

Mitesh Soni

Jenkins Essentials

Second Edition

Setting the stage for a DevOps culture



Packt

Contents

- [1: Exploring Jenkins](#)
 - [b'Chapter 1: Exploring Jenkins'](#)
 - [b'Introduction of Jenkins 2'](#)
 - [b'Installation of Jenkins 2'](#)
 - [b'Jumpstart tour of the Jenkins dashboard'](#)
 - [b'Configuration settings in Jenkins'](#)
 - [b'Overview of the CI/CD pipeline'](#)
 - [b'Summary'](#)
- [2: Installation and Configuration of Code Repository and Build Tools](#)
 - [b'Chapter 2: Installation and Configuration of Code Repository and Build Tools'](#)
 - [b'Overview of Jenkins'](#)
 - [b'Installing Java and configuring the environment variables'](#)
 - [b'Installing and configuring Ant'](#)
 - [b'Installing Maven'](#)
 - [b'Configuring Ant, Maven, and JDK in Jenkins'](#)
 - [b'First job in Jenkins'](#)
 - [b'Installing and configuring the Git repository on CentOS'](#)
 - [b'Eclipse and Jenkins integration'](#)
 - [b'Summary'](#)
- [3: Managing Code Quality and Notifications](#)
 - [b'Chapter 3: Managing Code Quality and Notifications'](#)
 - [b'Jenkins 2.x integration with Sonar 6.3'](#)
 - [b'Quality Gate plugin'](#)
 - [b'Email notifications on build status'](#)
 - [b'Summary'](#)
- [4: Continuous Integration with Jenkins](#)
 - [b'Chapter 4: Continuous Integration with Jenkins'](#)
 - [b'Dashboard View plugin'](#)
 - [b'Creating and configuring a build job for a Java application with Ant'](#)
 - [b'Creating and configuring a build job for a Java application with Maven'](#)
 - [b'Summary'](#)
- [5: Continuous Delivery - Implementing Automated Deployment](#)
 - [b'Chapter 5: Continuous Delivery - Implementing Automated Deployment'](#)

- [b'An overview of Continuous Delivery and Continuous Deployment'](#)
- [b'Installing Tomcat'](#)
- [b'Deploying a war file from Jenkins to Tomcat'](#)
- [b'Deploying a WAR file from Jenkins to AWS Elastic Beanstalk'](#)
- [b'Deploying a war file from Jenkins to Microsoft Azure App Services'](#)
- [b'Summary'](#)
- [6: Continuous Testing - Functional and Load Testing with Jenkins](#)
 - [b'Chapter 6: Continuous Testing - Functional and Load Testing with Jenkins'](#)
 - [b'Functional testing with Selenium'](#)
 - [b'Load testing with Apache JMeter'](#)
 - [b'Summary'](#)
- [7: Build Pipeline and Pipeline as a Code](#)
 - [b'Chapter 7: Build Pipeline and Pipeline as a Code'](#)
 - [b'Build Pipeline'](#)
 - [b'Overview of pipeline as a code'](#)
 - [b'Promoted builds'](#)
 - [b'Summary'](#)
- [8: Managing and Monitoring Jenkins](#)
 - [b'Chapter 8: Managing and Monitoring Jenkins'](#)
 - [b'Managing Jenkins master and slave nodes'](#)
 - [b'Monitoring Jenkins with JavaMelody'](#)
 - [b'Managing job-specific configurations - backup and restore'](#)
 - [b'Managing disk usage'](#)
 - [b'Build job-specific monitoring with the Build Monitor plugin'](#)
 - [b'Audit Trail plugin-overview and usage'](#)
 - [b'Workspace Cleanup plugin'](#)
 - [b'Conditional Build Step plugin'](#)
 - [b'EnvInject plugin'](#)
 - [b'Summary'](#)
- [9: Security in Jenkins](#)
 - [b'Chapter 9: Security in Jenkins'](#)
 - [b'User management'](#)
 - [b'Role-based security'](#)
 - [b'Project-based security'](#)

- b'Summary'

Chapter 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.

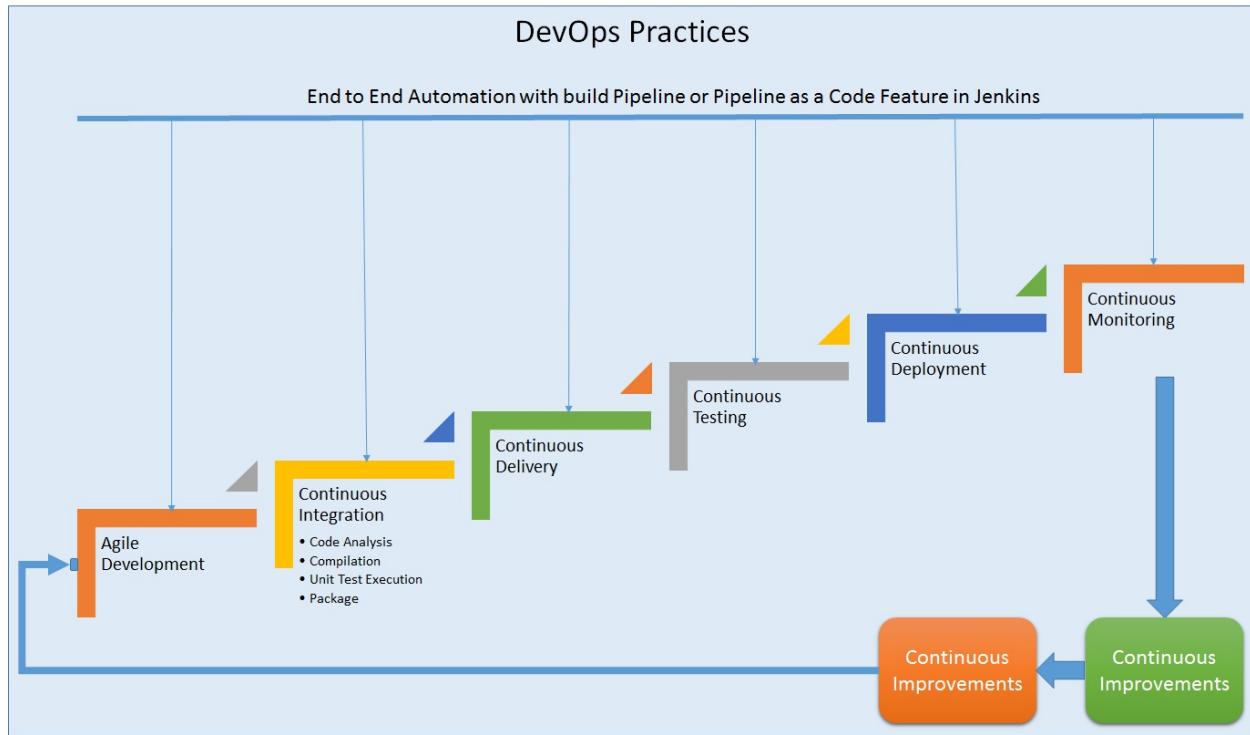
This chapter also describes installation and configuration of Jenkins 2. We are going to take a jumpstart tour through some of the key features of Jenkins and plugin installations as well.

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!

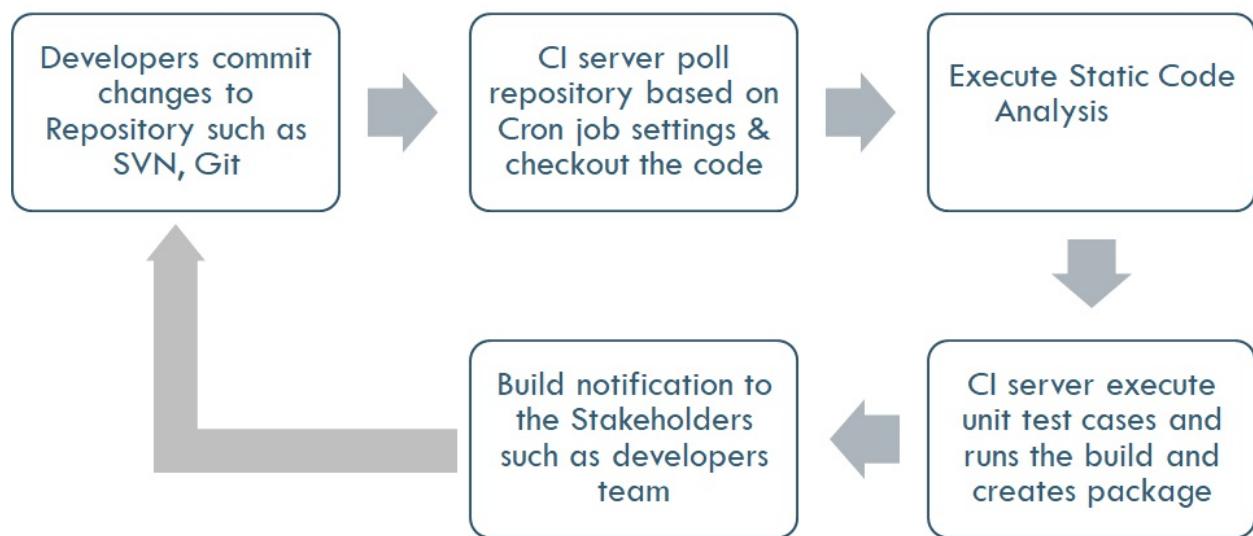
In this chapter, we will see what Jenkins 2 is and the new features introduced regarding CI as a part of our DevOps journey. We will cover the following steps to complete our DevOps journey. Each chapter is a stepping stone to reach the next one. It is always good to have an incremental approach so we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

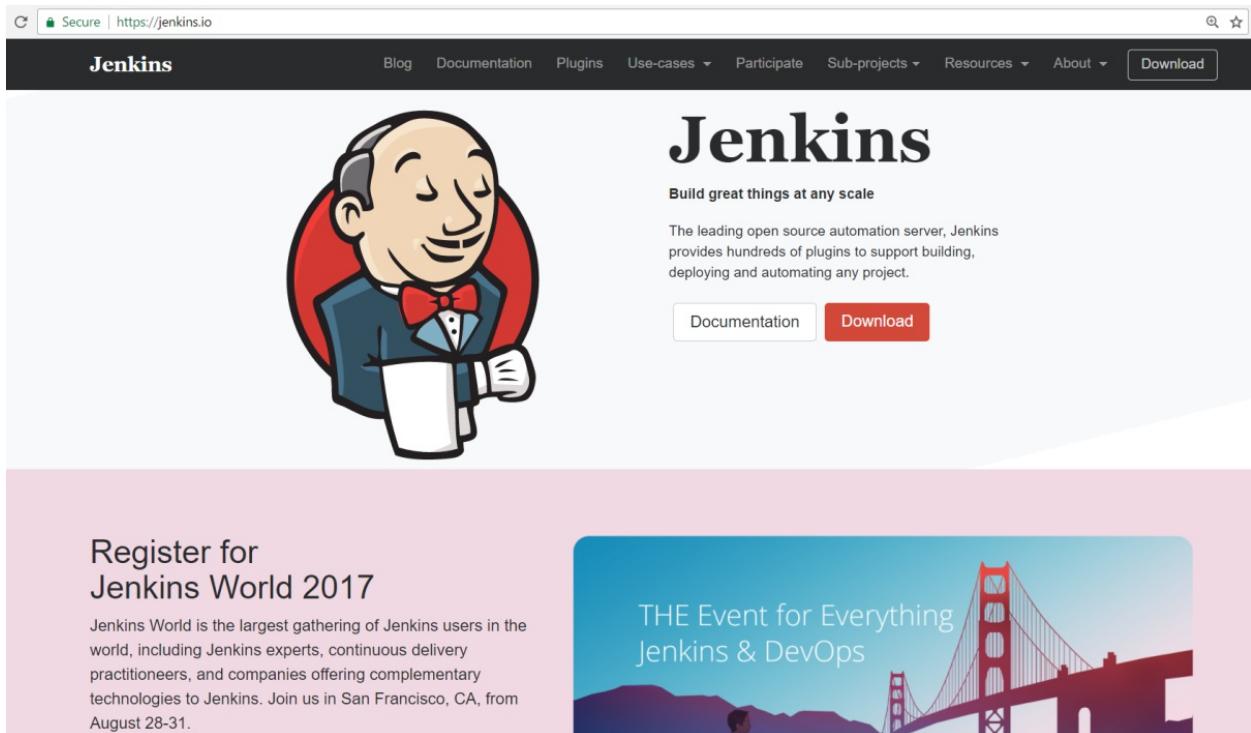
- Easy installation on different operating systems, Arch Linux, FreeBSD, Gentoo, MacOS X, openBSD, openSUSE, RedHAT/Fedora/CentOS, Ubuntu/Debian, Windows, and it is also available for Docker and as generic Java packages too.
- Easy upgrades: Jenkins has very speedy release cycles (long-term support and 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.

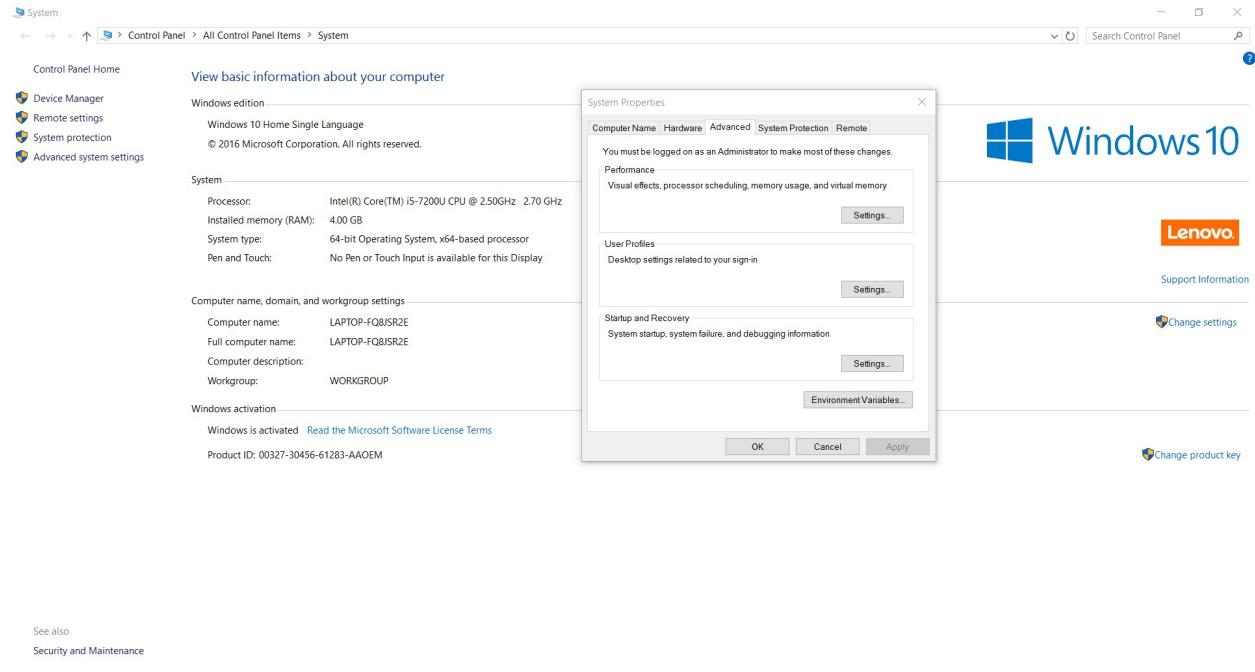
Download **Generic Java Package (.war)** from <https://jenkins.io/download/> on the local system as follows:

The screenshot shows the Jenkins website's download page. At the top, there's a navigation bar with links for Blog, Documentation, Plugins, Use-cases, Participate, Sub-projects, Resources, About, and a prominent Download button. Below the navigation, the main content area has a heading "Getting started with Jenkins". It explains that Jenkins produces two release lines: LTS and weekly. It notes that both are distributed as .war files, native packages, installers, and Docker containers. A "Long-term Support (LTS)" section details that LTS releases are chosen every 12 weeks from the regular stream. A "Weekly" section states that new releases are produced weekly for bug fixes and features. Below these sections are links for Changelog, Upgrade Guide, and Past Releases. On the left, a sidebar titled "Download Jenkins 2.46.3" lists various platforms: Docker, FreeBSD, Gentoo, Mac OS X, OpenBSD, openSUSE, Red Hat/Fedora/CentOS, Ubuntu/Debian, Windows, and Generic Java package (.war). On the right, another sidebar titled "Download Jenkins 2.63 for:" lists supported operating systems: Arch Linux, Docker, FreeBSD, Gentoo, Mac OS X, OpenBSD, openSUSE, Red Hat/Fedora/CentOS, Ubuntu/Debian, Windows, and Generic Java package (.war).

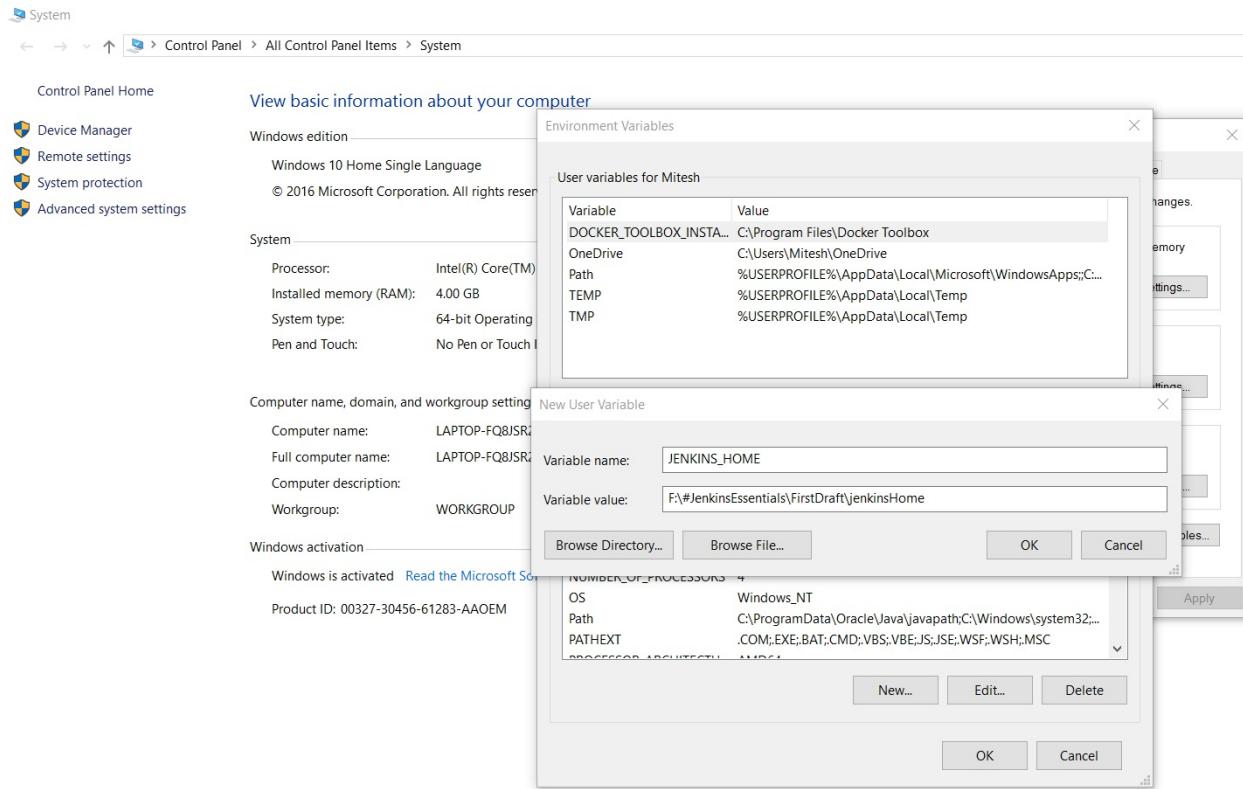
Before we start Jenkins, we will set the `JENKINS_HOME` environment variable. When we install and configure Jenkins, all the files reside in the `jenkins` directory by default. We often need to change the default location for our convenience. We can do that by setting the `JENKINS_HOME` environment variable.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

```

C:\Windows\System32\cmd.exe - 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")
@[33mMay 20, 2017 8:07:36 PM Main deleteWinstoneTempContents
WARNING: Failed to delete the temporary Winstone file C:/Users/Mitesh/AppData/Local/Temp/winstone/jenkins.war
@[0mMay 20, 2017 8:07:36 PM org.eclipse.jetty.util.log.Log initialized
INFO: Logging initialized @1132ms to org.eclipse.jetty.util.log.JavaUtilLog
May 20, 2017 8:07:36 PM winstone.Logger logInternal
INFO: Beginning extraction from war file
@[33mMay 20, 2017 8:07:40 PM org.eclipse.jetty.server.handler.ContextHandler setContextPath
WARNING: Empty contextPath
@[0mMay 20, 2017 8:07:40 PM org.eclipse.jetty.server.Server doStart
INFO: jetty-9.4.z-SNAPSHOT
May 20, 2017 8:07:43 PM org.eclipse.webapp.StandardDescriptorProcessor visitServlet
INFO: NO JSP Support for /, did not find org.eclipse.jsp.JettyJspServlet
May 20, 2017 8:07:43 PM org.eclipse.jetty.server.session.DefaultSessionIdManager doStart
INFO: DefaultSessionIdManager workerName=node0
May 20, 2017 8:07:43 PM org.eclipse.jetty.server.session.DefaultSessionIdManager doStart
INFO: No SessionScavenger set, using defaults
May 20, 2017 8:07:43 PM org.eclipse.jetty.server.session.HouseKeeper startScavenging
INFO: Scavenging every 66000ms
Jenkins home directory: F:\#JenkinsEssentials\FirstDraft\jenkinsHome found at: EnvVars.masterEnvVars.get("JENKINS_HOME")
May 20, 2017 8:07:45 PM org.eclipse.server.handler.ContextHandler doStart
INFO: Started w:@6cae0256d/,file:///F:/%23JenkinsEssentials/FirstDraft/jenkinsHome/war/,AVAILABLE}{F:\#JenkinsEssentials\FirstDraft\jenkinsHome\war}
May 20, 2017 8:07:45 PM org.eclipse.jetty.server.AbstractConnector doStart
INFO: Started ServerConnector@58670130[HTTP/1.1,[http/1.1]]{0.0.0.0:8080}
May 20, 2017 8:07:45 PM org.eclipse.jetty.server.Server doStart
INFO: Started @10267ms
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 onAttained
INFO: Listed all plugins
May 20, 2017 8:07:47 PM jenkins.InitReactorRunner$1 onAttained
INFO: Prepared all 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:

```

C:\Windows\System32\cmd.exe - java -jar jenkins.war
root of factory hierarchy
May 20, 2017 8:07:56 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.web.context.support.StaticWebApplicationContext@18078206: display name [Root WebApplicationContext]; startup date [Sat May 20 20:07:56 IST 2017]; root of context hierarchy
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@18078206]: org.springframework.beans.factory.support.DefaultListableBeanFactory@62f26c47
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@62f26c47: defining beans [filter,legacy]; root of factory hierarchy
May 20, 2017 8:07:56 PM jenkins.install.SetupWizard init
INFO:

*****
Denkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
75495e17d897498b8db53d27121080a

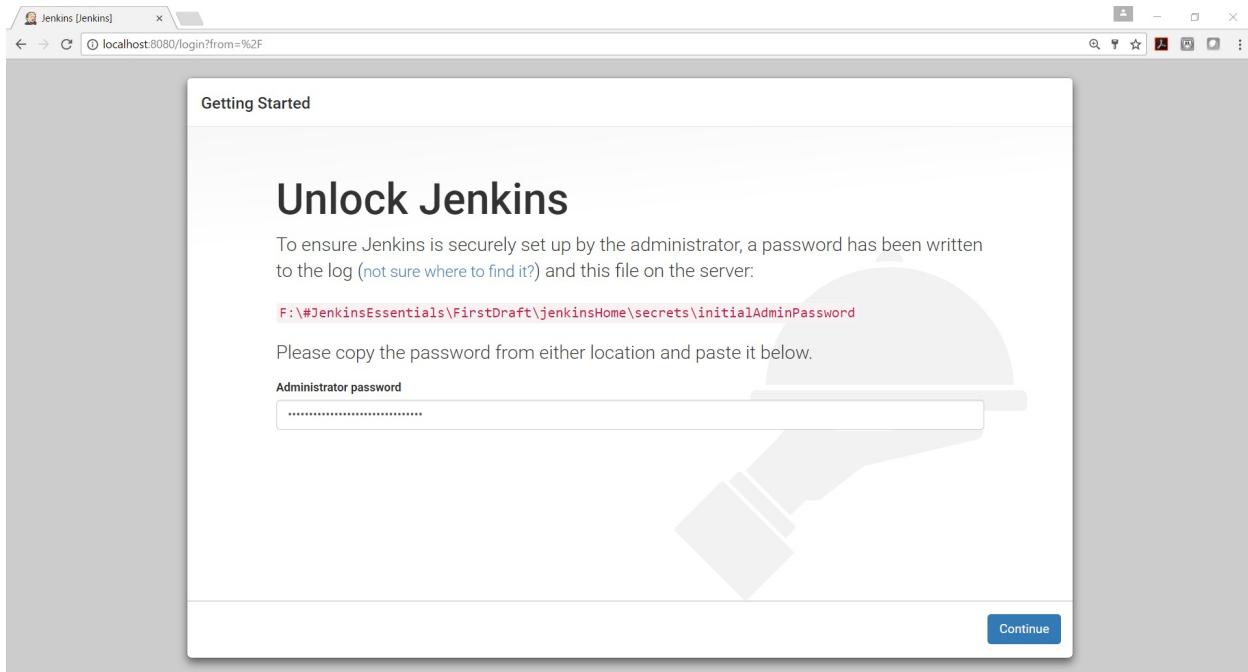
This may also be found at: F:\#JenkinsEssentials\FirstDraft\jenkinsHome\secrets\initialAdminPassword

*****
May 20, 2017 8:08:09 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$3 run
INFO: Jenkins is fully up and running
May 20, 2017 8:08:10 PM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
May 20, 2017 8:08:12 PM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
May 20, 2017 8:08:15 PM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tools.JDKInstaller
May 20, 2017 8:08:15 PM hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata. 20,242 ms

```

6. 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:

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.61

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	** bouncycastle API Plugin Folders Plugin ** Structs Plugin ** JUnit Plugin OWASP Markup Formatter Plugin PAM Authentication plugin ** Windows Slaves Plugin ** Display URL API Jenkins Mailer Plugin
⌚ Timestamper	⌚ Workspace Cleanup Plugin	⌚ Ant Plugin	⌚ Gradle 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
⌚ 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
✓ Timestamper	✓ Workspace Cleanup Plugin	✓ Ant Plugin	✓ Gradle 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
⌚ Pipeline	⌚ GitHub Branch Source Plugin	⌚ Pipeline: GitHub Groovy Libraries	⌚ Pipeline: Stage View Plugin	** - required dependency
⌚ 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

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

Getting Started

Create First Admin User

Username:	<input type="text" value="admin"/>
Password:	<input type="password" value="....."/>
Confirm password:	<input type="password" value="....."/>
Full name:	<input type="text" value="admin"/>
E-mail address:	<input type="text" value="████████@████████.cc"/>

Jenkins 2.61

[Continue as admin](#)

[Save and Finish](#)

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

Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

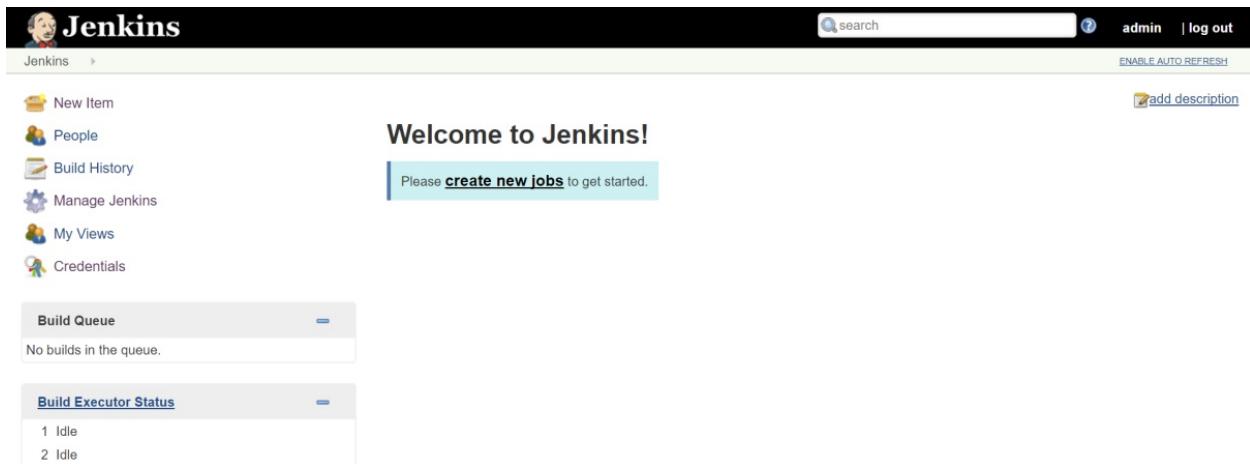
[Start using Jenkins](#)

Jenkins 2.61

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

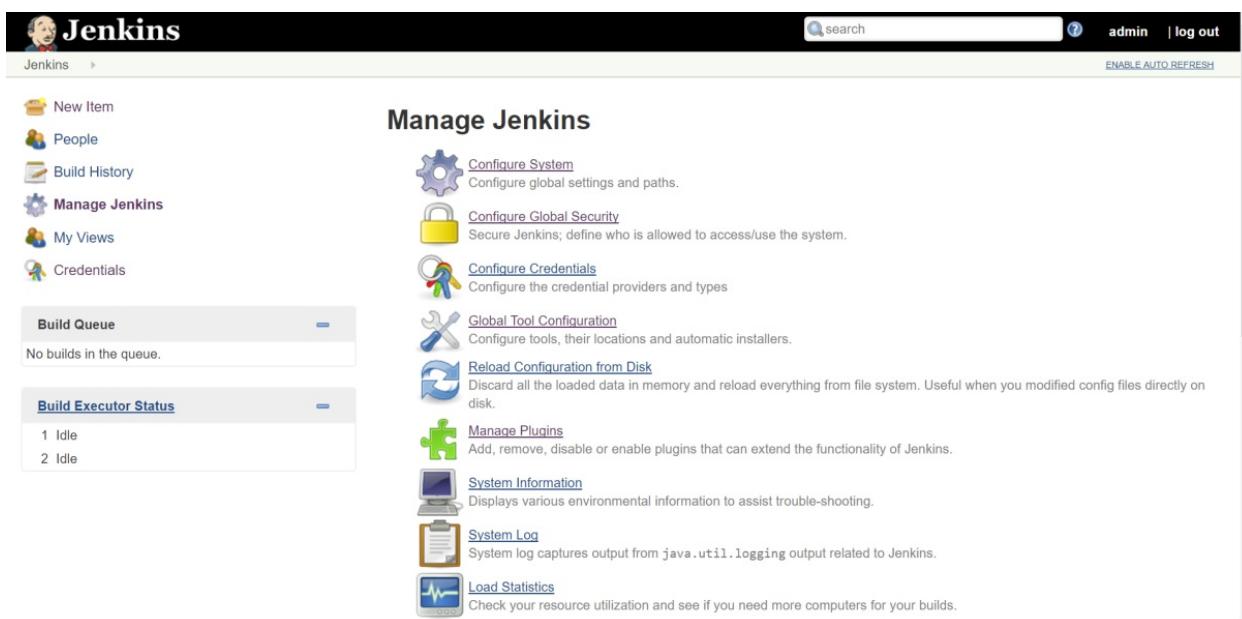
Jumpstart tour of the Jenkins dashboard

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



The screenshot shows the Jenkins dashboard with a dark header bar. On the left, there's a sidebar with icons for New Item, People, Build History, Manage Jenkins, My Views, and Credentials. The main area has a "Welcome to Jenkins!" message with a button to "create new jobs". Below it, there are two sections: "Build Queue" (empty) and "Build Executor Status" (2 Idle). The top right shows the user "admin" logged in.

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



The screenshot shows the "Manage Jenkins" page. On the left, there's a sidebar with icons for New Item, People, Build History, Manage Jenkins, My Views, and Credentials. The main area lists several configuration options with icons: "Configure System" (gear), "Configure Global Security" (padlock), "Configure Credentials" (key and user), "Global Tool Configuration" (wrench and wrench), "Reload Configuration from Disk" (disk), "Manage Plugins" (puzzle piece), "System Information" (monitor), "System Log" (log icon), and "Load Statistics" (graph). The top right shows the user "admin" logged in.

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 is a navigation bar with links for 'Back to Dashboard', 'Manage Jenkins', and 'Update Center'. Below this, a search bar and user information ('admin | log out') are visible. The main area has four tabs: 'Updates' (selected), 'Available', 'Installed' (selected), and 'Advanced'. The 'Installed' tab displays a table of currently enabled plugins. The columns are 'Enabled', 'Name', 'Version', 'Previously installed version', and 'Uninstall'. The table lists several plugins:

Enabled	Name	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	Ant Plugin Adds Apache Ant support to Jenkins	1.5		Uninstall
<input checked="" type="checkbox"/>	Authentication Tokens API Plugin This plugin provides an API for converting credentials into authentication tokens in Jenkins.	1.3		Uninstall
<input checked="" type="checkbox"/>	bouncycastle API Plugin This plugin provides an stable API to Bouncy Castle related tasks.	2.16.1		Uninstall
<input checked="" type="checkbox"/>	Branch API Plugin This plugin provides an API for multiple branch based projects.	2.0.9		Uninstall
<input checked="" type="checkbox"/>	build timeout plugin This plugin allows builds to be automatically terminated after the specified amount of time has elapsed.	1.18		Uninstall
<input checked="" type="checkbox"/>	Credentials Binding Plugin Allows credentials to be bound to environment variables for use from miscellaneous build steps.	1.11		Uninstall
<input checked="" type="checkbox"/>	Credentials Plugin This plugin allows you to store credentials in Jenkins.	2.1.13		Uninstall
	Display URL 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 interface for managing nodes. At the top, there's a navigation bar with the Jenkins logo, a search bar, and user information for 'admin'. Below the navigation, a link 'ENABLE AUTO REFRESH' is visible. On the left sidebar, there are links: 'Back to Dashboard', 'Manage Jenkins', 'New Node', and 'Configure'. The main content area is titled 'Nodes' and displays a table of nodes. The table has columns: S, Name (sorted by name), Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. One node, 'master', is listed. It has an icon of a computer monitor, the name 'master', architecture 'Windows 10 (amd64)', clock difference 'In sync', free disk space '231.76 GB', free swap space '10.66 GB', free temp space '248.36 GB', and response time '0ms'. Below the table, it says 'Data obtained 27 min'. A 'Refresh status' button is at the bottom right of the table. Underneath the table, there are two collapsed sections: 'Build Queue' (which says 'No builds in the queue.') and 'Build Executor Status' (which lists '1 Idle' and '2 Idle').

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Windows 10 (amd64)	In sync	231.76 GB	10.66 GB	248.36 GB	0ms
		Data obtained	27 min	27 min	27 min	27 min	27 min

In the next section, we will cover different kinds of configurationÂ 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 'Configure System' page. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', and 'Credentials'. Below that are sections for 'Build Queue' (empty) and 'Build Executor Status' (2 Idle). The main area has several configuration fields:

- Home directory:** F:\#JenkinsEssentials\FirstDraft\jenkinsHome
- Workspace Root Directory:** \${JENKINS_HOME}/workspace/\${ITEM_FULLNAME}
- Build Record Root Directory:** \${ITEM_ROOTDIR}/builds
- System Message:** (Empty text area)
- # of executors:** 2
- Labels:** (Empty text area)
- Usage:** Use this node as much as possible
- Quiet period:** 5
- SCM checkout retry count:** 0

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 Jenkins 'Configure Global Security' configuration page. At the top, there is a header with the Jenkins logo, a search bar, and user information for 'admin'. Below the header, the title 'Configure Global Security' is displayed next to a yellow padlock icon. The main content area contains several configuration sections:

- Enable security**: A checked checkbox.
- TCP port for JNLP agents**: Radio buttons for 'Fixed' (selected), 'Random', and 'Disable'.
- Agent protocols...**: A button.
- Disable remember me**: A checkbox.
- Access Control**: A section containing:
 - Security Realm**: Radio buttons for 'Delegate to servlet container' (disabled), 'Jenkins' own user database' (selected), and 'Allow users to sign up'.
 - Authorization**: Radio buttons for 'Anyone can do anything' (disabled), 'Legacy mode' (disabled), and 'Logged-in users can do anything' (selected).
- Save** and **Apply** buttons at the bottom.

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 page. It includes sections for Maven Configuration, JDK installations, and Git installations. Under Maven Configuration, 'Default settings provider' is set to 'Use default maven settings'. Under JDK, there is a list of installations with 'Add JDK' and 'List of JDK installations on this system' buttons. Under Git, there is a single entry for 'Git' with 'Name' set to 'Default' and 'Path to Git executable' set to 'git.exe'. Buttons for 'Save' and 'Apply' are at the bottom.

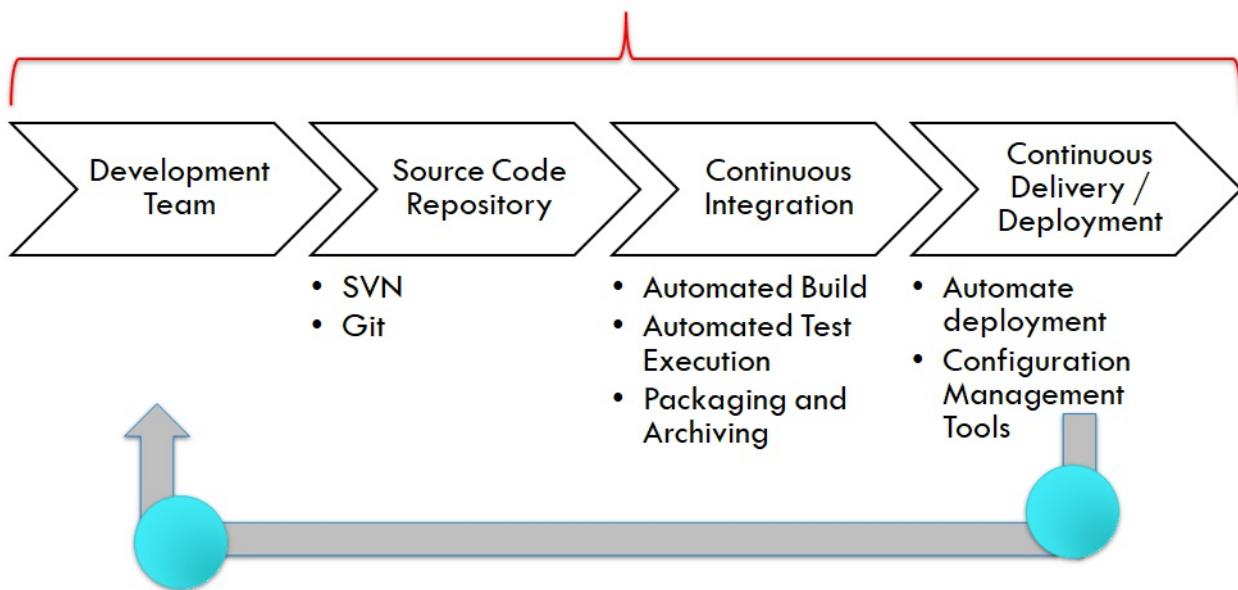
As discussed earlier in this chapter, the **Manage Plugins** section has an **Advanced** tab that allows us to configure proxy details.

The screenshot shows the Jenkins Plugin Manager Advanced tab. It features the 'HTTP Proxy Configuration' section. Fields include 'Server' (empty), 'Port' (empty), 'User name' (set to 'admin'), 'Password' (set to '****'), and 'No Proxy Host' (empty). A 'Submit' button is at the bottom left, and an 'Advanced...' button is at the bottom right.

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.

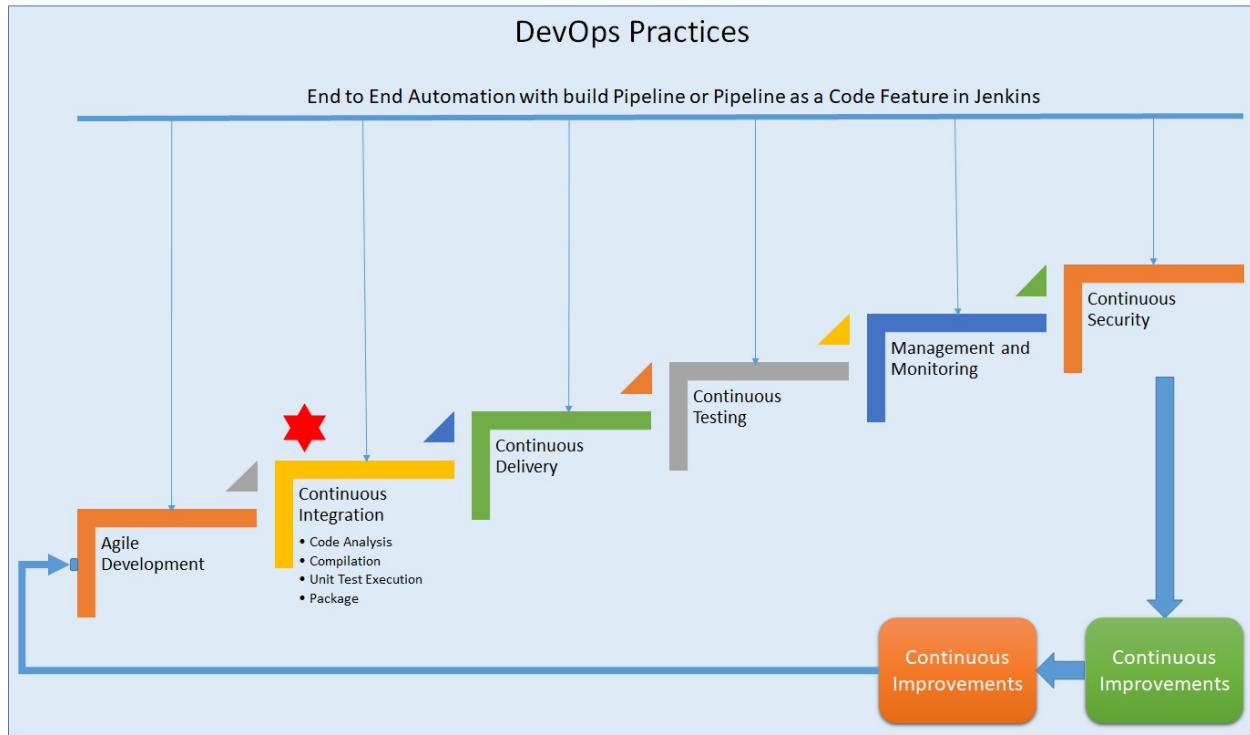
Chapter 2. Installation and Configuration of Code Repository and Build Tools

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 Jenkinsâan 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 coverÂ 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 eitherÂ 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 installedÂ 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, securi
.
.
.
ant-javamail.x86_64 : Optional javamail tasks for ant
eclipse-mylyn-java.x86_64 : Mylyn Bridge: Java Developme
.
.
.
java-1.5.0-gcj.x86_64 : JPackage runtime compatibility la
GCJ
java-1.5.0-gcj-devel.x86_64 : JPackage development compat
layer for GCJ
```

```
java-1.5.0-gcj-javadoc.x86_64 : API documentation for lib
java-1.6.0-openjdk.x86_64 : OpenJDK Runtime Environment
java-1.6.0-openjdk-devel.x86_64 : OpenJDK Development Env
java-1.6.0-openjdk-javadoc.x86_64 : OpenJDK API Documenta
java-1.7.0-openjdk.x86_64 : OpenJDK Runtime Environment
jcommon-serializer.x86_64 : JFree Java General Serializat
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, securi
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.
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
  1/1
  Verifying   : 1:java-1.7.0-openjdk-1.7.0.3-2.1.el6.7.x86
  1/1

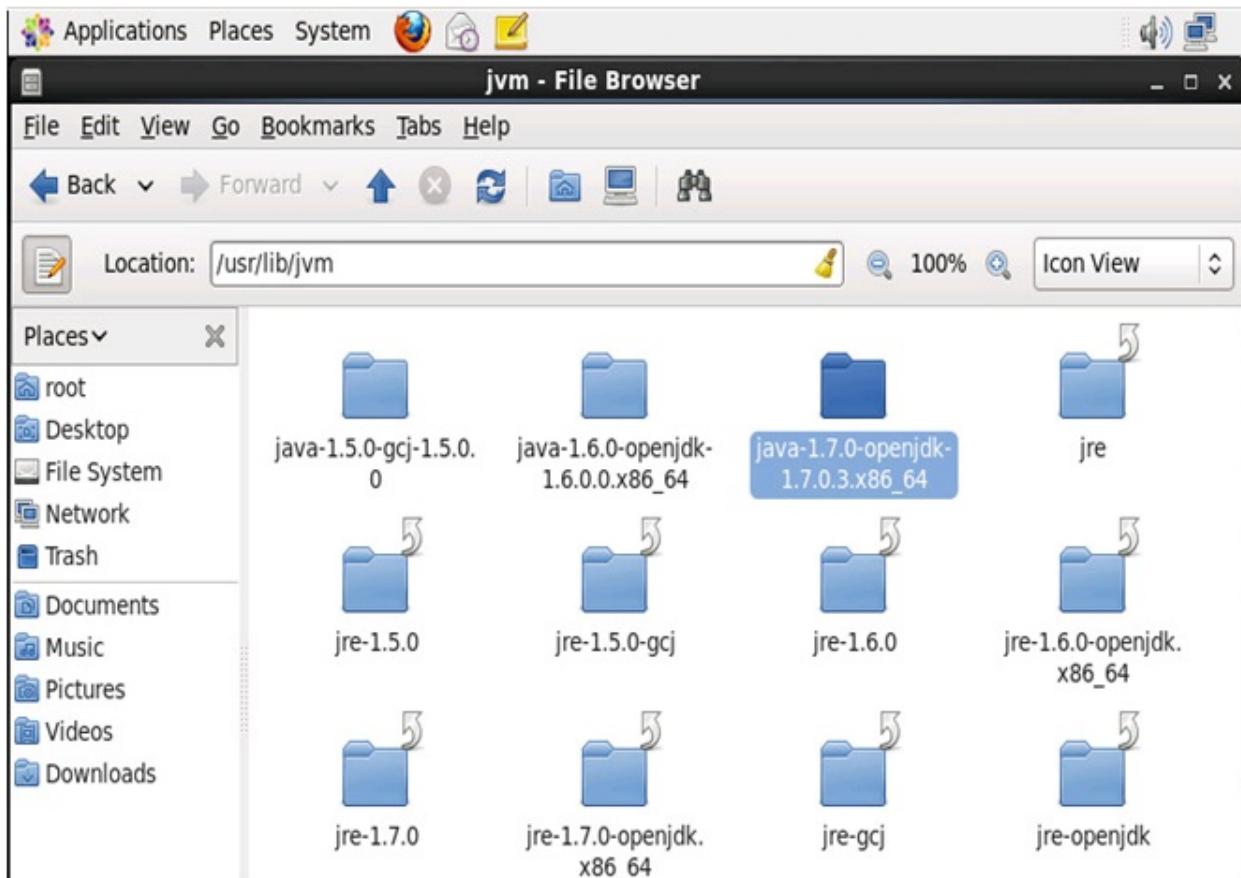
Installed:
  java-1.7.0-openjdk.x86_64 1:1.7.0.3-2.1.el6.7
Complete!
```

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.

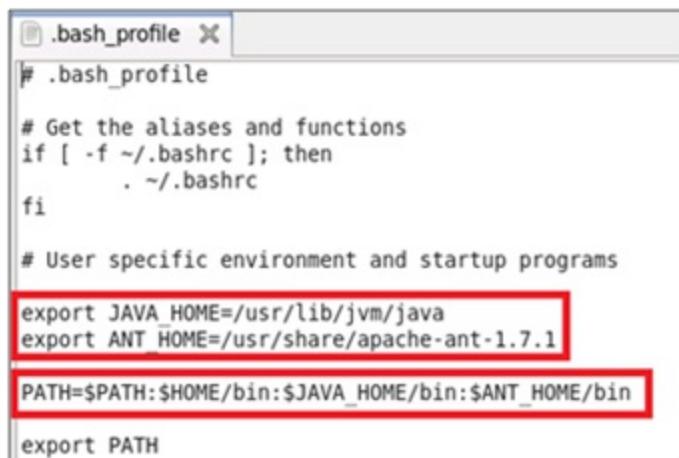
Configuring Ant in Windows

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

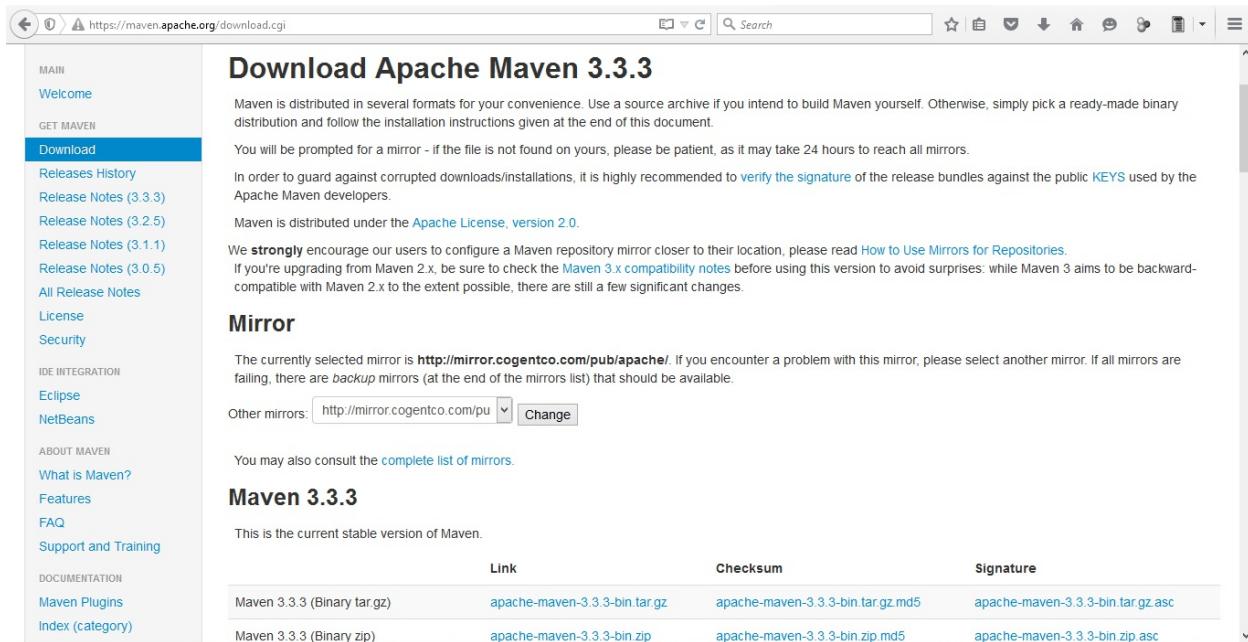
PATH=$PATH:$HOME/bin:$JAVA_HOME/bin:$ANT_HOME/bin

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 a web browser displaying the Apache Maven download page at <https://maven.apache.org/download.cgi>. The left sidebar contains links for MAIN, GET MAVEN, and various releases (3.3.3, 3.2.5, 3.1.1, 3.0.5). The main content area is titled "Download Apache Maven 3.3.3". It provides instructions for distribution formats and mirrors. A "Mirror" section lists the selected mirror as <http://mirror.cogentco.com/pub/apache/>. Below this, a table lists two download options for Maven 3.3.3:

	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:

The screenshot shows the Jenkins Global Tool Configuration page. Under the 'JDK' section, there is a 'JDK installations' table with one row. The row has a 'Name' field containing 'Java SE Development Kit 8u131' with a red 'Required' indicator. Below the table is an 'Install from java.sun.com' section with a dropdown for 'Version' set to 'Java SE Development Kit 8u131'. There are checkboxes for 'I agree to the Java SE Development Kit License Agreement' and 'Installing JDK requires Oracle account. Please enter your username/password'. At the bottom are 'Save' and 'Apply' buttons.

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:

Global Tool Configuration

Maven Configuration

Default settings provider

Default global settings provider

JDK

JDK installations

 **JDK**
Name

JAVA_HOME

Install automatically



Delete JDK

Add JDK

List of JDK installations on this system

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:

Jenkins > Global Tool Configuration

Git

Git installations

 **Git**
Name
Path to Git executable

Install automatically

Delete Git

Add Git

Gradle

Gradle installations

Add Gradle

List of Gradle installations on this system

Ant

Ant installations

Add Ant

List of Ant installations on this system

Maven

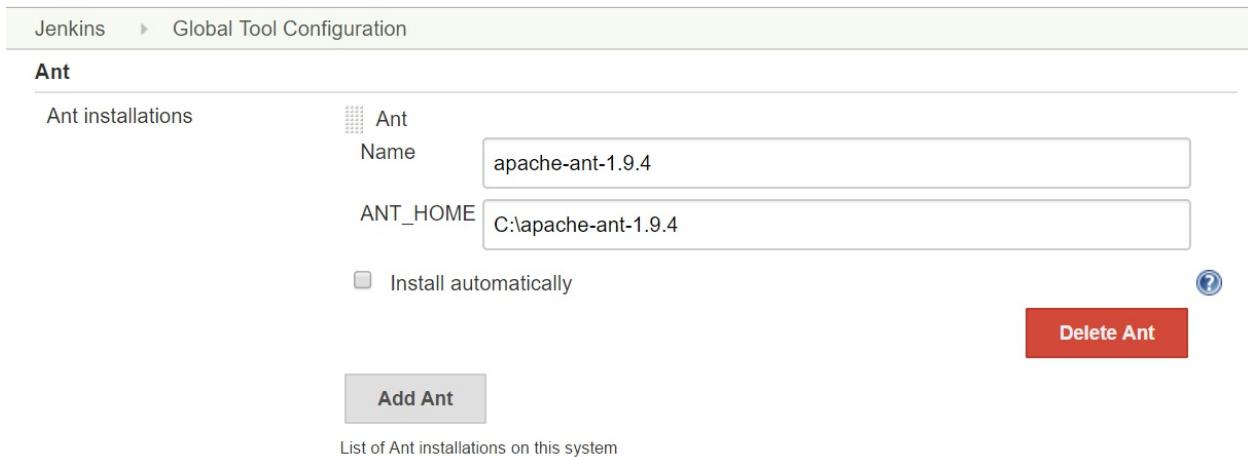
Maven installations

Add Maven

Save **Apply**

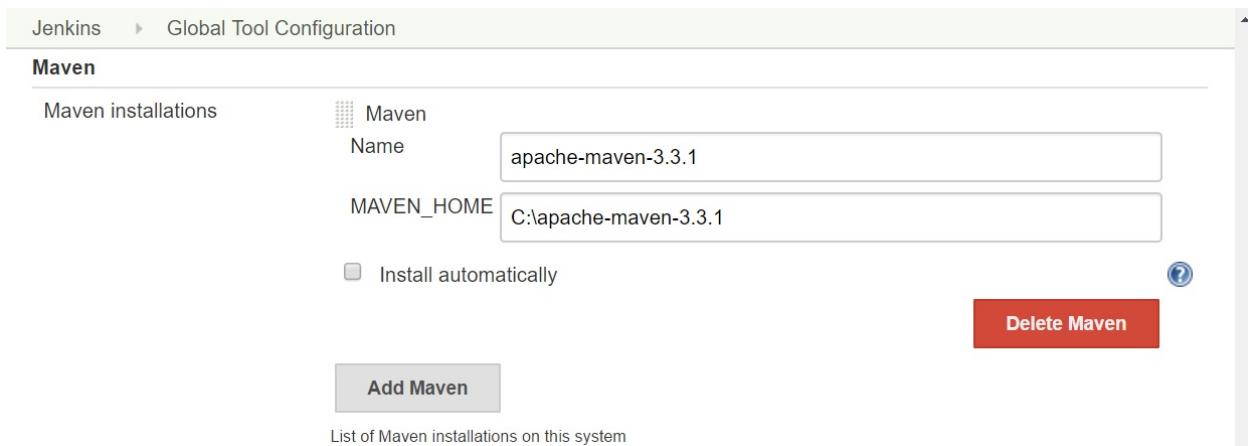
5. Click on **Add Ant** and provide a **Name** and **ANT_HOME** location. The value that we give in the **Name** box will be used in build job to identify the Ant version we want to use. This is similar practice for any tool that we will

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 Ant. At the top, there's a breadcrumb navigation: Jenkins > Global Tool Configuration. Below it, a section titled "Ant" contains a table for managing Ant installations. A single entry is listed: "Name" is "apache-ant-1.9.4" and "ANT_HOME" is "C:\apache-ant-1.9.4". There's a checkbox for "Install automatically" which is unchecked. On the right side of the entry are a blue help icon and a red "Delete Ant" button. Below the table is a grey "Add Ant" button. A note at the bottom says "List of Ant installations on this system".

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 Maven. The layout is identical to the Ant configuration above. It has a "Maven" section with a table for managing Maven installations. A single entry is listed: "Name" is "apache-maven-3.3.1" and "MAVEN_HOME" is "C:\apache-maven-3.3.1". There's a checkbox for "Install automatically" which is unchecked. On the right side of the entry are a blue help icon and a red "Delete Maven" button. Below the table is a grey "Add Maven" button. A note at the bottom says "List of Maven installations on this system".

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**:

Jenkins search admin | log out

Enter an item name

FirstJob » Required field

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 build.

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

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

OK

5. Give a **Project name**:

The screenshot shows the Jenkins 'General' configuration page for a job named 'FirstJob'. The 'General' tab is selected. The 'Project name' field contains 'FirstJob'. The 'Description' field is empty. Below it, there's a note about previewing the description in plain text or rich text. Under the 'Strategy' section, the 'Log Rotation' strategy is selected. It includes fields for 'Days to keep builds' (empty), 'Max # of builds to keep' (set to 5), and a note that only up to 5 build records will be kept. There are 'Save' and 'Apply' buttons at the bottom.

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 'Source Code Management' and 'Build Triggers' configuration pages. In the 'Source Code Management' section, the 'None' option is selected. In the 'Build Triggers' section, the 'Build periodically' option is selected, and the 'Schedule' field contains '52 8 * * *'. A warning message below the schedule notes: 'Would last have run at Saturday, 20 May, 2017 8:52:55 AM IST; would next run at Sunday, 21 May, 2017 8:52:55 AM IST.' There are 'Save' and 'Apply' buttons at the bottom.

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:

The screenshot shows the Jenkins configuration interface for a job. The top navigation bar includes tabs for General, Source Code Management, Build Triggers, Build Environment (which is selected), Build, and Post-build Actions. The Build Environment tab contains several checkboxes: Delete workspace before build starts, Abort the build if it's stuck, Add timestamps to the Console Output, and Use secret text(s) or file(s). Below this is a section titled "Build" with a dropdown menu labeled "Add build step". The dropdown menu is open, showing options: Execute Windows batch command (which is highlighted with a blue selection bar), Execute shell, Invoke Ant, Invoke Gradle script, Invoke top-level Maven targets, Run with timeout, and Set build status to "pending" on GitHub commit. At the bottom of the "Build" section are two buttons: "Save" and "Apply".

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

General Source Code Management Build Triggers Build Environment **Build** Post-build Actions

Build

Execute Windows batch command

Command `ipconfig`

See [the list of available environment variables](#)

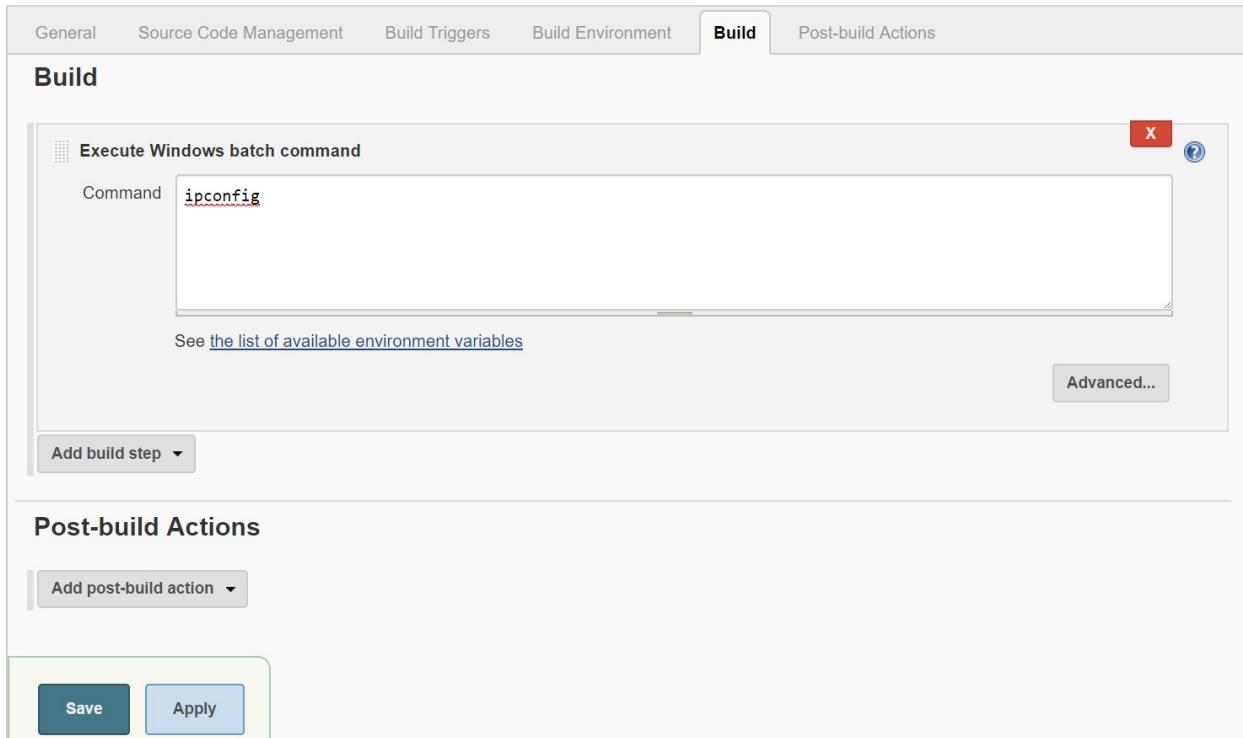
[Advanced...](#)

Add build step ▾

Post-build Actions

Add post-build action ▾

[Save](#) [Apply](#)



11. Click on **Build** now:

Project FirstJob

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

[Workspace](#)

[Recent Changes](#)

Build History

[x](#)

[trend](#)

 #1 May 21, 2017 8:52 AM

[RSS for all](#) [RSS for failures](#)

Permalinks

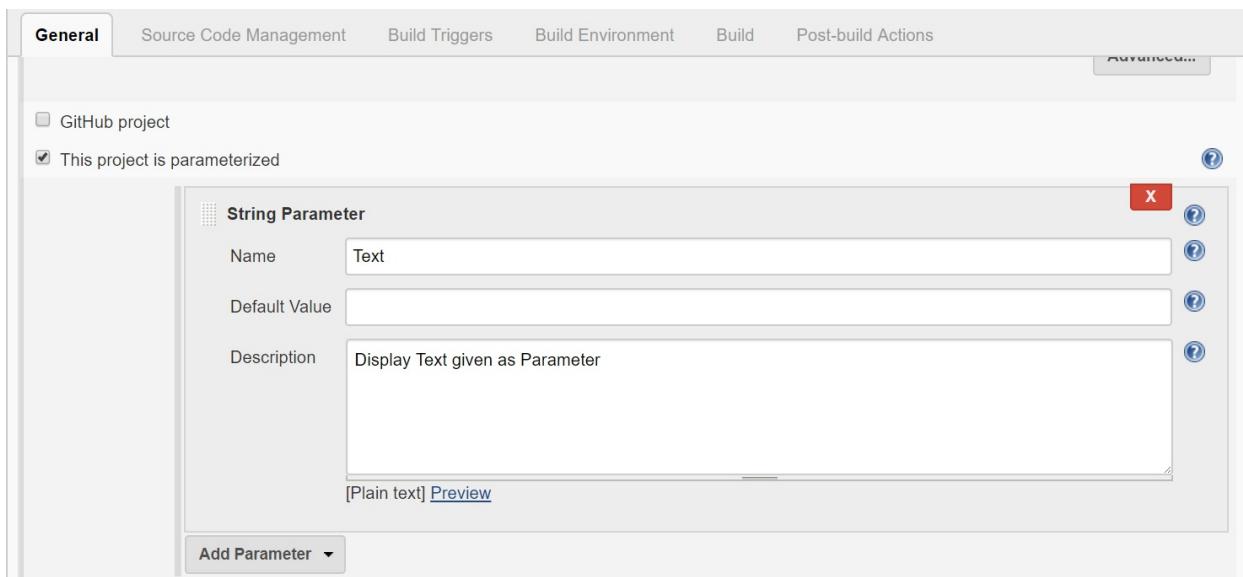
12. Observe the **Build History** and click on the blue ball to go to **Console Output**. Bingo! We have created our first job in Jenkins:

The screenshot shows the Jenkins interface. At the top, there's a navigation bar with the Jenkins logo, the project name 'FirstJob', and the build number '#1'. On the right side of the bar are links for 'admin' and 'log out'. Below the navigation bar, there's a sidebar on the left 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 build log, starting with 'Started by timer' and 'Building in workspace F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob [FirstJob] \$ cmd /c call C:\Users\Mitesh\AppData\Local\Temp\jenkins6369707303111927477.bat'. The log then shows the output of the 'ipconfig' command, detailing network adapter configurations for Ethernet, VirtualBox Host-Only Network, and 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**:

The screenshot shows the Jenkins 'General' configuration page for a project. At the top, there are tabs for 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build', and 'Post-build Actions'. The 'General' tab is selected. In the 'General' section, there are several checkboxes: 'GitHub project' (unchecked), 'This project is parameterized' (checked), 'Throttle builds' (unchecked), 'Disable this project' (unchecked), and 'Execute concurrent builds if necessary' (unchecked). Below these checkboxes, there's a section for 'Source Code Management' with a radio button set to 'None'. To the right of the checkboxes, there's a 'Parameters...' button. A dropdown menu is open over this button, titled 'Add Parameter'. The menu lists several parameter types: 'Boolean Parameter', 'Choice Parameter', 'Credentials Parameter', 'File Parameter', 'List Subversion tags (and more)', 'Multi-line String Parameter', 'Password Parameter', 'Run Parameter', and 'String Parameter'. The 'String Parameter' option is highlighted with a blue selection bar and has a small mouse cursor icon pointing at it.

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**:

Project FirstJob

[Workspace](#)

[Recent Changes](#)

Permalinks

- [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](#)
- [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 'FirstJob' project. On the left, there's a sidebar with links like 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build with Parameters', 'Delete Project', and 'Configure'. The main area is titled 'Project FirstJob' and displays a message: 'This build requires parameters:'. Below this is a text input field with the value 'Hello etutorialsworld.com!' and a placeholder 'Display Text given as Parameter'. A large blue 'Build' button is centered below the input field. To the left of the main content, there's a 'Build History' section showing four previous builds, each with a status icon and a timestamp: May 21, 2017 8:57 AM (#4), May 21, 2017 8:55 AM (#3), May 21, 2017 8:55 AM (#2), and May 21, 2017 8:52 AM (#1). At the bottom of the history section are two RSS feed links: 'RSS for all' and 'RSS for failures'.

18. Verify the `Console Output`:

Console Output

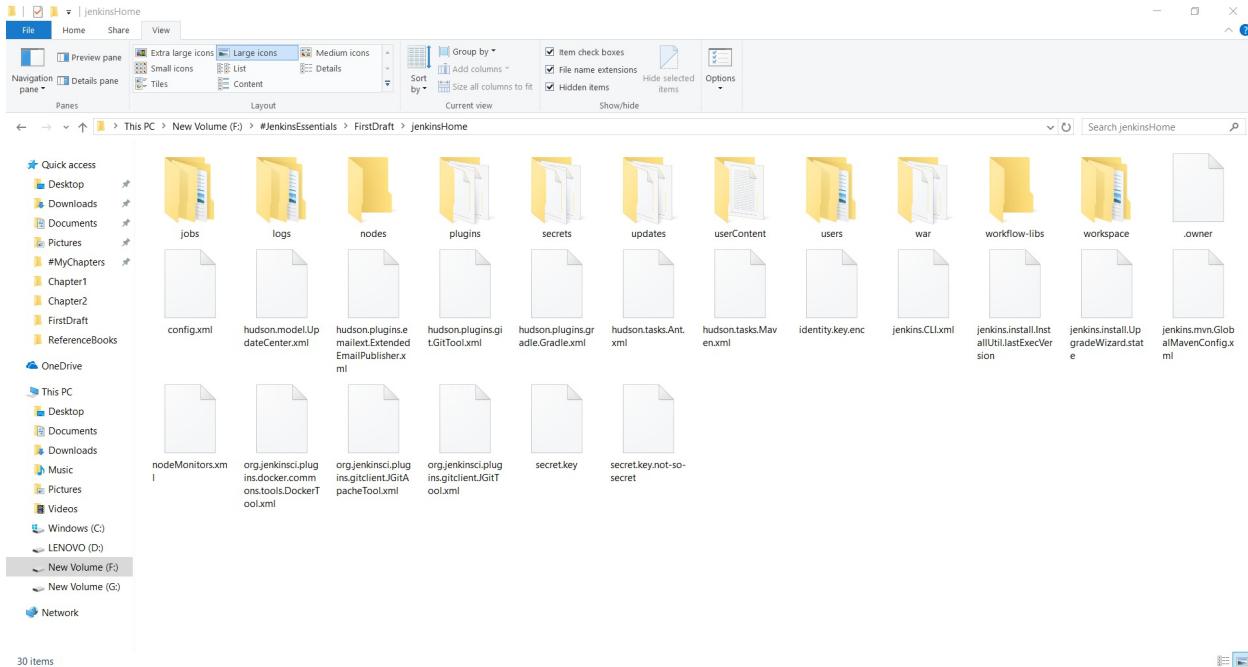
```
Started by user admin
Building in workspace F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob
[FirstJob] $ cmd /c call C:\Users\Mitesh\AppData\Local\Temp\jenkins3630239299526944525.bat

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

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

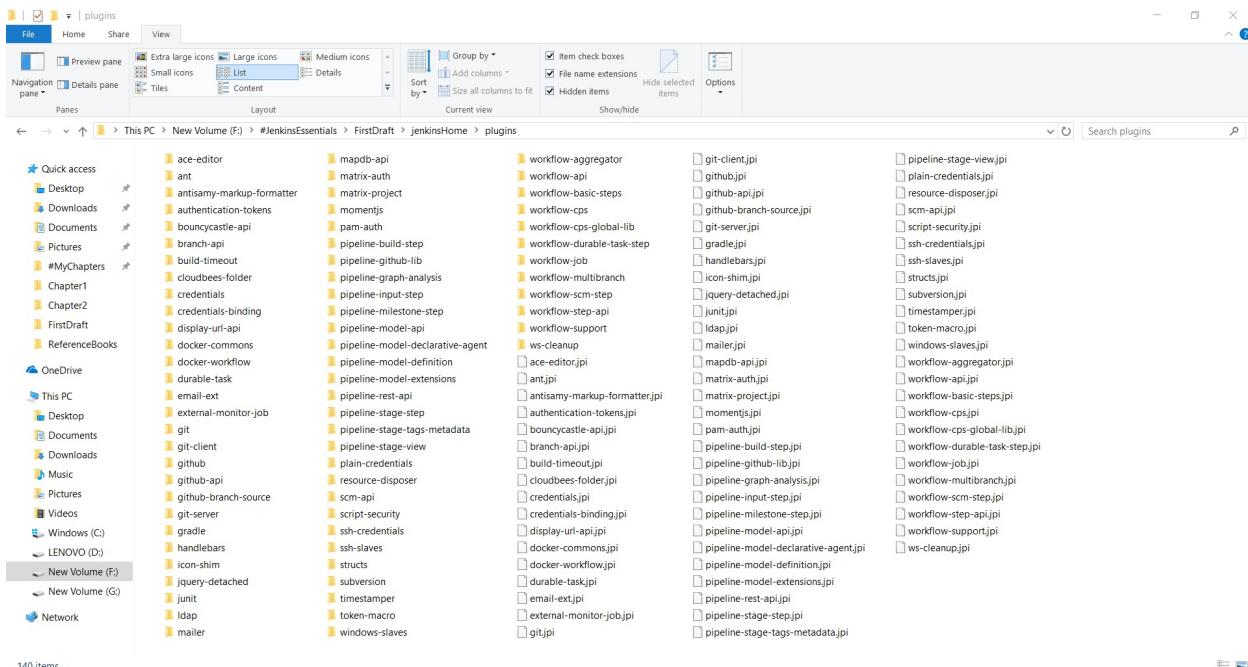
F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\FirstJob>exit 0
Finished: SUCCESS
```

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



All the Directories available in JENKINS_HOME folder

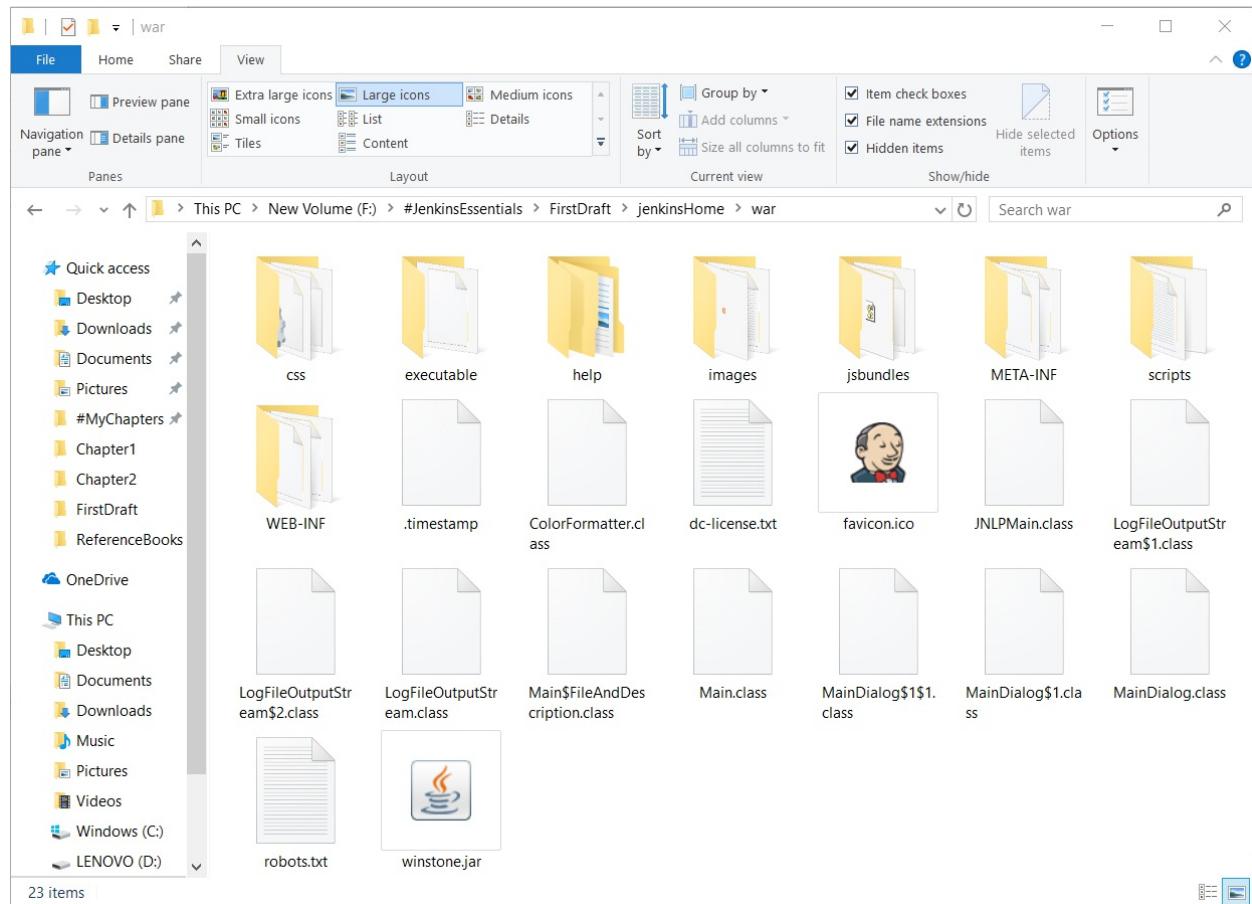
20. In the plugins directory, all installed plugins are available:



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:

```
root@localhost:~/git/development/AntExample1
File Edit View Search Terminal Tabs Help
root@localhost:~/git/development/AntExa... X root@localhost:/usr X
[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]#
```

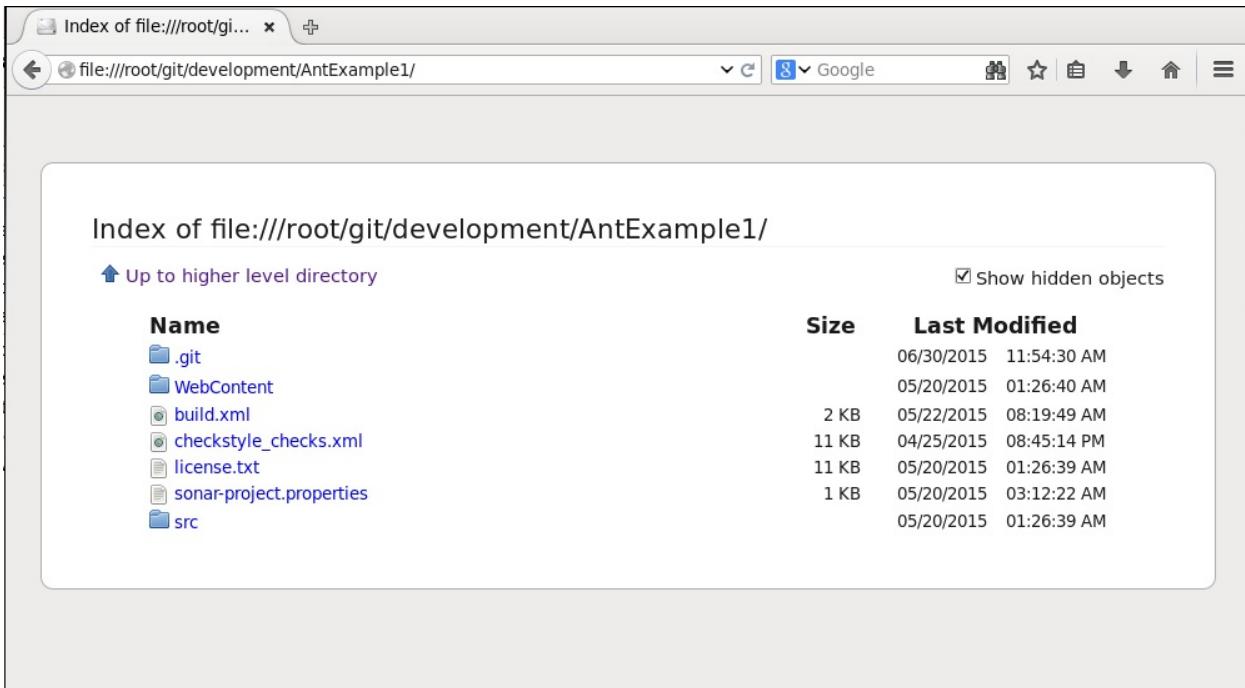
10. Commit by executing `git commit -m "Initial Commit" -a`:

```
root@localhost:~/git/development/AntExample1
File Edit View Search Terminal Tabs Help
root@localhost:~/git/development/AntExa...  X root@localhost:/usr
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.jar
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:

Name	Size	Last Modified
.git	1 KB	02/23/2015 11:22:31 AM
AntExample1	1 KB	02/23/2015 11:22:31 AM
HEAD	1 KB	02/23/2015 11:22:31 AM
branches	1 KB	02/23/2015 11:22:31 AM
config	1 KB	02/23/2015 11:22:31 AM
description	1 KB	02/23/2015 11:22:31 AM
hooks	1 KB	02/23/2015 11:22:31 AM
info	1 KB	02/23/2015 11:22:31 AM
objects	1 KB	02/23/2015 11:22:31 AM
refs	1 KB	02/23/2015 11:22:31 AM

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:

Enabled	Name ↓	Version	Previously installed version	Pinned	Uninstall
<input checked="" type="checkbox"/>	GitHub API Plugin This plugin provides GitHub API for other plugins.	1.67		Downgrade to 1.67	Uninstall
<input checked="" type="checkbox"/>	GitHub plugin This plugin integrates GitHub to Jenkins.	1.11.3		Downgrade to 1.8	Uninstall

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

Item name: AntExampleGit

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 build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

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. See the documentation for more details.

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

Copy existing Item
Copy from:

Build Queue: No builds in the queue.

Build Executor Status: master (1 Idle, 2 Idle), WindowsNode (offline)

OK

5. Configure Git in the **Source Code Management** section:

Jenkins > AntExampleGit > configuration

Advanced Project Options

Source Code Management

None
 CVS
 CVS Projectset
 Git

Repositories Repository URL: file:///root/git/development/AntExample1/

Credentials - none -

Branches to build Branch Specifier (blank for 'any') */master

Repository browser (Auto)

Additional Behaviours

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

Source Code Management

None
 File System
 Git

Repositories

Repository URL: https://github.com/mitesh51/AntExample.git

Branches to build

Branch Specifier (blank for 'any') */master

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

Jenkins > AntExampleGit > configuration

Generic-Artifactory Integration
Gradle-Artifactory Integration
Inject environment variables to the build process
Inject passwords to the build as environment variables
Maven3-Artifactory Integration
Run buildstep before SCM runs

Build

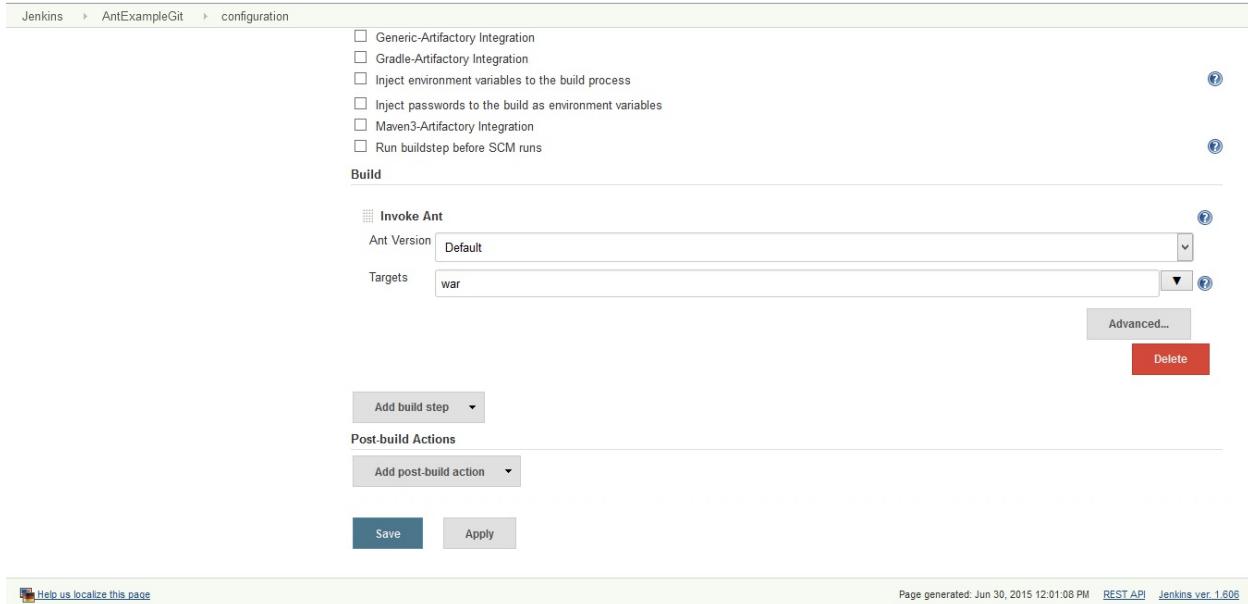
Invoke Ant
Ant Version Default
Targets war

Add build step Advanced... Delete

Post-build Actions
Add post-build action

Save Apply

Help us localize this page Page generated: Jun 30, 2015 12:01:08 PM REST API Jenkins ver. 1.606



8. Execute the build:

Jenkins

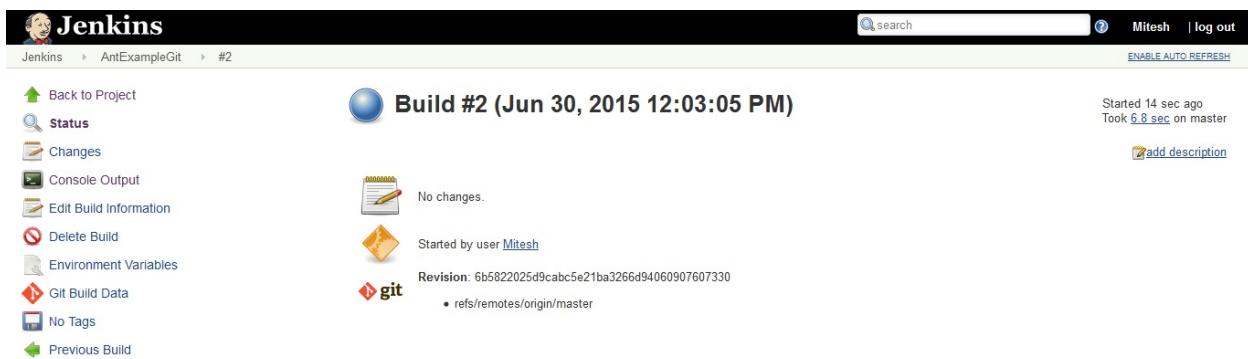
Back to Project Status Changes Console Output Edit Build Information Delete Build Environment Variables Git Build Data No Tags Previous Build

Build #2 (Jun 30, 2015 12:03:05 PM)

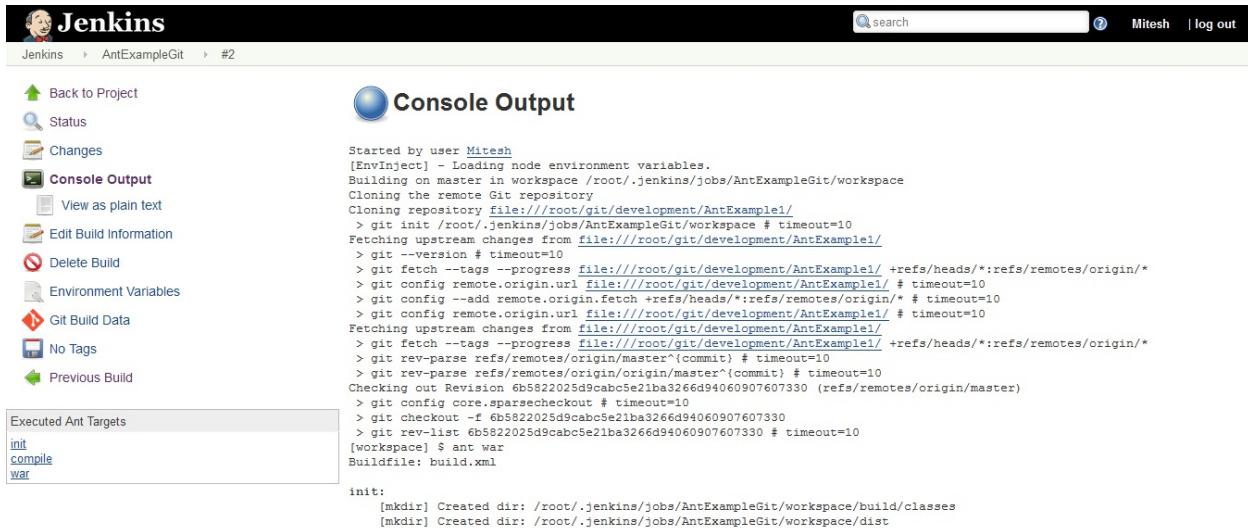
No changes.

Started by user Mitesh
Revision: 6b5822025d9cabc5e21ba3266d94060907607330
refs/remotes/origin/master

Started 14 sec ago Took 6.8 sec on master Add description



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



The screenshot shows the Jenkins interface for a build job named 'AntExampleGit'. The left sidebar contains links for Back to Project, Status, Changes, Console Output (which is selected), View as plain text, Edit Build Information, Delete Build, Environment Variables, Git Build Data, No Tags, and Previous Build. A section titled 'Executed Ant Targets' lists 'init', 'compile', and 'war'. The main content area is titled 'Console Output' and displays the command-line output of the build process. The output shows the execution of Ant targets, cloning a Git repository, and building the project.

```
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/AntExample1/
> git init /root/.jenkins/jobs/AntExampleGit/workspace # timeout=10
Fetching upstream changes from file:///root/git/development/AntExample1/
> git --version # timeout=10
> git fetch --tags --progress file:///root/git/development/AntExample1/ +refs/heads/*:refs/remotes/origin/*
> git config remote.origin.url file:///root/git/development/AntExample1/ # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url file:///root/git/development/AntExample1/ # timeout=10
Fetching upstream changes from file:///root/git/development/AntExample1/
> git fetch --tags --progress file:///root/git/development/AntExample1/ +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/master#(commit) # timeout=10
Checking out Revision 6b5822025d9cabc5e21ba3266d94060907607330 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 6b5822025d9cabc5e21ba3266d94060907607330
> git rev-list 6b5822025d9cabc5e21ba3266d94060907607330 # timeout=10
[workspace] $ ant war
Buildfile: build.xml

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

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

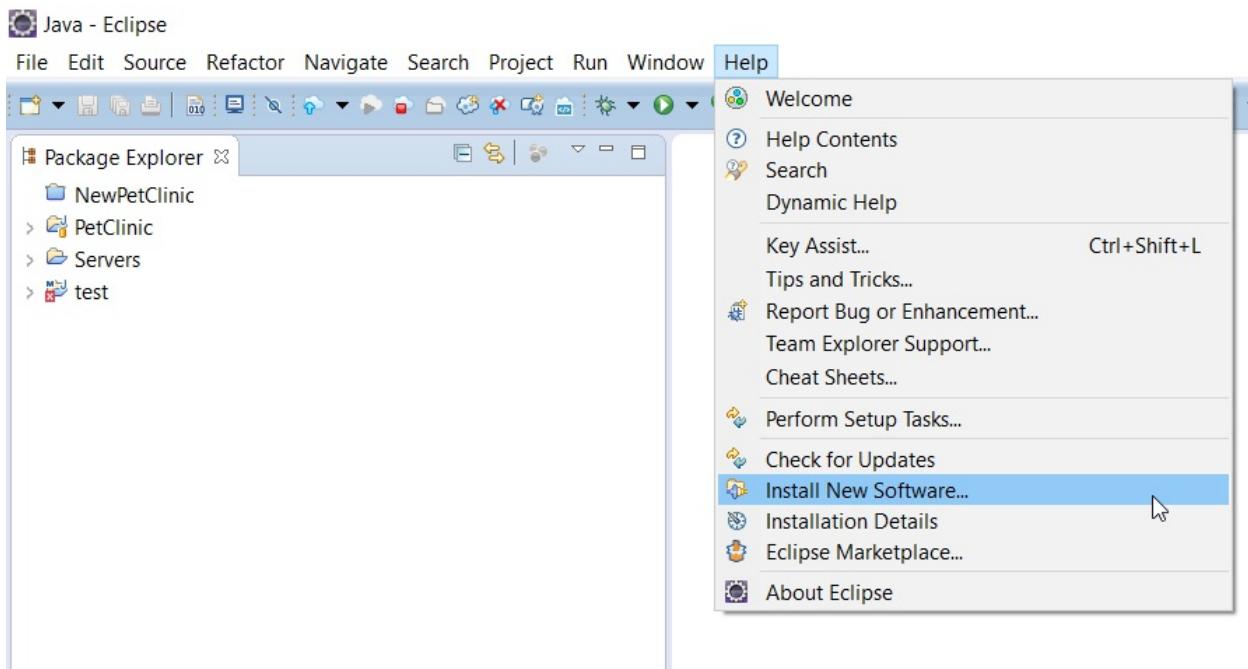
Done!

Eclipse and Jenkins integration

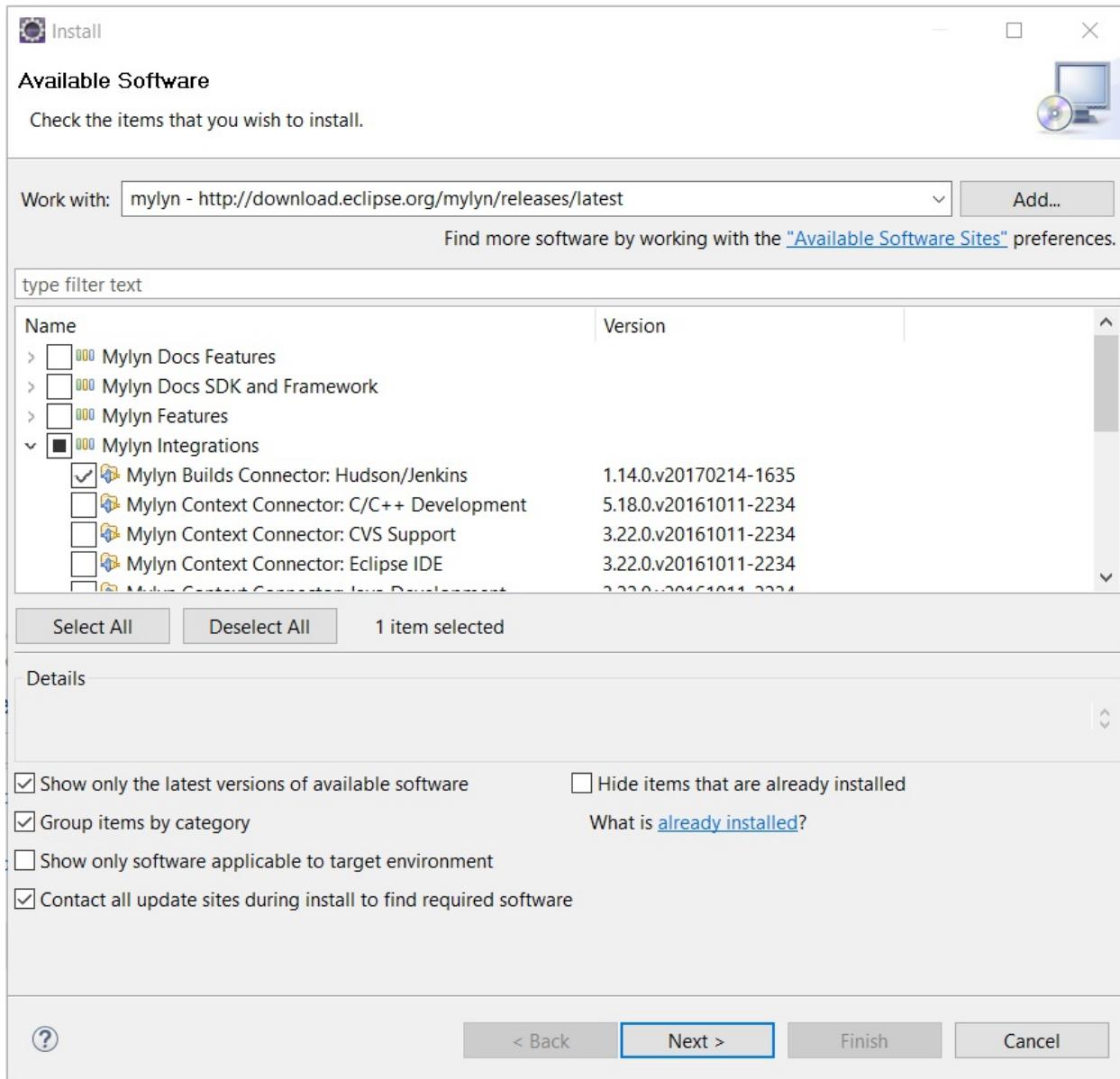
Can we execute a Jenkins job from Eclipse?

Yes, by following these steps:

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

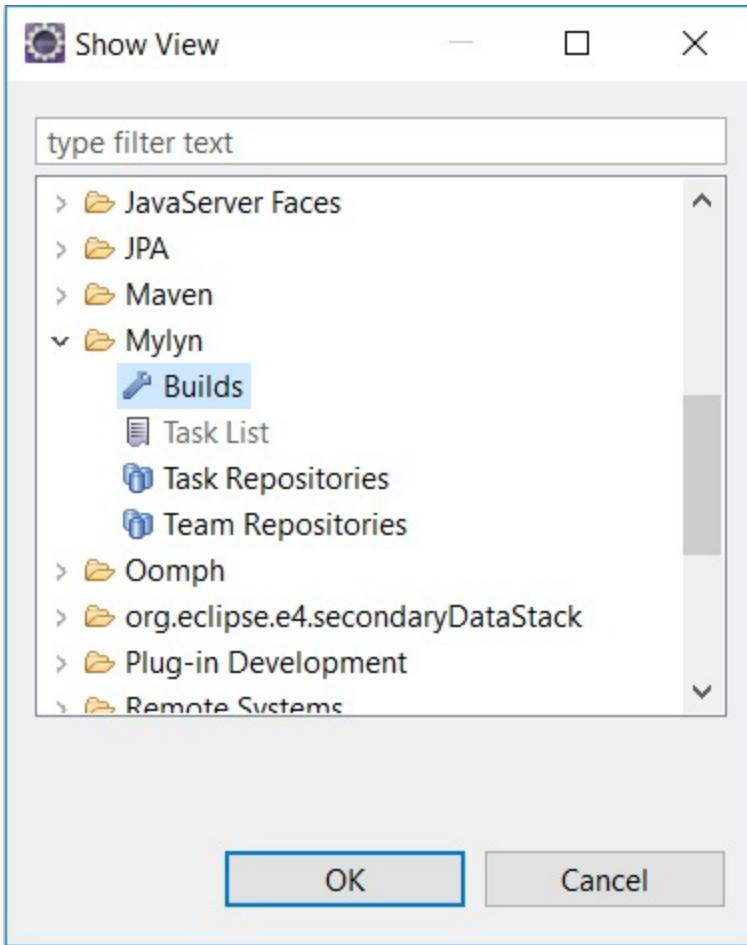


2. Add a site for Mylyn and click on **Next**:

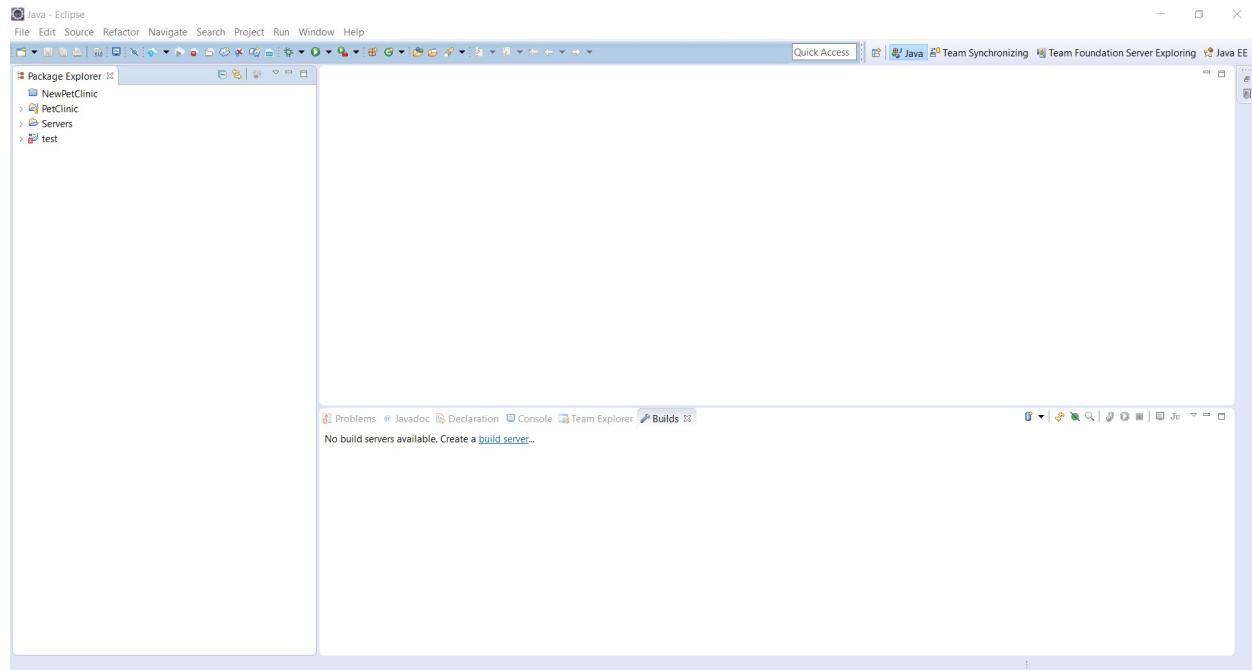


3. Review the items to be installed and click on **Next**.
4. Accept the terms of the license agreement.
5. Click on **Finish**.
6. It will start installing the **Mylyn** package. Once it is finished, restart Eclipse.

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

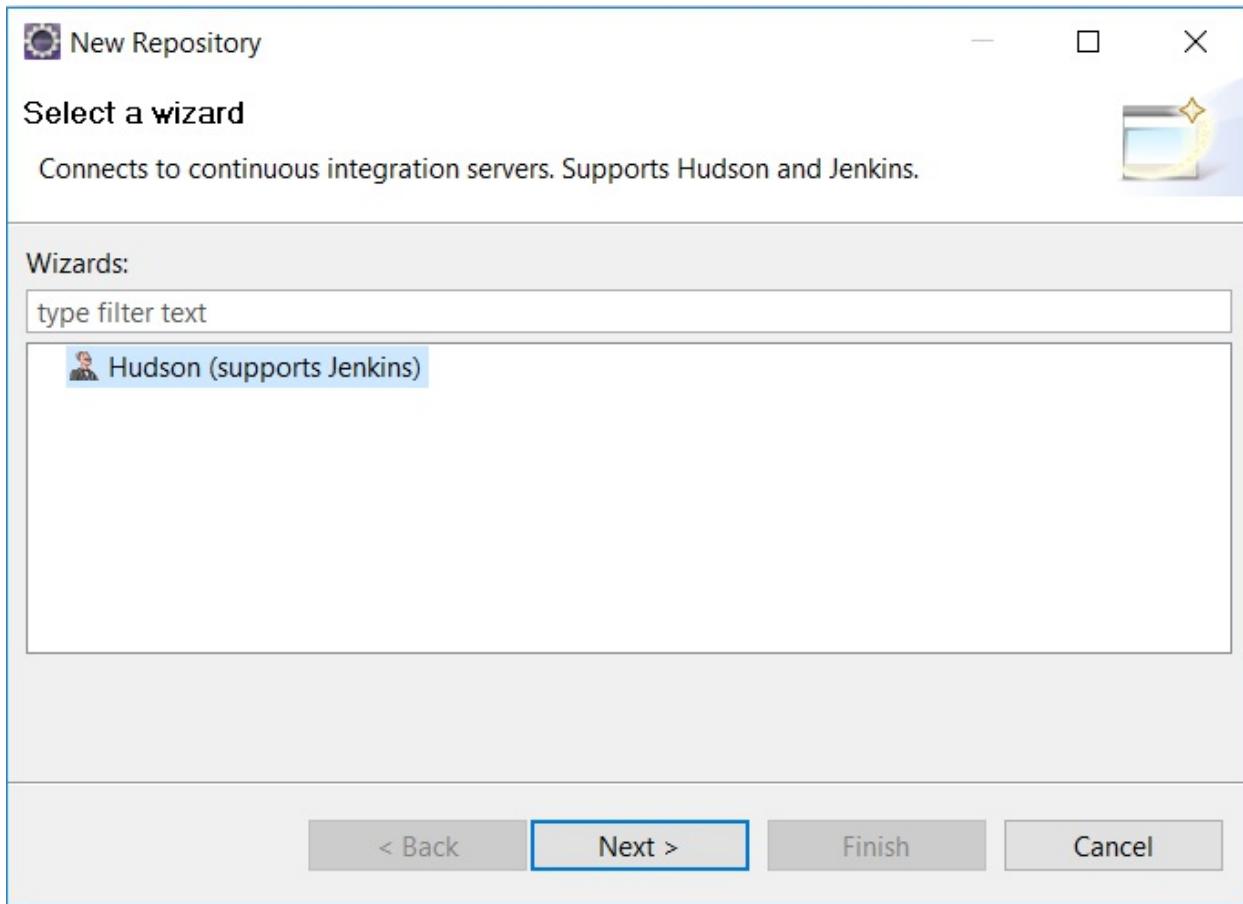


10. In the **Builds** section, click on the **build server** link:

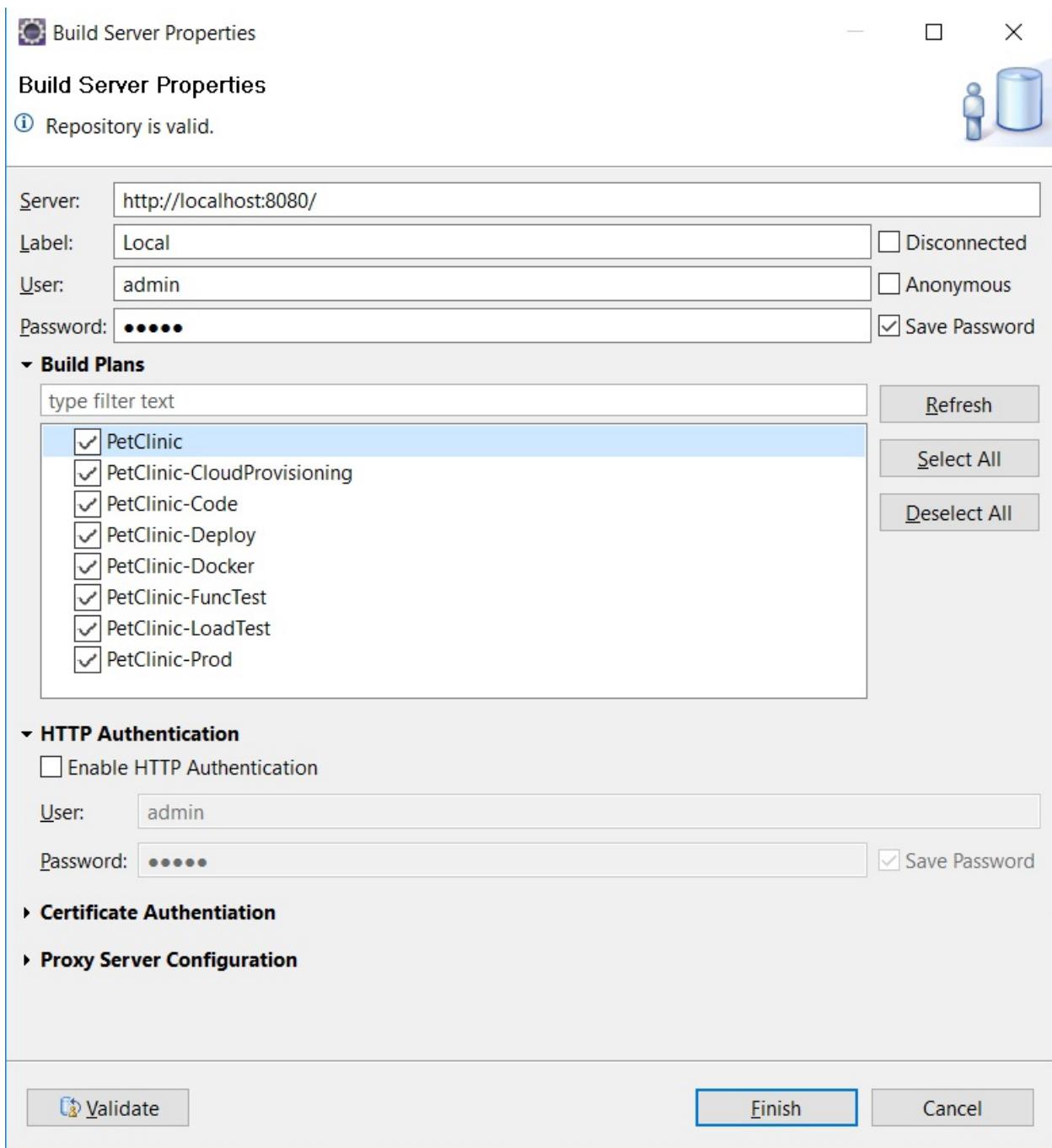


Build Section is Eclipse IDE with No build server configured

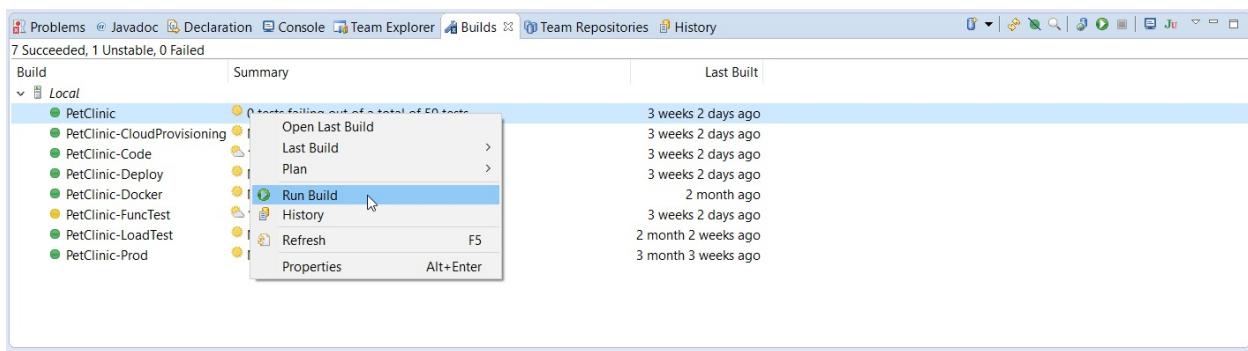
11. Select **Hudson (supports Jenkins)** and click on **Next**:



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.

Chapter 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.

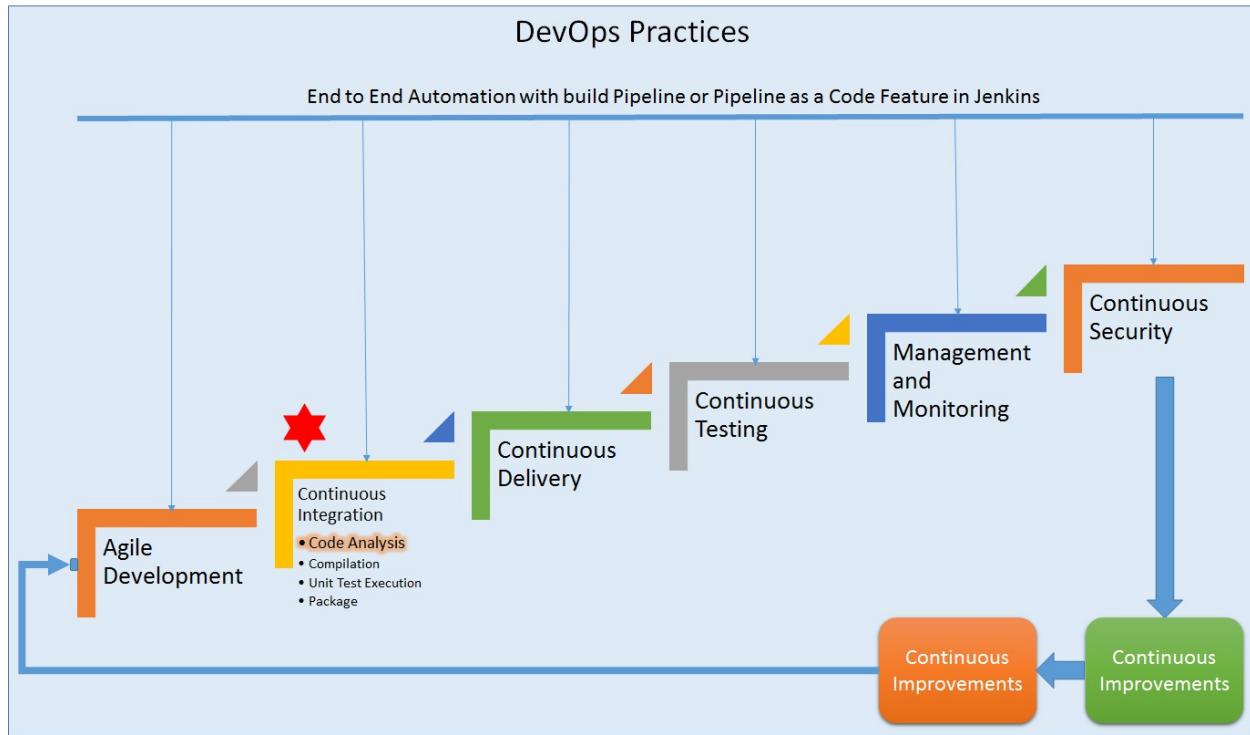
We will start our journey with Continuous Code Qualityâstatic code analysis- and it will be followed by Continuous Integration, Continuous Delivery, Continuous Testing, Continuous Deployment, Continuous Monitoring, and Continuous Security. For static code analysis, we will use **SonarQube** to analyze 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:

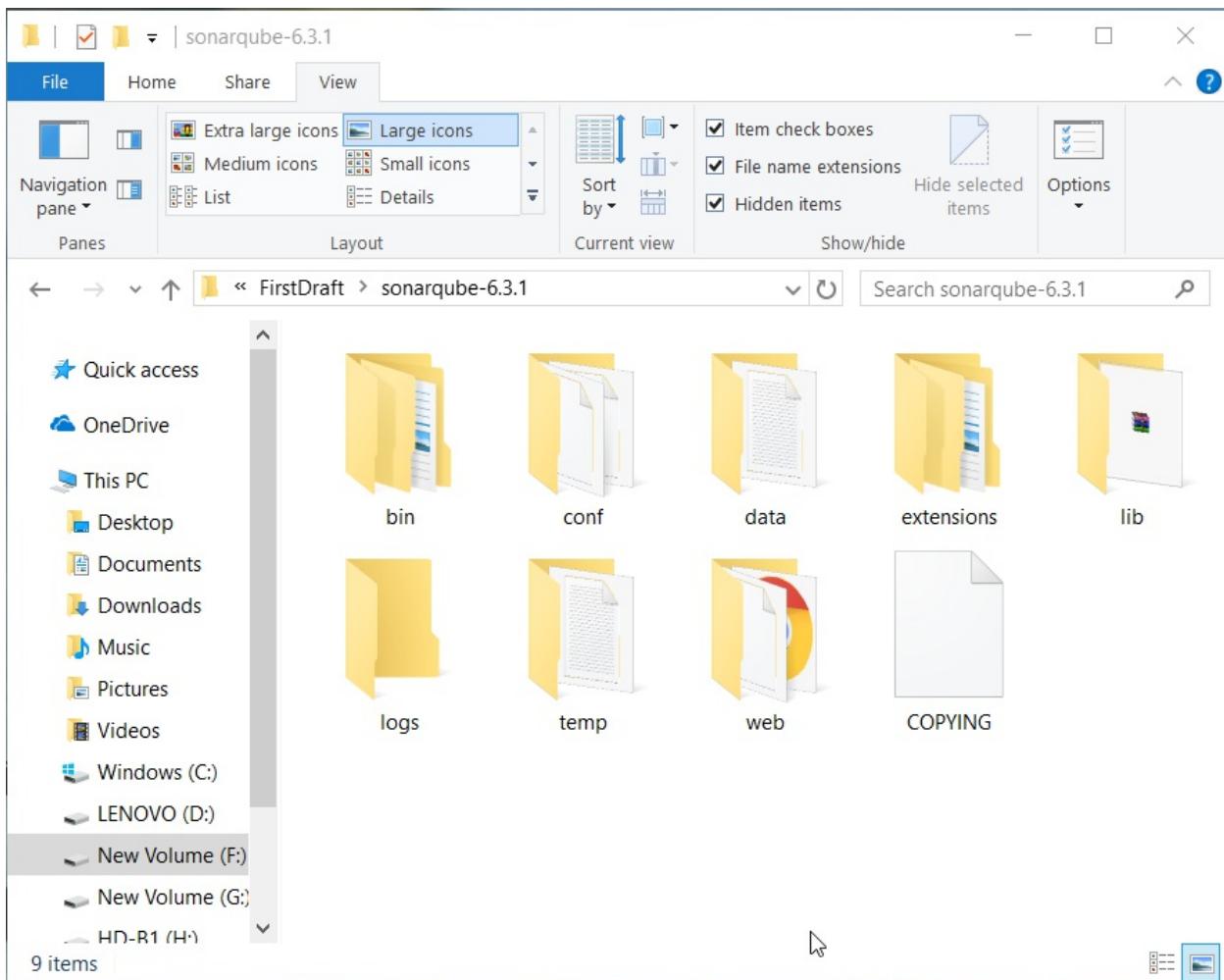


At the end of this chapter, we will know how to configure a SonarQube server with Jenkins and perform static code analysis.

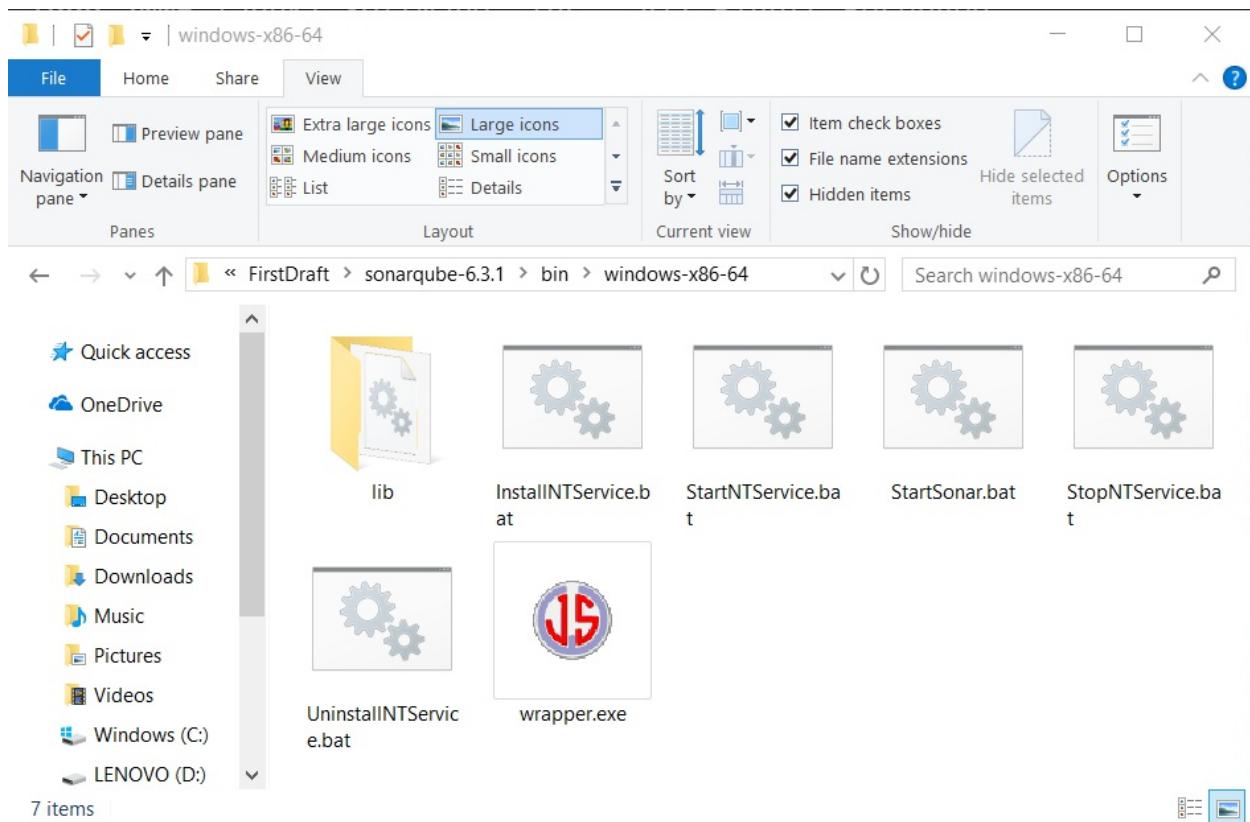
Jenkins 2.x integration with Sonar 6.3

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

SonarQube

localhost:9000/about

sonarqube Projects Issues Rules Quality Profiles Quality Gates Log in

Continuous Code Quality

0 Bugs

0 Vulnerabilities

0 Code Smells

Projects Analyzed

Log in Read documentation

Multi-Language

20+ programming languages are supported by SonarQube thanks to our in-house code analyzers, including:

Java C/C++ C# COBOL ABAP HTML RPG JavaScript Objective C XML
VB.NET PL/SQL Flex Python Groovy PHP Swift Visual Basic PL/I

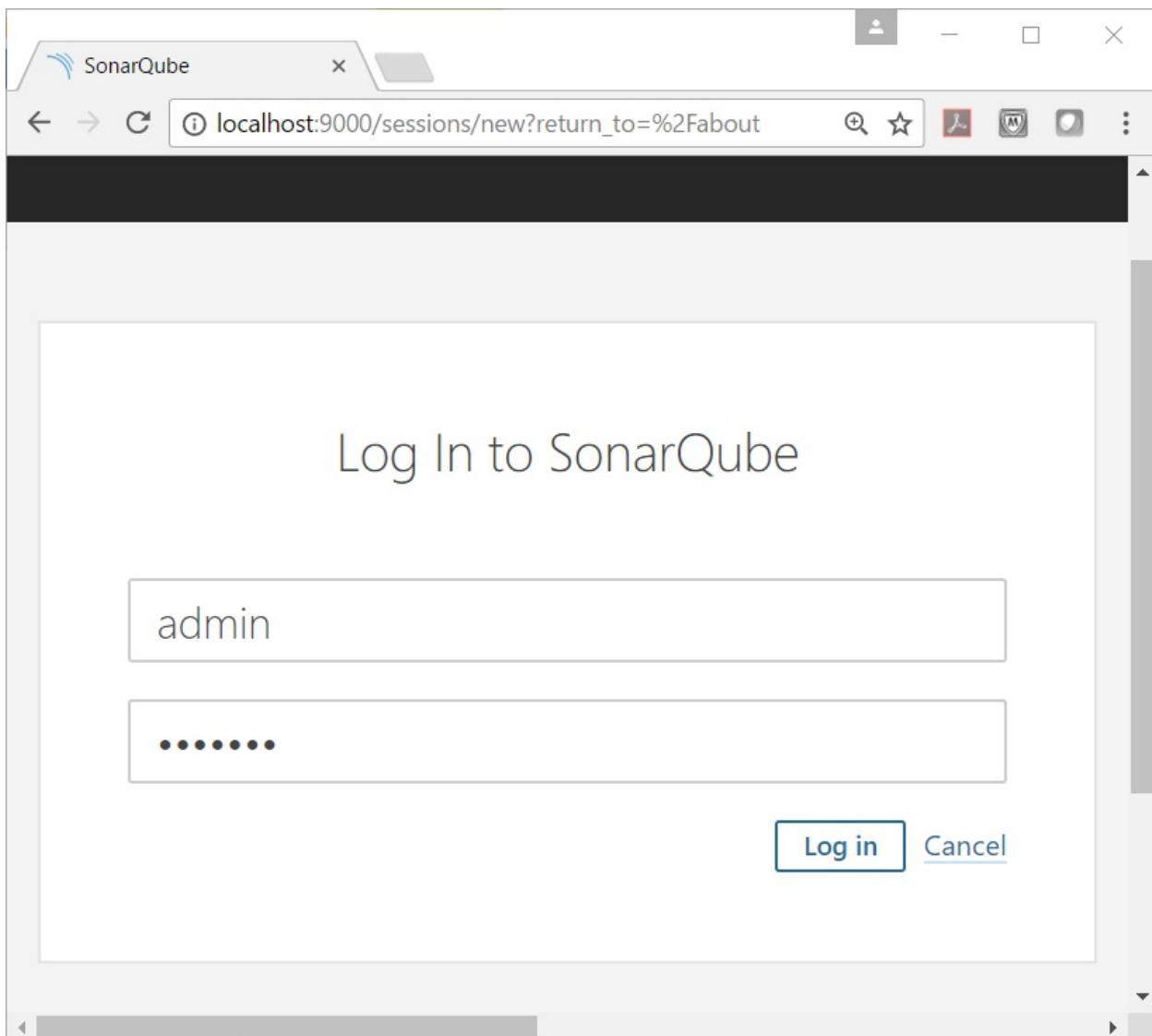
Quality Model

BUGS: Bugs track code that is demonstrably wrong or highly likely to yield unexpected behavior.

VULNERABILITIES: Vulnerabilities are raised on code that is potentially vulnerable to exploitation by hackers.

CODE SMELLS: Code Smells will confuse maintainers or give them pause. They are 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:

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**:

Language	Profile	Projects	Rules	Updated	Used
C#	Sonar way	Default	134	Never	Never
Flex	Sonar way	Default	60	Never	Never
Java	Sonar way	Default	277	Never	Never
JavaScript	Sonar way	Default	277	Never	Never

Recently Added Rules

- Skipped unit tests should be either removed or fixed. C#, not yet activated
- Failed unit tests should be fixed. C#, not yet activated
- Source files should not have any duplicated blocks. C#, not yet activated
- Source files should have a sufficient density of comments. C#, 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 blocks. 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

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

SonarQube way

Conditions

Only project measures are checked against thresholds. Sub-projects, directories and files are ignored. [More](#)

Metric	Over Leak Period	Operator	Warning	Error
Coverage on New Code	Always	is less than	80	Update Delete
Maintainability Rating on New Code	Always	is worse than	A X	Update Delete
Reliability Rating on New Code	Always	is worse than	A X	Update Delete
Security Rating on New Code	Always	is worse than	A X	Update Delete

[Add Condition](#)

Projects

You must not select specific projects for the default quality gate.

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

Rules

1 / 397 rules

Reload New Search Bulk Change

Language

- Java 397
- Python 238
- C# 189
- JavaScript 184
- PHP 126
- Flex 79
- Search

Type

- Bug 149
- Vulnerability 31
- Code Smell 217
- Tag
- Repository
- Default Severity
- Status
- Available Since

".equals()" should not be used to test the values of "Atomic" classes	Java	Bug	multi-threading
"@Deprecated" code should not be used	Java	Code Smell	cert, cwe, obsolete
"@NonNull" values should not be set to null	Java	Bug	
"@Override" should be used on overriding and implementing methods	Java	Code Smell	bad-practice
"action" mappings should not have too many "forward" entries	Java	Code Smell	brain-overload, struts
"Arrays.stream" should be used for primitive arrays	Java	Bug	performance
"assert" should only be used with boolean variables	Java	Bug	cert, suspicious
"BigDecimal(double)" should not be used	Java	Bug	cert
"catch" clauses should do more than rethrow	Java	Code Smell	cert, clumsy, finding, unused
"clone" should not be overridden	Java	Code Smell	suspicious
"Cloneables" should implement "clone"	Java	Bug	
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	Java	Code Smell	obsolete, pitfall
"compareTo" results should not be checked for specific values	Java	Bug	unpredictable
"compareTo" should not return "Integer.MIN_VALUE"	Java	Bug	

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

The screenshot shows the SonarQube interface for managing quality profiles. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles (which is the active tab), Quality Gates, and Administration. On the right, there are user icons for Administrator, a search bar, and a refresh button.

The main content area displays the 'Sonar way' profile details. It includes a summary table for rules:

Rules	Active	Inactive
Total	277	120
Bugs	129	20
Vulnerabilities	18	13
Code Smells	130	87

A blue button labeled 'Activate More' is located at the bottom of this table.

Below the table, there are sections for 'Inheritance' and 'Projects'. The 'Inheritance' section shows 'Sonar way' with '277 active rules'. A 'Change Parent' button is available. The 'Projects' section states that every project not specifically associated with this profile will be associated by default. A note at the bottom of this section says: 'Default Every project not specifically associated with a quality profile will be associated to this one by default.'

A red warning box at the bottom left states: '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.'

At the bottom right, footer information includes: '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 tabs include 'Update Center [Jenkins]', 'Jenkins Plugins', 'Projects - SonarQube', and 'New Tab'. The main title bar says 'Jenkins' with a count of '1' in a red box. The top navigation bar includes a search field, user 'admin', and 'log out'. Below the navigation is a breadcrumb 'Jenkins > Plugin Manager'. On the left, there are links for 'Back to Dashboard' and 'Manage Jenkins'. A search bar at the top right has the placeholder 'Filter: Sonarqube'. Below the search bar are four tabs: 'Updates' (disabled), 'Available' (selected), 'Installed', and 'Advanced'. The main content area shows a table with columns 'Install ↓', 'Name', and 'Version'. One row is selected: 'SonarQube Scanner for Jenkins' (version 2.6.1). A tooltip for 'Mashup Portlets' explains it as a group of portlets including Generic JS Portlet, Recent Changes Portlet, SCM changes, SonarQube Portlets, and Test Results Portlet. At the bottom are buttons for 'Install without restart' (disabled), 'Download now and install after restart', and 'Check now'. A status message says 'Update information obtained: 3 days 10 hr ago'. The footer contains the page generation time 'Page generated: May 25, 2017 11:46:58 PM IST', links to 'REST API' and 'Jenkins ver. 2.61', and a copyright notice '© 2017 Jenkins Software'.

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 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

Install without restart Download now and install after restart Check now

Update information obtained: 3 days 10 hr ago

Page generated: May 25, 2017 11:46:58 PM IST REST 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 configuration interface for SonarQube servers. At the top, there are tabs for 'Configure System' (Jenkins) and 'Projects - SonarQube'. The main content area is titled 'SonarQube servers' and contains two sections: 'Environment variables' and 'SonarQube installations'. Under 'SonarQube installations', there is a single entry for 'Sonarqube6.3' with fields for 'Server URL' (http://localhost:9000), 'Server version' (5.3 or higher), and 'Server authentication token'. Below this, there are fields for 'SonarQube account login' and 'SonarQube account password', both of which are currently empty. A red 'Delete SonarQube' button is visible. At the bottom left 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 'sonarqube', 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. The 'Administration' tab is active. Below it, a secondary navigation bar has 'Configuration' selected. A dropdown menu for 'Security' is open, showing options: 'Users' (which is highlighted with a cursor icon), 'Groups', 'Global Permissions', and 'Permission Templates'. A tooltip for 'Users' states: 'Manage users for this instance.'

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

The screenshot shows the SonarQube Administration interface under the Security tab. In the 'Tokens' section, there is a table with one row. The table has three columns: 'SCM Accounts' (containing 'Administrator admin'), 'Groups' (containing 'sonar-administrators', 'sonar-users'), and 'Tokens' (containing '0' and three icons). A 'Create User' button is located at the top right of the 'Tokens' section.

21. Click on **Tokens**:

The screenshot shows the same SonarQube Administration interface as above, but the 'Tokens' section now has a '0' value and three icons. A 'Update Tokens' button is visible in the 'Tokens' column. The rest of the interface remains the same, including the 'Create User' button.

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

This is a screenshot of a modal dialog titled 'Tokens'. It contains a table with two columns: 'Name' and 'Created'. Below the table is a section labeled 'Generate Tokens' with a text input field containing 'JenkinsEssentials' and a blue 'Generate' button with a hand cursor icon. At the bottom right of the dialog is a 'Done' button.

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

Tokens

Name	Created	
JenkinsEssentials	May 25, 2017	Revoke

Generate Tokens

Enter Token Name Generate

New token "JenkinsEssentials" has been created. Make sure you copy it now, you won't be able to see it again!

[Copy](#) `4dc45d3a67f0c445845d53dc1d47e1e8279ff2ca`

[Done](#)

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

SCM Accounts	Groups	Tokens
Administrator admin	sonar-administrators sonar-users	1 Edit Lock Delete

1/1 shown

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

The screenshot shows the Jenkins 'Configure System' page under the 'Jenkins' section. The title bar says 'Configure System [Jenkins] > Projects - SonarQube'. The main content area is titled 'SonarQube servers'. It contains two sections: 'Environment variables' and 'SonarQube installations'. Under 'Environment variables', there is a checkbox for 'Enable injection of SonarQube server configuration as build environment variables', with a note explaining it allows job administrators to inject a SonarQube server configuration as environment variables in the build. Under 'SonarQube installations', there is a form for configuring a single instance. The fields are: 'Name' (set to 'Sonarqube6.3'), 'Server URL' (set to 'http://localhost:9000/'), 'Server version' (set to '5.3 or higher'), 'Server authentication token' (a redacted password), 'SonarQube account login' (a redacted account name), and 'SonarQube account password' (a redacted password). At the bottom are 'Save' and 'Apply' buttons.

SonarQube servers

Environment variables

Enable injection of SonarQube server configuration as build environment variables
If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name: Sonarqube6.3

Server URL: http://localhost:9000/
Default is http://localhost:9000

Server version: 5.3 or higher

Server authentication token:
Configuration fields depend on the SonarQube server version.

SonarQube account login:
SonarQube authentication token. Mandatory when anonymous access is disabled.

SonarQube account password:
SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.

SonarQube account password:
SonarQube account used to perform analysis. Mandatory when anonymous access is

Save **Apply**

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

The screenshot shows the Jenkins Global Tool Configuration page. Under the 'SonarQube Scanner' section, there is a table with one row. The row contains a SonarQube Scanner icon, the text 'SonarQube Scanner', a 'Name' field set to 'SonarQube Scanner 3.0.3', a checked checkbox for 'Install automatically', and a 'Delete Installer' button. Below this table is a 'Delete SonarQube Scanner' button. At the bottom of the page, there are sections for 'Ant' and 'Maven' with 'Installations...' buttons, and 'Save' and 'Apply' buttons.

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:

The screenshot shows the Jenkins 'Enter an item name' dialog. The input field contains 'PetClinic-Code'. Below the input field is a 'Required field' message. A list of job types is shown in a scrollable area:

- Freestyle project**: Description: 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 build.
- Maven project**: Description: Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- Pipeline**: Description: 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**: Description: 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**: Description: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

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

The screenshot shows the Jenkins interface for the 'PetClinic-Code' project. The 'Build Environment' tab is active. In the 'Build' section, a dropdown menu is open, showing various build step options. 'Execute SonarQube Scanner' is highlighted with a blue background.

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

The screenshot shows the Jenkins interface for the 'PetClinic-Code' project. The 'Build' tab is active. The 'Execute SonarQube Scanner' step is configured with 'JDK 8' selected for the JDK and 'sonar-project.properties' specified for the Path to project properties. The 'Save' and 'Apply' buttons are visible at the bottom of the configuration panel.

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

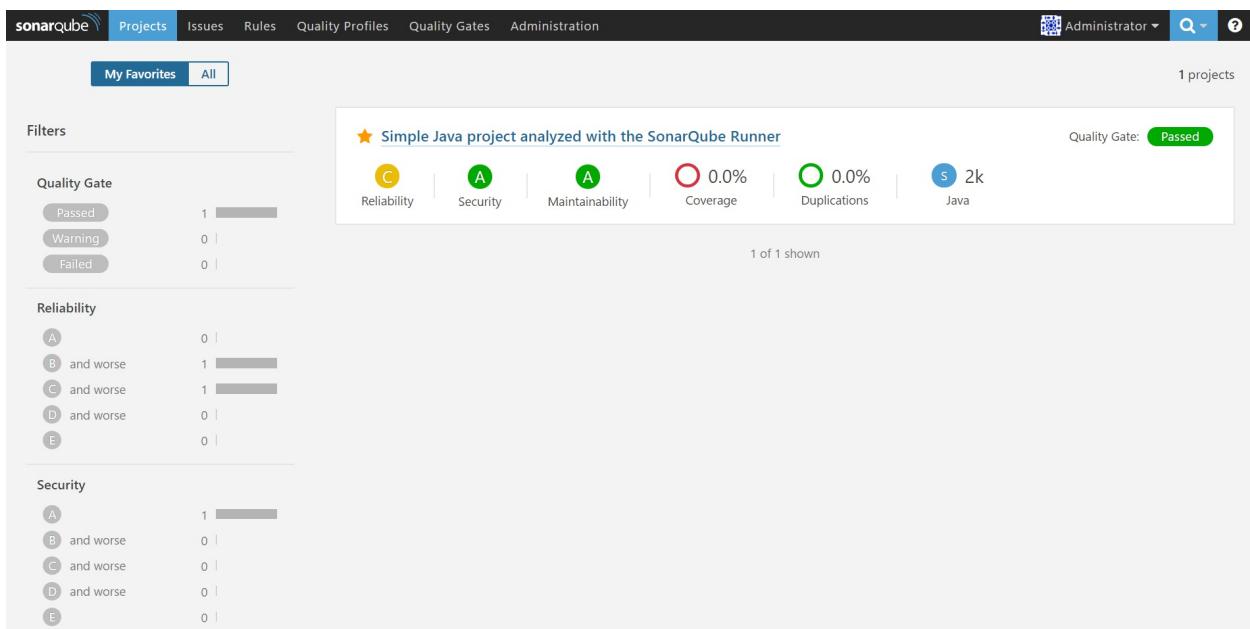
# Comma-separated paths to directories with sources (required)
sonar.sources=src

# 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:

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Administrator May 26, 2017 10:44 AM Version 1.0

Simple Java project analyzed with the SonarQube Runner

Issues Measures Code Activity Administration

Quality Gate Passed

Bugs & Vulnerabilities

33 C Bugs 0 A Vulnerabilities

Code Smells

7h A Debt 35 Code Smells

started 5 minutes ago

Coverage 0.0% Coverage

Lines of Code 2k Java 2k

Quality Gate (Default) SonarQube way

Quality Profiles (Java) Sonar way

Key java-sonar-runner-simple

Activity May 26, 2017 1.0 Show More

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:

Language	Name	Projects	Rules	Updated	Used
C#, 1 profile(s)	Sonar way	Default	134	Never	Never
Flex, 1 profile(s)	Sonar way	Default	60	Never	Never
Java, 1 profile(s)	Sonar way	Default	277	Never	6 minutes ago
JavaScript, 1 profile(s)	Sonar way	Default	111	Never	
PHP, 3 profile(s)			0	20	Never

localhost:9000/profiles#

Recently Added Rules

- Skipped unit tests should be either removed or fixed
- C#, not yet activated
- Failed unit tests should be fixed
- C#, not yet activated
- Source files should not have any duplicated b...
- C#, not yet activated
- Source files should have a sufficient density o...
- C#, 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 b...
- PHP, activated on 1 profile(s)
- Lines should have sufficient coverage by tests
- PHP, not yet activated
- Source files should have a sufficient density o...
- PHP, not yet activated
- Branches should have sufficient coverage by t...
- PHP, not yet activated
- See All 1.2k

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

Copy Profile Sonar way - Java

New name* PetClinic

Copy **Cancel**

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

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration Administrator ▾ Q ?

Quality Profiles / Java
PetClinic Updated: a few seconds ago Used: Never Changelog Actions ▾

Rules	Active	Inactive
Total	277	120
Bugs	129	20
Vulnerabilities	18	13
Code Smells	130	87

Activate More

Inheritance
PetClinic 277 active rules Change Parent

Projects
No projects are explicitly associated to the profile. Change Projects

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:

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration Administrator ▾ Q ? May 26, 2017 10:44 AM Version 1.0

Simple Java project analyzed with the SonarQube Runner Issues Measures Code Activity Administration ▾

Quality Profiles
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.)

Language	Quality Profile
C#	Default: Sonar way
Flex	Default: Sonar way
Java	Default: Sonar way
JavaScript	Default: Sonar way PetClinic
PHP	Default: Sonar way
Python	Default: Sonar way

- 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.
- Go to SonarQube and disable the SCM sensor:

General Settings
Edit global settings for this SonarQube instance.

Analysis Scope	SCM
C#	Disable the SCM Sensor Disable the retrieval of blame information from Source Control Manager Key: sonar.scm.disabled <input checked="" type="checkbox"/>
Flex	
General	
Java	
PHP	
Python	
Scanner for MSBuild	
SCM	SVN
Security	Passphrase Optional passphrase of your private key file Key: sonar.svn.passphrase.secured <input type="button" value="Set"/>
SonarJS	Username Username to be used for SVN server or SVN+SSH authentication Key: sonar.svn.username <input type="text"/>

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

Quality Gate plugin

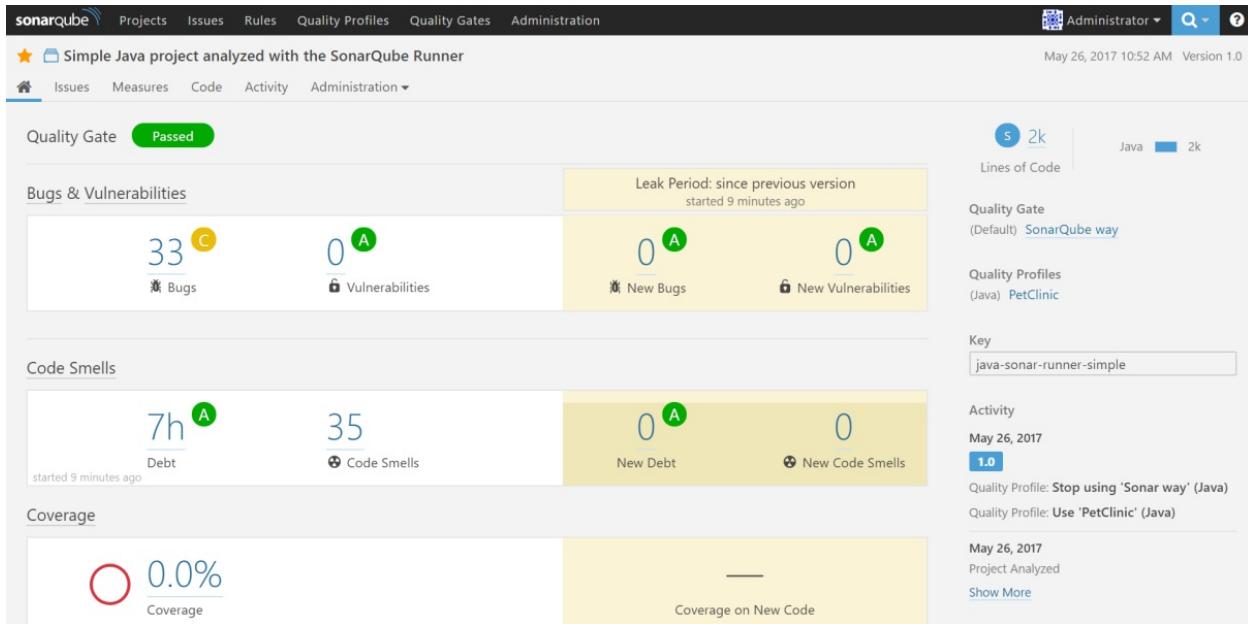
The `Quality Gate` plugin is useful if we want to fail the Jenkins job based on the result of Quality Gate:

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

Install ↓	Name	Version
<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
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 Fails the build whenever the Quality Gates criteria in the Sonar analysis aren't met (the project Quality Gates status is different than "Passed")	2.5
<input 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")	1.0.4

[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:

The screenshot shows the Jenkins 'Post-build Actions' configuration page for the 'PetClinic-Code' build job. The 'Post-build Actions' tab is selected. A 'Quality Gates Sonarqube Plugin' section is displayed, stating: 'There are no Quality Gates instances configured. Please configure a Quality Gate instance in the system configuration. By default http://localhost:9000, Username: admin, Password: admin will be used.' Below this, a 'Project Key' field is shown with an error message: 'Please insert project key.' A note says 'Enter your project key.' At the bottom, there is a 'Save' button and an 'Apply' button.

5. Go to **Manage Jenkins**, click on **Configure system**, and configure

Sonar instance for **Quality Gates**:

The screenshot shows the Jenkins configuration interface for setting up a Sonar instance. The URL is `localhost:8080/configure`. The path is `Jenkins > configuration`. The section title is **Quality Gates - Sonarqube**.

Setting	Value	Description
Name	Sonaqube6.3	Make sure the name is unique value
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)		Default value is '10000'

Buttons at the bottom:

- Delete (red button)
- Add Sonar instance (grey button)

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

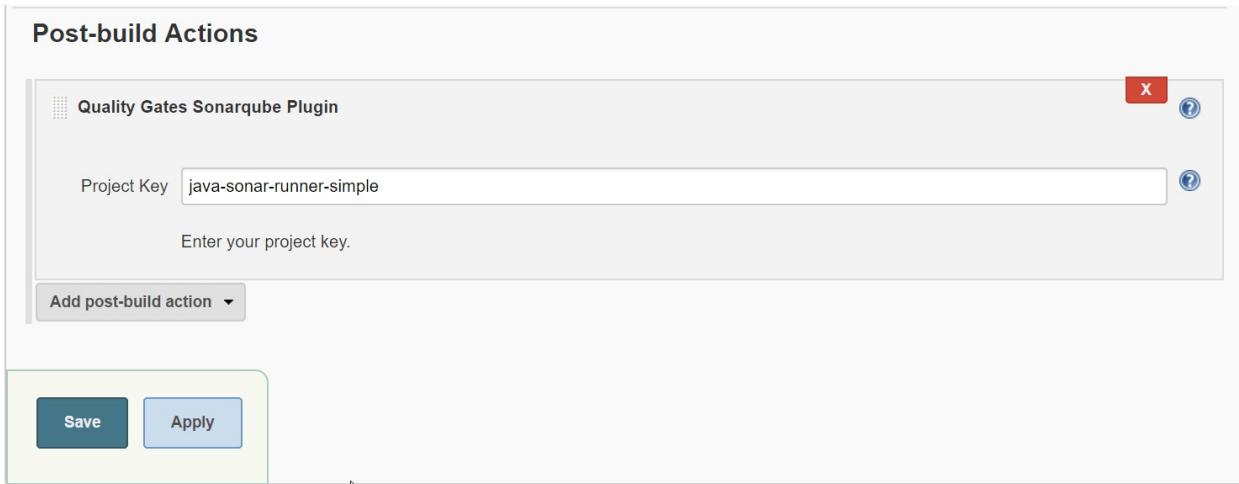
```
# Required metadata
sonar.projectKey=java-sonar-runner-simple
sonar.projectName=Simple Java project analyzed with the SonarQube Runner
sonar.projectVersion=1.0

# 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**:

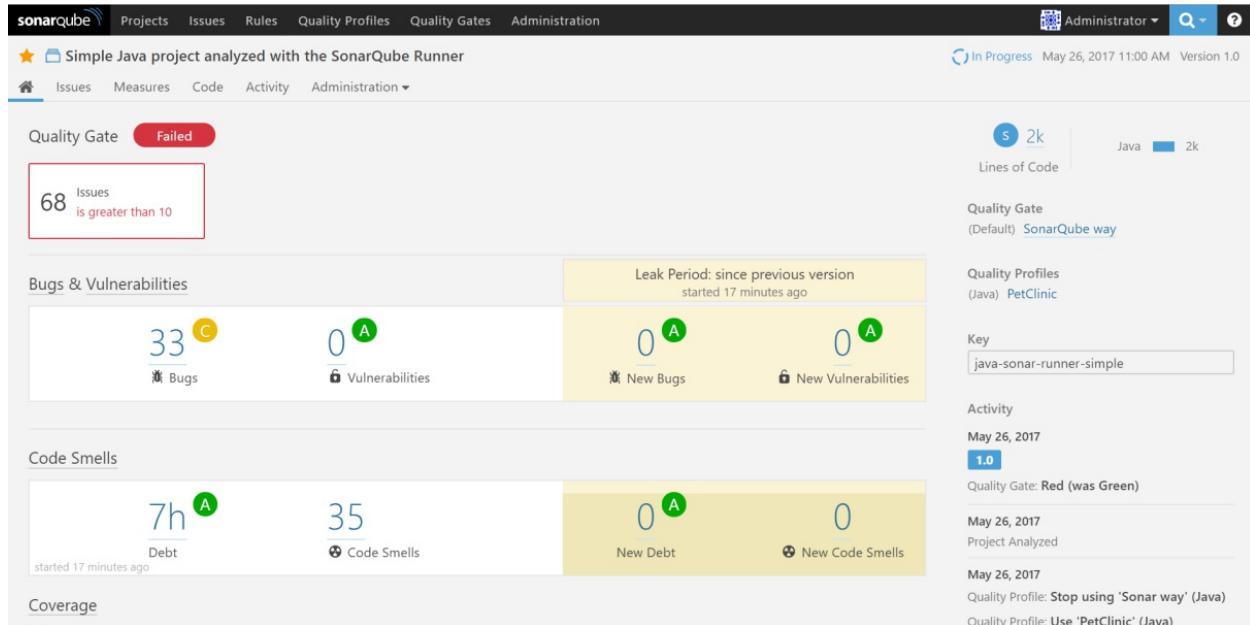


8. Click on **Build now** to execute a Jenkins build job.
9. The Jenkins job has failed. Go through the console output and the reason will be Quality Gate failure:

```
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

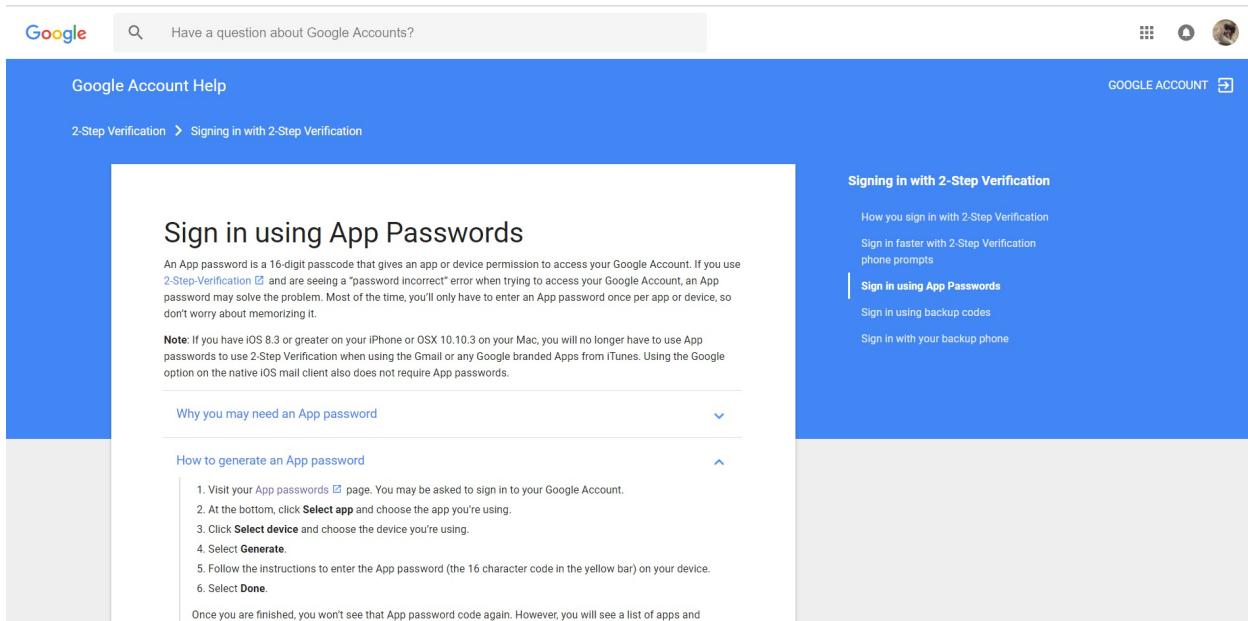
What if we want to send a notification of the build status from Jenkins? We can configure mail notifications in such scenarios.

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:

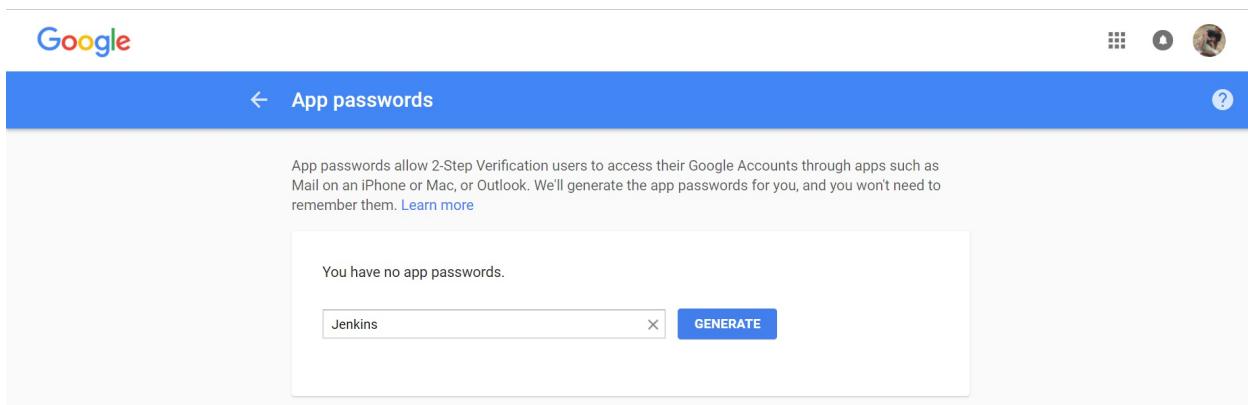
The screenshot shows a web browser window with the URL <https://myaccount.google.com/lesssecureapps?pli=1>. The page title is "Less secure apps". On the right, there is a user profile picture of Mitesh. The main content area contains text about less secure sign-in technology and links to "Learn more". A note at the bottom states that this setting is not available for accounts with 2-Step Verification enabled, with a link to "Learn more".

3. Click on the **App Password** page:



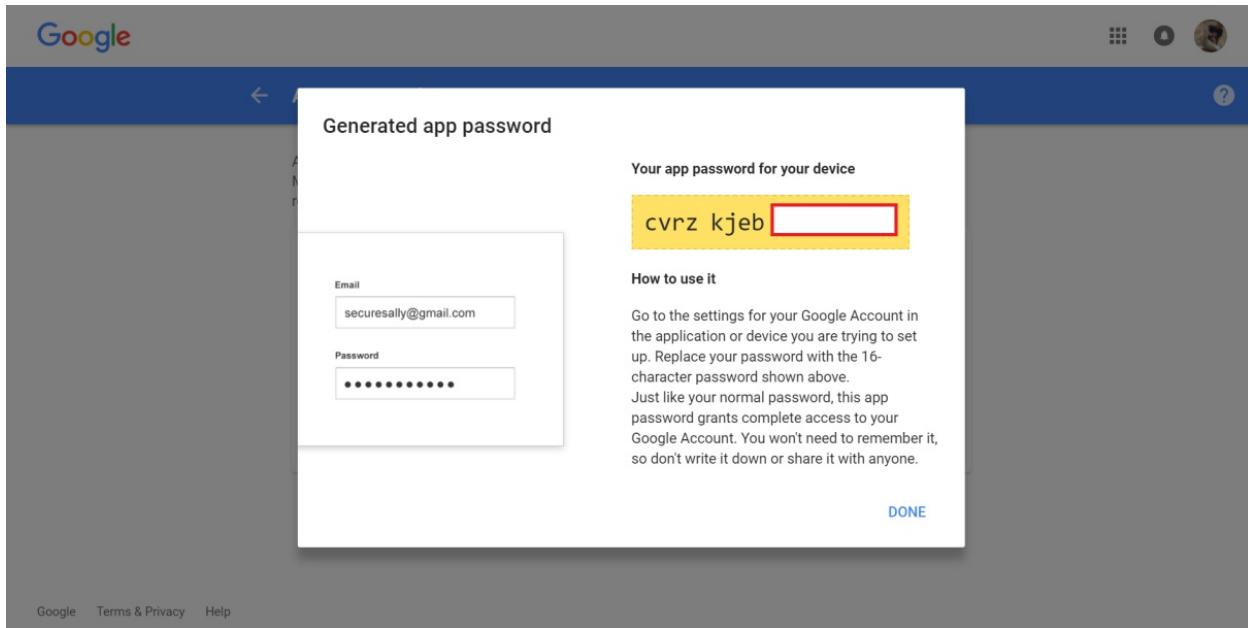
The screenshot shows the Google Account Help page for 2-Step Verification. At the top, there's a search bar with the placeholder "Have a question about Google Accounts?". On the right, there are account settings icons and a "GOOGLE ACCOUNT" button. The main content area has a blue header "Sign in using App Passwords". Below it, a note explains what an App password is and how it can help if 2-Step Verification fails. A "Note" section mentions iOS 8.3+ and OS X 10.10.3 support. There are two expandable sections: "Why you need an App password" and "How to generate an App password". The "How to generate an App password" section contains a numbered list of steps: 1. Visit the App passwords page. 2. Click "Select app" and choose the app. 3. Click "Select device" and choose the device. 4. Click "Generate". 5. Follow the instructions to enter the App password. 6. Click "Done". A note at the bottom says once finished, the code won't appear again but a list of apps will. To the right, a sidebar titled "Signing in with 2-Step Verification" lists related topics like "How you sign in with 2-Step Verification", "Sign in faster with 2-Step Verification phone prompts", and "Sign in using App Passwords".

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



The screenshot shows the "App passwords" page. At the top, there's a back arrow and the title "App passwords". On the right, there's a help icon and a question mark icon. The main content area explains that app passwords allow 2-Step Verification users to access their Google Accounts through apps like Mail on an iPhone or Mac, or Outlook. It links to "Learn more". Below this, a message says "You have no app passwords." There's a text input field containing "Jenkins" with a clear button "X" and a blue "GENERATE" button.

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**:

Jenkins > PetClinic-Code >

General Source Code Management Build Triggers Build Environment Build **Post-build Actions**

Add build step ▾

Post-build Actions

E-mail Notification

Recipients [REDACTED]@gmail.com

Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

Send e-mail for every unstable build

Send separate e-mails to individuals who broke the build

Add post-build action ▾

Save **Apply**

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. The subject is "Build failed in Jenkins: PetClinic-Code #20". The body of the email contains the error message from the previous step, indicating a failure due to an unconfigured address. It also includes the Jenkins log output, which details the SonarQube scanner configuration and execution process.

```
address not configured yet <[REDACTED]@gmail.com>  
to me [REDACTED]  
See <http://localhost:8080/job/PetClinic-Code/20/display/redirect>  
  
Started by user admin  
Building in workspace <http://localhost:8080/job/PetClinic-Code/ws/>  
FSSCM checkout F:\#JenkinsEssentials\FirstDraft\PetClinic to <http://localhost:8080/job/PetClinic-Code/ws/>  
FSSCM.check completed in 4.606 seconds  
[PetClinic-Code] $ F:\#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:\#JenkinsEssentials\FirstDraft\jenkinsHome\tools\hudson.plugins.sonar.SonarRunnerInstallation\SonarQube_Scanner_3.0.3\bin\conf\sonar-scanner.properties  
INFO: Project root configuration file: <http://localhost:8080/job/PetClinic-Code/ws/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  
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.scanner.cli.Main.execute(Main.java:74)  
    at org.sonarsource.scanner.cli.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.

Chapter 4. Continuous Integration with Jenkins

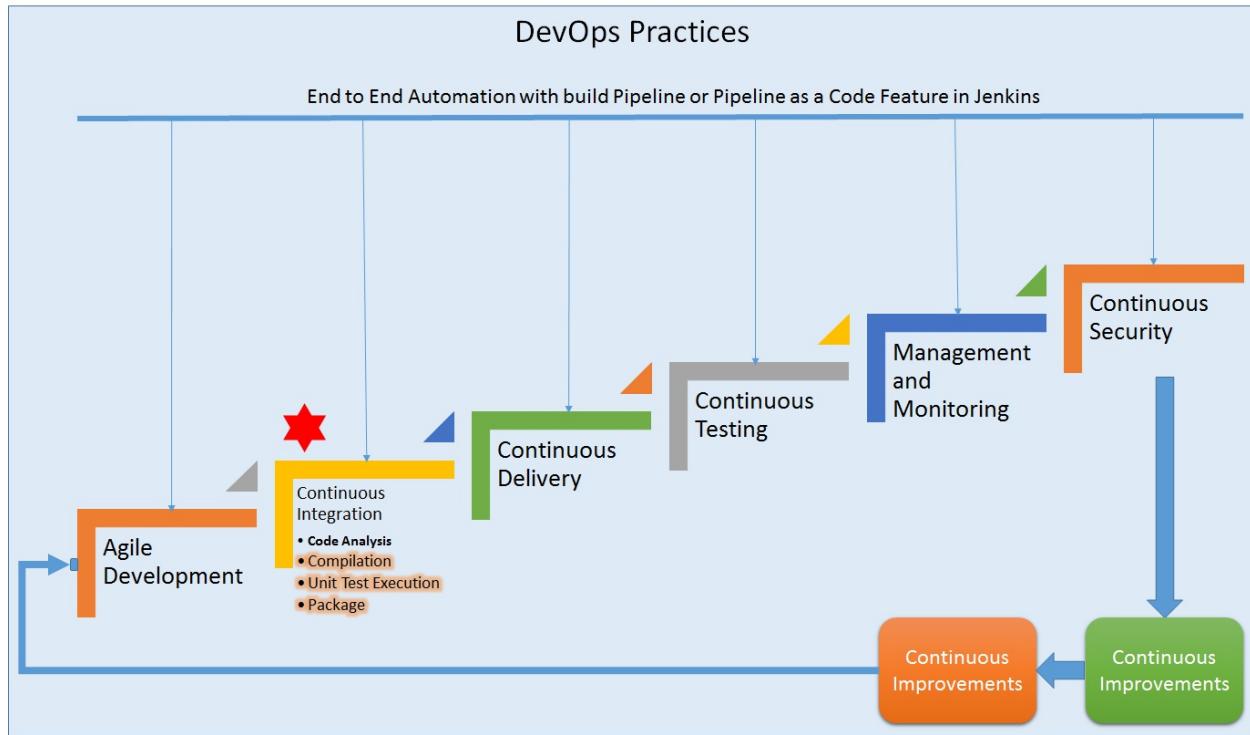
Continuous Integration is one of the most important DevOps practices and serves as a base to implement DevOps culture in any organization. It is all about committing code into shared repositories such as Git or SVN multiple times, based on feature completion or bug fixes, and then verifying it with static 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.

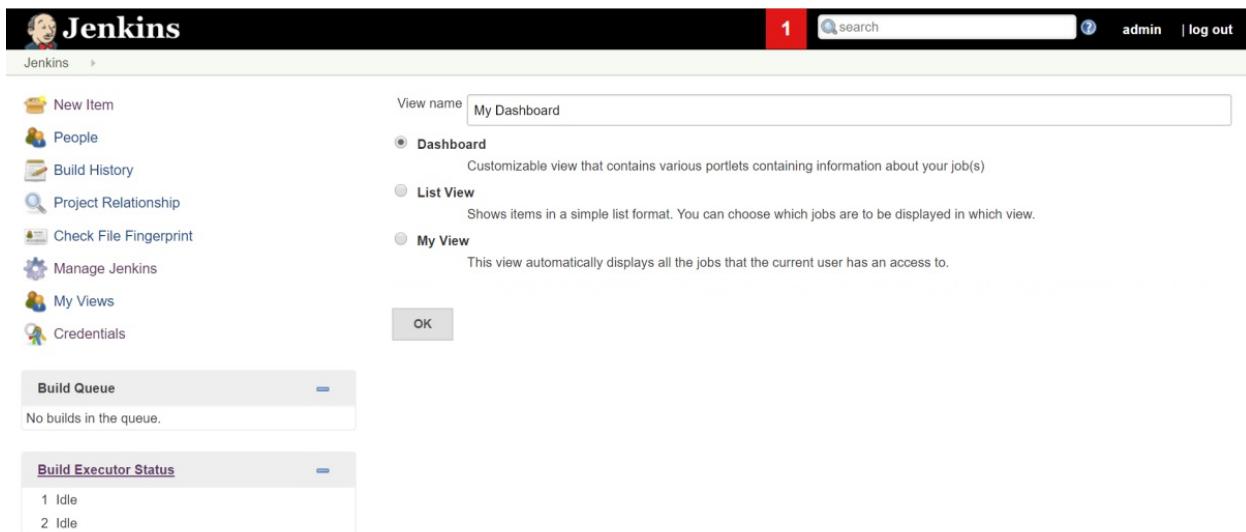
Dashboard View plugin

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:

The screenshot shows the Jenkins 'My Dashboard' configuration interface. On the left, there's a sidebar with various icons for 'New Item', 'People', 'Build History', 'Edit View', 'Delete View', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', and 'Credentials'. Below these are two collapsed sections: 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 Idle, 2 Idle). The main right panel has a 'Name' field set to 'My Dashboard' and a 'Description' text area. Under 'Job Filters', a 'Status Filter' dropdown is set to 'All selected jobs', and a 'Recurse in subfolders' checkbox is checked. The 'Jobs' section lists several Jenkins jobs: FirstJob, Main-PetClinic, Maven-Sample, PetClinic-Code (which is checked), and SpringBoot. At the bottom are 'OK' and 'Apply' buttons.

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 after configuration. The sidebar and 'Build Queue' and 'Build Executor Status' sections are identical to the previous screenshot. The main dashboard area now features a table titled 'My Dashboard' with columns: S, W, Name, Last Success, Last Failure, and Last Duration. It contains one row for the 'PetClinic-Code' job, which has a red icon, is labeled 'S M L', and has a status of 'Last Success: 11 hr - #18, Last Failure: 19 min - #20, Last Duration: 1 min 34 sec'. Below the table are links for 'Legend', 'RSS for all', 'RSS for failures', and 'RSS for just latest builds'. The top navigation bar includes a search bar, user info for 'admin', and a 'log out' link.

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.

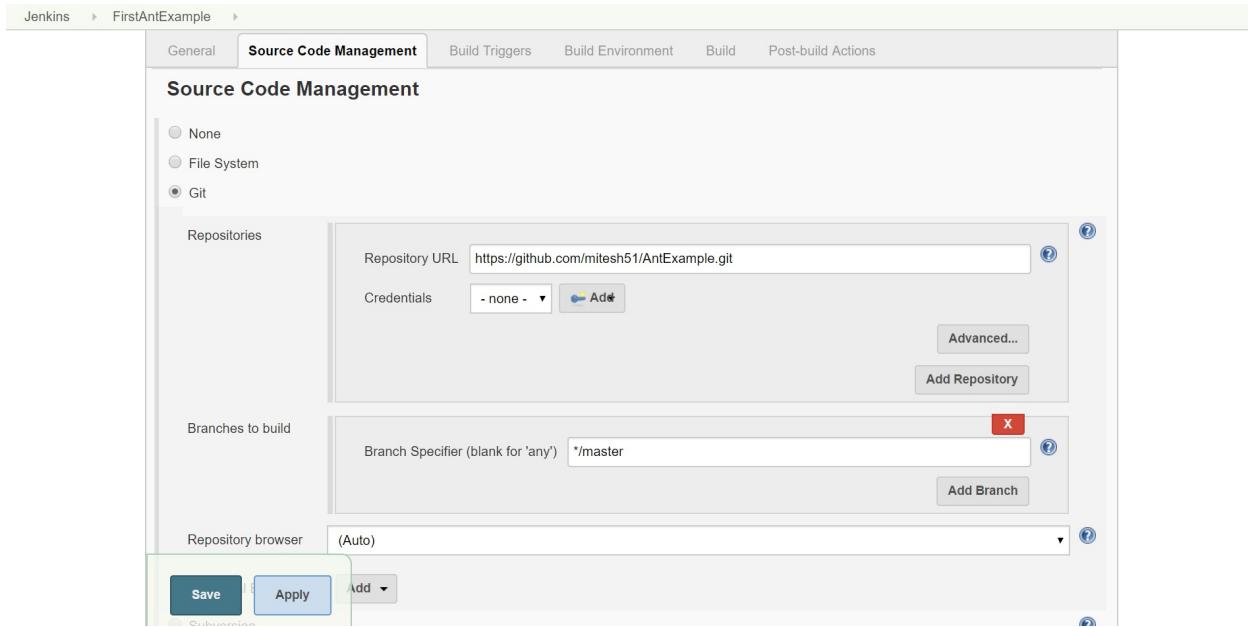
We will now create a **Freestyle project** in Jenkins for our first Ant project integration in Jenkins:

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

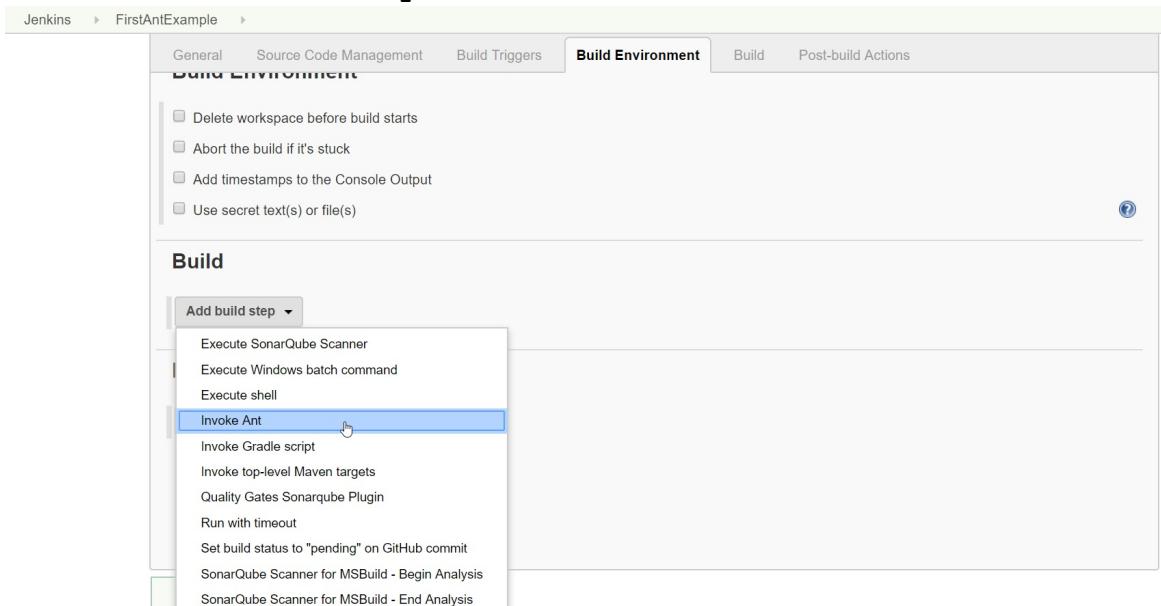
The screenshot shows the Jenkins interface for creating a new item. At the top, there's a navigation bar with the Jenkins logo, a search bar, and user information ('admin | log out'). Below the header, the main content area has a title 'Enter an item name' and a text input field containing 'FirstAntExample'. A note below the field says '» Required field'. Below the input field, there's a list of project types with icons: 'Freestyle project' (selected), 'Maven project', 'Pipeline', 'External Job', and 'Multi-configuration project'. Each item has a brief description and a link to its configuration page.

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.

```

Jenkins > FirstAntExample > #2
Edit Build Information
Git Build Data
No Tags
Previous Build

Executed Ant Targets
• init
• compile
• war

> git.exe config remote.origin.url https://github.com/mitesh51/FirstAntExample.git # timeout=10
Fetching upstream changes from https://github.com/mitesh51/FirstAntExample.git
> git.exe -v # timeout=10
> git.exe fetch --tags --progress https://github.com/mitesh51/FirstAntExample.git +refs/heads/*:refs/remotes/origin/*
> git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
> git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision b88fd734baee7a009d404e478171871629e7ec84 (refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f b88fd734baee7a009d404e478171871629e7ec84
> git.exe rev-list b88fd734baee7a009d404e478171871629e7ec84 # timeout=10
[FirstAntExample] $ 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.sysclasspath=last; 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:

Workspace of FirstAntExample on master

dist / [AntExample.war](#) 4.19 MB [view](#)
[\(all files in zip\)](#)

Build History

#	Build Number	Date
2	#2	May 26, 2017 10:47 PM
1	#1	May 26, 2017 10:36 PM

[RSS for all](#) [RSS for failures](#)

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" xm
          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.outputEncodin
    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
    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**:

The screenshot shows the Jenkins interface for creating a new item. The title bar says 'Jenkins'. The main area has a yellow-highlighted input field containing 'PetClinic-Package'. Below it, a note says '» Required field'. A list of project types is shown:

- Freestyle project**: Described as the central feature of Jenkins, combining any SCM with any build system.
- Maven project**: Described as building a Maven project where Jenkins takes advantage of POM files.
- Pipeline**: Described as orchestrating long-running activities for building pipelines.
- External Job**: Described as recording the execution of a process run outside Jenkins.
- Multi-configuration project**: Described as suitable for projects with many configurations.

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

The screenshot shows the Jenkins configuration page for 'PetClinic-Package'. The top navigation bar includes 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Pre Steps', **Build**, 'Post Steps', and 'Build Settings'. The 'Build' tab is active. Under 'Build', the 'Root POM' is set to 'pom.xml' and the 'Goals and options' is set to 'package'. An 'Advanced...' button is visible. The 'Post Steps' section contains radio buttons for 'Run only if build succeeds', 'Run only if build succeeds or is unstable', and 'Run regardless of build result'. The 'Build Settings' section has an 'E-mail Notification' checkbox. At the bottom, there are 'Save' and 'Apply' buttons.

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.

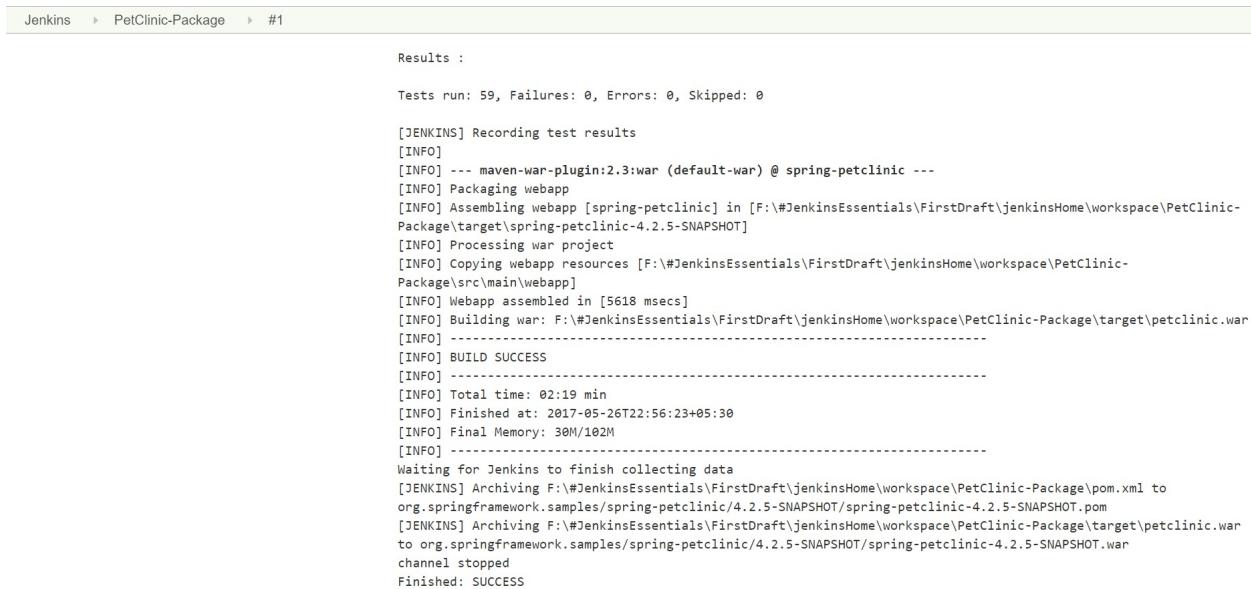
6. Go to the `conf` directory and open `settings.xml`.

Uncomment the following block:

```
<proxies>
<proxy>
<id>optional</id>
<active>true</active>
<protocol>http</protocol>
<username>testuser</username>
<password>sdbjsdhbjw</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.



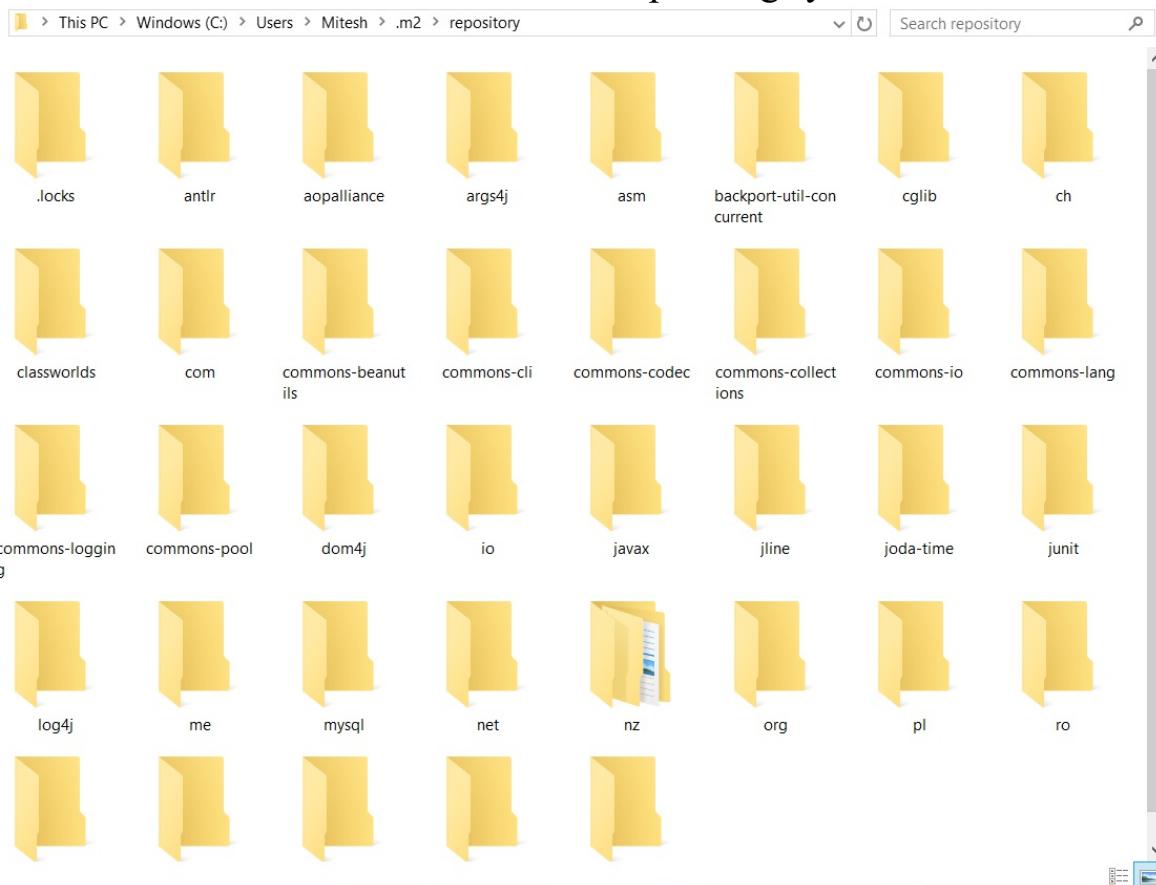
The screenshot shows the Jenkins interface with the URL `Jenkins > PetClinic-Package > #1`. The console output window displays the following log entries:

```
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 [5618 msec]
[INFO] Building war: F:\#JenkinsEssentials\FirstDraft\jenkinsHome\workspace\PetClinic-Package\target\petclinic.war
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:19 min
[INFO] Finished at: 2017-05-26T22:56:23+05:30
[INFO] Final Memory: 30M/102M
[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.

Chapter 5. Continuous Delivery - Implementing Automated Deployment

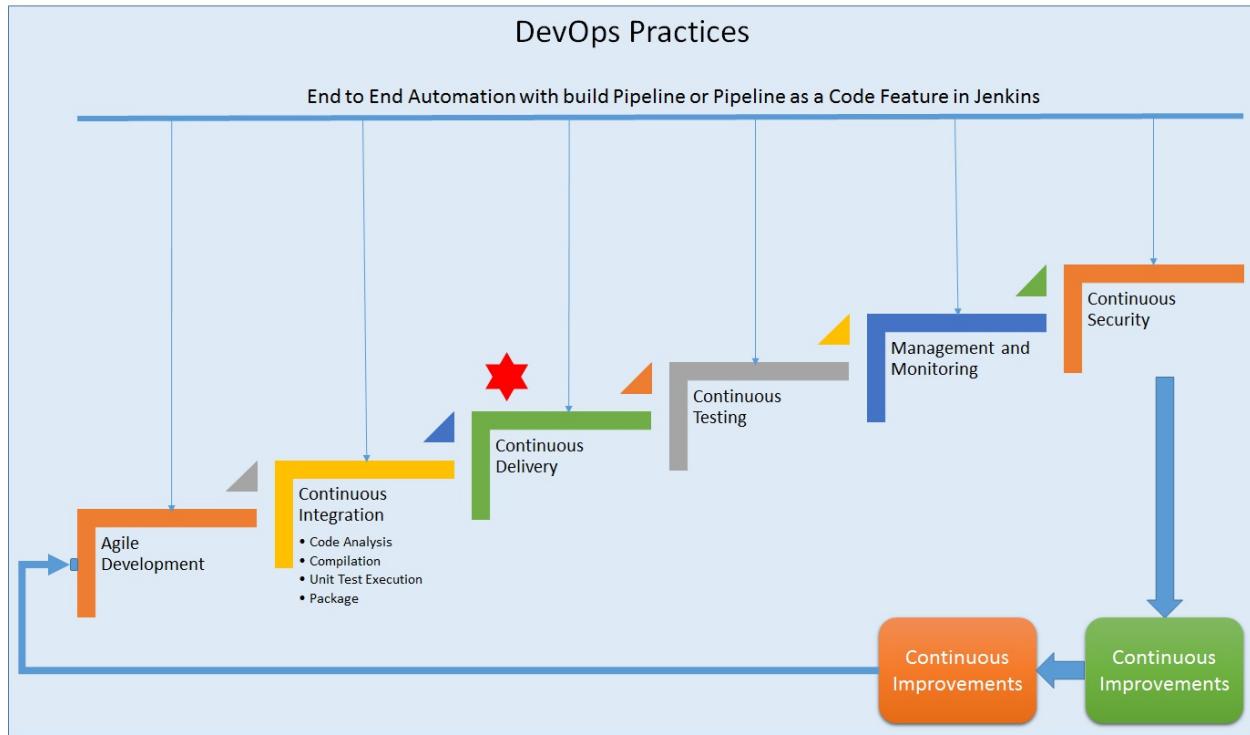
Once we have a package ready for deployment after Continuous Integration, our next step should be deployment of that package in to a web or application server.

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. The 'Available' tab is selected. A search bar at the top right is set to 'Deploy to'. Below the tabs, there's a table with columns 'Name' and 'Version'. Two plugins are listed: 'Deploy to container Plugin' (version 1.10) and 'Deploy to Websphere container Plugin' (version 1.0). At the bottom, there are buttons for 'Install without restart' and 'Download now and install after restart', along with a 'Check now' button and a note about update information.

Name	Version
Deploy to container Plugin	1.10
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

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

Dashboard View



Success

Deploy to container 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

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

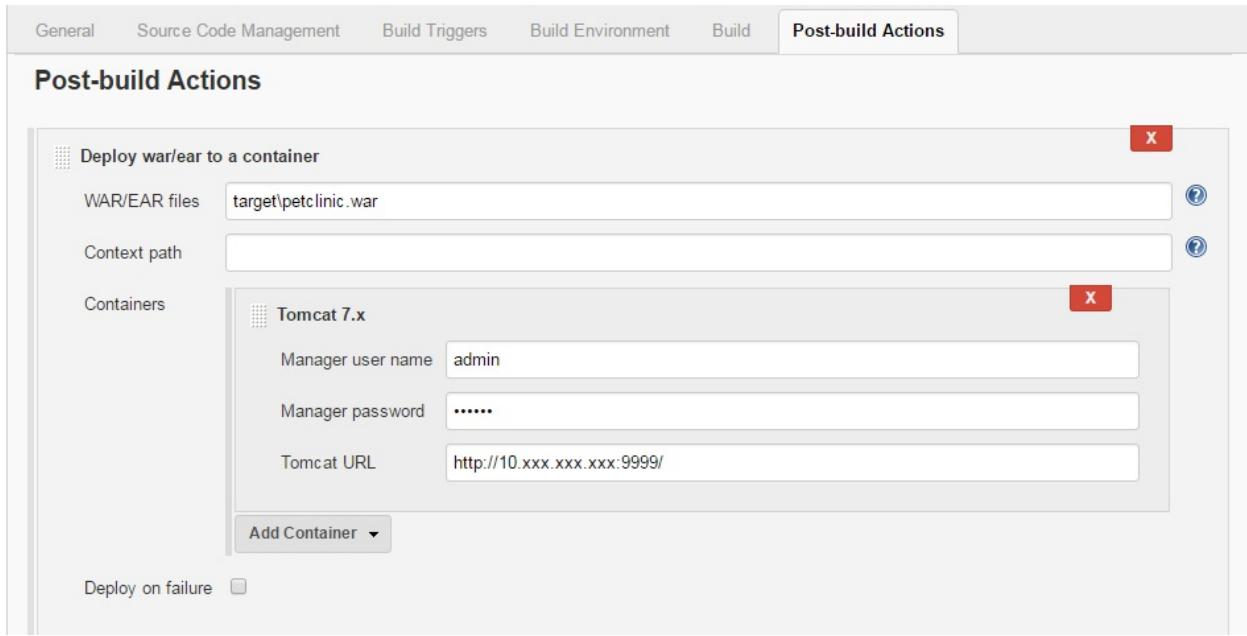
perform this action. Give the project a name and path from which we need to copy the WAR file:

The screenshot shows the Jenkins 'Build' configuration page. The 'Build' tab is selected. A 'Copy artifacts from another project' step is configured with the following settings:

- Project name:** PetClinic-Package
- Which build:** Latest successful build
- Artifacts to copy:** target\petclinic.war
- Parameter filters:** Flatten directories, Optional, Fingerprint Artifacts

At the bottom, there are 'Save' and 'Apply' buttons.

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`:



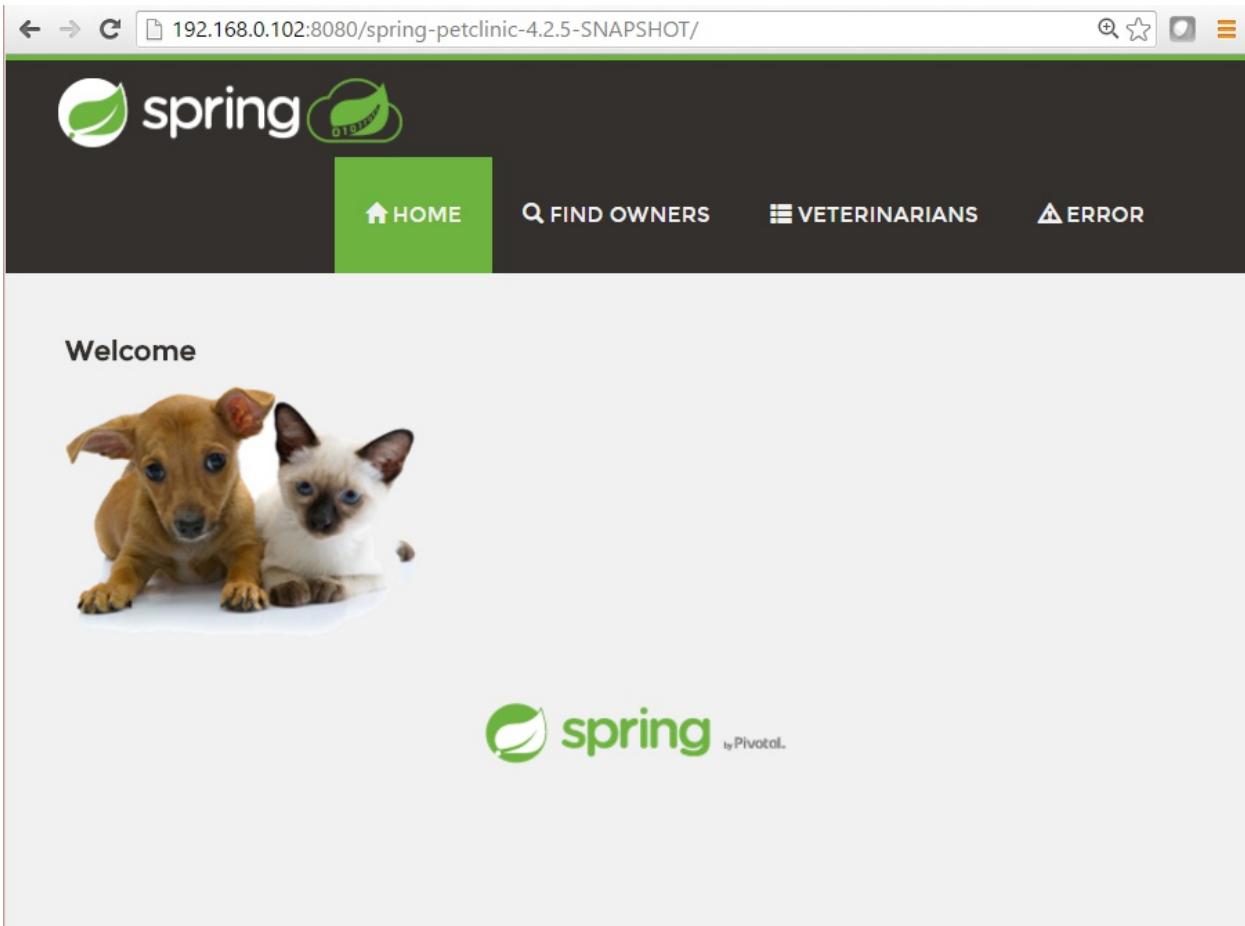
9. Execute the build by clicking on **Build now**. Verify the logs for application deployment:

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
```

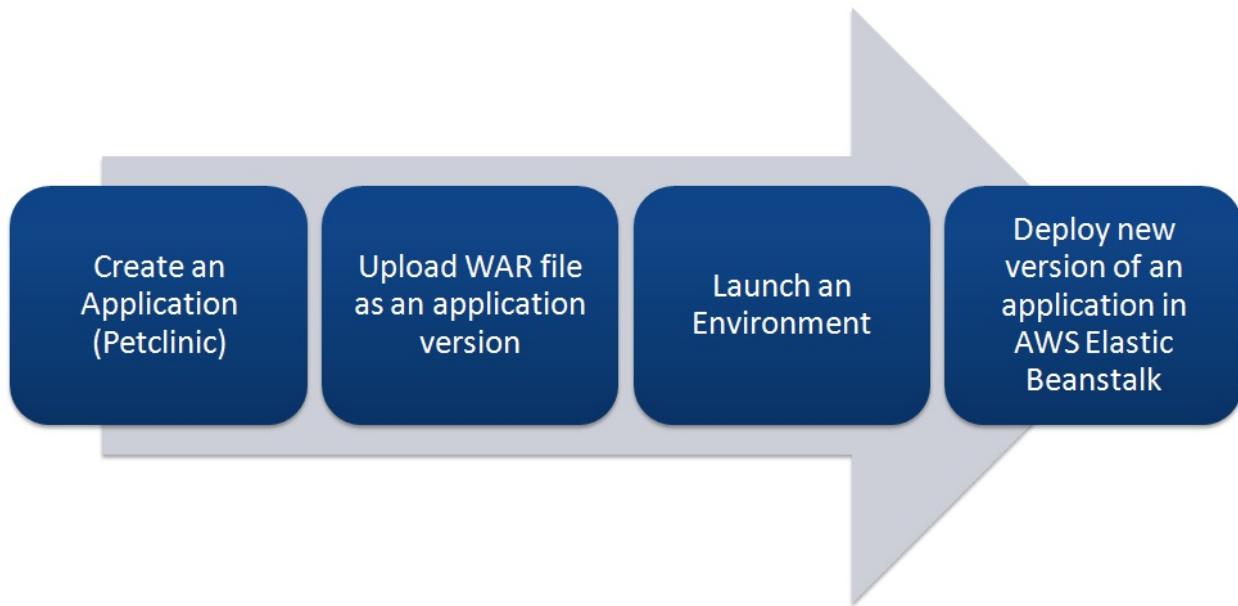
10. Go to a browser and visit the application with the Tomcat URL and the context of an application:



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 header, the title "Resources" is displayed with a refresh icon. A sidebar on the left lists various VPC components: 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". Two buttons are present: "Start VPC Wizard" (blue) and "Launch EC2 Instances" (grey). 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:

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:

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 app later.

Upload your own code

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

2. Verify the sequence of events for the creation of a sample application:

Elastic Beanstalk petclinic Create New Application

All Applications > petclinic > petclinic (Environment ID: e-y2fmvwi3n, 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:
aws eb e-y2fmvwi3n stack AWSEBESecurityGroup-E1E762FFSL5Q
11:01pm Using elasticbeanstalk-us-east-1-685239287657 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. At the top, there are navigation links for AWS, Services, and Edit, along with user information for Mitesh, N. Virginia, and Support. Below the navigation bar, there's a search bar with the text 'Elastic Beanstalk' and a dropdown menu set to 'petclinic'. On the right side of the search bar is a 'Create New Application' button. The main content area shows the 'All Applications > petclinic' path. To the right of the path is an 'Actions' dropdown. On the left, there's a sidebar with links for Environments, Application Versions, Saved Configurations, and a large green box containing detailed information about the 'petclinic' environment.

All Applications > petclinic

Actions ▾

Environments
Application Versions
Saved Configurations

petclinic

Environment tier: Web Server
Running versions: Sample Application
Last modified: 2016-07-07 23:05:19 UTC+0530
URL: petclinic.mjczcu3cvp.us-east-1.elasticbe...

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 console with the 'petclinic' environment selected. The top navigation bar and search bar are identical to the previous screenshot. The main content area shows the 'All Applications > petclinic > petclinic' path, followed by '(Environment ID: e-y2fmvwr3n, URL: petclinic.mjczcu3cvp.us-east-1.elasticbeanstalk.com)'. An 'Actions' dropdown is on the right. The left sidebar has links for Dashboard, Configuration, Logs, Health, Monitoring, Alarms, Managed Updates (with a 'NEW' badge), and Events. The right side of the dashboard displays the 'Overview' section. It features a large green circle with a white checkmark icon, labeled 'Health' and 'Ok'. Below it is a 'Causes' link. To the right, there's a 'Running Version' section showing 'Sample Application' and a 'Upload and Deploy' button. Further right is a 'Configuration' section with a cartoon cat icon, showing '64bit Amazon Linux 2016.03 v2.1.3 running Tomcat 8 Java 8' and a 'Change' button.

All Applications > petclinic > petclinic (Environment ID: e-y2fmvwr3n, URL: petclinic.mjczcu3cvp.us-east-1.elasticbeanstalk.com)

Actions ▾

Dashboard
Configuration
Logs
Health
Monitoring
Alarms
Managed Updates NEW
Events

Overview

Refresh



Health

Ok

Causes

Running Version

Sample Application

Upload and Deploy



Configuration

64bit Amazon Linux 2016.03
v2.1.3 running Tomcat 8 Java 8

Change

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

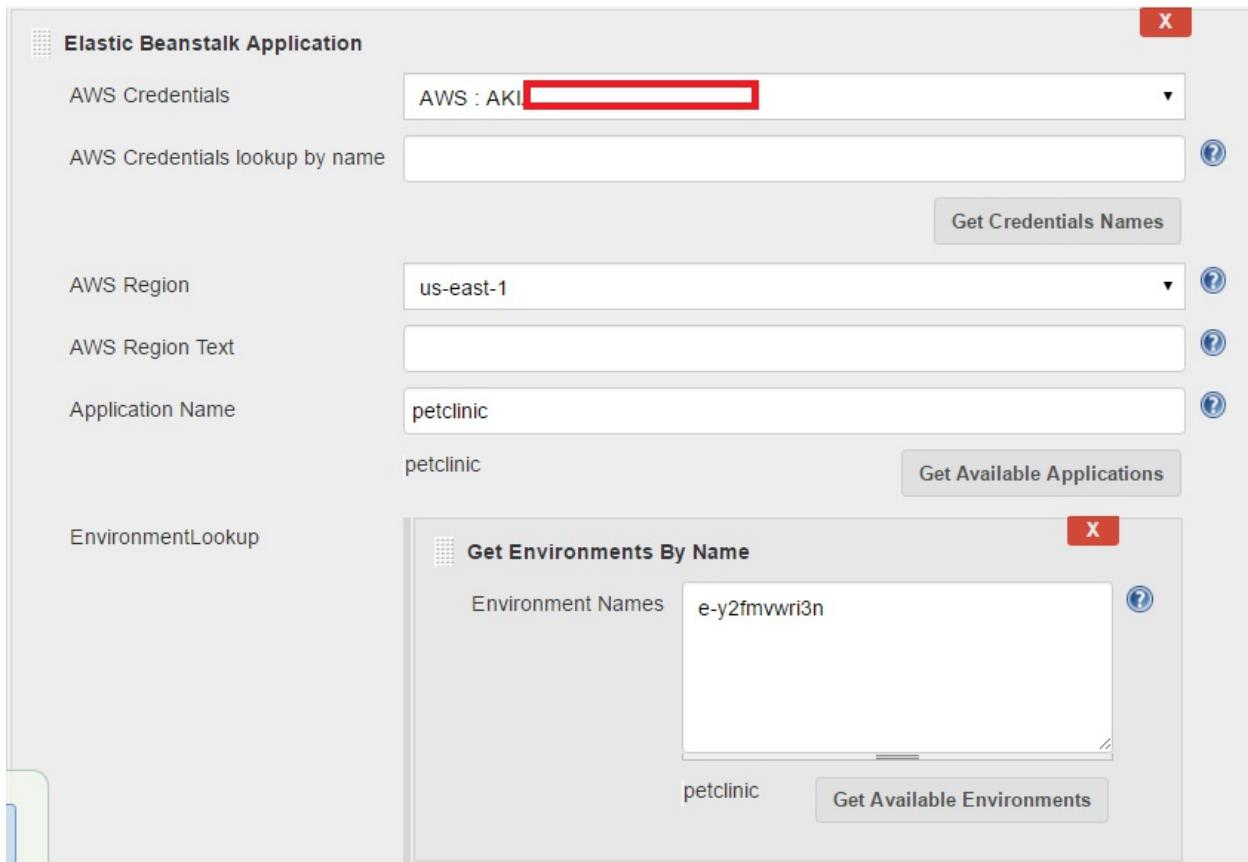
The screenshot shows a web browser window with the URL petclinic.mjczcu3cvp.us-east-1.elasticbeanstalk.com. The main content is a large blue banner with the word "Congratulations" in white. Below it, a message says: "Your first AWS Elastic Beanstalk Application is now running on your own dedicated environment in the AWS Cloud". To the right, there's a "What's Next?" section with links to learn about building, deploying, and managing applications using AWS Elastic Beanstalk, as well as links to AWS Elastic Beanstalk concepts, creating new application versions, and managing environments. There are also sections for downloading the AWS Reference Application and the AWS Toolkit for Eclipse.

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

The screenshot shows a Jenkins job configuration page for "PetClinic". In the "Post-build Actions" section, the "Deploy into AWS Elastic Beanstalk" option is selected. It shows fields for "Access Key" (admin), "Secret Key" (redacted), and "Endpoint" (http://192.168.0.102:8080). At the bottom, there are "Save" and "Apply" buttons.

7. A new section will come up in **Post-build Actions** for Elastic Beanstalk.

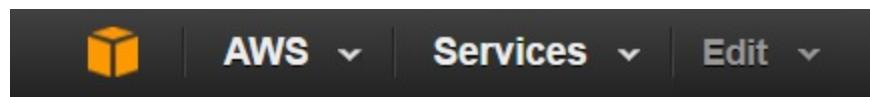
8. Click on the Jenkins dashboard and select **Credentials**; add your AWS credentials.
9. Go to your Jenkins build and select AWS Credential, which is set in the global configuration.
10. Select **AWS Region** from the list and click on **Get Available Applications**. As we have created a sample application, 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:



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:

	Name	Storage Class	Size	Last Modified
<input type="checkbox"/>	.elasticbeanstalk	Standard	0 bytes	Thu Jul 07 23:01:35 GMT+5
<input type="checkbox"/>	petclinic-jenkins-PetClinic-Test-39.zip	Standard	39.9 MB	Fri Jul 08 00:52:04 GMT+53
<input type="checkbox"/>	resources	--	--	--

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:

Screenshot of the AWS Elastic Beanstalk dashboard showing the PetClinic application.

The dashboard includes a sidebar with links for "Learn More", "Get started using Elastic Beanstalk", "Modify the code", "Create and connect to a database", "Add a custom domain", "Command Line Interface (v3)", and "Installing the AWS EB CLI" and "EB CLI Command Reference".

The main area displays the "All Applications" section for "petclinic". It shows a green card for the "petclinic" environment tier, which is a Web Server. The card details the "Running versions" as "jenkins-PetClinic-Test-39", "Last modified" as "2016-07-08 01:04:41 UTC+0530", and the "URL" as "petclinic.mjczcu3cv.us-east-1.elasticbeanstalk.com". A "Actions" button is also present.

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:

Screenshot of a web browser displaying the PetClinic application's homepage.

The URL in the address bar is "petclinic.mjczcu3cv.us-east-1.elasticbeanstalk.com".

The page features a dark header with the Spring logo and navigation links for "HOME", "FIND OWNERS", "VETERINARIANS", and "ERROR".

The main content area is titled "Welcome" and features a photograph of a brown puppy and a white cat sitting together.

At the bottom, there is a footer with the Spring logo and the text "by Pivotal".

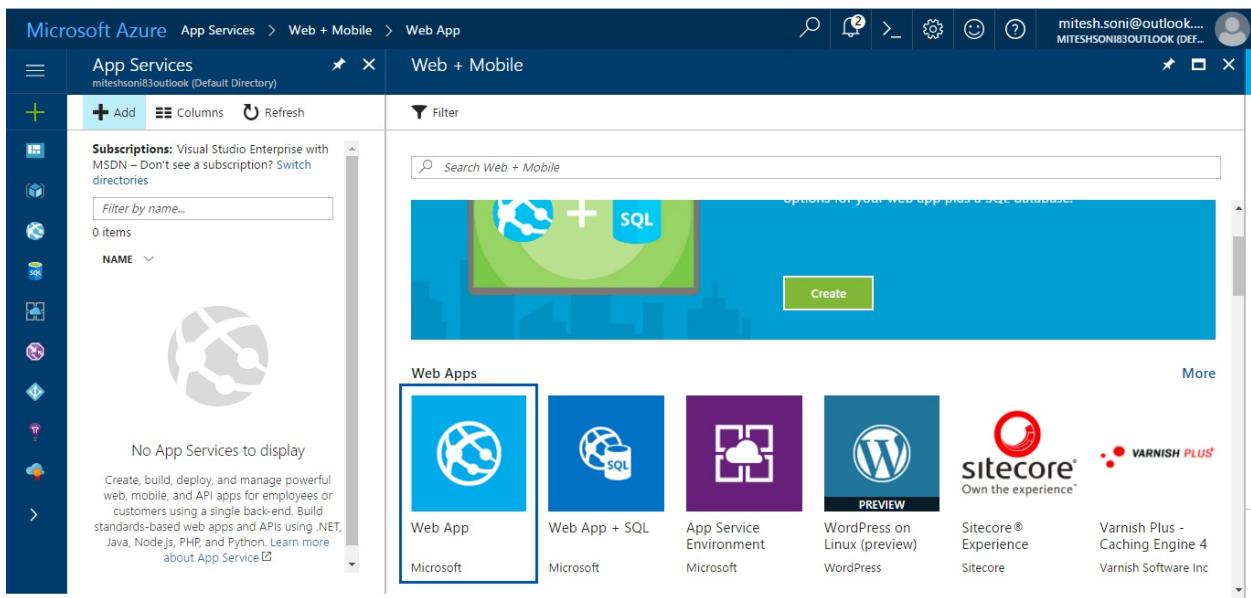
Once the application deployment is successful, terminate the environment.

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
Microsoft

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 mytestsite8 WEBSITE

CREATE BROWSE EDIT STOP DOWNGRADE ...

Summary Summary

Notifications MyTestGroup

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**:

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various icons. The main area is titled "Web App" under "App Services". A sub-menu bar at the top says "Microsoft Azure App Services > Web + Mobile > Web App > Web App". The central part of the screen displays a "Create" dialog for a new web app. The "App name" field is filled with "DevOpsPetClinic". The "Subscription" dropdown shows "Visual Studio Enterprise with MSDN". Under "Resource Group", the "Create new" option is selected with "DevOpsPetClinic" as the name. The "App Service plan/Location" section shows "ServicePlan5cff78bd-99a2(South ...)" and a "Service Plan" dropdown. The "Application Insights" section has an "On" button. At the bottom right of the dialog are "Create" and "Automation options" buttons.

- 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 dashboard for the "DevOpsPetClinic" app service. The left sidebar has a "New" button and links to "Dashboard", "Resource groups", "App Services", "SQL databases", "App Service Environ...", "Traffic Manager prof..", "Azure Active Directo...", "Application Insights", and "Advisor". The main area has a "Search (Ctrl+/" input field. Below it is a navigation bar with "Overview" (highlighted), "Activity log", "Access control (IAM)", "Tags", "Diagnose and solve problems", "DEPLOYMENT" (with "Quickstart", "Deployment credentials", "Deployment slots", "Deployment options", and "Continuous Delivery (Preview)"), and "More services >". To the right, there's a "Click here to access our Quickstart guide for deploying code to your app" button. The "Essentials" section shows the resource group ("DevOpsPetClinic"), status ("Running"), location ("South Central US"), subscription ("Visual Studio Enterprise with MSDN"), and deployment ID ("b88f4447-ad0e-44d4-a662-2eb5c950f091"). The "Monitoring" section includes a "Requests and errors" chart with values 100, 80, and 60.

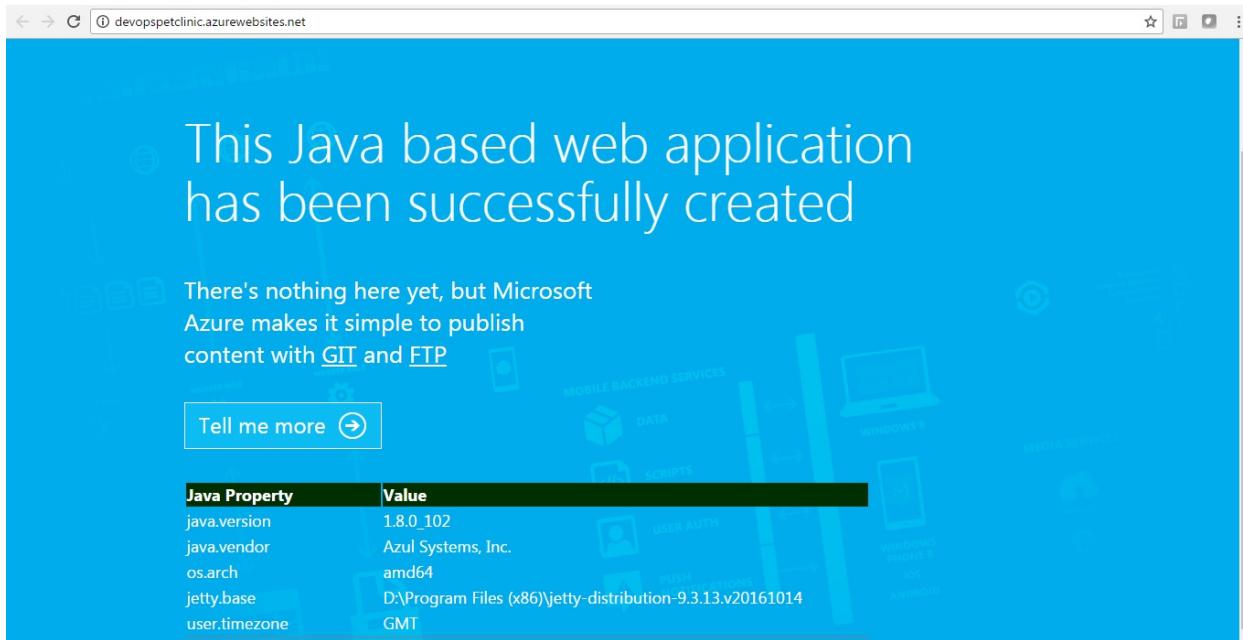
- Verify you can see this in the **App Services** section too:

The screenshot shows the Microsoft Azure App Services dashboard. On the left sidebar, under the 'App Services' category, there is a single item: 'DevOpsPetClinic'. The main pane displays the 'App Services' section with the title 'miteshsoni83outlook (Default Directory)'. It includes a search bar, filter options ('Filter by name...', 'All locations', 'No grouping'), and a table showing one item: 'DevOpsPetClinic' (Status: Running, Location: South Central US, App Type: Web app, App Service Plan: ServicePlan5cff78bd-...). The table has columns for NAME, STATUS, LOCATION, APP TYPE, and APP SERVICE PLAN.

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**:

The screenshot shows the 'Application settings' page for the 'DevOpsPetClinic' app service. The left sidebar lists various settings categories: Deployment options, Continuous Delivery (Preview), Application settings (selected), Authentication / Authorization, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The main pane is titled 'DevOpsPetClinic - Application settings' and contains a 'General settings' section. It includes fields for .NET Framework version (v4.6), PHP version (5.6), Java version (Java 8), Java Minor version (Newest), Web container (Newest Jetty 9.3), Python version (Off), Platform (32-bit selected), Web sockets (Off), and Always On (On selected). There is also a Managed Pipeline Version section with Integrated and Classic options.

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:



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

A screenshot of the Azure portal showing the "Deployment credentials" settings for the "DevOpsPetClinic" app service. The left sidebar shows various Azure services like App Services, SQL databases, and Application Insights. The main panel is titled "DevOpsPetClinic - Deployment credentials" and shows a "Deployment" section with "Deployment credentials" selected. On the right, there is a form to enter deployment credentials:

New name and password
Git and FTP can't authenticate using the account you're signed in with, so create a new user name and password to use with those technologies
Use this user name and password to deploy to any apps for all subscriptions associated with your Microsoft Azure account

* FTP/deployment username:

* Password: ✓

* Confirm password: ✓

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:

Publish over FTP

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

	Name	Modified	Size
...	ROOT	7/31/2016, 12:34:13 PM	
...	spring-petclinic-4.2.5-SNAPSHOT	7/31/2016, 3:11:54 PM	
...	spring-petclinic-4.2.5-SNAPSHOT.war	7/31/2016, 3:11:50 PM	40946 KB

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.

Chapter 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.

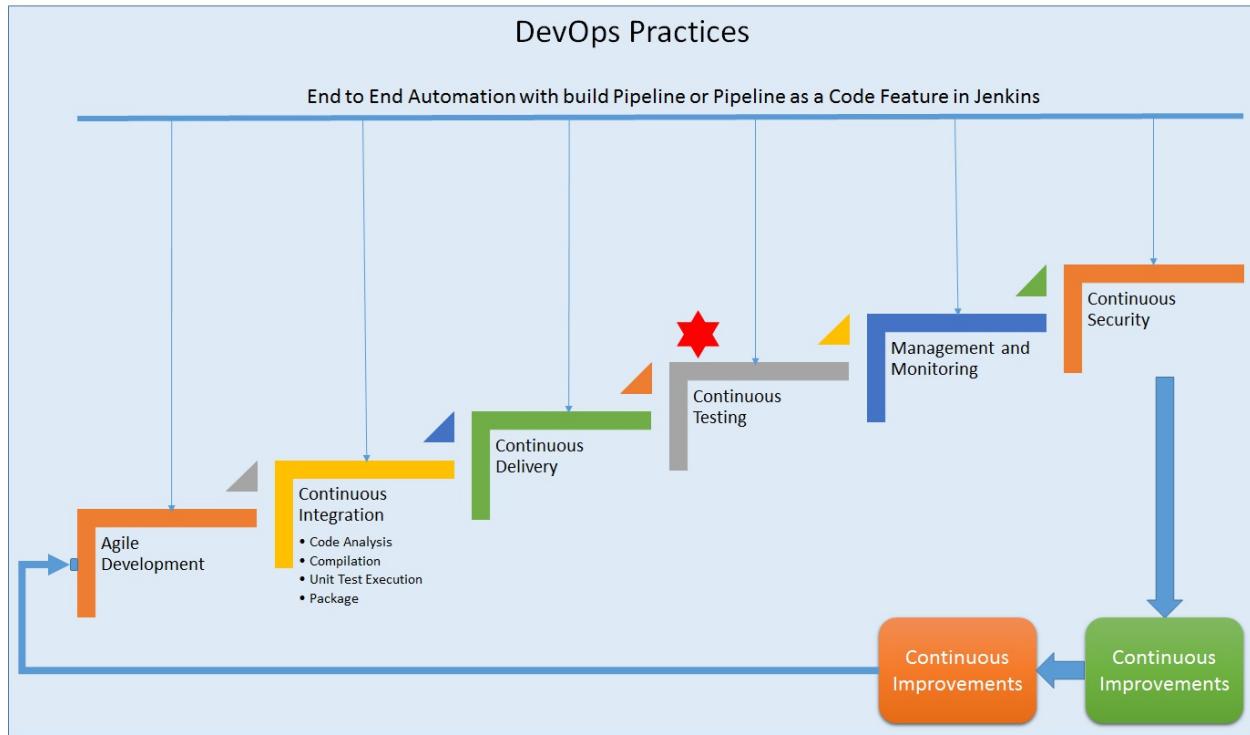
It not only considers automation, but it also includes aspects such as culture change and tools. It is essential to integrate automated tests into application lifecycle management early, to test quickly and in a timely manner, and to repeat the test execution process efficiently.

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

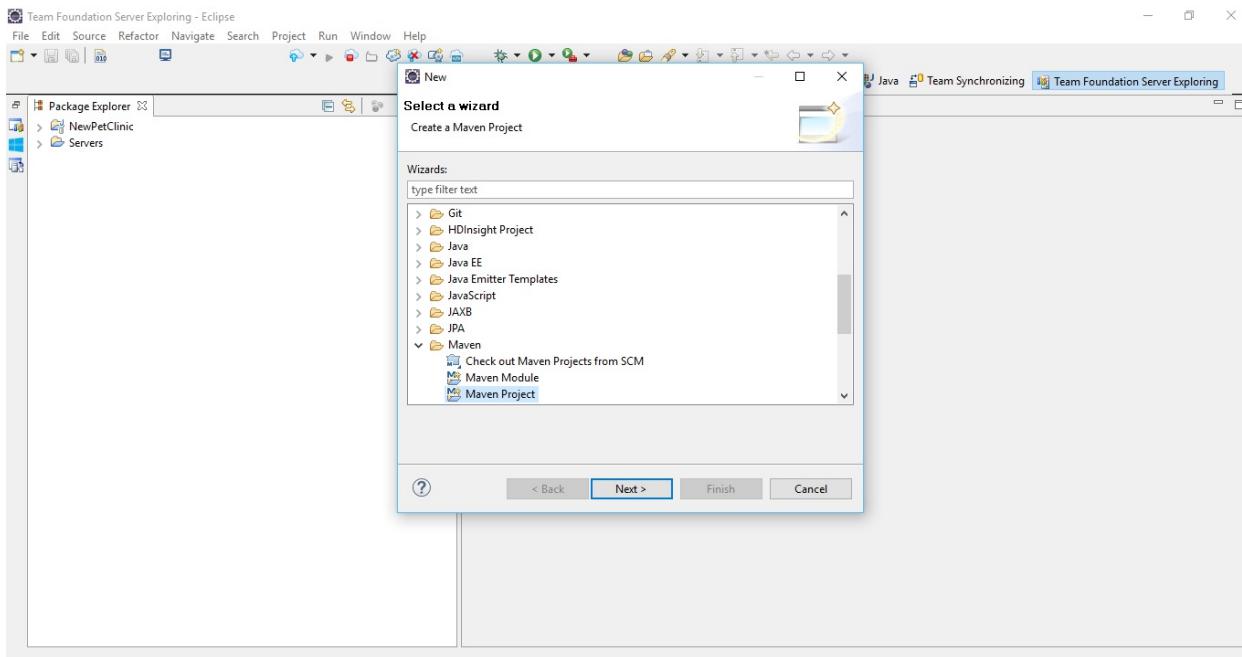
In this chapter, we will use Selenium and Eclipse for a functional test case execution. We have already deployed the application in the Tomcat, so we can perform functional tests and load tests on that deployment.

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

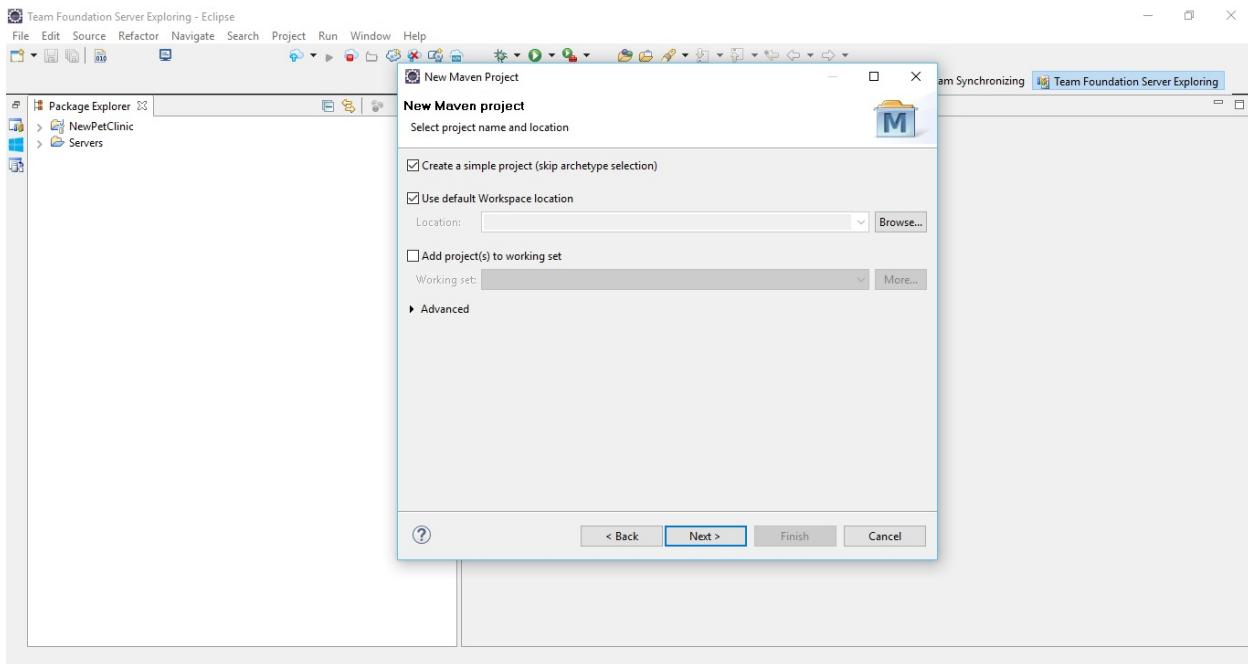
The PetClinic project is a Maven-based Spring application, and we will create a test case using Eclipse and Maven. We will utilize the `m2eclipse` plugin in Eclipse.

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`:

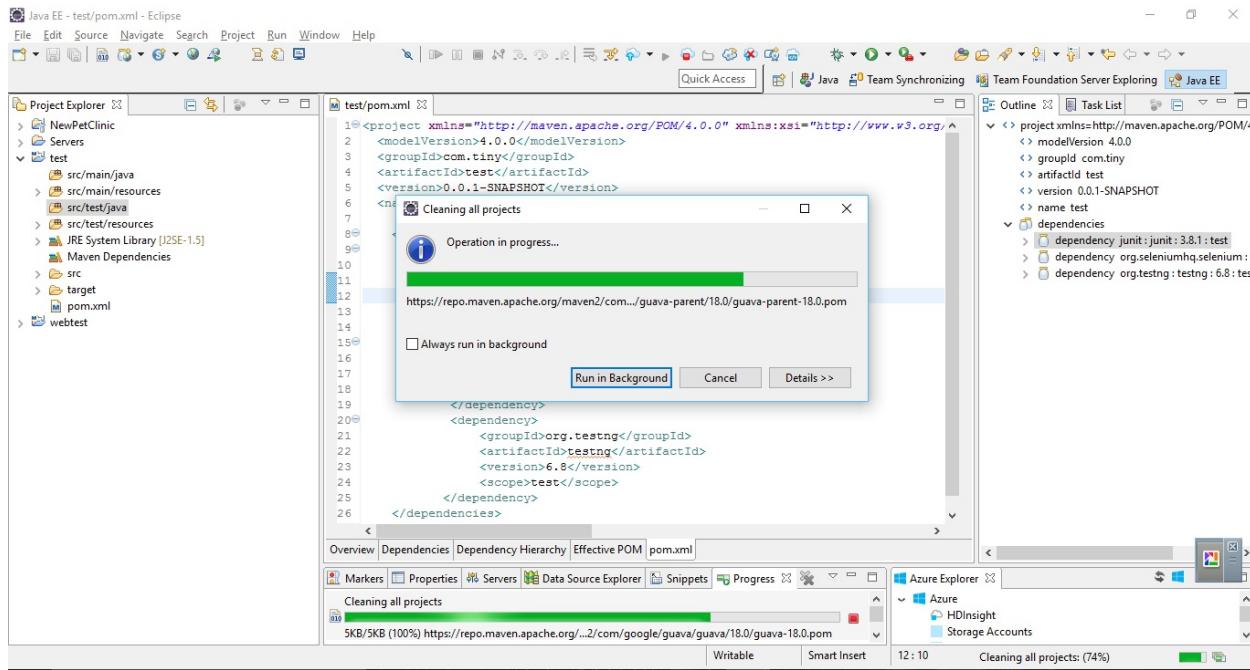
```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tiny</groupId>
  <artifactId>test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

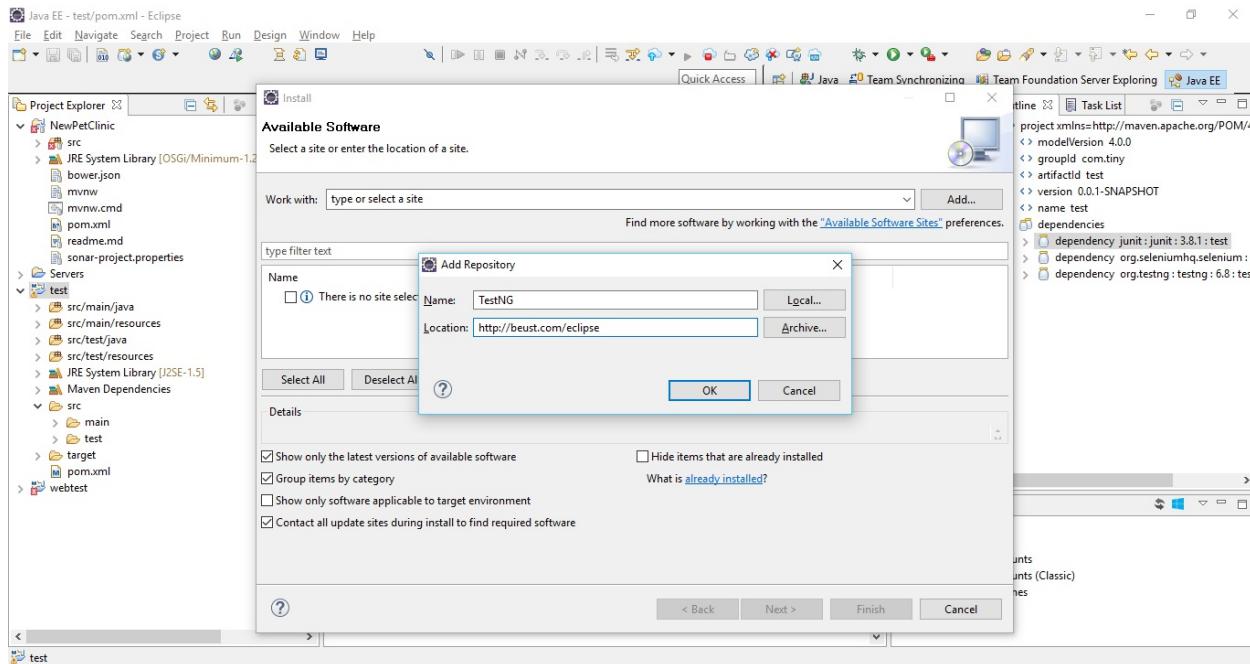
```

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

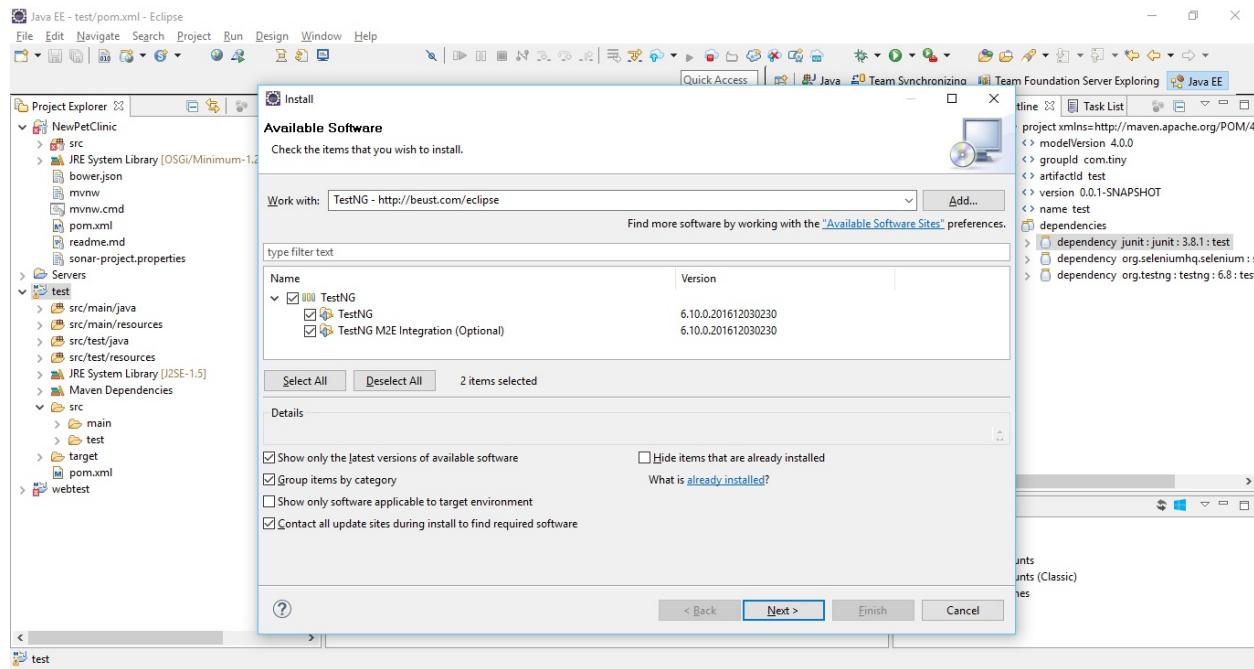


Create a Maven Project in Eclipse IDE

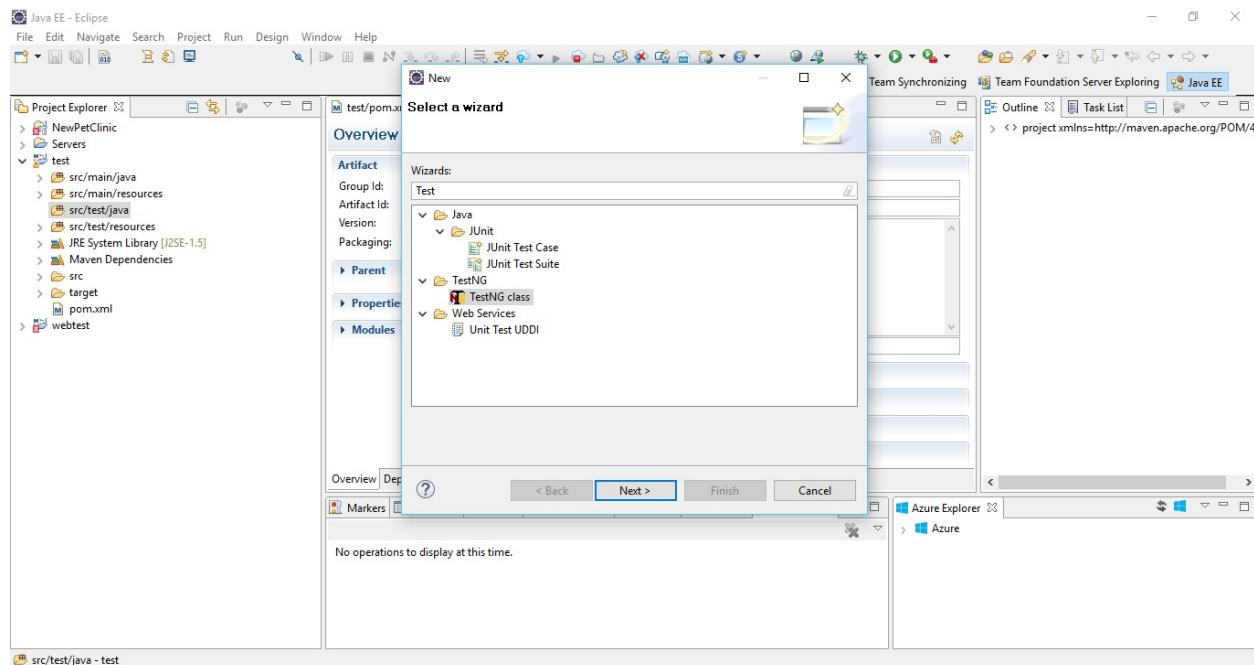
8. Click on the **Details** button of the dialog box to verify the operation in progress.
9. The next task is to write the `TestNG` class. Install the `TestNG` plugin. Go to `Help` and click on `Install New Software` and `Add Repository`:



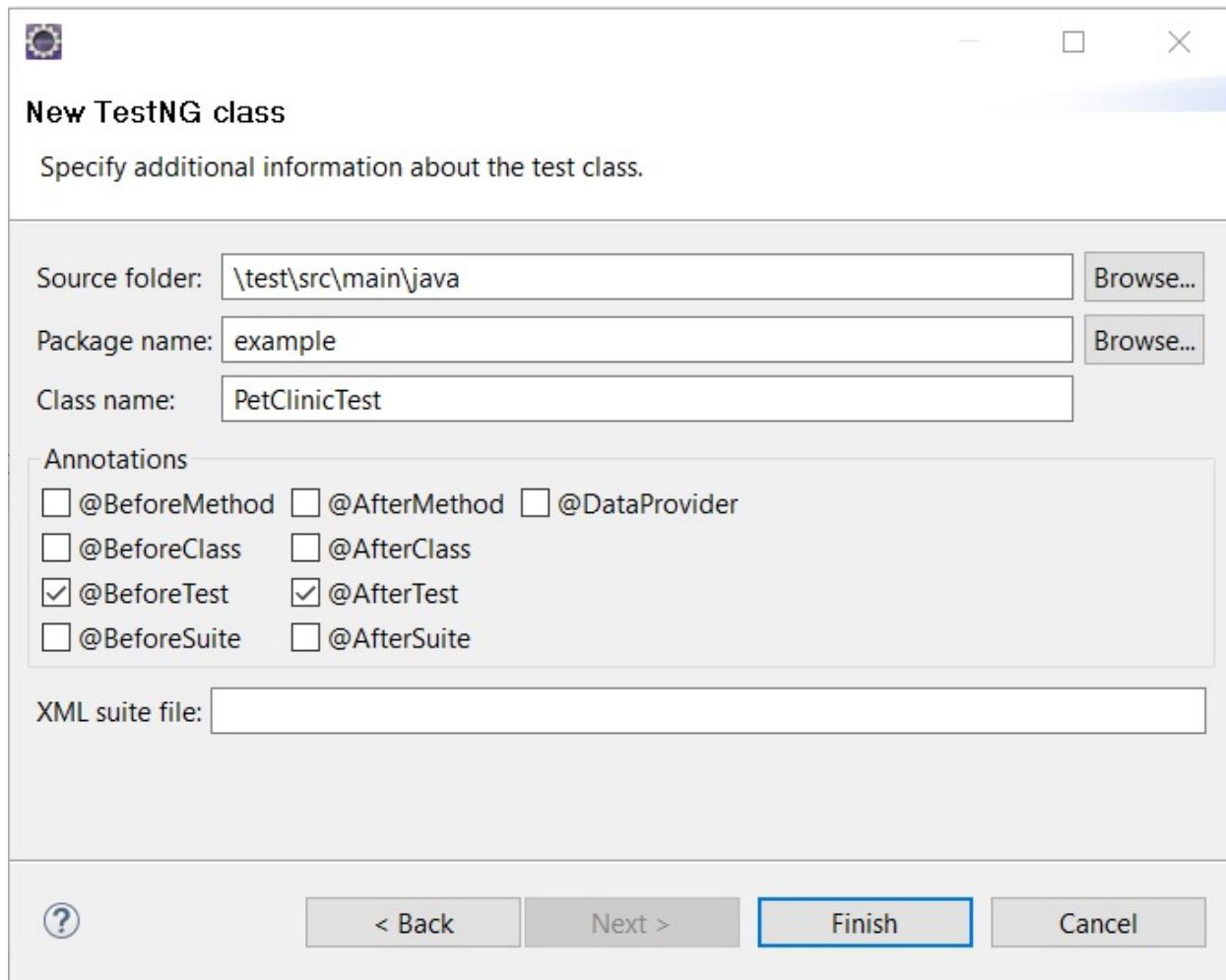
10. Select the items we need to install:



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**:

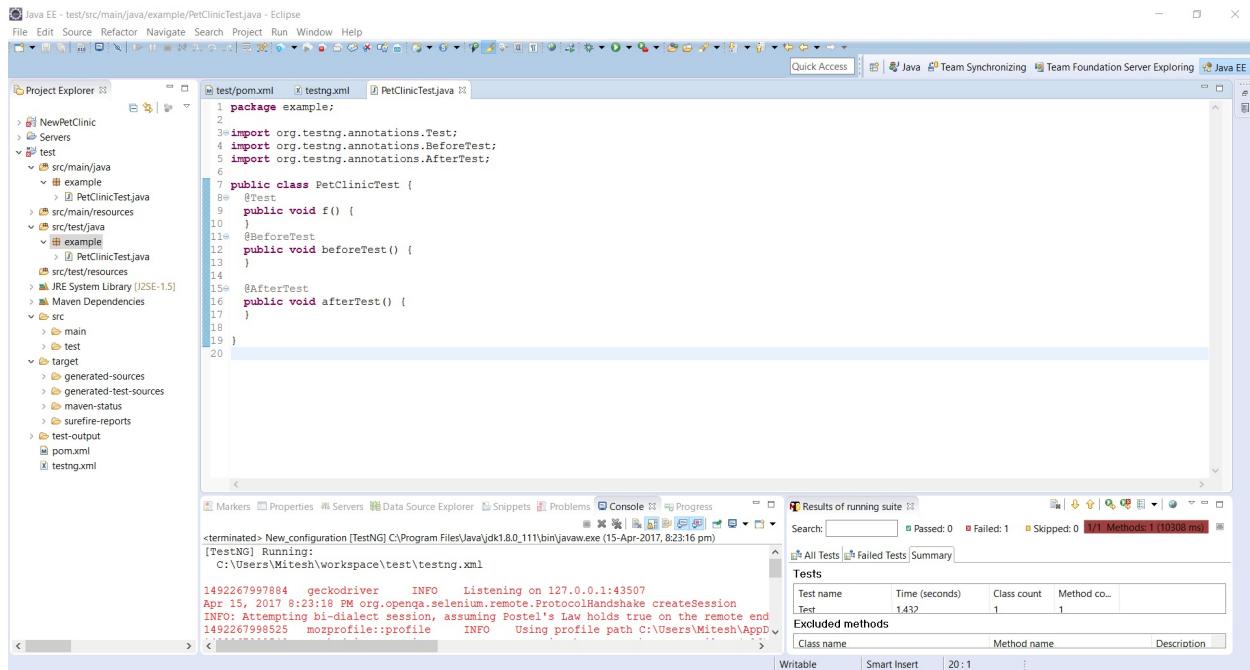


15. Provide a class name:

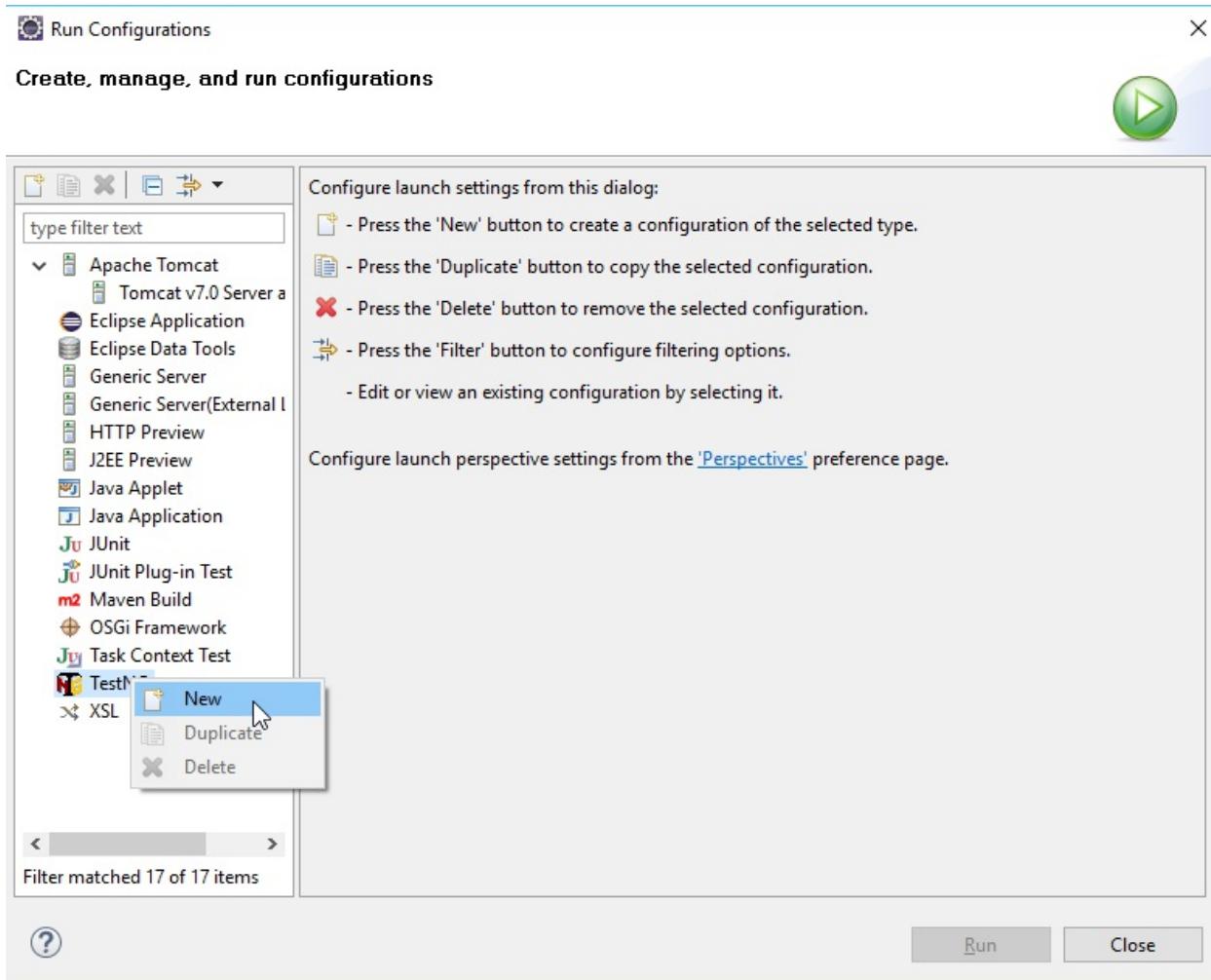


16. Give a package name and click on **Finish**.

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

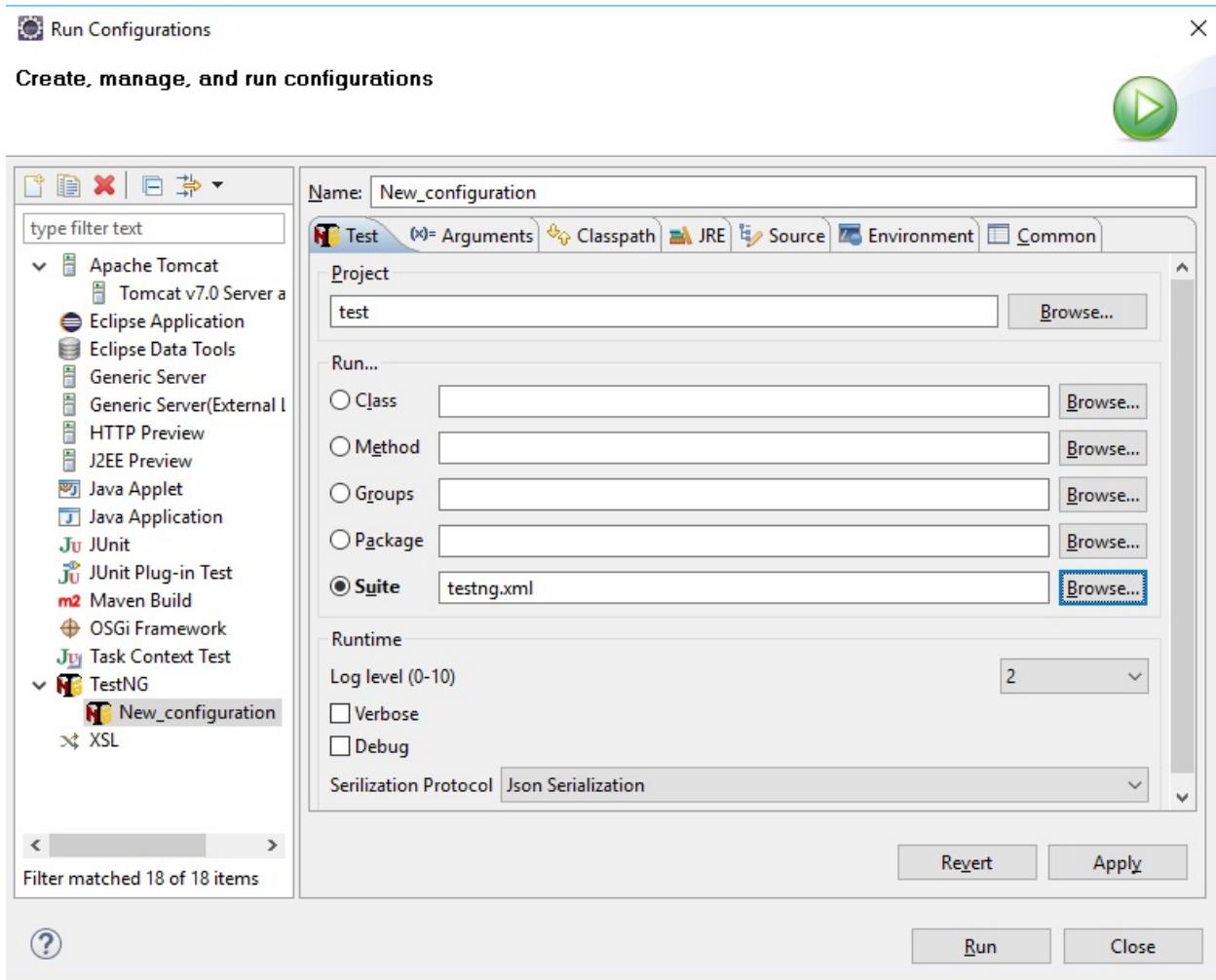


18. Right-click on the test file and click on **TestNG**, convert to TestNG.
19. This will create a `testing.xml` file that has details about the test suite.
20. Right-click on project and click on **Run Configurations**.
21. Right-click on **TestNG** and click on **New**:



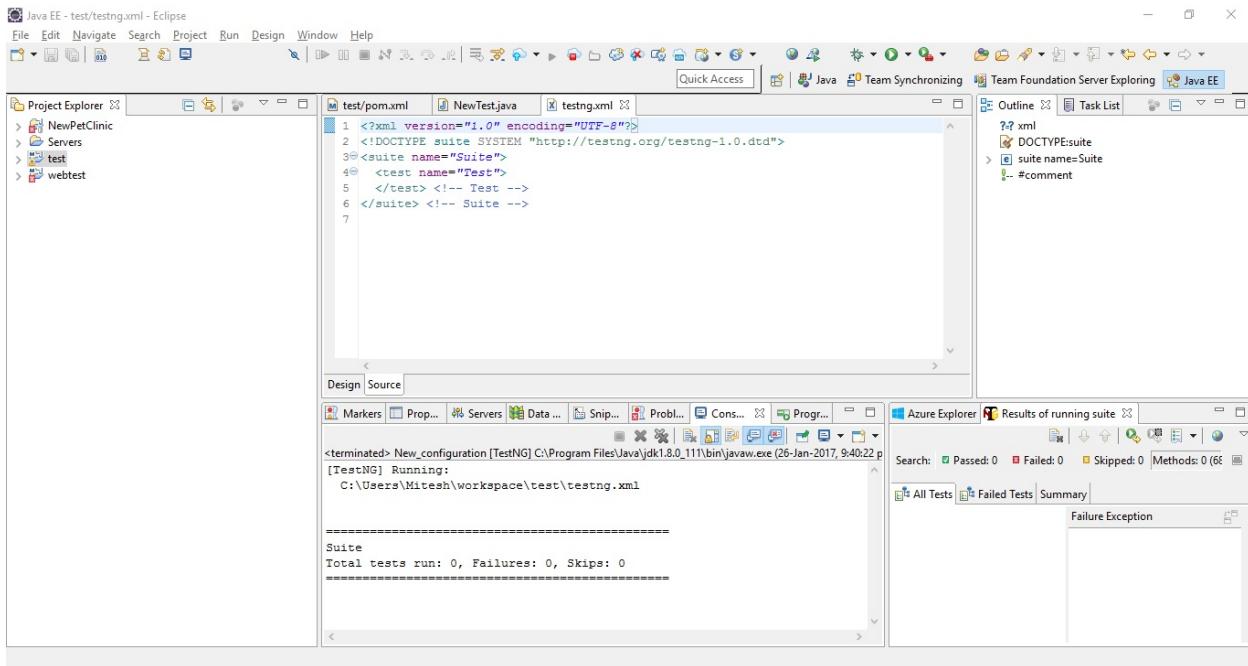
22. Provide the project name and select `testing.xml` in the suite.
23. Click **OK** and **Apply**.

24. Click on **Run**:



25. If Windows Firewall blocks it, then click on **Allow Access**.
26. There is no configuration available in `testing.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 in
    }
}

```

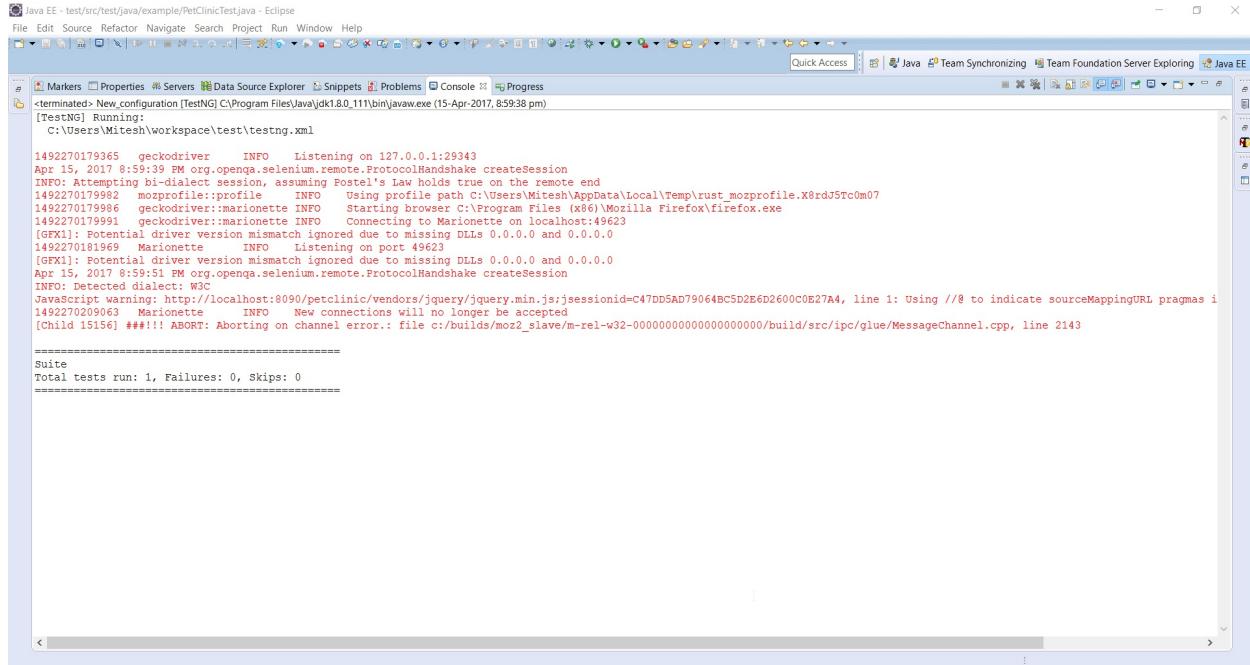
```

        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 b
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:



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following log message:

```

java EE - test/src/test/java/example/PetClinicTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Markers Properties Servers Data Source Explorer Snippets Problems Console Progress
<terminated> New_configuration [TestNG] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (15-Apr-2017 8:59:38 M)
[TestNG] Running:
C:\Users\Witesh\workspace\test\testng.xml

1492270179365 geckodriver INFO Listening on 127.0.0.1:29343
Apr 15, 2017 8:59:39 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Attempting bi-dialect session, assuming Postel's Law holds true on the remote end
1492270179982 mozprofile:profile INFO Using profile path C:\Users\Witesh\AppData\Local\Temp\rust_mozprofile.X8rdj5Tc0m07
1492270179986 geckodriver::marionette INFO Starting browser C:\Program Files (x86)\Mozilla Firefox\firefox.exe
1492270179991 geckodriver::marionette INFO Connecting to Marionette on localhost:49623
[GFX1]: Potential driver version mismatch ignored due to missing DLLs 0.0.0.0 and 0.0.0.0
1492270181969 Marionette INFO Listening on port 49623
[GFX1]: Potential driver version mismatch ignored due to missing DLLs 0.0.0.0 and 0.0.0.0
Apr 15, 2017 8:59:51 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
JavaScript warning: http://localhost:8090/petclinic/vendors/jquery/jquery.min.js:jsessionid=C47DD5AD79064BC5D2E6D2600C0E27A4, line 1: Using //@ to indicate sourceMappingURL pragmas is
1492270209063 Marionette INFO New connections will no longer be accepted
[Child 15156] !!!!!! ABORT: Aborting on channel error.: file c:/builds/moz2_slave/m-rel-w32-00000000000000000000/build/src/ipc/glue/MessageChannel.cpp, line 2143

=====
Suite
Total tests run: 1, Failures: 0, Skips: 0
=====
```

34. Check the **All Tests** tab in the **Results** of the running suite section in

Eclipse. We can see successful execution here:

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Displays the project structure. The `PetClinicTest.java` file is open in the editor.
- Editor:** Shows the code for `PetClinicTest.java`. The code includes imports, class definition, and several test methods (`testPetClinic`, `beforeTest`, `afterTest`, `beforeTest0`, `afterTest0`, `testPetClinic0`) with annotations like `@Test`, `@BeforeTest`, and `@AfterTest`.
- Console:** Shows the output of the test execution. It includes logs from Selenium WebDriver and the TestNG framework. The log ends with "INFO: Detected dialect: W3C".
- Results of running suite:** A table showing the execution results:

Category	Count	Time
Passed	1	17.237 ms
Failed	0	
Skipped	0	
- Summary:** A summary table showing the overall status:

Category	Count	Time
All Tests	1	17.237 ms
Suite	1	17.237 ms
Test	1	17.237 ms
example.PetClinicTest	1	17.237 ms
testPetClinic	1	17.237 ms

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:

java EE - test/src/test/java/example/PetClinicTest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Markers Properties Servers Data Source Explorer Snippets Problems Console Progress

<terminated> New_configuration [TestNG] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (15-Apr-2017, 9:01:31 pm)

[TestNG] Running:
C:\Users\Mitesh\workspace\test\testng.xml

```
1492270293305 geckodriver INFO Listening on 127.0.0.1:48101
Apr 15, 2017 9:01:33 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Attempting bi-dialect session, assuming Postel's Law holds true on the remote end
1492270294014 mozprofile:profile INFO Using profile path C:\Users\Mitesh\AppData\Local\Temp\rust_mozprofile.7DuMKZqPmoby
1492270294018 geckodriver:marionette INFO Starting browser C:\Program Files (x86)\Mozilla Firefox\firefox.exe
1492270294100 geckodriver:marionette INFO Connecting to Marionette on localhost:49679
[GECKO]: Potential driver version mismatch ignored due to missing DLLs 0.0.0.0 and 0.0.0.0
1492270301548 geckodriver:marionette INFO Listening on port 49679
[GFWX]: Potential driver version mismatch ignored due to missing DLLs 0.0.0.0 and 0.0.0.0
Apr 15, 2017 9:01:54 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
JavaScript warning: http://localhost:8090/petcclinic/vendors/jquery/jquery.min.js:jsessionid=C30BB777FF19B0D7CP6040969C6E7601, line 1: Using //@ to indicate sourceMappingURL pragmas is deprecated
1492270326107 Marionette INFO New connections will no longer be accepted
[Child 4368] ##### ABORT: Aborting on channel error.: file c:/builds/moz2_slave/m-rel-w32-00000000000000000000/build/src/ipc/glue/MessageChannel.cpp, line 2143
```

=====

Suite
Total tests run: 1, Failures: 1, Skips: 0
=====

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

java EE - test/src/test/java/example/PetClinicTest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Markers Properties Servers Data Source Explorer Snippets Problems Console Progress

Project Explorer

- NewPetClinic
- Servers
- test
 - src/main/java
 - src/main/resources
 - src/test/java
 - example
 - PetClinicTest.java
 - PetClinicTest
 - driver
 - afterTest0 : void
 - beforeTest0 : void
 - testPetClinic0 : void
 - src/test/resources
 - JRE System Library [JSE-1.5]
 - Maven Dependencies
 - src
 - main
 - test
 - target
 - generated-sources
 - generated-test-sources
 - maven-status
 - surefire-reports
 - test-output
 - pom.xml
 - testing.xml

PetClinicTest.java

```
package example;
import java.io.File;
public class PetClinicTest {
    private WebDriver driver;
    @Test
    public void testPetClinic() {
        driver.get("http://localhost:8090/petcclinic/");
        String title = driver.getTitle();
        Assert.assertTrue(title.contains("a Spring Framework"));
    }
    @BeforeTest
    public void beforeTest() {
        File file = new File("F:\\##DevOpsBootCamp\\geckodriver-v0.13.0-win64\\geckodriver.exe");
        System.setProperty("webdriver.gecko.driver", file.getAbsolutePath());
        driver = new FirefoxDriver();
    }
    @AfterTest
    public void afterTest() {
        driver.quit();
    }
}
```

<terminated> New_configuration [TestNG] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (15-Apr-2017, 9:01:31 pm)

[GECKO]: Potential driver version mismatch ignored due to missing DLLs 0.0.0.0 and 0.0.0.0

Apr 15, 2017 9:01:54 PM org.openqa.selenium.remote.ProtocolHandshake createSession

INFO: Detected dialect: W3C

JavaScript warning: http://localhost:8090/petcclinic/vendors/jquery/jquery.min.js:jsessionid=C30BB777FF19B0D7CP6040969C6E7601, line 1: Using //@ to indicate sourceMappingURL pragmas is deprecated

1492270326107 Marionette INFO New connections will no longer be accepted

[Child 4368] ##### ABORT: Aborting on channel error.: file c:/builds/moz2_slave/m-rel-w32-00000000000000000000/build/src/ipc/glue/MessageChannel.cpp, line 2143

=====

Suite
Total tests run: 1, Failures: 1, Skips: 0
=====

Results of running suite

Search: | Passed: 0 Failed: 1 Skipped: 0 / 1 Methods: (0.440 ms)

All Tests Failed Tests Summary

1492270301548 Marionette INFO Listening on port 49679

[GECKO]: Potential driver version mismatch ignored due to missing DLLs 0.0.0.0 and 0.0.0.0

Apr 15, 2017 9:01:54 PM org.openqa.selenium.remote.ProtocolHandshake createSession

INFO: Detected dialect: W3C

JavaScript warning: http://localhost:8090/petcclinic/vendors/jquery/jquery.min.js:jsessionid=C30BB777FF19B0D7CP6040969C6E7601, line 1: Using //@ to indicate sourceMappingURL pragmas is deprecated

1492270326107 Marionette INFO New connections will no longer be accepted

[Child 4368] ##### ABORT: Aborting on channel error.: file c:/builds/moz2_slave/m-rel-w32-00000000000000000000/build/src/ipc/glue/MessageChannel.cpp, line 2143

=====

Suite
Total tests run: 1, Failures: 1, Skips: 0
=====

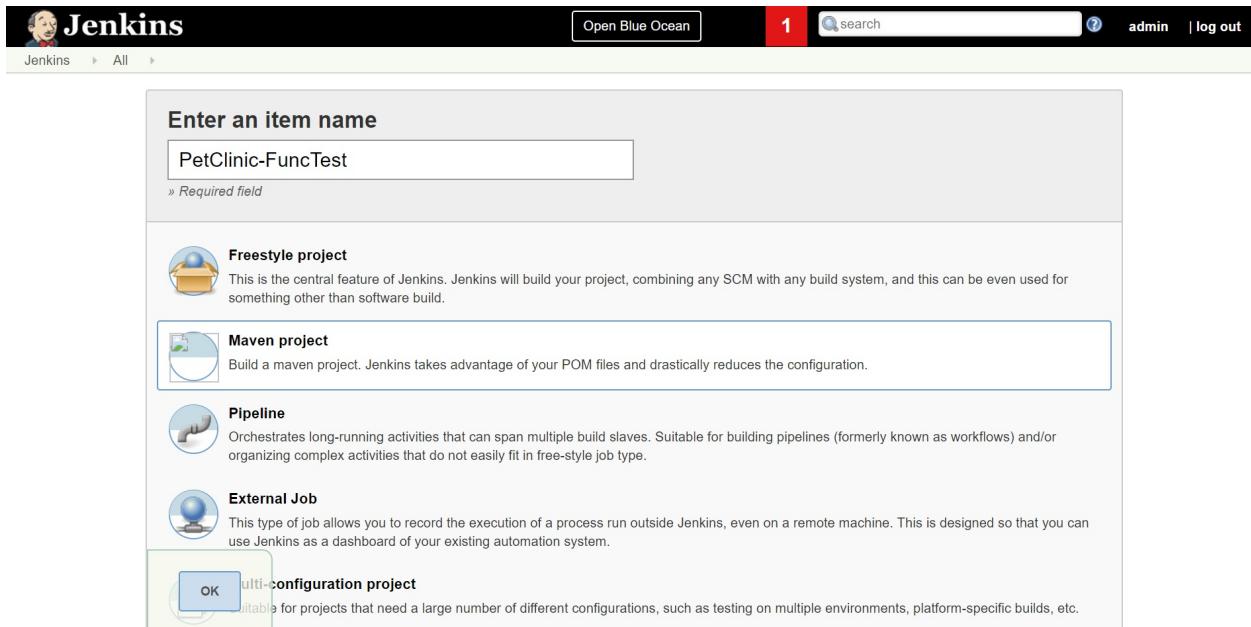
40. Observe the **Failed Tests** tab in the **Results** of the running suite section in Eclipse.

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:



The screenshot shows the Jenkins dashboard with a search bar and a notification badge indicating 1 new item. A form titled "Enter an item name" contains the text "PetClinic-FuncTest" with a note "» Required field". Below this, there are five project creation options:

- Freestyle project**: Described as the central feature of Jenkins, combining any SCM with any build system.
- Maven project**: Described as building a maven project, utilizing POM files.
- Pipeline**: Described as orchestrating long-running activities across multiple build slaves.
- External Job**: Described as recording a process run outside Jenkins, suitable for dashboards.
- Multi-configuration project**: Described as being suitable for projects with many configurations, like testing across environments.

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:

Jenkins > PetClinic-FuncTest >

General Source Code Management Build Triggers Build Environment Pre Steps **Build** Post Steps Build Settings

Post-build Actions

Build

Root POM: pom.xml
Goals and options: test

Post Steps

Run only if build succeeds Run only if build succeeds or is unstable Run regardless of build result
Should the post-build steps run only for successful builds, etc.

Add post-build step ▾

Build Settings

E-mail Notification

Post-build Actions

Save **Apply**

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.t3uOSiy560nn
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 0.0.0.0 and 0.0.0.0
[GFX1]: Potential driver version mismatch ignored due to missing DLLs 0.0.0.0 and 0.0.0.0
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;jsessionid=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 with the text "Testng". Below the search bar, there are four tabs: "Updates", "Available" (which is selected), "Installed", and "Advanced". Under the "Available" tab, there is a table with one row. The row contains a checkbox (which is checked), the name "TestNG Results Plugin", and the version "1.14". At the bottom of the table, there are three buttons: "Install without restart", "Download now and install after restart", and "Check now". To the right of the "Check now" button, it says "Update information obtained: 17 hr ago".

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

Install without restart Download now and install after restart Check now

Update information obtained: 17 hr ago

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

PetClinic-FuncTest

General Source Code Management Build Triggers Build Environment Pre Steps **Build** Post Steps Build Settings

Post-build Actions

- Aggregate downstream test results
- Archive the artifacts
- Build other projects
- Calculate disk usage of build
- Deploy artifacts to Maven repository
- Publish Performance test result report
- Publish TestNG Results**
- Quality Gates Sonarqube Plugin
- Record fingerprints of files to track usage
- Git Publisher
- SonarQube analysis with Maven
- Deploy war/ear to a container
- Editable Email Notification
- Set GitHub commit status (universal)

Run only if build succeeds Run only if build succeeds or is unstable Run regardless of build result
is run only for successful builds, etc.

Add post-build action ▾

Save **Apply**

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

Post-build Actions

Publish TestNG Results

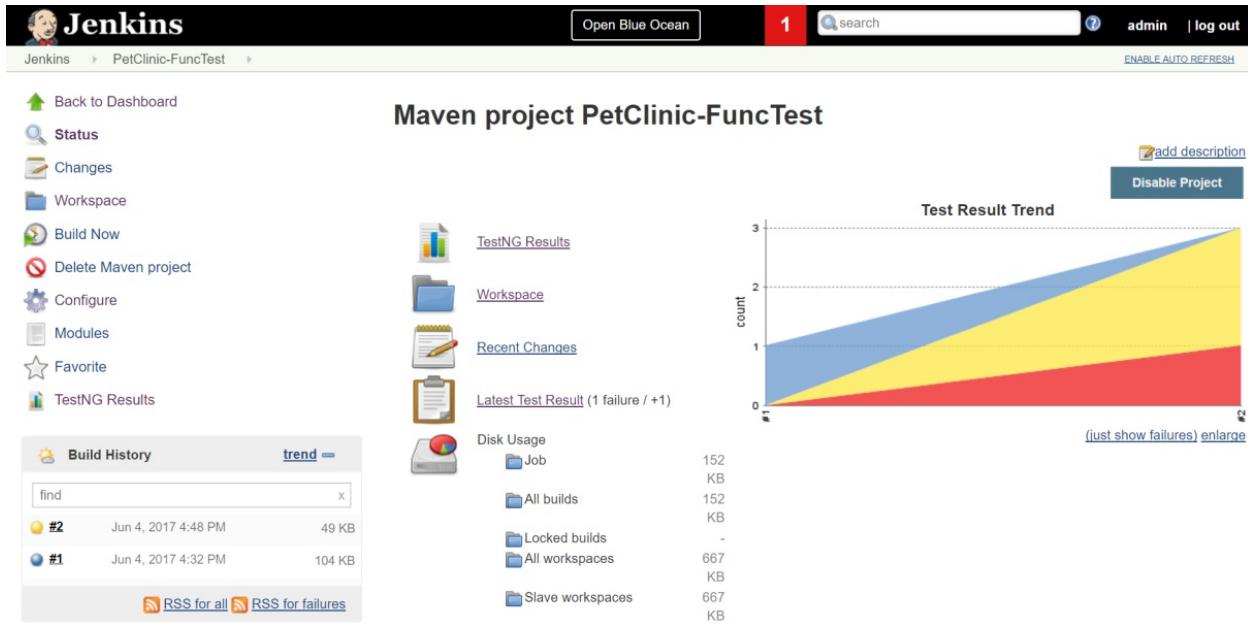
TestNG XML report pattern `**/testng-results.xml`

Advanced...

Add post-build action ▾

Save **Apply**

9. Click on **Build now**:



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

TestNG Results Trends

Need at least 2 builds with results to show trend graph

Latest Test Results ([build #2](#))

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

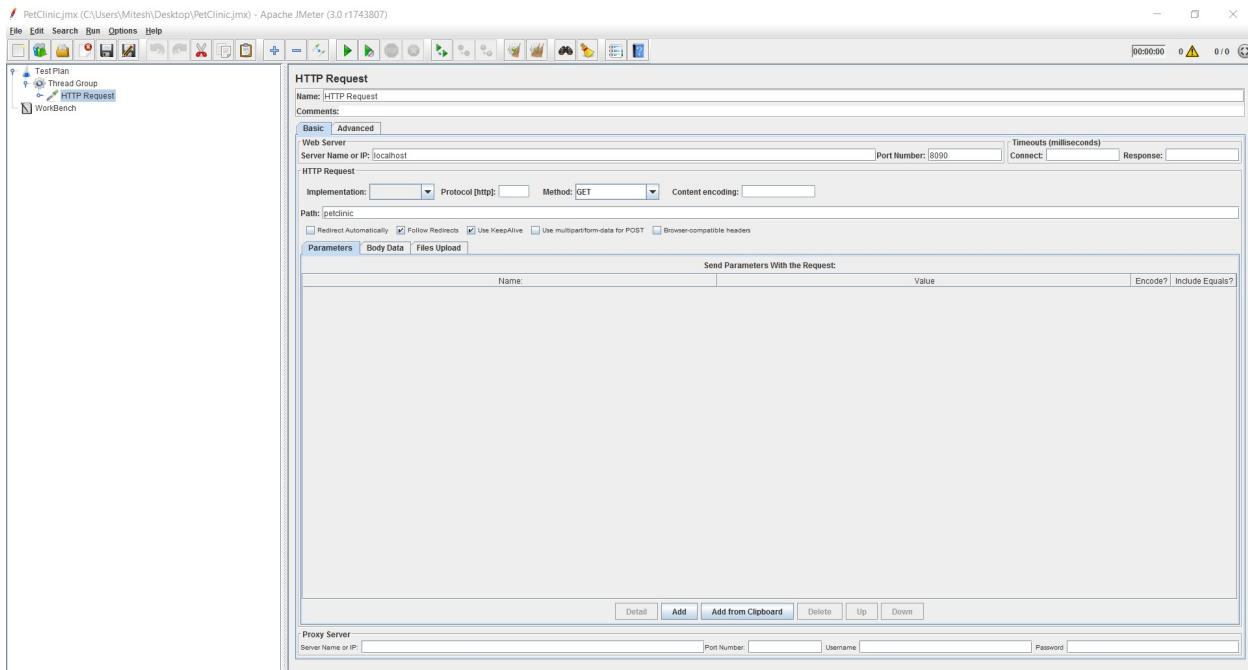
We have seen how to execute Selenium-based test cases in Jenkins. In the next section, we will see how to execute a load test using Jenkins.

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:

Enter an item name

PetClinic-LoadTest

» Required field

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 build.

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

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
```

Jenkins > PetClinic-LoadTest >

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build**
- Post-build Actions

Build

Execute Windows batch command

Command

```
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
```

See [the list of available environment variables](#)

Advanced...

Add build step ▾

Post-build Actions

Add post-build action ▾

Save Apply

3. Add Post-build Actions:

Jenkins > PetClinic-LoadTest >

- Triggers
- Build Environment
- Build
- Post-build Actions**

Publish Performance test result report

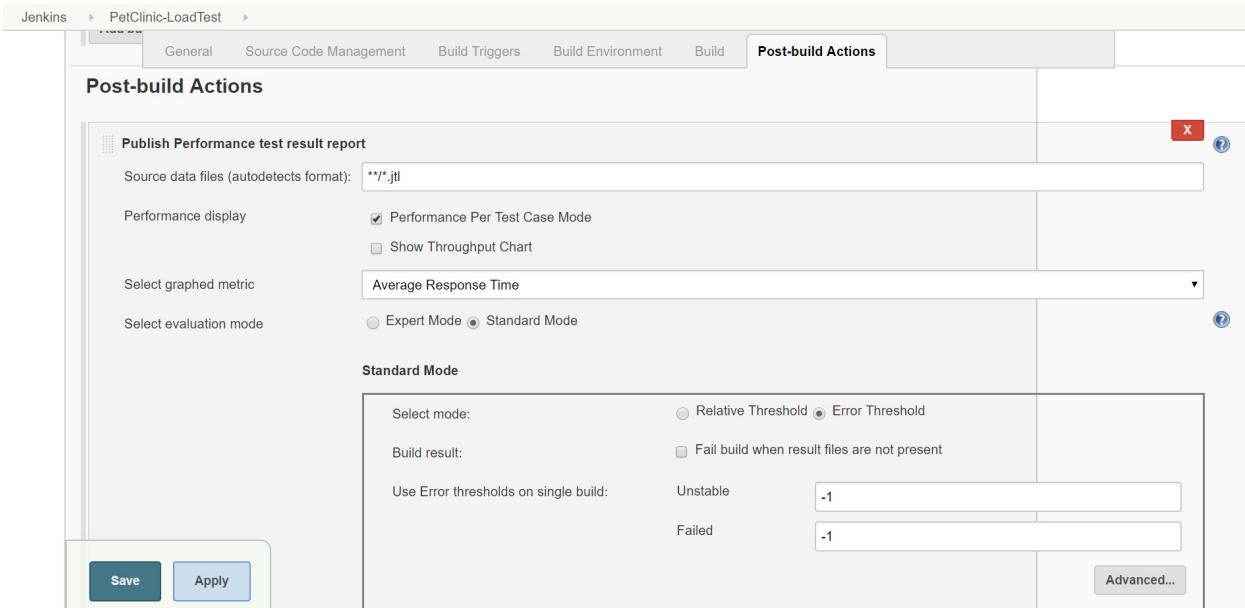
See [the list of available environment variables](#)

Advanced...

Add post-build action ▾

Save Apply

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



5. Click on Build now:

Console Output

```
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\jenkins1396957989406107476.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
```

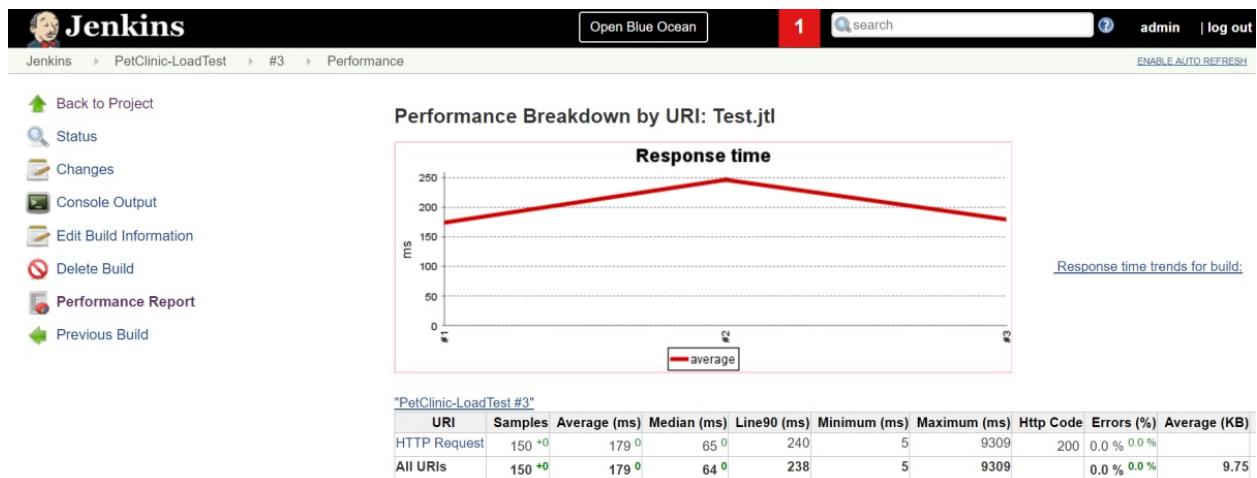
- Verify **Performance Trend** on the Project dashboard by clicking on the Test results graph.
- Click on **Performance Trend**:

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, Favorite, and Performance Trend. The main area has a title "Project PetClinic-LoadTest". Below it, there are sections for "Workspace", "Last Successful Artifacts" (with a link to "dashBoard_Test.xml" and size 453 B), "Recent Changes", and "Disk Usage". To the right, there are two performance trend graphs: "Response time" (a line graph with three lines: 90% line in red, average in blue, and median in green) and "Percentage of errors" (a bar chart showing 0% errors across three categories). At the bottom, there's a "Permalinks" section with links to the last build, stable build, and successful build.

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

The screenshot shows the Jenkins Performance Trend page for the test file "Test.jtl". The sidebar is identical to the previous dashboard. The main area has a title "Performance Trend" with "Last Report" and "Filter trend data" buttons. It features two large graphs: "Response time" (a line graph with a red line labeled "HTTP Request") and "Percentage of errors" (a bar chart with a red bar labeled "errors"). Below the graphs, there's a link "Testcase Trend". The sidebar also includes a "Build History" section with the same three builds as the dashboard.

- 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.

Chapter 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?

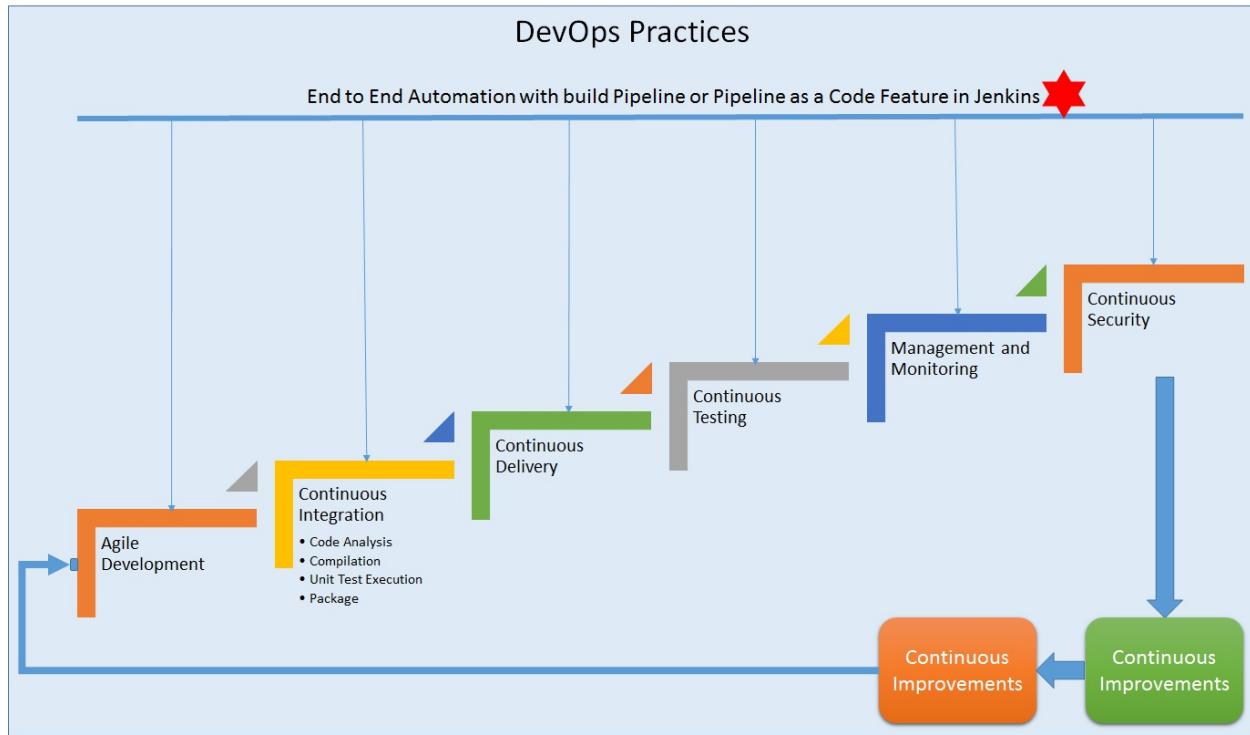
What if we want to create a pipeline where one successful execution of Job can lead to another execution of Job?

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

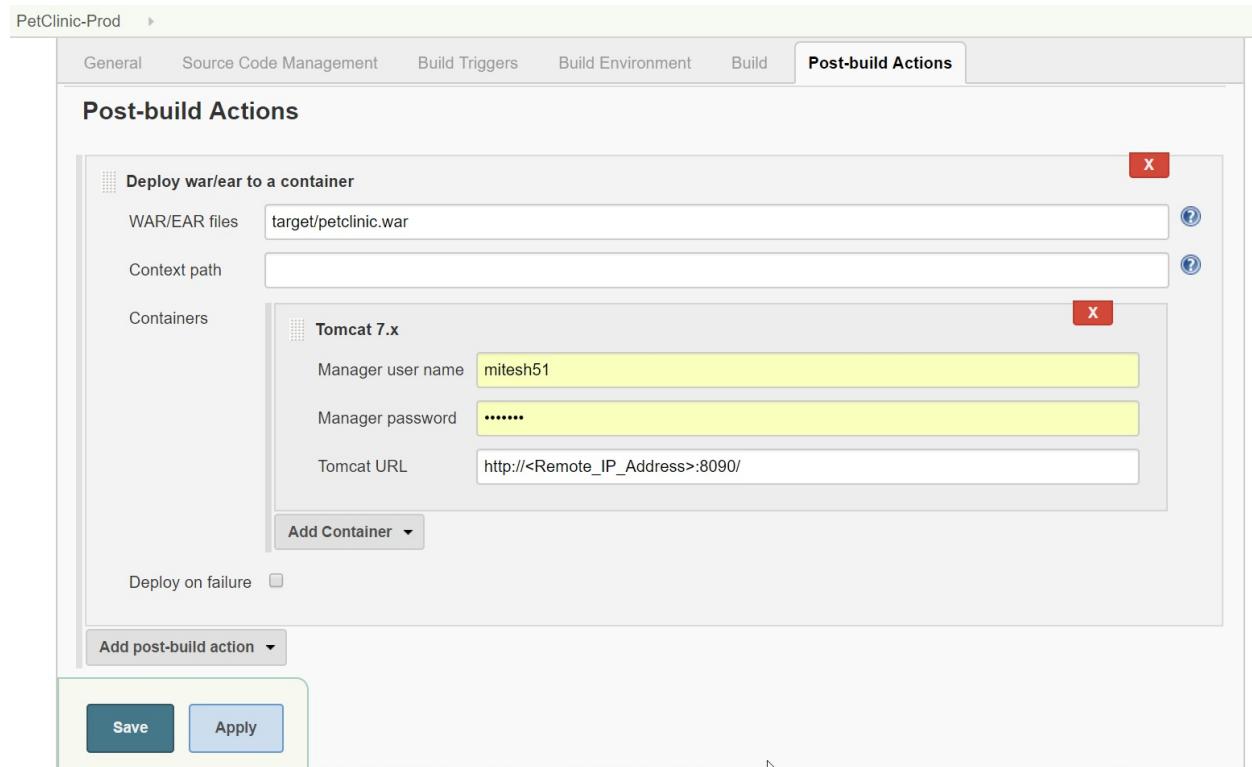
Continuous Integration and Continuous Delivery have become popular practices for application development. The Build Pipeline plugin provides a pipeline view of upstream and downstream connected jobs that typically form a build pipeline, with the ability to define manual triggers or an approval process. 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**:

The screenshot shows the Jenkins dashboard with a new item creation dialog open. The 'Item name' field is filled with 'PetClinic-Prod'. Below the field, there are five options: 'Freestyle project', 'Maven project', 'Pipeline', 'External Job', and 'Multi-configuration project'. The 'Freestyle project' option is highlighted with a blue border. A tooltip for 'Freestyle project' states: '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 build.' The 'Maven project' option has a tooltip: 'Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.' The 'Pipeline' option has a tooltip: '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.' The 'External Job' option has a tooltip: '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.' The 'Multi-configuration project' option has a tooltip: 'Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.' At the bottom right of the dialog, there is an 'OK' button.

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



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

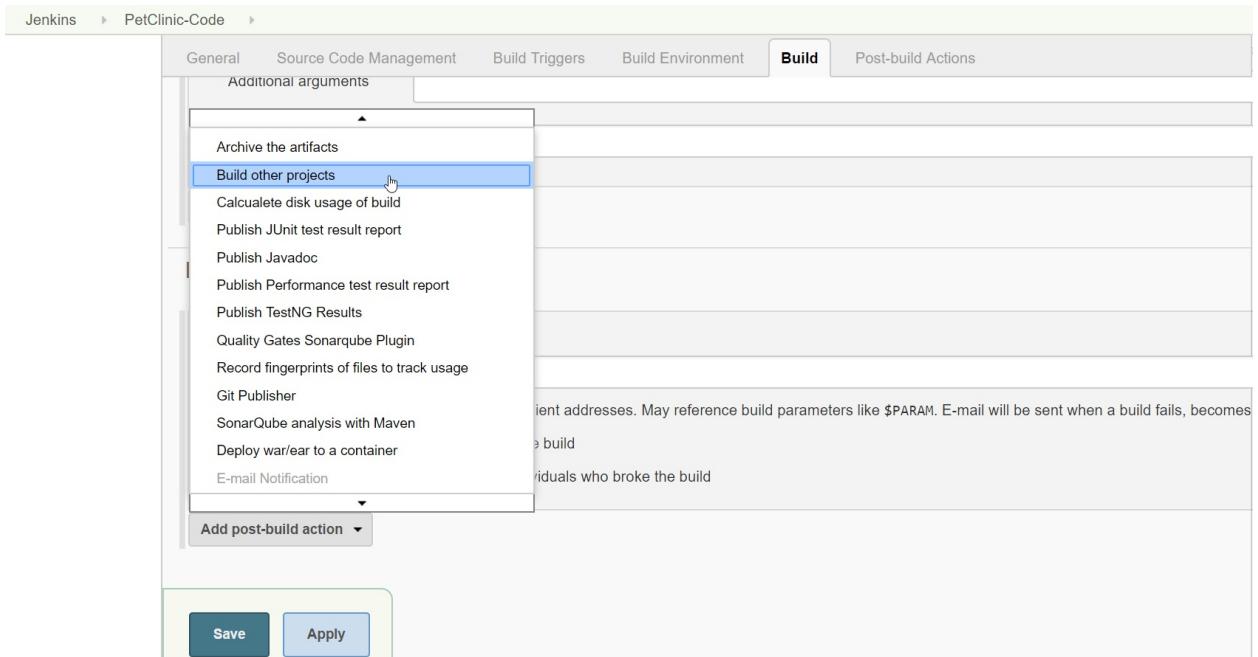
In the next section, we will configure upstream and downstream relationships between all the jobs we have created so far.

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:

PetClinic-Code

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Post-build Actions

E-mail Notification

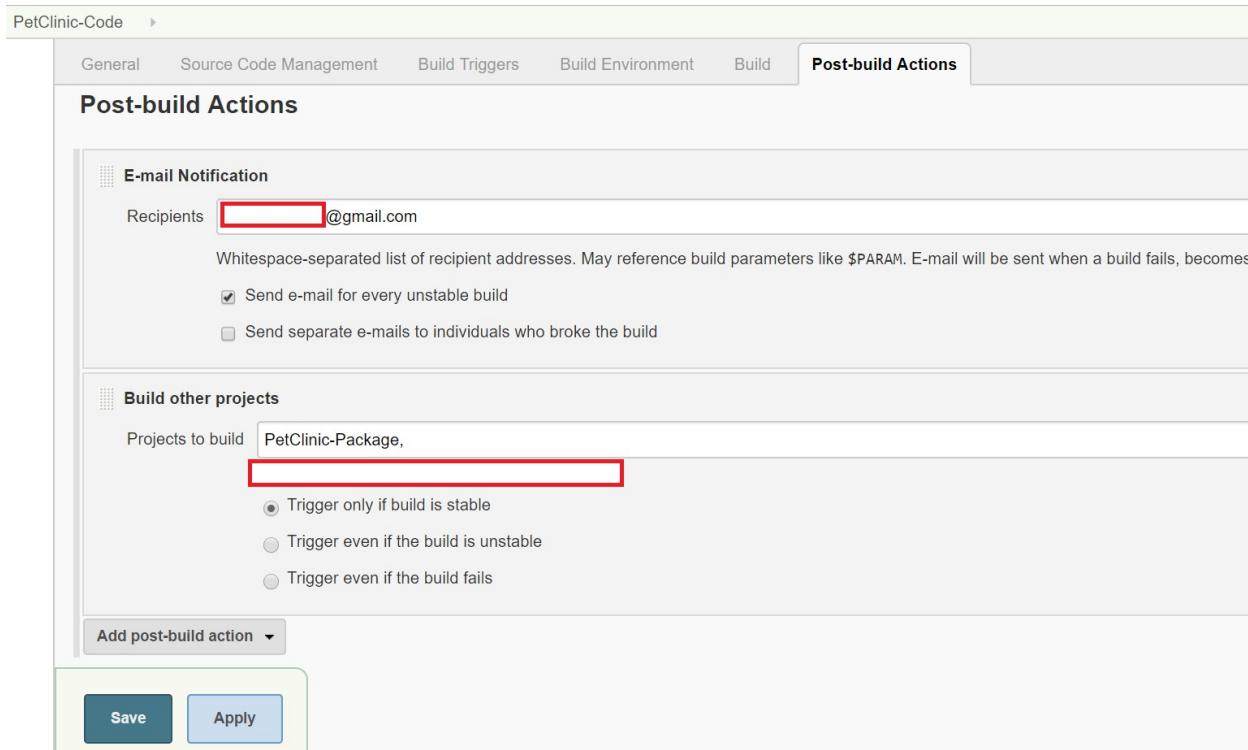
Recipients [REDACTED]@gmail.com
Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or succeeds.
 Send e-mail for every unstable build
 Send separate e-mails to individuals who broke the build

Build other projects

Projects to build PetClinic-Package, [REDACTED]
 Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Add post-build action ▾

Save Apply



- 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**:

PetClinic-Package

General Source Code Management Build Triggers Build Environment Pre Steps Build Post Steps Build Settings

Post-build Actions

E-mail Notification

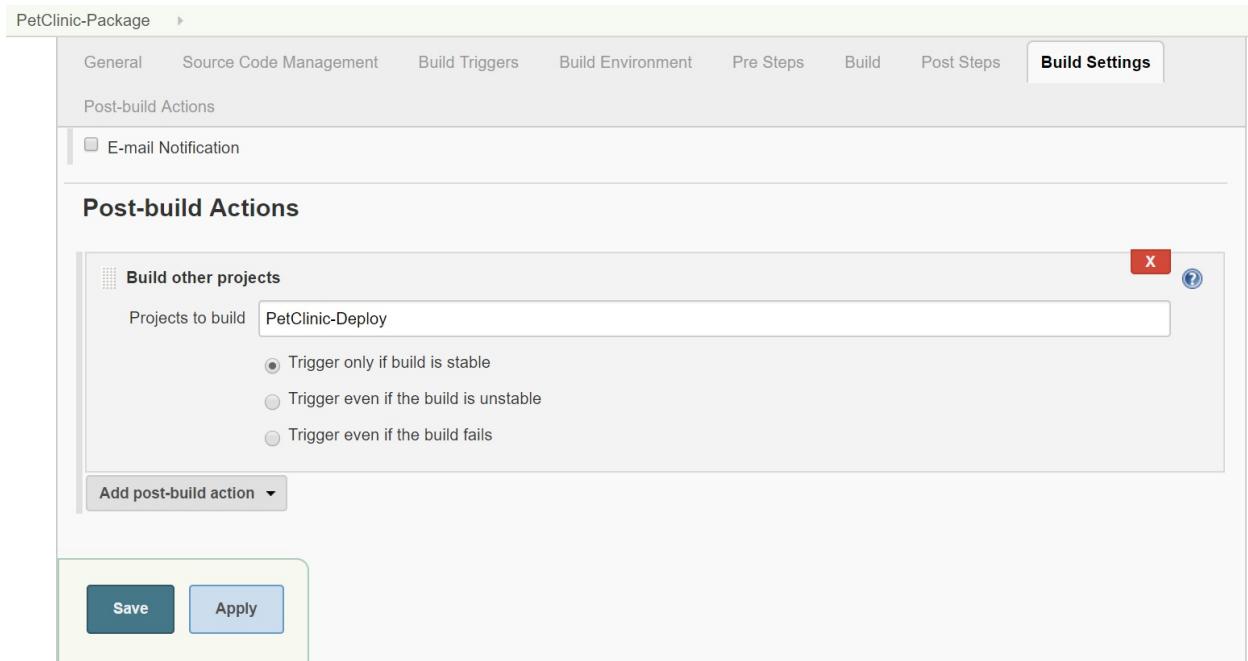
Post-build Actions

Build other projects

Projects to build PetClinic-Deploy
 Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

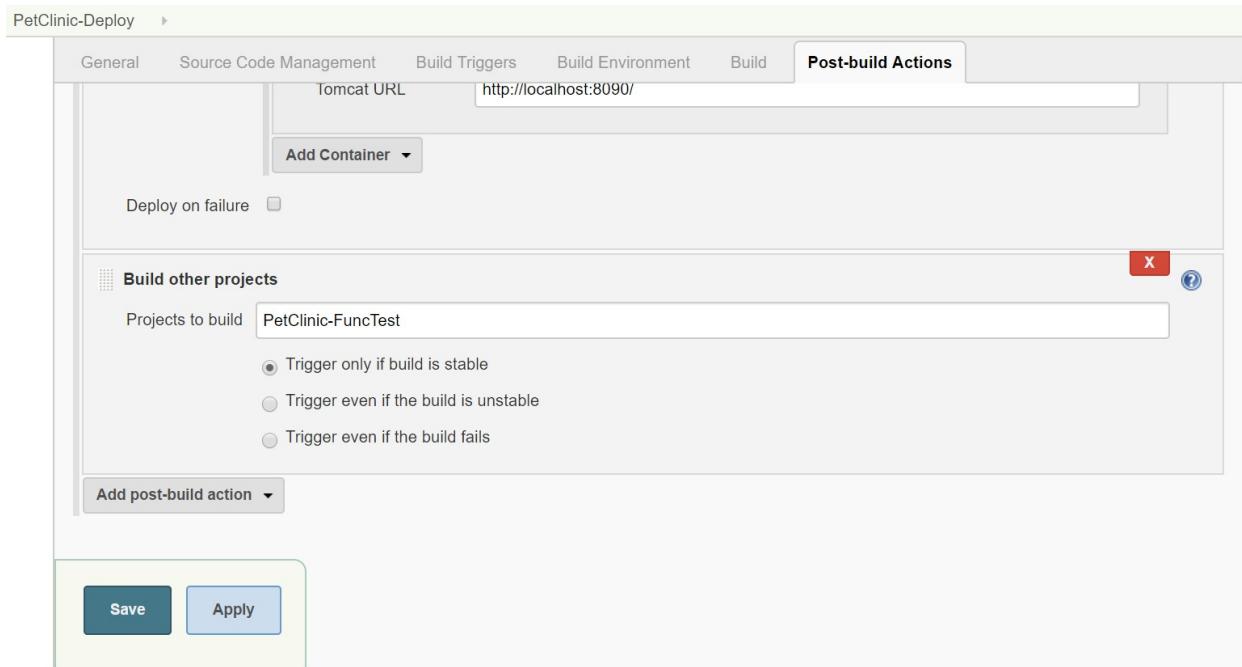
Add post-build action ▾

Save Apply



- 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**:



- 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**:

PetClinic-FuncTest

General Source Code Management Build Triggers Build Environment Pre Steps Build Post Steps Build Settings

Post-build Actions

Post-build Actions

Publish TestNG Results

TestNG XML report pattern `**/testng-results.xml`

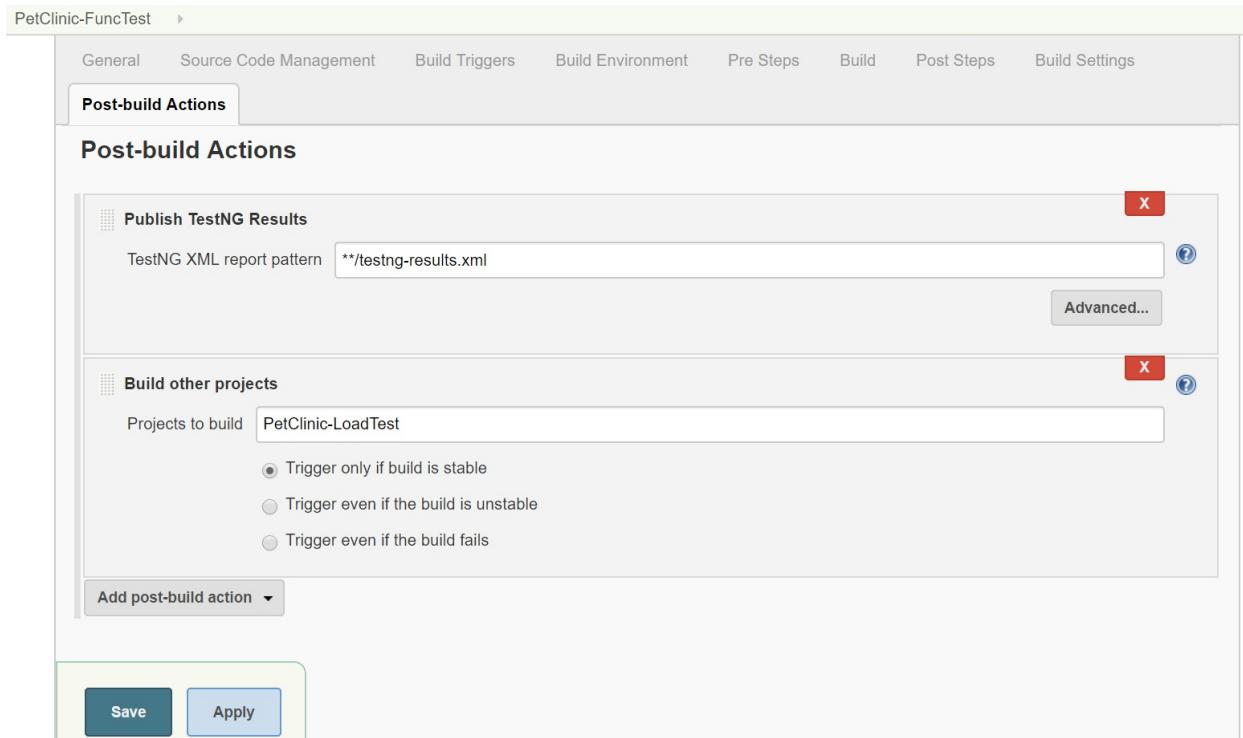
Build other projects

Projects to build PetClinic-LoadTest

Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Add post-build action ▾

Save Apply



- 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**:

PetClinic-LoadTest

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Ignore Unstable Builds
 Save constraint log to workspace

Constraints Add a new constraint ▾

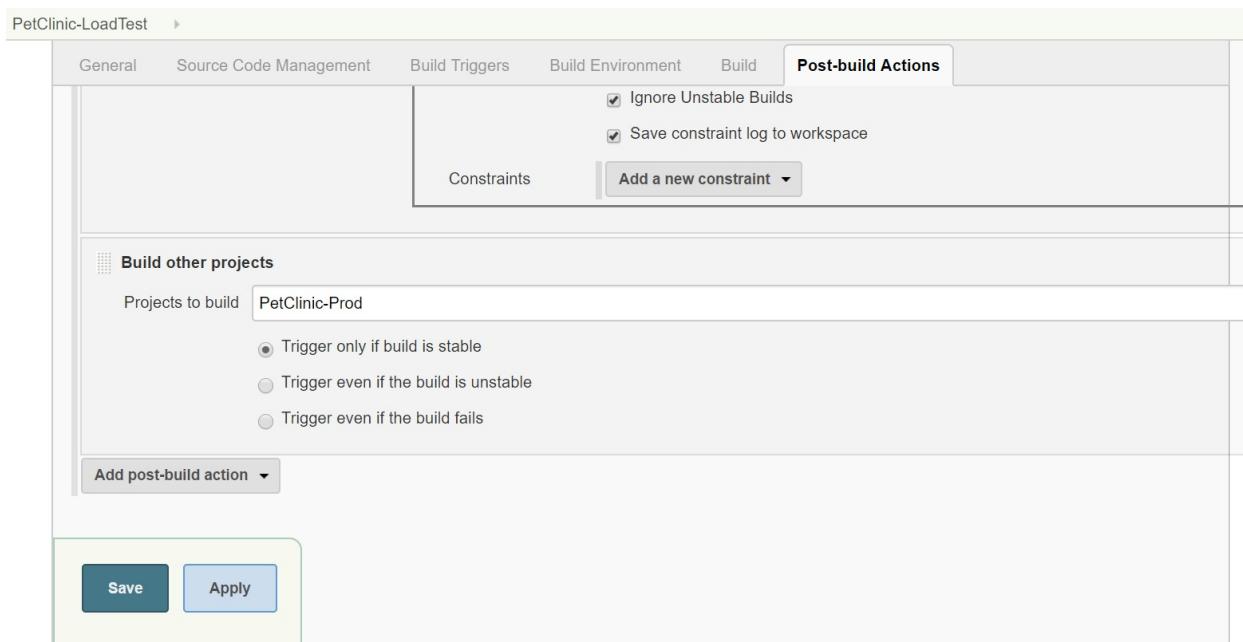
Build other projects

Projects to build PetClinic-Prod

Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Add post-build action ▾

Save Apply



9. Install the plugin from **Manage Jenkins** | **Manage Plugins**:

The screenshot shows the Jenkins 'Manage Plugins' interface. The 'Available' tab is selected. A search bar at the top right contains the text 'Build Pip'. Below the tabs is a table with columns 'Install', 'Name', and 'Version'. A single row is visible for the 'Build Pipeline Plugin', which is described as providing build pipeline functionality to Hudson and Jenkins. It includes a brief description, a checked checkbox, the version '1.5.6', and two buttons: 'Install without restart' and 'Download now and install after restart'. At the bottom right of the table area, it says 'Update information obtained: 17 hr ago' and has a 'Check now' button.

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

Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

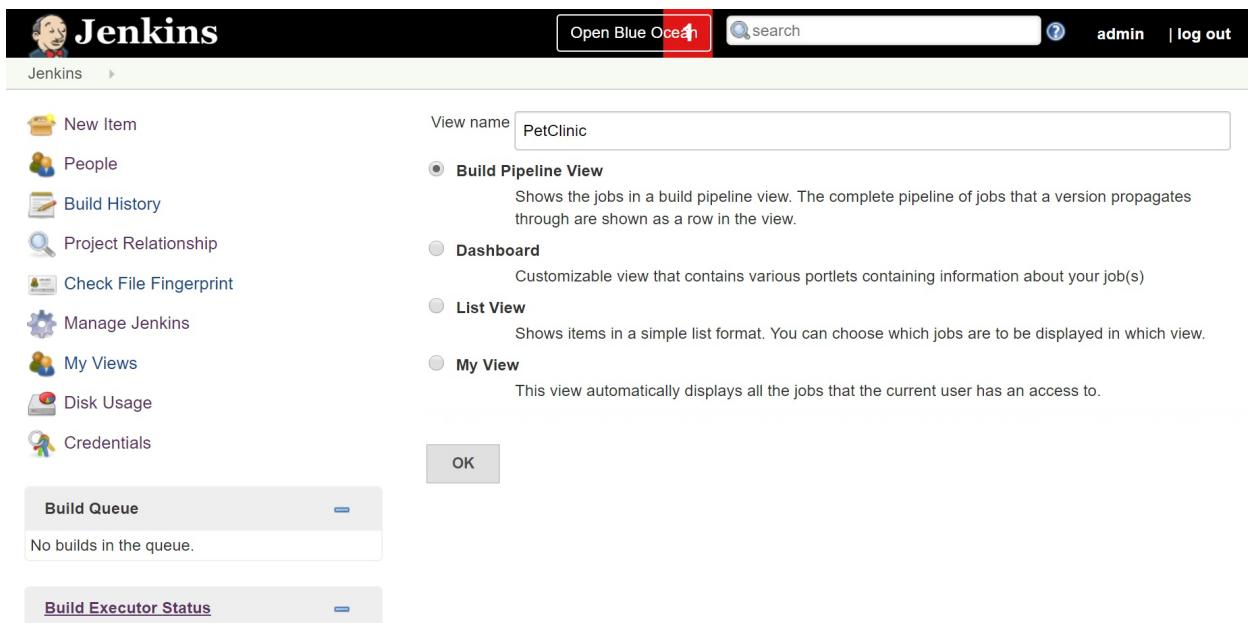
TestNG Results Plugin		Success
promoted builds plugin		Success
Run Condition Plugin		Success
Conditional BuildStep		Success
Parameterized Trigger plugin		Success
Build Pipeline Plugin		Success

→ [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

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**:

Name: PetClinic

Description:

Pipeline Flow:

Layout: Based on upstream/downstream relationship

Build Queue: No builds in the queue.

Build Executor Status: 1 Idle, 2 Idle

OK **Apply**

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**:

Trigger Options

Restrict triggers to most recent successful builds Yes No

Always allow manual trigger on pipeline steps Yes No

Display Options

No Of Displayed Builds: 3

Row Headers: Just the pipeline number

Column Headers: No header

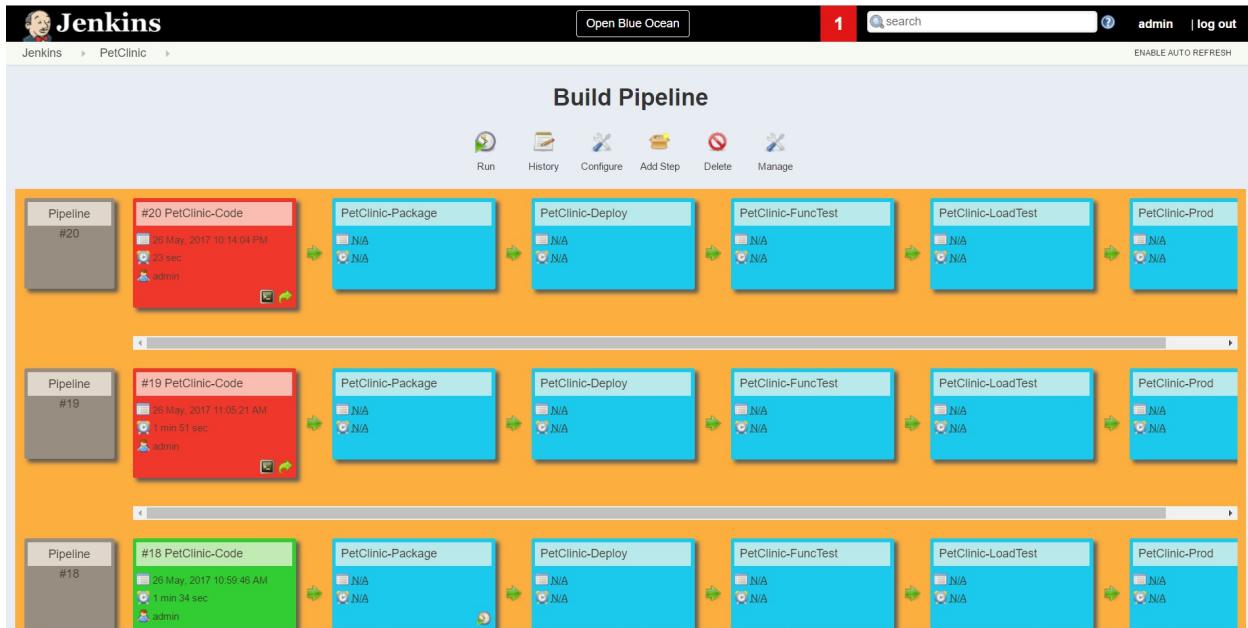
Refresh frequency (in seconds): 3

URL for custom CSS files:

Console Output Link Style: Lightbox

OK **Apply**

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

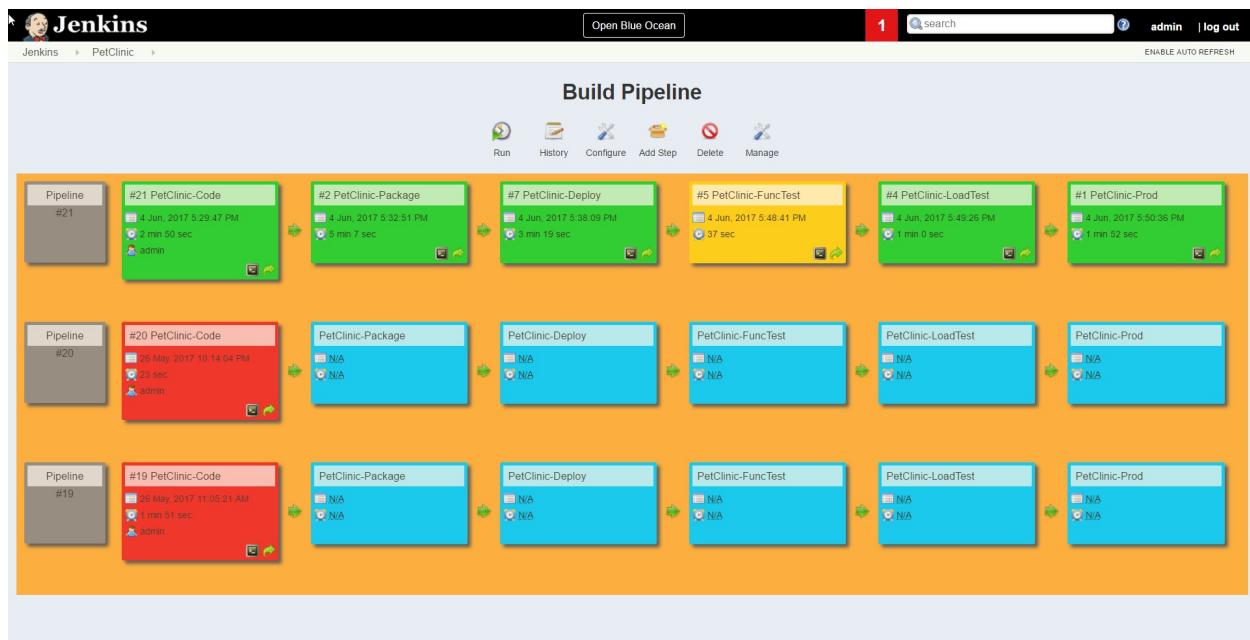


Build Pipeline with configured upstream and downstream job

- 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:

The screenshot shows the configuration page for the PetClinic-FuncTest job. The 'Post-build Actions' tab is selected. It contains two sections: 'Publish TestNG Results' and 'Build other projects'. In the 'Build other projects' section, the 'Projects to build' field contains 'PetClinic-LoadTest'. Below this, there are three radio button options: 'Trigger only if build is stable' (unchecked), 'Trigger even if the build is unstable' (checked), and 'Trigger even if the build fails' (unchecked). At the bottom, there is a 'Add post-build action' dropdown menu.

- 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:

Updates	Available	Installed	Advanced
Install ↓	Name	Version	
	<u>Blue Ocean</u>		
<input checked="" type="checkbox"/>	Blue Ocean is a new project that rethinks the user experience of Jenkins. Designed from the ground up for Jenkins Pipeline and compatible with Freestyle jobs, Blue Ocean reduces clutter and increases clarity for every member of your team.	1.0.1	
<input type="checkbox"/>	<u>Common API for Blue Ocean</u>	1.0.1	
<input type="checkbox"/>	<u>Config API for Blue Ocean</u>	1.0.1	
<input type="checkbox"/>	<u>Dashboard for Blue Ocean</u>	1.0.1	
<input type="checkbox"/>	<u>Events API for Blue Ocean</u>	1.0.1	
<input type="checkbox"/>	<u>Git Pipeline for Blue Ocean</u>	1.0.1	
<input type="checkbox"/>	<u>GitHub Pipeline for Blue Ocean</u>	1.0.1	
<input type="checkbox"/>	<u>i18n for Blue Ocean</u>	1.0.1	
<input type="checkbox"/>	<u>JWT for Blue Ocean</u>	1.0.1	
<input type="checkbox"/>	<u>Personalization for Blue Ocean</u>	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**:

Enter an item name

FirstPipeline

» Required field

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

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

General Build Triggers Advanced Project Options Pipeline

Pipeline name: FirstPipeline

Description:

[Plain text] [Preview](#)

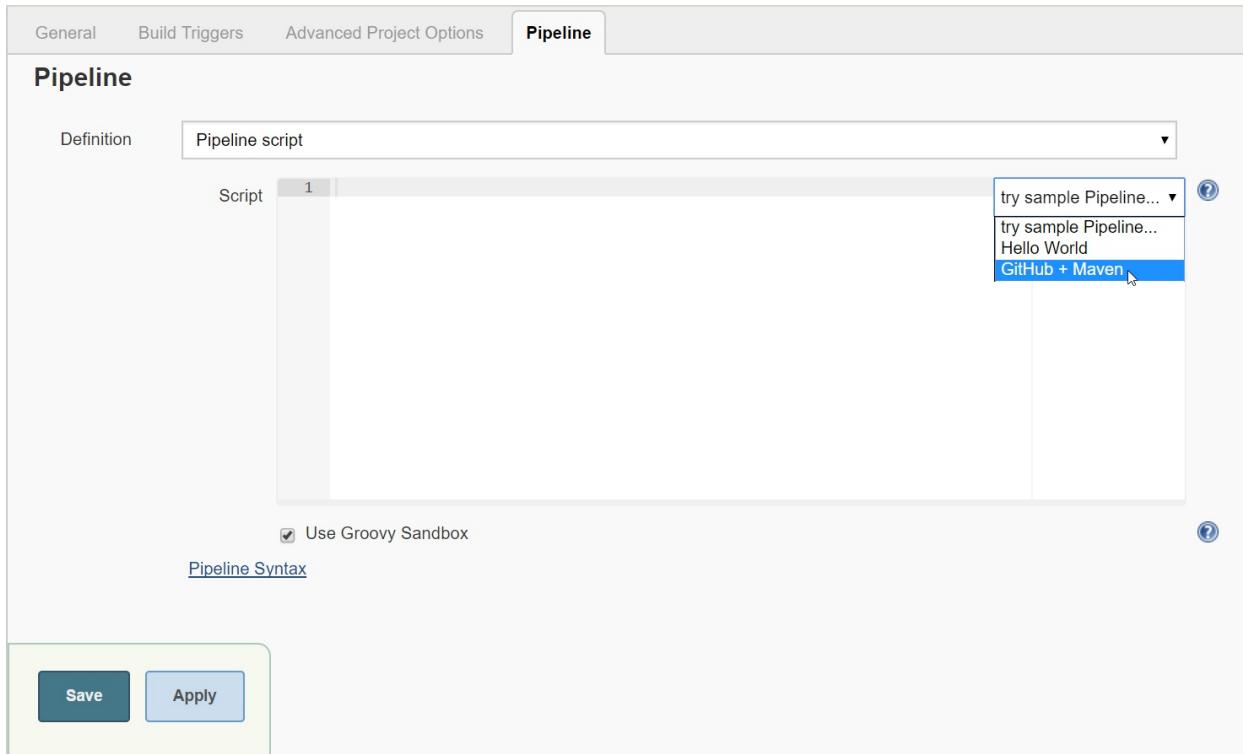
Enable project-based security
 Discard old builds
 Do not allow concurrent builds
 GitHub project
 This project is parameterized
 Throttle builds

Build Triggers

Save Apply

Build periodically

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



7. 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:

General Build Triggers Advanced Project Options **Pipeline**

Pipeline

Definition Pipeline script

```

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       }
}

```

Use Groovy Sandbox

[Pipeline Syntax](#)

Save **Apply**

8. Click on **Save** and **Execute** the build to verify it.
9. However, we will create our own pipeline with the same sequence we tried with Build Pipeline.
10. Click on **Pipeline syntax** to generate the syntax for specific tasks we want to execute.
11. 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('win'
    // 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 "${scannerHome}/bin/sonar-scanner -Dsonar.host.url=http://localhost:9000/ -Dsonar.login=1335c62cbfceab5954a5101ab7477cc974f58d56 -Dsonar.projectVersion=1.0 -Dsonar.projectKey=petclinicKey -Dsonar.sources=src"
        }
    } stage('Build') {
        // Run the maven build based on the Operating system
        if (isUnix()) {
            sh "'${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore clean package"
        } else {
            bat("${mvnHome}\bin\mvn" clean package)
        }
        // Publish JUnit Report
        junit '**/target/surefire-reports/TEST-*.xml'
    }
    // 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 to tomcat
    // Copy Artifact from this Pipeline Project into PetC 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
    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
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 function
bat("${mvnHome}\\bin\\mvn" "test") }
// This stage can be optional based on the requirements s
Test'){
// Execute command to perform load testing with the use o
JMeter; In our case we are using JMeter that is already in
Windows hence the bat file is used. Make sure to change th
location based
on the Apache JMeter installation directory available in y
system.
bat "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"
// 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:

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
Average stage times:	11s	1min 13s	1min 58s	1min 13s	2min 5s	7s
#20 Jun 04 11:58 No Changes	9s	1min 2s	2min 11s	2min 38s	2min 1s	12s
#19 Jun 04 11:41 No Changes	4s	52s	1min 39s	52s	2min 8s	1s failed
#18 Jun 04 11:35 No Changes	19s	1min 44s	2min 4s	8s failed		

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

	Success Preparation	SonarQube analysis	Build	Deploy	Functional Test	Load Test
Average stage times:	Success Preparation	1min 13s	1min 58s	1min 13s	2min 5s	7s
#20 Jun 04 11:58 No Changes	9s ↳	1min 2s	2min 11s	2min 38s	2min 1s	12s
#19 Jun 04 11:41 No Changes	4s	52s	1min 39s	52s	2min 8s	1s failed
#18 Jun 04 11:35 No Changes	19s	1min 44s	2min 4s	8s failed		

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

The screenshot shows the Jenkins Stage view for the 'FirstPipeline'. A modal window titled 'Stage Logs (Preparation)' is open, displaying two log entries:

- Git -- https://github.com/mitesh51/spring-petclinic.git -- (self time 9s)
- Use a tool from a predefined Tool Installation -- apache-maven-3.3.1 -- (self time 16ms)

Below the modal, the main Stage view table shows average stage times for various stages: Success Preparation (9s), SonarQube analysis (1min 13s), Build (1min 58s), Deploy (1min 13s), Functional Test (2min 5s), and Load Test (7s). The Preparation stage is highlighted with a green background.

	Success Preparation	SonarQube analysis	Build	Deploy	Functional Test	Load Test
Average stage times:	9s	1min 13s	1min 58s	1min 13s	2min 5s	7s
#20 Jun 04 11:58 No Changes	9s	1min 2s	2min 11s	2min 38s	2min 1s	12s
#19 Jun 04 11:41 No Changes	4s	52s	1min 39s	52s	2min 8s	1s failed
#18 Jun 04 11:35 No Changes	19s	1min 44s	2min 4s	8s failed		

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

The screenshot shows the Jenkins Stage view for the 'FirstPipeline'. The 'Preparation' stage log is expanded, revealing detailed command-line output:

```

> 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/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

```

At the bottom of the expanded log, there is a note: 'Use a tool from a predefined Tool Installation -- apache-maven-3.3.1 -- (self time 16ms)'

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:3.0: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] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:03 min
[INFO] Finished at: 2017-06-04T12:01:43+05:30
[INFO] Final Memory: 27M/88M
[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:

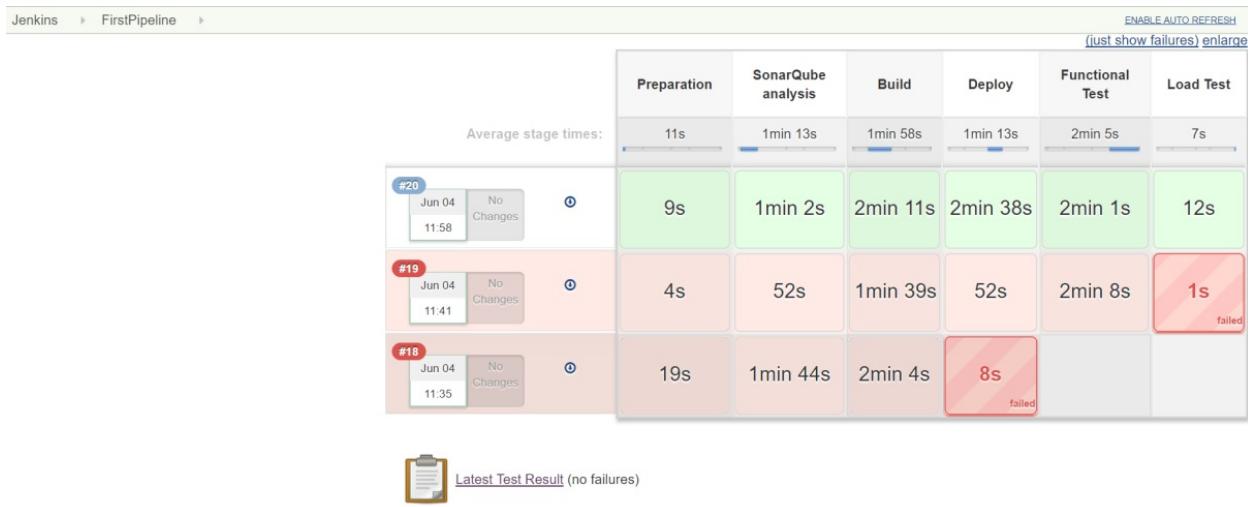
Results :

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 -
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
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 Blue Ocean interface. At the top, there's a navigation bar with links for Pipelines, Administration, Logout, Activity, Branches, and Pull Requests. Below the navigation is a header for 'FirstPipeline' with icons for cloud, star, and gear. The main area displays a table of pipeline runs. The table has columns for Status, Run, Commit, Message, Duration, Completed, and a refresh icon. Three runs are listed: Run 20 (Status: green checkmark, Commit: 7262ae6, Message: -, Duration: 8m 19s, Completed: 3 hours ago); Run 19 (Status: red X, Commit: 7262ae6, Message: -, Duration: 5m 44s, Completed: 4 hours ago); and Run 18 (Status: red X, Commit: 7262ae6, Message: -, Duration: 4m 18s, Completed: 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:

This screenshot shows the detailed view of Pipeline #20. The top bar includes a checkmark icon, the pipeline name, and various navigation icons. Below the bar, it shows 'Branch: -', 'Commit: 7262ae6', and the duration '8m 19s'. A timeline below shows stages: Preparation, SonarQube analysis, Build, Deploy, Functional Test, and Load Test, all marked with green checkmarks. Under the 'Load Test' stage, there's a 'Steps - Load Test' section. It lists a step 'Windows Batch Script' with a log output. The log shows the command running JMeter and its output, including the creation of a summariser and test results. The total duration for this step is 11s.

```
[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
```

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

✓ FirstPipeline #20

Pipeline Changes Tests Artifacts ⌂ ⚙ ⌂ ✎

Branch: — 8m 19s No changes
Commit: 7262ae6 3 hours ago

Preparation SonarQube analysis Build Deploy Functional Test Load Test

Steps - Deploy

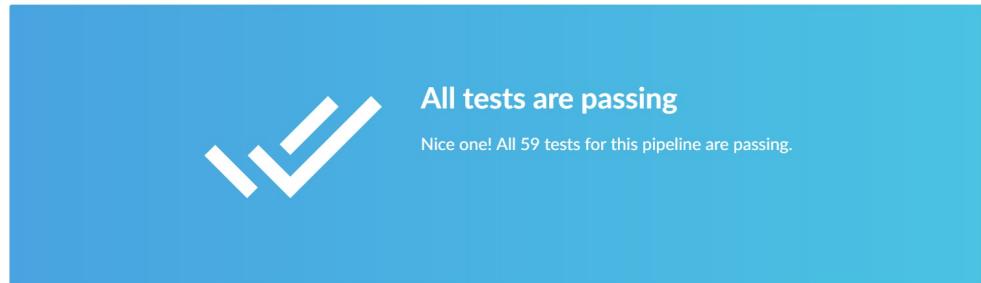
✓	General Build Step	<1s
1	Recording test results	
✓	Archive artifacts	<1s
✓	Building PetClinic-Deploy	2m 36s
1	Scheduling project: PetClinic-Deploy	
2	Starting building: PetClinic-Deploy #6	

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

✓ FirstPipeline #20

Pipeline Changes Tests Artifacts ⌂ ⚙ ⌂ ✎

Branch: — 8m 19s No changes
Commit: 7262ae6 4 hours ago



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

✓ [FirstPipeline #20](#)

Pipeline Changes Tests Artifacts ⚡ ⚙️ ⌂ ✕

Branch: — ⏱ 8m 19s No changes

Commit: 7262ae6 ⏱ 4 hours ago

Name	Size	Download
pipeline.log	-	⬇️
dashBoard_Test.xml	452 bytes	⬇️
target/petclinic.war	40 MB	⬇️

[Download All](#)

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**:

Install ↓	Name	Version
<input checked="" type="checkbox"/>	promoted builds plugin	2.28.1
<input type="checkbox"/>	Promoted Builds (Simple)	1.9

Install without restart Download now and install after restart Check now

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:

Category	Sub-Category	Value
Disk Usage	Job	152 KB
Disk Usage	All builds	152 KB
Disk Usage	Locked builds	-
Disk Usage	All workspaces	667 KB
Disk Usage	Slave workspaces	667 KB

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.

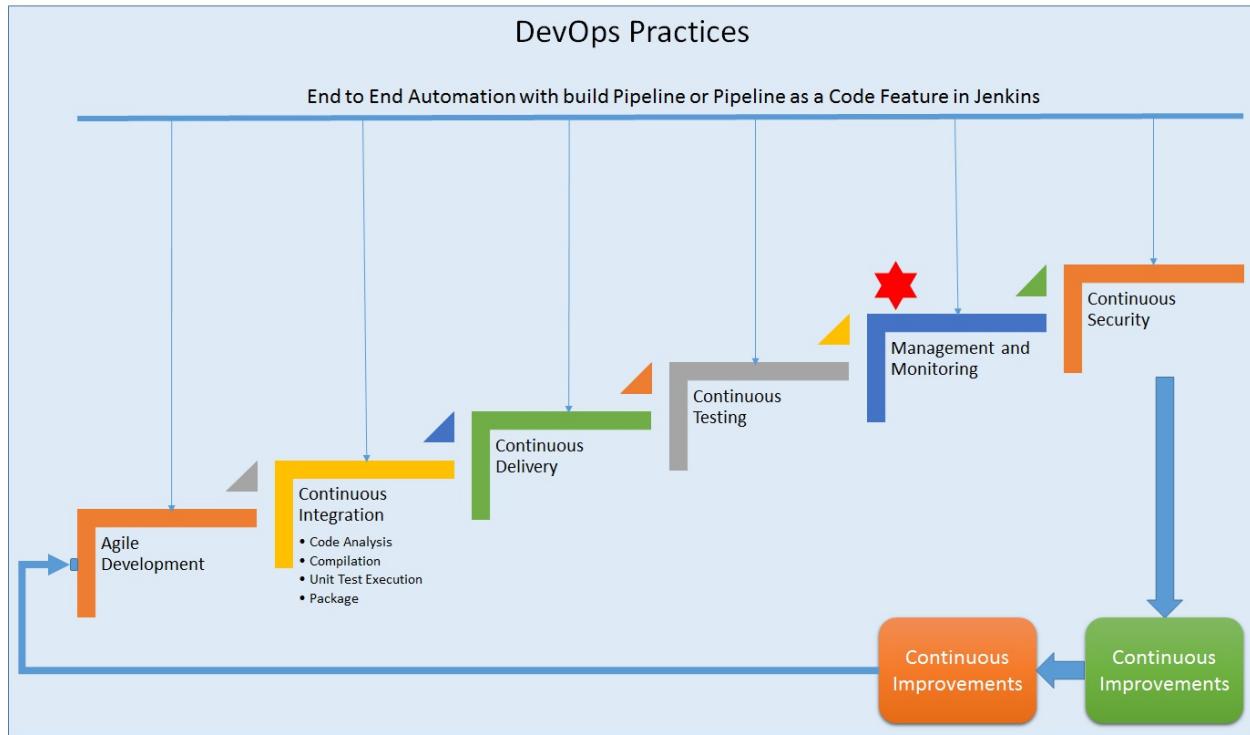
Chapter 8. Managing and Monitoring Jenkins

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.

Note

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

Nodes.

The screenshot shows the Jenkins 'Nodes' page. At the top, there's a navigation bar with the Jenkins logo, a search bar, and user information ('admin | log out'). Below the navigation is a table titled 'Nodes' with one row. The table columns are: S, Name (sorted by name), Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The single node listed is 'master' (Windows 10 (amd64)), which is 'In sync' with a clock difference of 0ms. It has 230.57 GB of free disk space, 1.98 GB of swap space, and 238.77 GB of temp space. The response time is 0ms. A 'Refresh status' button is at the bottom right of the table. On the left side of the main content area, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 Idle, 2 Idle).

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 'New Node' configuration dialog in Jenkins. The 'Node name' field is filled with 'WindowsNode'. The 'Permanent Agent' radio button is selected. A tooltip explains that this adds a plain, permanent agent to Jenkins. Below the configuration fields are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 Idle, 2 Idle). At the bottom right is an 'OK' button.

Provide details as required and click on Save:

The screenshot shows the Jenkins 'Nodes' configuration page. On the left, there's a sidebar with links like 'Back to Dashboard', 'Manage Jenkins', 'New Node', and 'Configure'. Below that are two sections: 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 Idle, 2 Idle). The main right panel is for configuring the 'WindowsNode'. It includes fields for Name (WindowsNode), Description, # of executors (1), Remote root directory (d:\tempJenkins), Labels (Windows, Test, Android), Usage (Use this node as much as possible), Launch method (Launch agent via execution of command on the master), Launch command (empty field with a note: 'No launch command specified'), Availability (Keep this agent online as much as possible), and Node Properties (Environment variables, Tool Locations). A 'Save' button is at the bottom.

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

Here we can specify the maximum number of concurrent builds that Jenkins may execute on this agent node.

of executors 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.

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 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

Labels	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.
Launch method	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. 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.

This setting controls when Jenkins starts and stops this agent.

Keep this agent online as much as possible.

Availability

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:

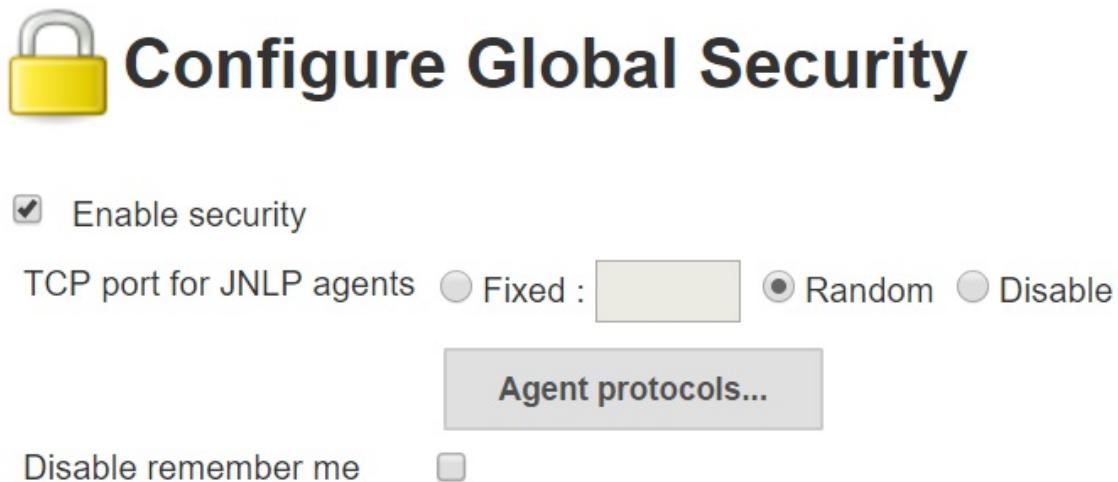
S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time	
	master	Windows 10 (amd64)	In sync	230.57 GB	1.98 GB	238.77 GB	0ms	
	WindowsNode		N/A	N/A	N/A	N/A	N/A	
	Data obtained	13 ms	2 ms	12 min	12 min	12 min	12 min	

[Refresh status](#)

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. At the top, there's a navigation bar with 'Jenkins', 'Nodes', and 'WindowsNode'. On the left, a sidebar lists actions: 'Back to List', 'Status', 'Delete Agent', 'Configure', 'Build History', 'Load Statistics', and 'Log'. The main content area has a title 'Agent WindowsNode' with an icon showing a computer monitor and a red X. A message says 'This agent is offline because Jenkins failed to launch the agent process on it. See log for more details.' Below this is a 'Launch agent' button. Under 'Labels', it shows 'Android', 'Test', and 'Windows'. A section titled 'Projects tied to WindowsNode' shows 'None'. At the bottom, there's a 'Build Executor Status' button.

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



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:

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:

General

Source Code Management Build Triggers Build Environment Build Post-build Actions

Days to keep builds: 3
if not empty, build records are only kept up to this number of days

Max # of builds to keep: 3
if not empty, only up to this number of build records are kept

GitHub project
 This project is parameterized
 Throttle builds
 Disable this project
 Execute concurrent builds if necessary
 Restrict where this project can be run

Label Expression: WindowsNode
Label WindowsNode is serviced by 1 node

Save Apply Advanced...

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

Jenkins Nodes WindowsNode

1 search admin | log out ENABLE AUTO REFRESH

Back to List Status Delete Agent Configure Build History Load Statistics Log

Agent WindowsNode

This agent is offline because Jenkins failed to launch the agent process on it. See log for more details

Connect agent to Jenkins one of these ways:

- Launch Launch agent from browser
- Run from agent command line:
java -jar slave.jar -jnlpUrl http://localhost:8080/computer/WindowsNode/slave-agent.jnlp -secret 3c0443e6a97a503ba24b1f6b2ca4727462d785a64990c7f61e6033fe29ef0498

Build Executor Status

Labels: Android, Test, Windows

Projects tied to WindowsNode

S	W	Name ↓	Last Success	Last Failure	Last Duration
		PetClinic-Code	1 day 3 hr - #18	16 hr - #20	1 min 34 sec

Icon: S M L Legend: RSS for all RSS for failures RSS for just latest builds

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:

Updates	Available	Installed	Advanced
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/Slaves Monitoring Plugin	1.4	

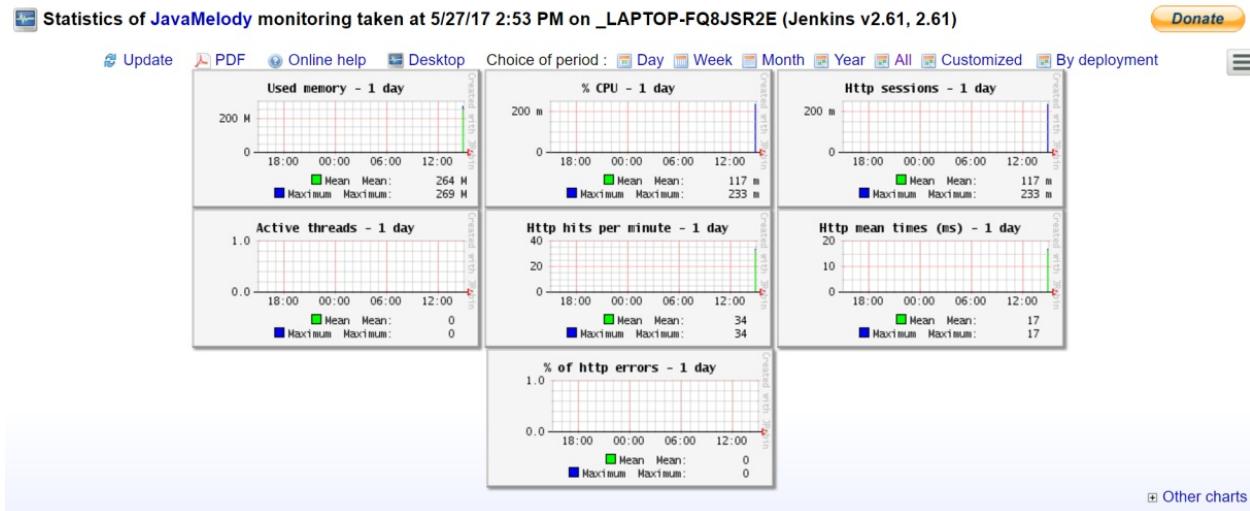
Filter:

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

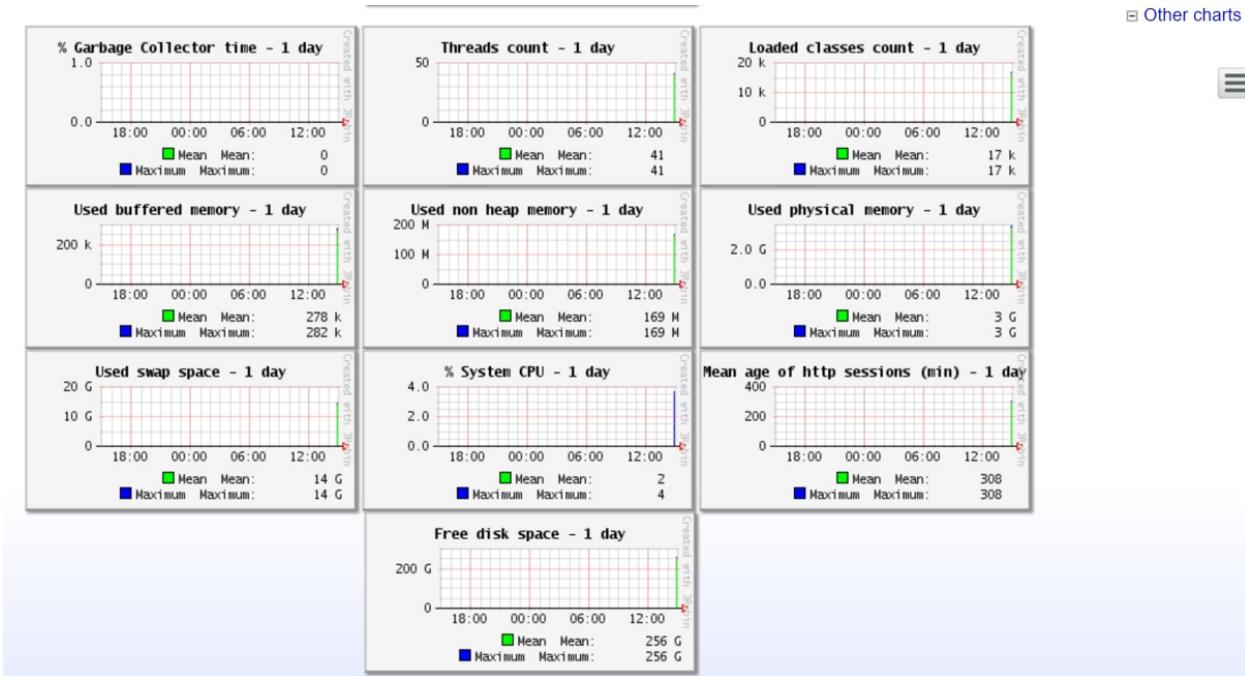
1. On the Jenkins dashboard, click on **Manage Jenkins**. Click on **Monitoring of Jenkins master**, as shown in the following screenshot:

-  [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.
-  [Monitoring of Jenkins master](#)
Monitoring of memory, cpu, http requests and more in Jenkins master.
You can also view the monitoring of [Jenkins nodes](#).
-  [Prepare for Shutdown](#)
Stops executing new builds, so that the system can be eventually shut down safely.

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



3. 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	17	128	34	100	5	0.00	20
http warning	50	5	60	114	52	24	9	0.00	54
http severe	0	0	-1	0	-1	0	-1	0.00	0

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

Statistics http system errors - 1 day

None

Statistics system errors logs - 1 day

None

Current requests

None

System information

Host: LAPTOP-FQ8JSR2E@192.168.99.1
Java memory used: 290 Mb / 889 Mb
Nb of http sessions: 1
Nb of active threads (current http requests): 0
% System CPU 16.97

[Execute the garbage collector](#) [Generate a heap dump](#) [View deployment descriptor](#) [View memory histogram](#) [Invalidate http sessions](#) [View http sessions](#)
[MBeans](#) [View OS processes](#) [JNDI tree](#)

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 [Details](#)

Thread	Daemon ?	Priority	State	Executed method	Cpu time (ms)	User time (ms)	Kill
Attach Listener	yes	5	● RUNNABLE		0	0	
Computer.threadPoolForRemoting [#1]	yes	5	● TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	187	109	
DestroyJavaVM	no	5	● RUNNABLE		1,875	1,328	
FilePath.localPool [#1]	yes	5	● TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	78	46	
Finalizer	yes	8	● WAITING	java.lang.Object.wait(Native Method)	31	31	
Handling GET /monitoring from 0:0:0:0:0:1 : RequestHandlerThread[#2]	yes	5	● RUNNABLE	java.lang.Thread.dumpThreads(Native Method)	125	109	
Java2D Disposer	yes	10	● WAITING	java.lang.Object.wait(Native Method)	0	0	
javamelody	yes	5	● TIMED_WAITING	java.lang.Object.wait(Native Method)	187	62	
Jenkins cron thread	no	5	● WAITING	java.lang.Object.wait(Native Method)	0	0	
Jenkins UDP 33848 monitoring thread	no	5	● RUNNABLE	java.net.TwoStacksPlainDatagramSocketImpl.receive0(Native Method)	0	0	
jenkins.util.Timer [#10]	yes	5	● WAITING	sun.misc.Unsafe.park(Native Method)	15	15	
jenkins.util.Timer [#1]	yes	5	● WAITING	sun.misc.Unsafe.park(Native Method)	15	15	
jenkins.util.Timer [#2]	yes	5	● TIMED_WAITING	sun.misc.Unsafe.park(Native Method)	15	0	
jenkins.util.Timer [#3]	yes	5	● WAITING	sun.misc.Unsafe.park(Native Method)	31	15	
jenkins.util.Timer [#4]	yes	5	● WAITING	sun.misc.Unsafe.park(Native Method)	0	0	

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)



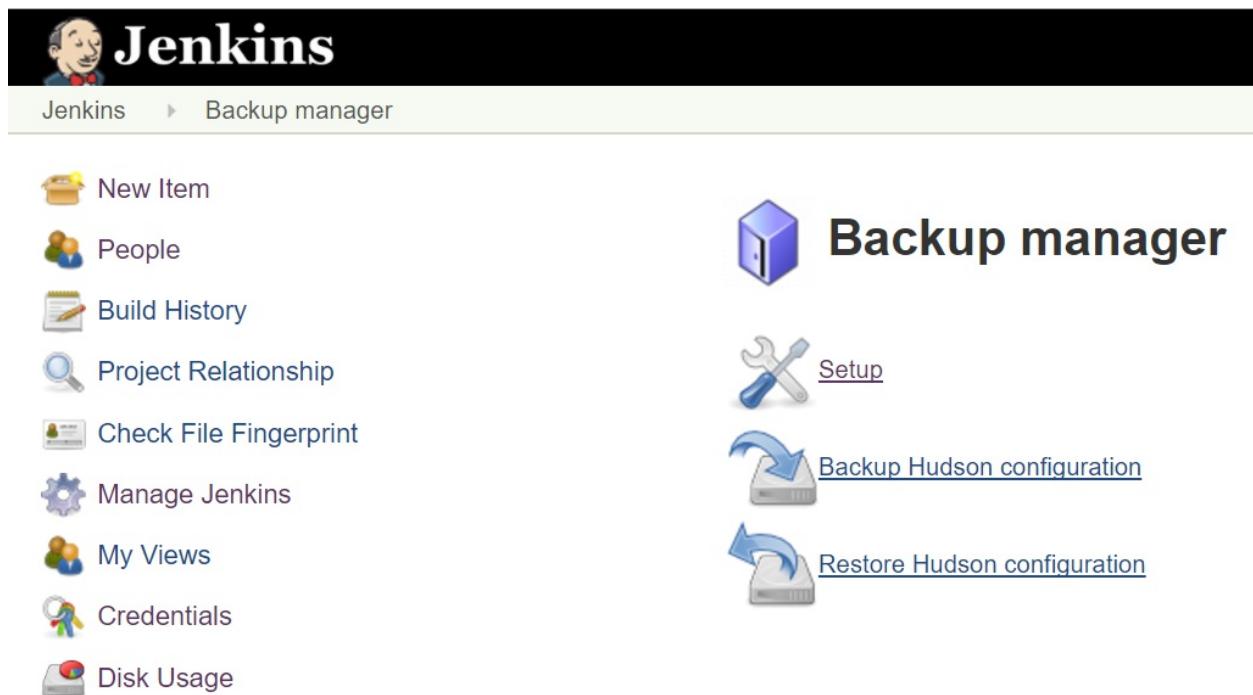
Restart Jenkins when installation is complete and no jobs are running

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**:**

	Disk Usage Displays per-project disk usage
	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.
	Backup manager Backup or Restore Jenkins configuration files
	Monitoring of Jenkins master Monitoring of memory, cpu, http requests and more in Jenkins master. You can also view the monitoring of Jenkins nodes .
	Prepare for Shutdown Stops executing new builds, so that the system can be eventually shut down safely.

2. Click on **Setup:**



Jenkins

Jenkins > Backup manager

New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

Credentials

Disk Usage

Backup manager

Setup

Backup Hudson configuration

Restore Hudson configuration

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

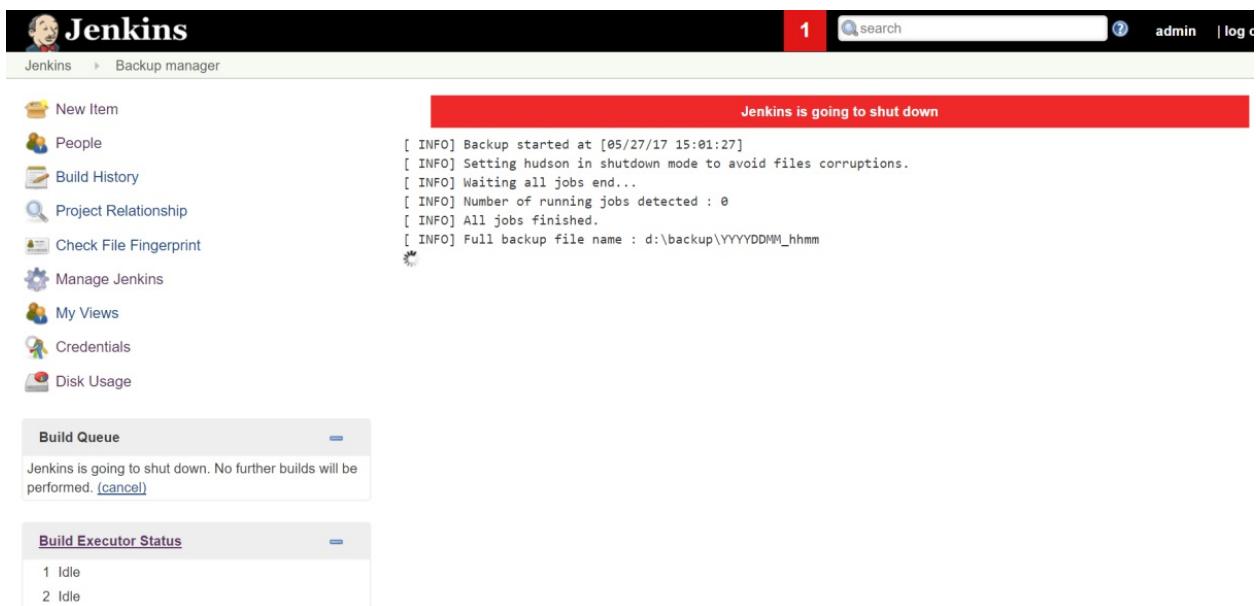


Case-sensitive

Backup builds history

Backup maven artifacts archives

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



The screenshot shows the Jenkins interface with the 'Backup manager' page selected. A red banner at the top right reads 'Jenkins is going to shut down'. The central log output shows the backup process starting and completing successfully:

```
[ INFO] Backup started at [05/27/17 15:01:27]
[ INFO] Setting hudson in shutdown mode to avoid files corruptions.
[ INFO] Waiting all jobs end...
[ INFO] Number of running jobs detected : 0
[ INFO] All jobs finished.
[ INFO] Full backup file name : d:\backup\YYYYDDMM_hhmm
```

The 'Build Queue' section shows a message: 'Jenkins is going to shut down. No further builds will be performed. ([cancel](#))'. The 'Build Executor Status' section shows 1 Idle and 2 Idle executors.

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

Jenkins is going to shut down

```
[ INFO] Backup started at [05/27/17 15:01:27]
[ INFO] Setting hudson in shutdown mode to avoid files corruptions.
[ INFO] Waiting all jobs end...
[ INFO] Number of running jobs detected : 0
[ INFO] All jobs finished.
[ INFO] Full backup file name : d:\backup\YYYYDDMM_hhmm
[ INFO] Saved files : 12482
[ INFO] Number of errors : 0
[ INFO] Cancel hudson shutdown mode
[ INFO] Backup end at [05/27/17 15:03:51]
[ INFO] [143.935s]
```

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

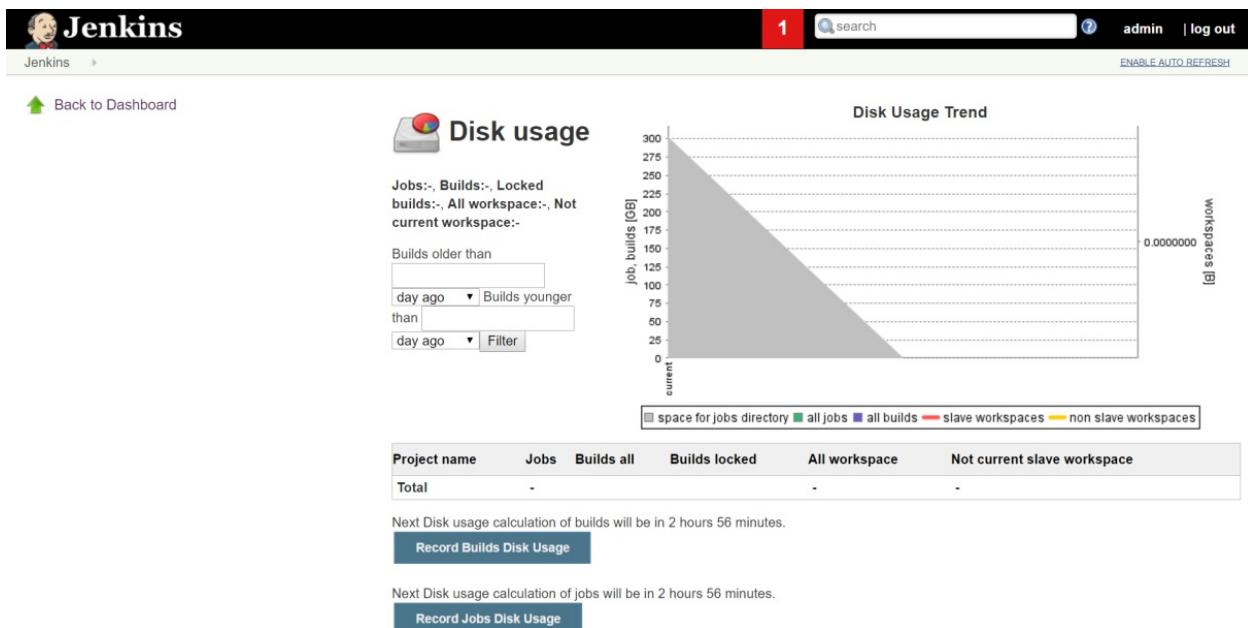
The screenshot shows the Jenkins Backup Manager page. At the top, there is a red banner with the text "Jenkins is going to shut down". Below the banner, a log message details a full backup process. On the left, a sidebar lists various Jenkins management options like "New Item", "People", and "Disk Usage". The main content area is titled "Backup manager" and displays a message "Available backup in d:\backup:". Below this, a blue button labeled "Launch restore" is visible. At the bottom, two sections show "Build Queue" (empty) 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:**

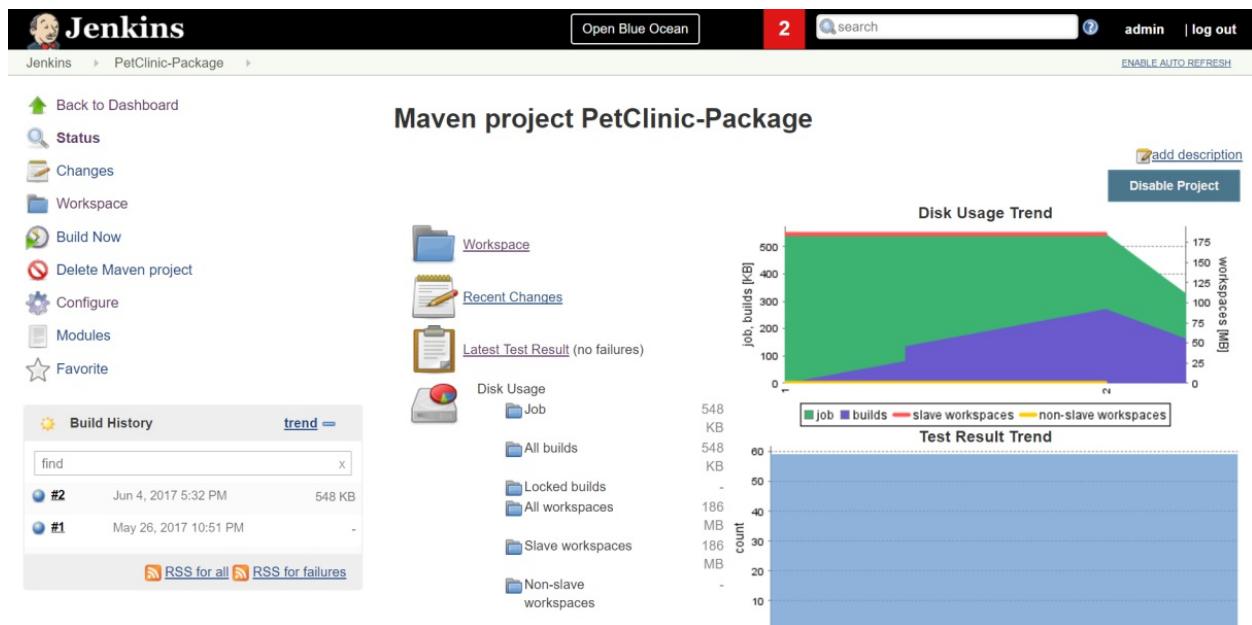


2. Go to **Manage Jenkins** and click on **Configure System**. Go to the **disk usage** section and click on **Show disk usage trend graph on the project page**:

Disk usage

<input checked="" type="checkbox"/> Enable calculation of builds	Time interval for calculation	0 */6 * * *
<input checked="" type="checkbox"/> Enable calculation of jobs	Time interval for calculation	0 */6 * * *
<input checked="" type="checkbox"/> Enable calculation of workspace	Time interval for calculation	0 */6 * * *
<input type="checkbox"/> Warn if some size is exceeded		
Show disk usage trend graph on the project page	<input checked="" type="checkbox"/>	
Show free space of jobs directory in global graph	<input checked="" type="checkbox"/>	
Length of global disk usage history	183	
Time out for calculation of slave workspace in minutes	5	
Control workspace from slave side too	<input type="checkbox"/>	
Jobs excluded for disk usage calculation	<input type="text"/>	

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



The screenshot shows the Jenkins interface for the 'PetClinic-Package' project. The left sidebar contains links like Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Maven project, Configure, Modules, and Favorite. The main area displays the 'Maven project PetClinic-Package' page. On the right, there is a 'Disk Usage Trend' chart showing disk usage over time. The chart has two stacked areas: a green area representing 'job' and a blue area representing 'builds'. A red line represents 'slave workspaces', which is very high, around 500-600 KB. A yellow line represents 'non-slave workspaces', which is much lower, around 25-50 KB. Below the chart is a 'Test Result Trend' chart showing the count of failures over time, with values ranging from 0 to 60.

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			
<input checked="" type="checkbox"/> 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`:

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

The screenshot shows the Jenkins 'New View' configuration dialog. On the left, there's 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, a radio button group shows 'Build Monitor View' is selected. A detailed description of this view follows. 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' page for 'PetClinic-M'. The 'Name' field is filled with 'PetClinic-M'. Under 'Job Filters', the 'Status Filter' is set to 'All selected jobs'. In the 'Jobs' section, several Jenkins jobs are listed with checkboxes: FirstAntExample, FirstJob, FirstPipeline, Main-PetClinic, Maven-Sample, mitesh51, PetClinic-Code (which is checked), PetClinic-Deploy (which is checked), and PetClinic-FuncTest (which is checked). Other jobs like PetClinic-UnitTest and PetClinic-IT are also listed but not checked.

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

The screenshot shows a build monitor interface for the PetClinic project. The title bar reads "PetClinic-M". There are six cards arranged in a grid:

- PetClinic-Code**: Status is green, build #21, 3 days ago, with a note "Back in the green!"
- PetClinic-Deploy**: Status is green, build #7, 3 days ago.
- PetClinic-FuncTest**: Status is orange, build #5, 3 days ago, with a note "4 builds have failed".
- PetClinic-LoadTest**: Status is green, build #4, 3 days ago.
- PetClinic-Package**: Status is green, build #2, 3 days ago.
- PetClinic-Prod**: Status is green, build #1, 3 days ago.

At the bottom right, there is a small note: "Build Monitor version 1.11+build.201701152243 brought to you by Jan Molak".

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:

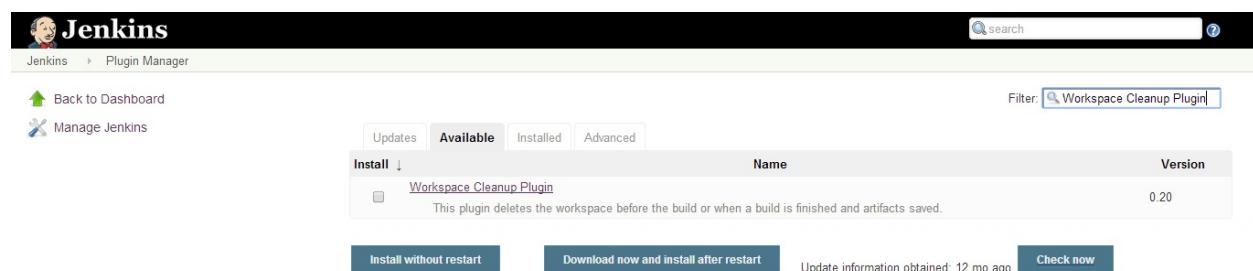
The screenshot shows the Jenkins configuration interface for the Audit Trail plugin. The top navigation bar has 'Audit Trail' selected. Below it, a sub-menu 'Loggers' is open. The main content area is titled 'Log file' and contains three configuration fields: 'Log Location' (set to 'd:\jenkinsLogs'), 'Log File Size MB' (set to '5'), and 'Log File Count' (set to '5'). Each field has a blue question mark icon to its right. At the bottom right of the configuration area is a red 'Delete' button. Below the configuration area is a grey 'Add Logger' button with a dropdown arrow. To the right of the 'Add Logger' button is a grey 'Advanced...' button.

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. At the top, there's a search bar and tabs for 'Updates', 'Available', 'Installed', and 'Advanced'. The 'Available' tab is selected. A filter bar below the tabs contains the text 'Workspace Cleanup Plugin'. Below the filter, a table lists the plugin details: Name 'Workspace Cleanup Plugin', Version '0.20', and a description 'This plugin deletes the workspace before the build or when a build is finished and artifacts saved.' At the bottom of the table are three buttons: 'Install without restart', 'Download now and install after restart', and 'Check now'. A note at the bottom right says 'Update information obtained: 12 mo ago'.

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 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 'Add post-build action ▾'.

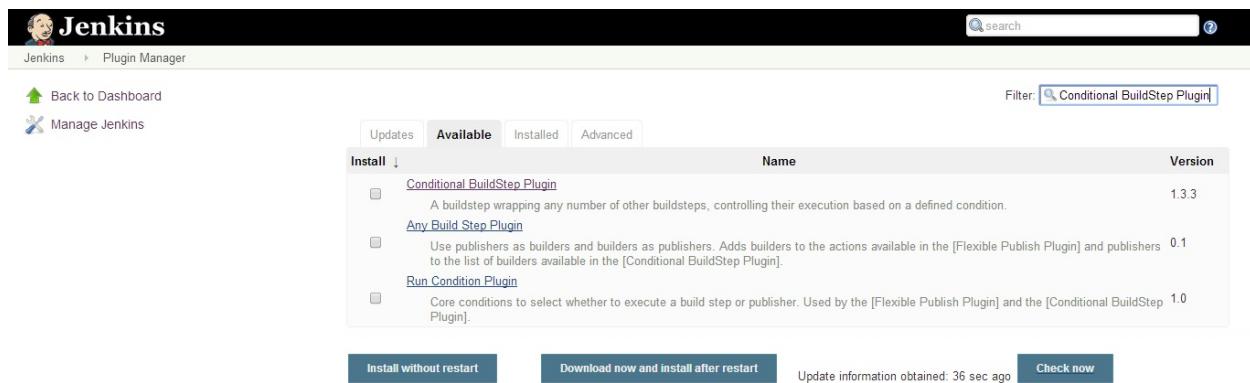
Note

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:



The screenshot shows the Jenkins Plugin Manager interface. At the top, there's a navigation bar with the Jenkins logo, a search bar, and a 'Plugin Manager' link. Below the navigation bar, there are links to 'Back to Dashboard' and 'Manage Jenkins'. The main area has tabs for 'Updates', 'Available' (which is selected), 'Installed', and 'Advanced'. A search bar at the top of the main area is set to 'Conditional BuildStep Plugin'. The results table has columns for 'Name', 'Version', and 'Description'. There are three entries:

Name	Version
Conditional BuildStep Plugin	1.3.3
Any Build Step Plugin	0.1
Run Condition Plugin	1.0

Below the table are three buttons: 'Install without restart', 'Download now and install after restart', and 'Check now'. A status message 'Update information obtained: 36 sec ago' is also present.

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**:

Conditional step (single)

Run? File exists

File /tmp/ec2.txt

Base directory Workspace

Advanced...

Builder EC2 Environment

AWS Credentials

AWS Credentials Id [redacted]

Please select the Jenkins Credentials Id for your AWS environment

aws.region [redacted]

Delete

Add build step ▾

Save Apply

Select **Conditional steps (multiple)** from **Add build step**. We can add multiple steps to a condition in this conditional step:

Conditional steps (multiple)

Run? Always

Advanced...

Steps to run if condition is met

Inject environment variables

Properties File Path

Properties Content

Delete

Execute shell

Command

See [the list of available environment variables](#)

Delete

Add step to condition ▾

Save Apply

The screenshot shows the Jenkins Conditional Build Step configuration interface. At the top, it says 'Conditional steps (multiple)' and 'Run? Always'. Below that is an 'Advanced...' button. The main area is titled 'Steps to run if condition is met'. It contains two entries: 'Inject environment variables' and 'Execute shell'. The 'Inject environment variables' entry has fields for 'Properties File Path' and 'Properties Content', both of which have help icons. It also has a 'Delete' button. The 'Execute shell' entry has a 'Command' field, a link to 'See the list of available environment variables', and a 'Delete' button. At the bottom, there's a 'Save' button, an 'Apply' button, and a 'Add step to condition' dropdown menu.

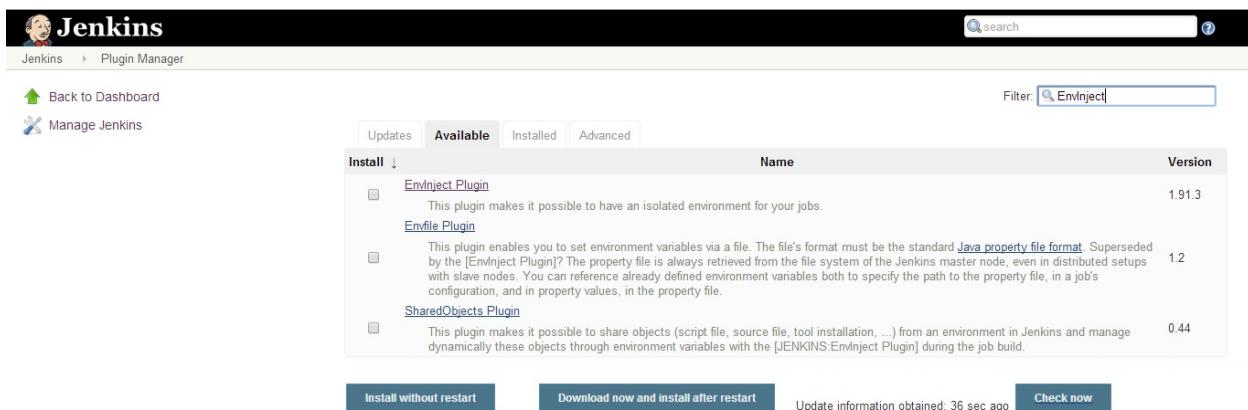
Note

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 search bar at the top right contains the text "EnvInject". Below the search bar, there are tabs for "Updates", "Available" (which is selected), "Installed", and "Advanced". A filter bar below the tabs also contains the text "EnvInject". The main table lists three available plugins:

Name	Version
EnvInject Plugin	1.91.3
Envfile Plugin	1.2
SharedObjects Plugin	0.44

Each row in the table has a brief description and a checkbox next to it. At the bottom of the page are buttons for "Install without restart", "Download now and install after restart", and "Check now".

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:

Inject environment variables to the build process ?

Properties File Path ?

Properties Content ?

Script File Path ?

Script Content ?

Evaluated Groovy script ?

Inject passwords to the build as environment variables ?

Global passwords ?

Job passwords

Save Apply

Note

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.

Chapter 9. Security in Jenkins

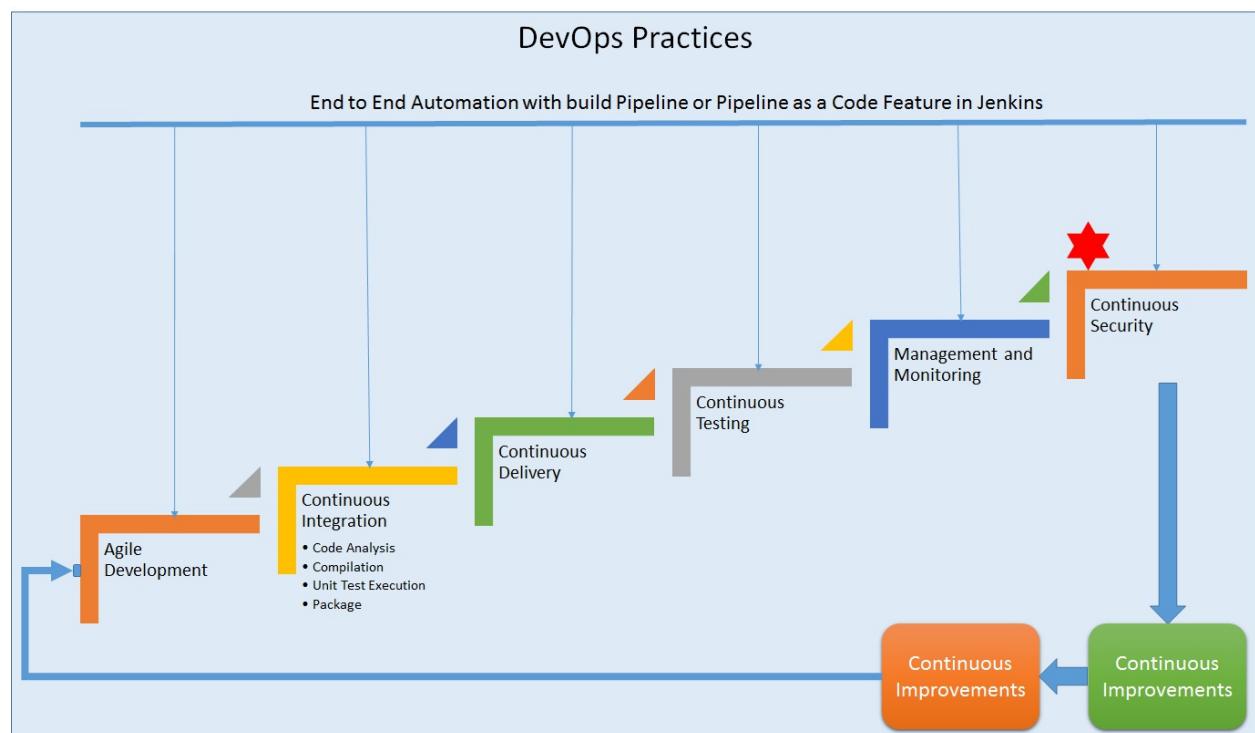
Up to now we have seen static code analysis, Continuous Integration, Continuous Delivery/Deployment, Continuous Testing, the orchestration of build jobs using the build pipeline plugin and pipeline as a code, and the management and monitoring of Jenkins resources.

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**:

-  [System Log](#)
System log captures output from `java.util.logging` output 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:

 Jenkins

1 admin | log out

Jenkins ▶ Jenkins' own user database [ENABLE AUTO REFRESH](#)

 Back to Dashboard

 Manage Jenkins

 Create User

Users

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.

User Id	Name	
 admin	admin	

3. Click on the **Create User** link and provide details:

 Jenkins

Jenkins ▶ Jenkins' own user database

 Back to Dashboard

 Manage Jenkins

 Create User

Create User

Username:	<input type="text" value="Shreyansh"/>
Password:	<input type="password" value="....."/>
Confirm password:	<input type="password" value="....."/>
Full name:	<input type="text" value="Shreyansh Soni"/>
E-mail address:	<input type="text" value="redacted@gmail.com"/>

Create User

4. Check the list of users in **Manage Jenkins** | **Manage Users**:

The screenshot shows a browser window with the address bar containing 'localhost:8080/securityRealm/'. The main content is the Jenkins 'Users' page. At the top right, there is a red button with the number '1' and the text 'admin | log out'. Below this, there is a link 'ENABLE AUTO REFRESH'. The page title is 'Jenkins' with a logo. Underneath the title, it says 'Jenkins > Jenkins' own user database'. On the left, there are three navigation links: 'Back to Dashboard', 'Manage Jenkins', and 'Create User'. The main section is titled 'Users' and contains a table with two rows. The table has columns for 'User Id' and 'Name'. The first row shows 'admin' as both the User Id and Name, with a gear icon to its right. The second row shows 'Shreyansh' as the User Id and 'Shreyansh Soni' as the Name, with a gear icon and a red circle with a slash icon to its right.

User Id	Name
admin	admin
Shreyansh	Shreyansh Soni

Users

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.

User Id	Name	
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**:

The screenshot shows the Jenkins 'Configure Global Security' configuration page. At the top left is a yellow padlock icon. The title 'Configure Global Security' is centered above a form area. The form includes sections for 'Enable security', 'TCP port for JNLP agents', 'Disable remember me', 'Access Control', 'Security Realm', and 'Authorization'. Buttons for 'Save' and 'Apply' are at the bottom.

Enable security

TCP port for JNLP agents Fixed : Random Disable

Disable remember me

Access Control

Security Realm

- Delegate to servlet container
- Jenkins' own user database
- Allow users to sign up
- LDAP

Authorization

- Anyone can do anything
- Legacy mode
- Logged-in users can do anything
- Allow anonymous read access

Buttons:

-
-

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 'Security Realm' section, 'Jenkins' own user database' is selected. Under 'Authorization', 'Matrix-based security' is selected. A table below shows permissions for 'User/group' (Anonymous, admin) across various Jenkins operations like Administer, Read, Create, Delete, etc. The 'admin' row has checkmarks in almost all columns, indicating full access. Below the table, there's a 'User/group to add:' input field containing 'admin' and an 'Add' button. At the bottom, there are 'Save' and 'Apply' buttons.

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**:

Jenkins > Configure Global Security

Access Control

Security Realm

- Delegate to servlet container
- Jenkins' own user database
 - Allow users to sign up
- LDAP

Authorization

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

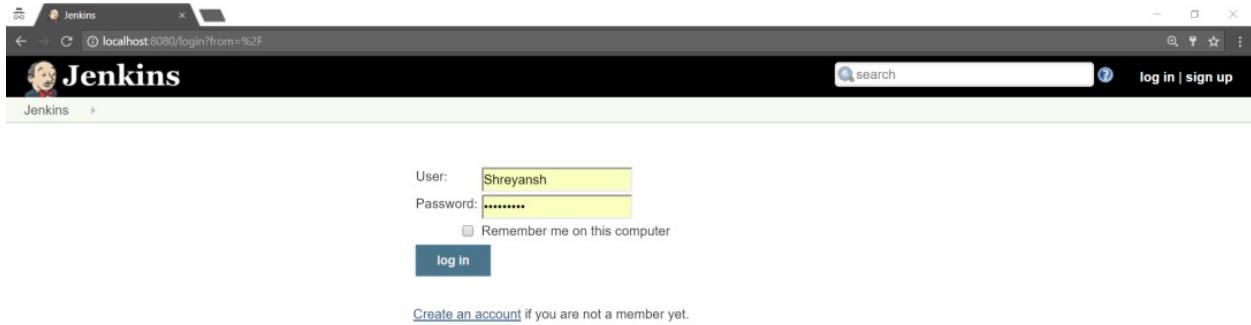
User/group	Overall	Credentials	Agent	Job																	
	Administer	Read	Create	Delete	Manage Domains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create	Delete	Discover	Move
Anonymous	<input type="checkbox"/>																				
admin	<input checked="" type="checkbox"/>																				
Shreyansh	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

User/group to add: Shreyansh

Project-based Matrix Authorization Strategy

Markup Formatter

- 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:



- Verify that limited access is available to the new user, and that the **New Item** and **Manage Jenkins** links are not available:

The screenshot shows the Jenkins dashboard at localhost:8080. The top navigation bar includes links for People, Build History, Project Relationship, Check File Fingerprint, and My Views. Below the navigation is a search bar and user information for Shreyansh Soni. The main content area features a "Welcome to Jenkins!" message and several status panels: "Build Queue" (No builds in the queue), "Build Executor Status" (1 Idle, 2 Idle), and other less visible panels.

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 "Configure Global Security" page under the "Manage Jenkins" section. It displays a matrix where users (Shreyansh, admin, Anonymous) are assigned permissions across various Jenkins roles (Overall, Credentials, Agent, Job). The "Job" row specifically shows checkboxes for "Create", "Delete", and "Read" under the "Job" column. Below the matrix, there are sections for "Plain text" (with a note about HTML escaping), "Crumb Algorithm" (set to "Default Crumb Issuer"), and "Advanced Options" (checkboxes for "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 with the following details:

- Left sidebar:**
 - People
 - Build History
 - Project Relationship
 - Check File Fingerprint
 - My Views
 - Build Queue:** No builds in the queue.
 - Build Executor Status:** 1 Idle, 2 Idle
- Central area:**
 - All** tab selected, **My Dashboard** tab available.
 - Jobs Table:**

S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☁️	FirstAntExample	14 hr - #2	14 hr - #1	14 sec
●	☀️	FirstJob	3 hr 41 min - #9	N/A	18 sec
●	☀️	Main-PetClinic	6 days 0 hr - #1	N/A	30 min
●	☀️	Maven-Sample	6 days 1 hr - #1	N/A	59 sec
●	🌧️	PetClinic-Code	1 day 2 hr - #18	14 hr - #20	1 min 34 sec
●	☀️	PetClinic-Package	14 hr - #1	N/A	4 min 51 sec
●	🌩️	SpringBoot	N/A	6 days 1 hr - #2	4.2 sec
 - Legend:** S M L
 - RSS Feeds:** RSS for all, RSS for failures, RSS for just latest builds

9. We can see the jobs, but we can't execute them as rights are not available:

The screenshot shows the Jenkins project page for **PetClinic-Code**:

- Left sidebar:**
 - [Back to Dashboard](#)
 - [Status](#)
 - [Changes](#)
 - [SonarQube](#)
- Central area:**
 - Project PetClinic-Code**
 - SonarQube** icon and link
 - Recent Changes** icon and link
 - Build History** section:
 - Search bar: find
 - Builds listed:
 - #20 May 26, 2017 10:14 PM
 - #19 May 26, 2017 11:05 AM
 - #18 May 26, 2017 10:59 AM
 - [RSS for all](#) and [RSS for failures](#)
 - Permalinks:**
 - [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](#)

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**:

Jenkins > Configure Global Security

Authorization

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

User/group	Overall	Credentials												Agent												Job											
		Administer	Read	Create	Delete	Manage Domains	Update View	Build	Configure	Connect	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create	Delete	Discover	Move	Renew															
Anonymous		<input type="checkbox"/>																																			
admin		<input checked="" type="checkbox"/>																																			

User/group to add: admin

Markup Formatter

Plain text

Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

Prevent Cross Site Request Forgery exploits

Crumbs

Crumb Algorithm

Default Crumb Issuer
 Enable proxy compatibility

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**:



Jenkins

Shreyansh Soni | log out

Jenkins >

Access Denied

 Shreyansh is missing the Overall/Read permission

5. We need to provide overall read rights so **Shreyansh** can access the Jenkins dashboard:

Jenkins > Configure Global Security

Authorization

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

User/group	Overall	Credentials	Agent	Job																
	Administer	Read	Create	Delete	Manage Domains	Update View	Build	Configure	Connect	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create	Delete	Discover	Monitor
admin	<input checked="" type="checkbox"/>																			
Anonymous	<input type="checkbox"/>																			
Shreyansh	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

User/group to add: Shreyansh

Markup Formatter

Prevent Cross Site Request Forgery exploits

Crumbs

Default Crumb Issuer

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**:

- 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**:

- Click on **Build** and verify all the rights are available to the user **Shreyansh**:

 Jenkins

Jenkins > PetClinic-Code >

search Shreyansh Soni | log out
ENABLE AUTO REFRESH

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

[SonarQube](#)

Project PetClinic-Code

[SonarQube](#)

[Workspace](#)

[Recent Changes](#)

[add description](#)

[Disable Project](#)

Build History trend —

find
X

#20 May 26, 2017 10:14 PM

#19 May 26, 2017 11:05 AM

#18 May 26, 2017 10:59 AM

RSS for all RSS for failures

Permalinks

- Last build (#20), 16 hr ago
- Last stable build (#18), 1 day 3 hr ago
- Last successful build (#18), 1 day 3 hr ago
- Last failed build (#20), 16 hr ago
- Last unsuccessful build (#20), 16 hr ago
- Last completed build (#20), 16 hr ago

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*.