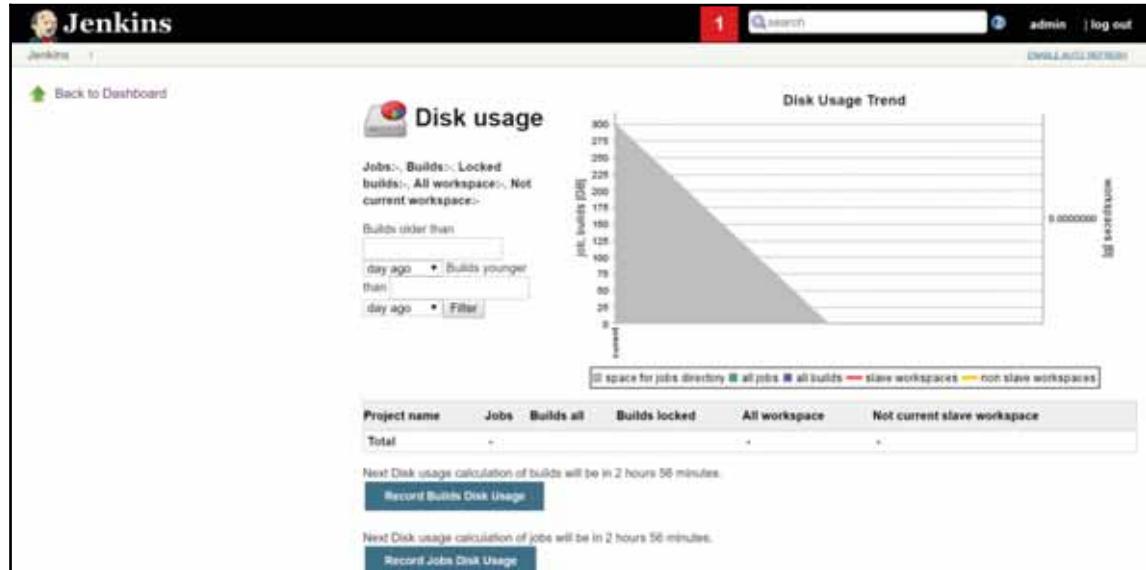
The screenshot shows the Jenkins interface with the 'Backup manager' plugin selected. On the left, there's a sidebar with various Jenkins management links like 'New Item', 'People', 'Build History', etc. The main content area is titled 'Backup manager' and displays the message 'Available backup in d:\backup :'. Below this, there's a redacted backup name followed by a 'Launch restore' button. At the bottom, there are two status sections: 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 Idle, 2 Idle).

In the next section, we will see the disk usage plugin.

Managing disk usage

This plugin gives details on disk usage for the system where Jenkins is installed.

1. Go to **Dashboard | Manage Jenkins | Disk Usage**:

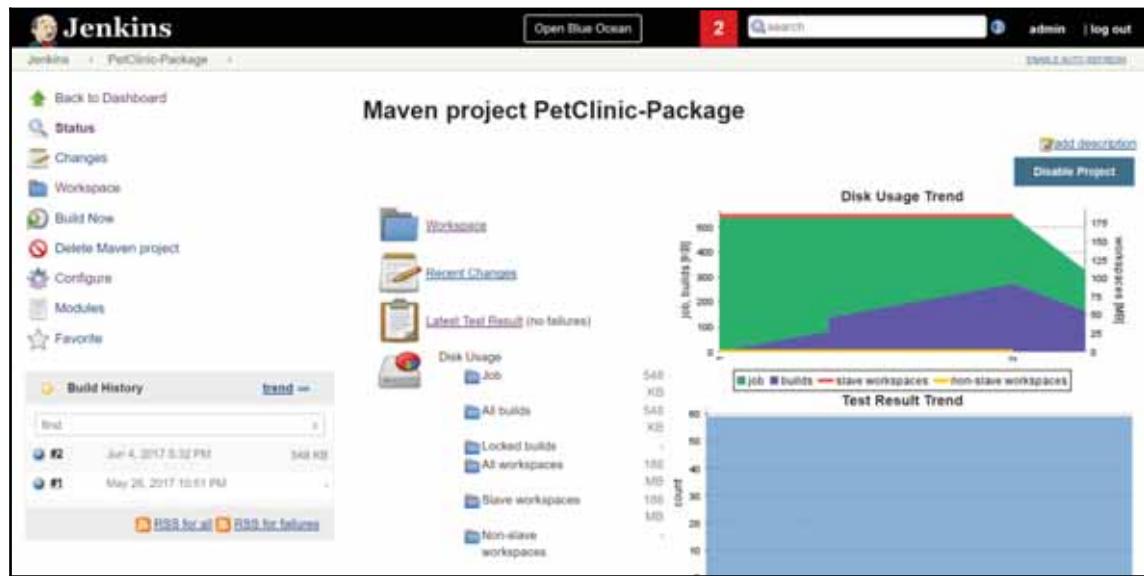


2. Go to **Manage Jenkins** and click on **Configure System**. Go to the **disk usage** section and click on **Show disk usage trend graph on the project page**:

The screenshot shows the 'Disk usage' configuration section within the Jenkins 'Configure System' page. It includes the following settings:

- Enable calculation of builds
Time interval for calculation: 0 */6 * * *
- Enable calculation of jobs
Time interval for calculation: 0 */6 * * *
- Enable calculation of workspace
Time interval for calculation: 0 */6 * * *
- Warn if some size is exceeded
- Show disk usage trend graph on the project page
- Show free space of jobs directory in global graph
- Length of global disk usage history: 183
- Time out for calculation of slave workspace in minutes: 5
- Control workspace from slave side too
- Jobs excluded for disk usage calculation: (empty text area)

3. Go to the specific project and see whether the disk usage trend chart is available or not:



In the next section, we will use the Build Monitor View plugin to keep track of the status and progress of different builds.

Build job-specific monitoring with the Build Monitor plugin

The Build Monitor plugin provides a visualization of the status and progress of selected Jenkins jobs. It displays an updated view automatically every couple of seconds, using AJAX. It can easily accommodate different computer screen sizes as well:

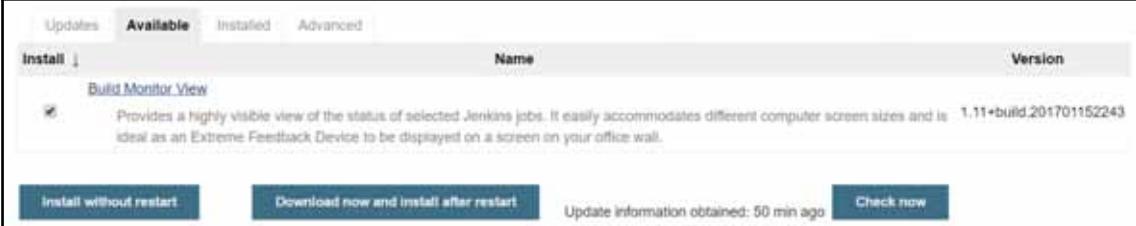
1. Go to the **Manage Jenkins | Manage Plugins | Available** tab. Install the **Build Monitor View** plugin:

Updates Available Installed Advanced

Install ↴ Build Monitor View

Provides a highly visible view of the status of selected Jenkins jobs. It easily accommodates different computer screen sizes and is ideal as an Extreme Feedback Device to be displayed on a screen on your office wall.

Install without restart **Download now and install after restart** Update information obtained: 50 min ago **Check now**



2. Go to the Jenkins dashboard. Click on **New View**:

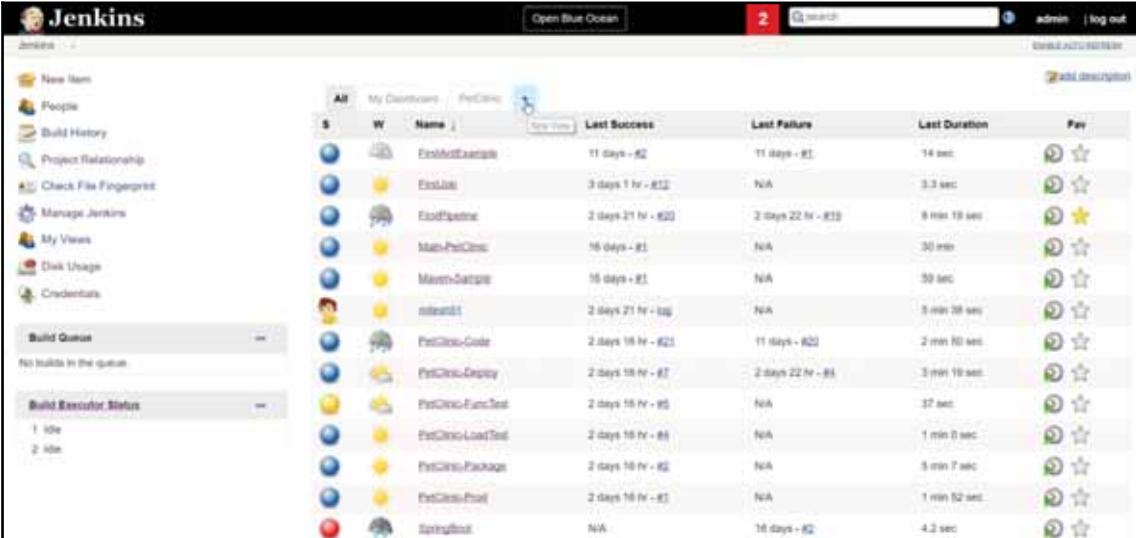
Jenkins | Open Blue Ocean | 2 | search | admin | log out

New Item People Build History Project Relationship Check File Fingerprint Manage Jenkins My Views Disk Usage Credentials

Build Queue — No builds in the queue.

Build Executor Status — 1 idle, 2 idle.

S	W	Name	Last Success	Last Failure	Last Duration	Key
1	1	FirstHelloExample	11 days - 82	N/A	14 sec	green star
2	2	FirstJob	3 days 1 hr - 812	N/A	3.3 sec	green star
3	3	FirstPipeline	2 days 21 hr - 820	2 days 22 hr - 823	8 min 19 sec	green star
4	4	Man-PetClinic	16 days - 81	N/A	30 min	green star
5	5	Maven-Sample	16 days - 81	N/A	59 sec	green star
6	6	resteasy	2 days 21 hr - 822	N/A	5 min 28 sec	green star
7	7	PetClinic-Code	2 days 18 hr - 821	11 days - 822	2 min 50 sec	green star
8	8	PetClinic-Deploy	2 days 18 hr - 87	2 days 22 hr - 84	3 min 19 sec	green star
9	9	PetClinic-FuncTest	2 days 18 hr - 85	N/A	37 sec	green star
10	10	PetClinic-LoadTest	2 days 18 hr - 86	N/A	1 min 5 sec	green star
11	11	PetClinic-Package	2 days 18 hr - 82	N/A	5 min 7 sec	green star
12	12	PetClinic-Profil	2 days 18 hr - 83	N/A	1 min 52 sec	green star
13	13	Springboot	N/A	16 days - 82	4.2 sec	green star



3. Provide a **View name** and select **Build Monitor View**. Click **OK**:

The screenshot shows the Jenkins 'New Item' creation interface. On the left is a sidebar with various Jenkins management links like 'New Item', 'People', 'Build History', etc. The main area has a 'View name' input field containing 'PetClinic-M'. Below it is a list of view types with radio buttons: 'Build Monitor View' (selected), 'Build Pipeline View', 'Dashboard', 'List View', and 'My View'. A note under 'Build Monitor View' says: 'Shows a highly visible status of selected jobs. Ideal as an Extreme Feedback Device to be displayed on a screen on your office wall.' A note under 'List View' says: 'Shows items in a simple list format. You can choose which jobs are to be displayed in which view.' A note under 'My View' says: 'This view automatically displays all the jobs that the current user has an access to.' At the bottom right is an 'OK' button.

4. Select the jobs to be displayed in the newly created **Build Monitor View**.

5. Click **Save**:

The screenshot shows the 'Edit View' configuration page for 'PetClinic-M'. The left sidebar includes links for 'New Item', 'Edit View' (which is active), 'Delete View', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', 'Disk Usage', and 'Credentials'. The main configuration area has fields for 'Name' (set to 'PetClinic-M') and 'Description' (empty). There are sections for 'Filter build queue' and 'Filter build executors'. Under 'Job Filters', there's a 'Status Filter' dropdown set to 'All selected jobs'. The 'Jobs' section lists several Jenkins jobs with checkboxes: 'FirstArtifactExample' (unchecked), 'FirstJob' (unchecked), 'FirstPipeline' (unchecked), 'Main-PetClinic' (unchecked), 'maven-sample' (unchecked), 'restest51' (unchecked), 'PetClinic-Code' (checked), 'PetClinic-Deploy' (checked), and 'PetClinic-Functest' (checked). At the bottom left is a 'Build Queue' section showing 'No builds in the queue.' and a 'Build Executor Status' section showing '1 idle'.

6. Verify the status of the **Build Monitor View** in the Jenkins dashboard:



In the next section, we will discuss the **Audit trail** plugin in brief.

Audit Trail plugin-overview and usage

Keep a log of who executed specific Jenkins operations, such as configuring jobs and so on. On the Jenkins configuration page, we need to configure the log file location and settings, such as file size and number of rotating log files:

Log file	Log Location	Log File Size MB	Log File Count
	d:\jenkinsLogs	5	5

Add Logger ▾ Advanced...

In the next section, we will discuss the workspace cleanup plugin in brief.

Workspace Cleanup plugin

The **Workspace Cleanup** plugin is used to delete the workspace from Jenkins before the build, or when a build is finished and artifacts saved. If we want to start a Jenkins build with a clean workspace, or we want to clean a particular directory before each build, then we can effectively use this plugin. Different options are available for deleting workspaces.

Install the plugin from the Jenkins dashboard:

The screenshot shows the Jenkins Plugin Manager interface. The 'Available' tab is selected. A search bar at the top right contains the text 'Workspace Cleanup Plugin'. Below the search bar, a table lists the plugin details. The table has columns for Name, Version, and two buttons: 'Install without restart' and 'Download now and install after restart'. The 'Name' column shows 'Workspace Cleanup Plugin' and the 'Version' column shows '0.20'. The 'Description' column contains the text: 'This plugin deletes the workspace before the build or when a build is finished and artifacts saved.'

We can apply patterns for files to be deleted based on the status of the build job. We can add post-build actions for workspace deletion:

The screenshot shows the configuration screen for the 'Delete workspace when build is done' post-build action. It includes fields for 'Patterns for files to be deleted' (with an 'Add' button), 'Apply pattern also on directories' (checkbox), 'Clean when status is' (checkboxes for Success, Unstable, Failure, Not Built, Aborted), 'Don't fail the build if cleanup fails' (checkbox), 'External Deletion Command' (text input field), and a 'Delete' button. At the bottom left is a dropdown menu for 'Add post-build action'.

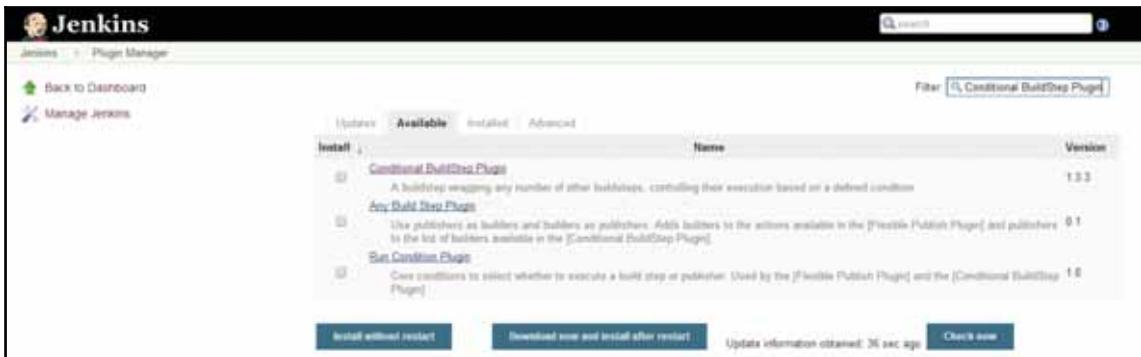


For more details on the **Workspace Cleanup** plugin, visit <https://plugins.jenkins.io/ws-cleanup>.

Conditional Build Step plugin

The Conditional Build Step plugin allows us to wrap any number of other build steps, controlling their execution based on a defined condition.

Install the plugin from the Jenkins dashboard:



This plugin defines a few core run conditions, such as:

- **Always/Never**: To disable a build step from the job configuration
- **Boolean condition**: To execute the step if a token expands to a representation of true
- **Current status**: To execute the build step if the current build status is within the configured/specific range
- **File exists/Files match**: To execute the step if a file exists or matches a pattern
- **Strings match**: To execute if the two strings are the same
- **Numerical comparison**: To execute the build step depending on the result of comparing two numbers
- **Regular expression match**: To execute the build step depending on the matching of regular expression
- **Provide a regular expression and a label**: To execute the build step if the expression matches the label

- **Time/Day of week:** To execute the build job during a specified period of the day, or day of the week
 - **And/Or/Not:** Logical operations to enable the combining and sense inversion of run conditions
 - **Build Cause:** To execute the build step depending on the cause of the build, such as triggered by timer, user, scm-change, and so on
 - **Script Condition:** Utilize a shell script to decide whether a step should be skipped
 - **Windows Batch Condition:** Utilize Windows Batch to decide whether a step should be skipped

Select **Conditional step (single)** from **Add build step**:

The screenshot shows the Jenkins Conditional step (single) configuration page. The 'Run?' section contains 'File exists' with 'File' set to '/tmp/ec2.txt' and 'Base directory' set to 'Workspace'. The 'Builder' section is set to 'EC2 Environment'. Under 'AWS Credentials', 'AWS Credentials Id' is selected (highlighted with a red box), and 'aws.region' is set to 'us-east-1' (also highlighted with a red box). A note below says 'Please select the Jenkins Credentials Id for your AWS environment'. At the bottom, there are 'Save' and 'Apply' buttons.

Select **Conditional steps (multiple)** from **Add build step**. We can add multiple steps to a condition in this conditional step:

The screenshot shows the Jenkins Conditional Build Step configuration interface. At the top, there's a dropdown menu labeled "Run?" with "Always" selected. A "Conditional steps (multiple)" section is expanded. Below it, a "Steps to run if condition is met" section is shown. This section contains two items: "Inject environment variables" and "Execute shell". Each item has a "Delete" button to its right. Under "Inject environment variables", there are fields for "Properties File Path" and "Properties Content". Under "Execute shell", there is a "Command" field. At the bottom of the configuration area, there are "Save" and "Apply" buttons.



For more details on Conditional Build Step plugin, visit
<https://wiki.jenkins-ci.org/display/JENKINS/Conditional+BuildStep+Plugin>.

EnvInject plugin

We know that different environments such as dev, test, production, and so on require different configurations.

Install the plugin from the Jenkins dashboard:

The screenshot shows the Jenkins Plugin Manager interface. The 'Available' tab is selected, and the search bar at the top right contains the text 'EnvInject'. A single result, 'EnvInject Plugin', is listed. It has a brief description, two versions (1.91 and 1.92), and a 'Download now and install after restart' button.

The EnvInject plugin provides a facility to have an isolated environment for different build jobs. The EnvInject plugin injects environment variables at node startup, before and/or after a SCM checkout for a run, as a build step for a run, and so on. **Select Inject environment variables to the build process** specific to the build job:

This screenshot shows the configuration page for the EnvInject plugin. The 'Inject environment variables to the build process' checkbox is checked. The 'Properties File Path' field contains '/usr/envVariables.properties'. The 'Properties Content' field contains 'MAVEN_HOME=/usr/maven'. Below these, there are fields for 'Script File Path' (empty), 'Script Content' (empty), and 'Evaluated Groovy script' (empty). At the bottom, the 'Inject passwords to the build as environment variables' checkbox is also checked. There are 'Save' and 'Apply' buttons at the bottom left.



For more details on EnvInject plugin, visit
<https://wiki.jenkins-ci.org/display/JENKINS/EnvInject+Plugin>.

Summary

In this chapter, we have seen how to configure a Master/Agent architecture to distribute the workload and to avoid a single point of failure; however, Jenkins does not have a high availability story yet and if the master goes down then it will still be a single point failure.

We have also seen different plugins that can enhance the monitoring and management of Jenkins, as well as plugins that can be utilized to extend the functionality of Jenkins. All these plugins and the Master/Agent architecture help to make automation more effective and broaden the scope of different minor innovations that can be done in automating different activities.

In the next chapter, we will see how to configure security in Jenkins. We will focus on user management, role-based access, and project-based access using Jenkins.

9

Security in Jenkins

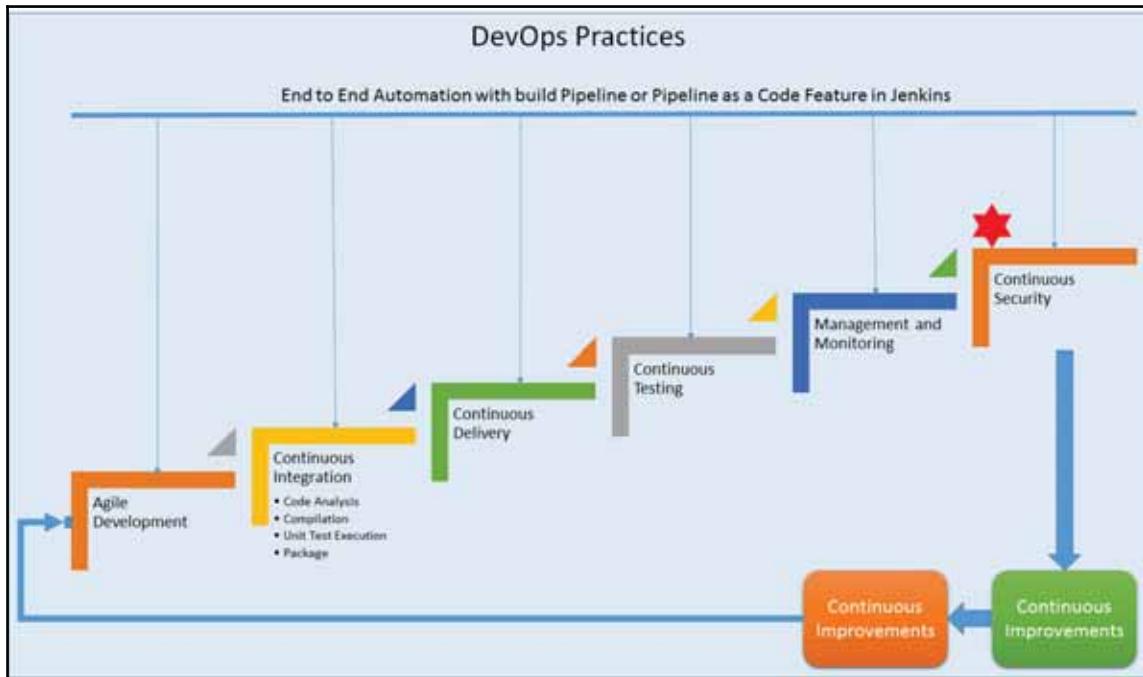
Up to now we have seen static code analysis, Continuous Integration, Continuous Delivery/Deployment, Continuous Testing, the orchestration of build jobs using the build pipeline plugin and pipeline as a code, and the management and monitoring of Jenkins resources.

This chapter will cover the security management options available in Jenkins.

It will help to perform user management, authentication, and authorization, including matrix-based security and role-based access. We will cover the following major topics in this chapter:

- User management
- Role-based security
- Project-based security

In this chapter, we will cover continuous security practices as a part of our DevOps journey:



Thread details available on Jenkins Master using Jenkins Monitoring Plugin

At the end of this chapter, we will know how to configure role-based and project-based security in Jenkins, as well as user management.

User management

In this section, we will cover how to manage multiple users. With user management, we can provide access to Jenkins for multiple users and provide them role-based or project-based access when it is required.

1. Go to **Manage Jenkins** and click on **Manage Users**:

The screenshot shows the left sidebar of the Jenkins 'Manage Jenkins' page. It contains several management links with corresponding icons:

- System Log**: System log captures output from `java.util.logging` related to Jenkins.
- Load Statistics**: Check your resource utilization and see if you need more computers for your builds.
- Jenkins CLI**: Access/manage Jenkins from your shell, or from your script.
- Script Console**: Executes arbitrary script for administration/trouble-shooting/diagnostics.
- Manage Nodes**: Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
- About Jenkins**: See the version and license information.
- Manage Old Data**: Scrub configuration files to remove remnants from old plugins and earlier versions.
- Install as Windows Service**: Installs Jenkins as a Windows service to this system, so that Jenkins starts automatically when the machine boots.
- Manage Users**: Create/delete/modify users that can log in to this Jenkins.
- In-process Script Approval**: Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.
- Prepare for Shutdown**: Stops executing new builds, so that the system can be eventually shut down safely.

2. Check the existing admin user available in Jenkins:

The screenshot shows the Jenkins 'Users' management interface. At the top, there's a navigation bar with the Jenkins logo, the word 'Jenkins', a user count '1', the name 'admin', and a 'log out' link. Below the navigation, there are links to 'Back to Dashboard', 'Manage Jenkins', and 'Create User'. The main section is titled 'Users' and contains a table with one row. The table has columns for 'User Id' and 'Name'. The single entry is 'admin'. To the right of the 'Name' column is a gear icon for configuration. The table has a light gray background with white borders.

User Id	Name
 admin	admin

3. Click on the **Create User** link and provide details:

Jenkins

Jenkins' own user database

Back to Dashboard

Manage Jenkins

Create User

Create User

Username:	Shreyansh
Password:
Confirm password:
Full name:	Shreyansh Soni
E-mail address:	redacted@gmail.com

Create User

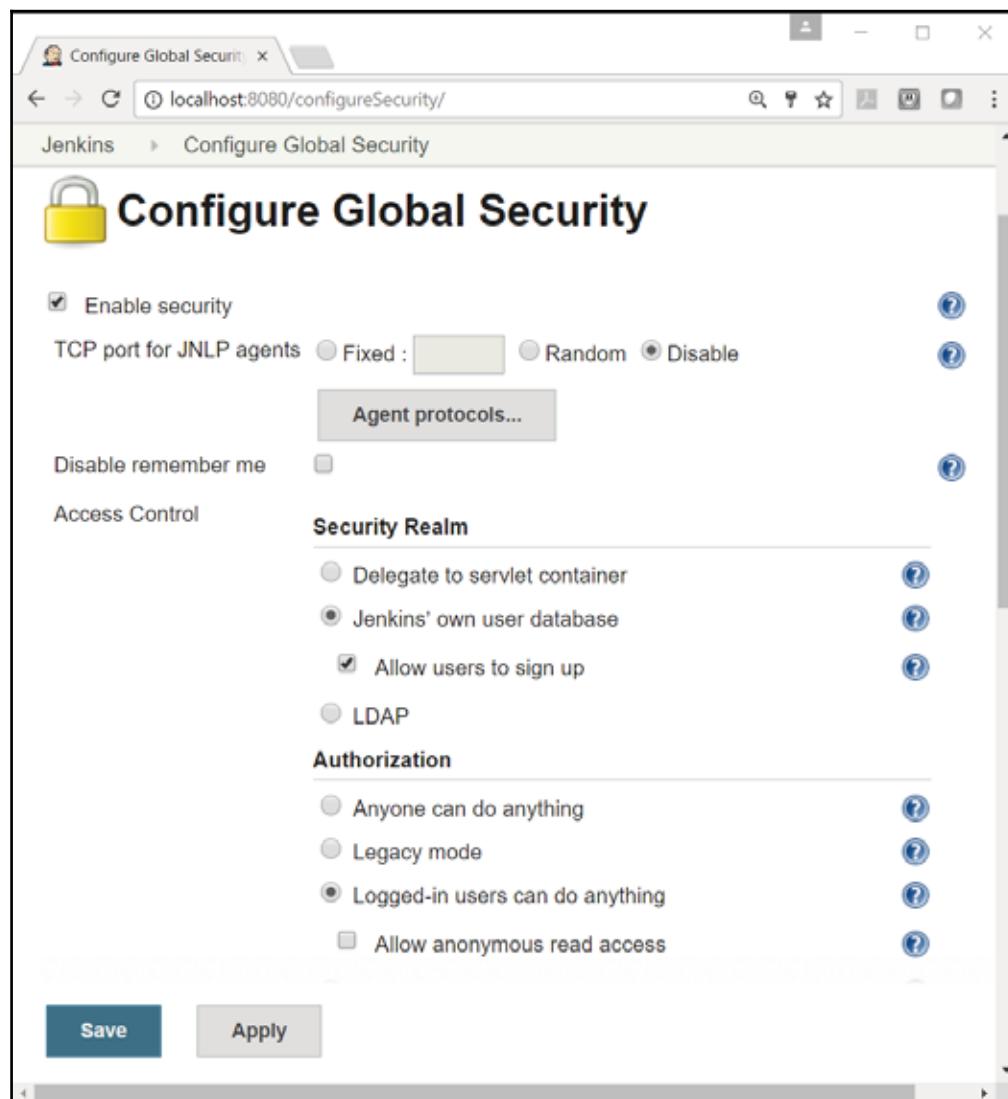
4. Check the list of users in **Manage Jenkins | Manage Users**:

The screenshot shows a browser window for the Jenkins 'Users' page at `localhost:8080/securityRealm/`. The page title is 'Users [Jenkins]'. The top navigation bar includes a Jenkins logo, a user icon, and the text 'localhost:8080/securityRealm/'. On the right side of the header, there is a red button with the number '1' and the word 'admin', along with a 'log out' link. Below the header, the page displays 'Jenkins' own user database' and links to 'Back to Dashboard', 'Manage Jenkins', and 'Create User'. The main content area is titled 'Users' and contains a table listing two users:

User Id	Name	Action
admin	admin	
Shreyansh	Shreyansh Soni	

5. To allow sign up and access to only logged in users, go to **Manage Jenkins | Configure Global Security**.

6. In the Access Control section, click on Jenkins' own user database and select Allow users to sign up:



So, this is how we can create users and allow users to sign up to access Jenkins.

Role-based security

In the **Authorization** section, we can configure matrix-based security so we can configure who can do what. We can configure the predefined roles available in Jenkins.

1. Select **Matrix-based security** and type a name in the User/group to add box.
Make sure that you give access to Admin before saving it, or the Jenkins account will be locked out:

The screenshot shows the Jenkins 'Configure Global Security' page under the 'Access Control' tab. In the 'Authorization' section, 'Matrix-based security' is selected. A matrix grid is displayed for users 'Anonymous' and 'admin'. The columns represent Jenkins features: Overall, Credentials, Agent, and Job. The rows represent user groups. For 'admin', all checkboxes are checked, indicating full access. For 'Anonymous', most checkboxes are unchecked, except for 'Overall' which has a question mark icon.

User/group	Overall	Credentials	Agent	Job
Anonymous	?	✗	✗	✗
admin	✓	✓	✓	✓

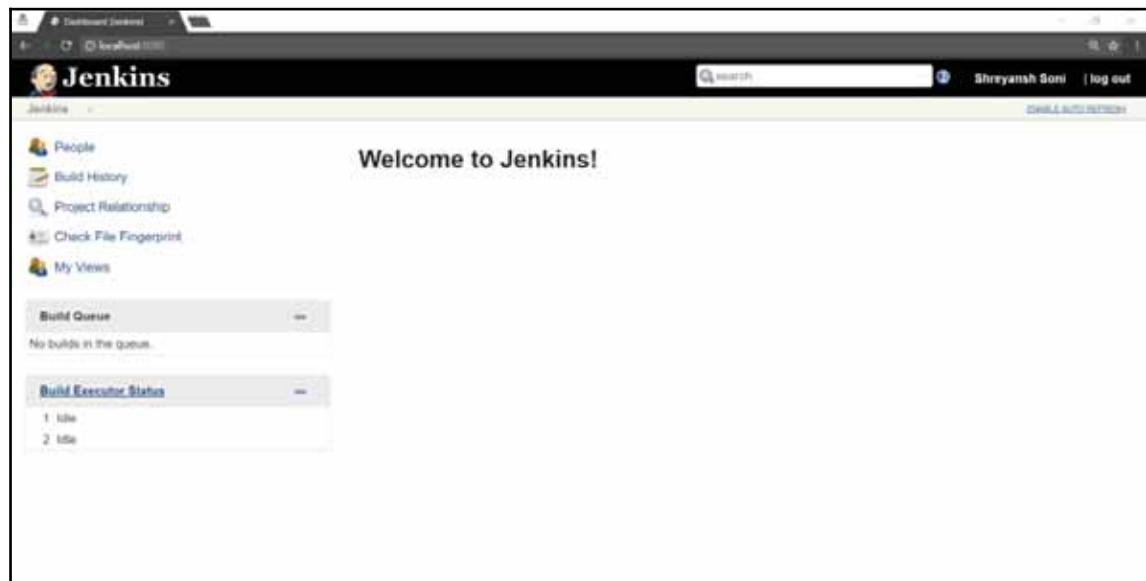
2. Type the name of our newly created user in the **User/group to add** text box, click on **Add**, and provide all the required rights. We can do the same things for different users.
3. Click on **Save**:

The screenshot shows the Jenkins 'Configure Global Security' page under the 'Access Control' section. It includes settings for 'Security Realm' (Jenkins' own user database selected), 'Authorization' (Matrix-based security selected), and a large 'Matrix Authorization Strategy' grid. The grid columns represent roles: Administer, Read, Create, Delete, Manage Domains, Update, View, Build, Configure, Connect, Create, Delete, Disconnect, Provision, Build, Cancel, Configure, Create, Delete, Discover, Move. The rows represent user groups: Anonymous, admin, and Shreyansh. The matrix shows specific permissions granted to each user group. At the bottom, there are 'Save' and 'Apply' buttons.

4. To verify access has been granted, open a new incognito window in your browser and log in with the username and password of the newly created user:

The screenshot shows the Jenkins login page. The URL is 'localhost:8080/login'. The page features a logo, a search bar, and links for 'log in | sign up'. The main area contains fields for 'User' (Shreyansh) and 'Password', a 'Remember me on this computer' checkbox, and a 'log in' button. Below the form is a link to 'Create an account'.

5. Verify that limited access is available to the new user, and that the **New Item** and **Manage Jenkins** links are not available:



6. Now go to **Manage Jenkins | Global Security Configuration**. Allow **Read** rights in the **Job** category for the user **Shreyansh**.
7. Click on **Save**:

The screenshot shows the Jenkins 'Configure Global Security' configuration page. At the top, there are tabs for Overall, Credentials, Agent, and Job. The Overall tab is selected, displaying a matrix for User/group (Shreyansh, admin, Anonymous) against permissions (Administer, Read, Create, Delete, Manage, Domains, Update, View, Build, Configure, Connect, Create, Delete, Disconnect, Provision, Build, Cancel, Configure, Create, Delete, Discover, Move, Read, Workspace, Delete). Below the matrix is a 'User/group to add:' input field and an 'Add' button. A note below says 'Plain text' and 'Treats all input as plain text; HTML unsafe characters like < and > are escaped to their respective character entities.' There are sections for 'Request Forgery exploits', 'Crumb Algorithm' (Default Crumb issuer, Enable proxy compatibility), 'Enable CLI over Remoting', and 'Use browser for metadata download'. At the bottom are 'Save' and 'Apply' buttons.

8. Go to the incognito window that we opened before and refresh the page. Now we have read access to the **Jobs** available in Jenkins:

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links for People, Build History, Project Relationship, Check File Fingerprint, My Views, Build Queue (empty), and Build Executor Status (1 idle, 2 idle). The main area displays a table of jobs under the 'All' tab. The columns are S (Status icon), W (Workflow icon), Name, Last Success, Last Failure, and Last Duration. The jobs listed are: FireAntExample (Last Success: 14 hr - #2, Last Failure: 14 hr - #1, Last Duration: 14 sec), FirstJob (Last Success: 3 hr 41 min - #2, Last Failure: N/A, Last Duration: 16 sec), Maven-PvtClinic (Last Success: 6 days 0 hr - #1, Last Failure: N/A, Last Duration: 30 min), Maven-Sonar (Last Success: 6 days 1 hr - #1, Last Failure: N/A, Last Duration: 59 sec), PetClinic-Code (Last Success: 1 day 2 hr - #18, Last Failure: 14 hr - #20, Last Duration: 1 min 34 sec), PetClinic-Package (Last Success: 14 hr - #1, Last Failure: N/A, Last Duration: 4 min 51 sec), and SpringBoot (Last Success: N/A, Last Failure: 6 days 1 hr - #2, Last Duration: 4.2 sec). A legend at the bottom indicates colors for success (green), failure (red), and latest build (blue).

9. We can see the jobs, but we can't execute them as rights are not available:

The screenshot shows the Jenkins interface for the 'PetClinic-Code' project. On the left, there's a sidebar with links to 'Back to Dashboard', 'Status', 'Changes', and 'SonarQube'. The main area is titled 'Project PetClinic-Code'. It features a 'SonarQube' icon with a link and a 'Recent Changes' icon with a link. Below this is a 'Permalinks' section with a list of recent builds:

- Last build (#20), 14 hr ago
- Last stable build (#18), 1 day 2 hr ago
- Last successful build (#18), 1 day 2 hr ago
- Last failed build (#20), 14 hr ago
- Last unsuccessful build (#20), 14 hr ago
- Last completed build (#20), 14 hr ago

At the bottom of the main area, there are 'RSS for all' and 'RSS for failures' links.

This is how we can manage users and authorization in Jenkins. In the next section, we will see how to give project-based access.

Project-based security

Project-based Matrix Authorization Strategy is an extension to **Matrix-based security**. It allows an access control list matrix to be defined for each project. This feature is very useful where we want to give access to specific jobs to specific users, so the security of Jenkins is not compromised.

1. Go to **Manage Jenkins | Global Security Configuration**. In the **Authorization** section, select **Project-based Matrix Authorization Strategy**.
2. Give admin all rights and **Save**:

The screenshot shows the Jenkins 'Configure Global Security' page. Under the 'Authorization' section, 'Project-based Matrix Authorization Strategy' is selected. A matrix table defines permissions for users and groups across various Jenkins features. The columns include Overall, Credentials, Agent, and Job. The rows include User/group (Anonymous, admin) and feature names like Administer, Read, Create, Delete, Manage, Domains, Update, View, Build, Configure, Connect, Create, Delete, Disconnected, Provision, Build, Cancel, Configure, Create, Delete, Discover, Move. The 'admin' user has full permissions (checkmarks in all cells). The 'Save' and 'Apply' buttons are at the bottom.

3. Go to the incognito window where we logged in using the credentials for Shreyansh.
4. Refresh the page and you will get **Access Denied**. The reason is we haven't given any rights to **Shreyansh** in **Project-based Matrix Authorization Strategy**:

The screenshot shows the Jenkins login page. The header includes the Jenkins logo and the user name 'Shreyansh Soni | log out'. The main content area displays the error message 'Access Denied' in large bold letters, followed by a red circular icon with a minus sign and the text 'Shreyansh is missing the Overall/Read permission'.

5. We need to provide overall read rights so **Shreyansh** can access the Jenkins dashboard:

The screenshot shows the Jenkins 'Configure Global Security' page under the 'Authorization' tab. It displays a matrix configuration for users and groups across various Jenkins features. A user named 'Shreyansh' has been added to the matrix.

User/group	Overall	Credentials	Agent	Job
Administrator	X	X	X	X
Read	X	X	X	X
Create	X	X	X	X
Delete	X	X	X	X
Manage Domains	X	X	X	X
Update	X	X	X	X
View	X	X	X	X
Build	X	X	X	X
Configure	X	X	X	X
Connect	X	X	X	X
Create Agent	X	X	X	X
Delete Agent	X	X	X	X
Disconnect Agent	X	X	X	X
Provision	X	X	X	X
Build	X	X	X	X
Cancel	X	X	X	X
Configure Job	X	X	X	X
Create Job	X	X	X	X
Delete Job	X	X	X	X
Discover Agents	X	X	X	X

Below the matrix, there is a text input field labeled 'User/group to add: Shreyansh' and a 'Add' button. The 'Plain text' section contains a note about HTML escaping. Under 'Crumb', there is a checkbox for 'Prevent Cross Site Request Forgery exploits' and a 'Crumb Algorithm' section with a 'Default CrumbIssuer' option. At the bottom are 'Save' and 'Apply' buttons.

6. Now, go to the individual build job as an admin and select **Enable project-based security** in the job configuration page.
7. Add **Shreyansh** as a **User** and click on **Save**:

The screenshot shows the Jenkins job configuration page for 'PetClinic-Code'. The 'General' tab is selected. Under 'Project name', it says 'PetClinic-Code'. Under 'Description', there is a large text area containing '[Plain text] Preview'. Below this, there is a section titled 'Enable project-based security' with a checked checkbox. A radio button next to it is labeled 'Block inheritance of global authorization matrix'. A table follows, showing permissions for 'User/group' (Anonymous and Shreyansh) across various Jenkins roles: Create, Delete, Manage, Domain, Update, View, Build, Cancel, Configure, Delete, Discover, Move, Read, Workspace, Delete, Replay, Update, Tag, and SCM. The 'Shreyansh' row has checkboxes checked for most of these permissions. At the bottom of the table is a 'User/group to add: Shreyansh' input field and an 'Add' button. Below the table are 'Save' and 'Apply' buttons.

- Now, go to the incognito window where **Shreyansh** is logged in and refresh the page. We can see one job that we have configured to give access to **Shreyansh**:

The screenshot shows the Jenkins dashboard under the user 'Shreyansh Soni'. On the left sidebar, there are links for People, Build History, Project Relationship, Check File Fingerprint, and My Views. The main area displays a table of jobs. The first job listed is 'PetClinic-Code', which has a red circular icon indicating it is failing. The table columns are 'All', 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The 'PetClinic-Code' row shows 'Icon: M L', 'Last Success: 1 day 3 hr - #18', 'Last Failure: 18 hr - #20', and 'Last Duration: 1 min 34 sec'. Below the table are links for 'Legend', 'RSS for all', 'RSS for failures', and 'RSS for just failed builds'. At the bottom of the dashboard, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 idle, 2 idle).

9. Click on **Build** and verify all the rights are available to the user **Shreyansh**:

The screenshot shows the Jenkins interface for the 'PetClinic-Code' project. At the top right, the user 'Shreyansh Soni' is logged in. On the left, there's a sidebar with links like 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', and 'SonarQube'. The main area is titled 'Project PetClinic-Code'. It features a 'SonarQube' icon with a 'Workspace' link below it, and a 'Recent Changes' icon with a 'Recent Changes' link. Below these are sections for 'Build History' and 'Permalinks'. The 'Build History' section lists three builds: #20 (May 26, 2017 10:14 PM), #19 (May 26, 2017 11:05 AM), and #18 (May 26, 2017 10:53 AM). The 'Permalinks' section lists several links related to the last build. At the bottom, there are 'RSS for all' and 'RSS for failures' buttons.

We have finished user management, role-based access, and project-based access in Jenkins as a part of securing Jenkins.

Summary

Up to now, we have covered static code analysis, Continuous Integration, Continuous Delivery/Deployment, Continuous Testing, the orchestration of build jobs using the build pipeline plugin and pipeline as a Code, the management and monitoring of Jenkins resources, security in Jenkins in the form of user management, role-based access, and project-based access for users.

By covering all these topics, we have ensured that we cover almost all major aspects of application lifecycle management. It is not that only Jenkins can deliver what we have achieved up to now. It is not about tools only. It is about people, processes, and tools in DevOps implementations. Another basic but very important thing is to remember that DevOps is not a tool, technology, model or framework; *DevOps is a CULTURE*.

Index

A

Ant

- build job, configuring for Java application 106, 108, 110, 111
- build job, creating for Java application 106, 108, 110, 111
- configuring, in CentOS 35
- configuring, in Jenkins 36, 38, 39
- configuring, in Windows 35
- URL 35
- URL, for example 107

Apache JMeter

- URL 167
- used, for load testing 167, 169, 170, 171, 173

Audit Trail plugin

- overview 234
- usage 234

AWS Elastic Beanstalk Publisher plugin

- URL 131

AWS Elastic Beanstalk

- war file, deploying from Jenkins 126, 127, 129, 130, 131, 133, 134, 135

B

backup

- job-specific configurations, managing 224, 227, 228

Blue Ocean 190

build job

- configuring, for Java application with Ant 106, 108, 110, 111
- configuring, for Java application with Maven 111, 114, 117
- creating, for Java application with Ant 106, 108, 110, 111
- creating, for Java application with Maven 111, 114, 117

C

CentOS

- Ant, configuring 35
- Git repository, configuring 51, 55
- Git repository, installing 51, 55
- Java, installing 32

CI/CD pipeline

- overview 27

Conditional Build Step plugin

- about 236
- installing 236, 238
- run conditions, defining 236
- URL 238

Continuous Delivery (CD)

- about 7
- overview 119

Continuous Deployment

- overview 119

Continuous Integration (CI) 7, 103

Continuous Testing 145

D

dashboard 22, 23, 24

Dashboard View plugin

installing 104, 105, 106
URL 104
disk usage
managing 229, 231
domain-specific language (DSL) 10
downstream job
configuring 178, 180, 182, 184, 185, 186, 188, 189

E

Eclipse
integrating, with Jenkins 59, 60, 63, 65
email notifications
sending, of build status 97, 99, 101
EnvInject plugin
about 238
installing 239
URL 239
environment variables
configuring 34

F

functional testing
with Selenium 146, 148, 150, 152, 153, 155, 157, 159, 160, 162, 163, 164, 167

G

Generic Java Package (.war)
URL 12
Git repository
build job, creating with Git 55, 57, 58
build job, creating with GitHub 55, 58
configuring, on CentOS 51, 55
installing, on CentOS 51, 55

J

Java application
build job, configuring with Ant 106, 108, 109, 111
build job, configuring with Maven 111, 114, 117
build job, creating with Ant 106, 108, 109, 111
build job, creating with Maven 111, 114, 117
Java installer
URL, for downloading 32
Java

installing 32
installing, on CentOS 32
JavaMelody
used, for monitoring Jenkins 220, 221, 222, 223, 224
JDK
configuring, in Jenkins 36, 38, 39
Jenkins 2.x
integration, with SonarQube 6.3 68, 70, 71, 72, 73, 74, 75, 77, 79, 80, 81, 83, 85, 86, 87, 89, 90
Jenkins 2
about 8, 10
features 10
installation 11, 13, 14, 16, 18, 20, 21
plugins, categories 9
URL 11
Jenkins
build, overview 32
Eclipse, integrating 59, 60, 63, 65
requisites 32
settings, configuring 26, 27
war file, deploying to Tomcat 120, 121, 123, 125
job-specific configurations
managing, for backup and restore 224, 227, 228
job
creating 40, 42, 43, 46, 47, 49, 51

K

Kudu console
URL 142

L

load testing
with Apache JMeter 167, 169, 170, 171, 173

M

master and slave nodes
managing 213, 214, 215, 216, 218, 219
Master/Agent architecture 213
Matrix-based security 252
Maven
build job, configuring for Java application 111, 114, 117
build job, creating for Java application 111, 114,

- 117
configuring, in Jenkins 36, 38, 39
installing 36
URL 36
Microsoft Azure App Services
war file, deploying from Jenkins 136, 139, 142, 143
Microsoft Azure
URL 138
- P**
- pipeline as code
implementation 190, 192, 194, 195, 198, 199, 201, 206, 207, 208, 209
overview 189
- Platform as a Service (PaaS) 126
- Project Object Model (POM) 111
- Project-based Matrix Authorization Strategy 252
- project-based security
configuring 252, 253, 254, 255, 256
- promoted builds plugin
identifying 209, 210
- Q**
- Quality Gate plugin
about 91
using 92, 94, 96, 97
- R**
- restore
job-specific configurations, managing 224, 227, 228
- role-based security
configuring 248, 249, 250, 251, 252
- S**
- Selenium
used, for functional testing 146, 148, 150, 152, 154, 155, 158, 159, 160, 162, 164, 165, 166, 167
- SonarQube 66
SonarQube 6.3
Jenkins 2.x, integration 68, 69, 71, 72, 73, 74, 75, 77, 79, 80, 81, 83, 85, 86, 87, 89, 90
URL 68
- T**
- Tomcat
installing 120
URL 120
war file, deploying from Jenkins 120, 121, 123, 125
- U**
- upstream job
configuring 178, 179, 182, 184, 185, 186, 188, 189
- user management 242, 244, 246, 247
- W**
- war file
deploying, from Jenkins to AWS Elastic Beanstalk 126, 127, 129, 130, 131, 133, 134, 135, 142, 143
deploying, from Jenkins to Microsoft Azure App Services 136, 138, 139
deploying, from Jenkins to Tomcat 120, 121, 123, 125
- Windows 10
Java, installing 32
- Windows
Ant, configuring 35
- Workspace Cleanup plugin
about 235
installing 235
URL 235