

# **Quality Thought**

# **DEVOPS**

## **(JENKINS, CHEF, AWS)**

**KONARK DIGITAL XEROX**

**Behind Mytrivanm, opp Aditya Enclave,  
Ameerpet, Hyd.**

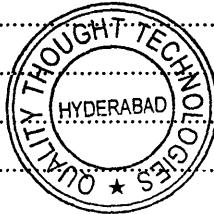
**(All Software Institute Materials are available)  
CELL: 9573162868, 9030502569**



# Continuous Integration with Jenkins

## Contents

|  |    |
|--|----|
| 1. Concepts of Continuous Integration .....  | 6  |
| The agile software development process ..... | 6  |
| Software development life cycle .....        | 6  |
| Continuous Integration .....                 | 7  |
| Note .....                                   | 8  |
| Agile runs on Continuous Integration .....   | 9  |
| How to achieve Continuous Integration .....  | 10 |
| Development operations .....                 | 10 |
| Use a version control system .....           | 11 |
| Use repository tools.....                    | 16 |
| Use a Continuous Integration tool .....      | 17 |
| Creating a self-triggered build .....        | 18 |
| Automate the packaging.....                  | 19 |
| Using build tools .....                      | 20 |
| Automating the deployments .....             | 20 |
| Note .....                                   | 21 |
| Automating the testing .....                 | 21 |
| Use static code analysis.....                | 23 |
| Continuous Integration benefits .....        | 23 |
| Freedom from long integrations.....          | 24 |
| Production-ready features .....              | 24 |
| Analyzing and reporting .....                | 24 |
| Catch issues faster.....                     | 25 |
| Spend more time adding features .....        | 25 |
| Rapid development .....                      | 25 |
| 2. Setting up Jenkins.....                   | 26 |
| Introduction to Jenkins .....                | 26 |
| What is Jenkins made of? .....               | 26 |
| Note .....                                   | 30 |



|  |    |
|--|----|
| <b>Why use Jenkins as a Continuous Integration server?</b>         | 31 |
| <b>Jenkins as a centralized Continuous Integration server</b>      | 32 |
| <b>Hardware requirements</b>                                       | 34 |
| <b>Running Jenkins inside a container</b>                          | 34 |
| <b>Installing Jenkins as a service on the Apache Tomcat server</b> | 35 |
| <b>Note</b>  | 35 |
| <b>Setting up the Jenkins home path</b>                            | 36 |
| <b>Why run Jenkins inside a container?</b>                         | 38 |
| <b>Running Jenkins as a standalone application</b>                 | 40 |
| <b>Setting up Jenkins on Ubuntu</b>                                | 41 |
| <b>Note</b>  | 41 |
| <b>Setting up Jenkins on Fedora/Centos</b>                         | 43 |
| <b>Note</b>  | 43 |
| <b>Note</b>  | 44 |
| <b>Tip</b>   | 44 |
| <b>Sample use cases</b>  | 46 |
| <b>Netflix</b>   | 46 |
| <b>Yahoo!</b>  | 46 |
| <b>Note</b>  | 47 |
| <b>3. Configuring Jenkins</b>                                      | 47 |
| <b>Creating your first Jenkins job</b>                             | 48 |
| <b>Note</b>  | 49 |
| <b>Note</b>  | 52 |
| <b>Adding a build step</b>   | 52 |
| <b>Note</b>  | 53 |
| <b>Adding post-build actions</b>                                   | 54 |
| <b>Configuring the Jenkins SMTP server</b>                         | 55 |
| <b>Running a Jenkins job</b>                                       | 56 |
| <b>Note</b>  | 57 |
| <b>Jenkins build log</b>   | 58 |
| <b>Note</b>  | 59 |
| <b>Jenkins home directory</b>                                      | 60 |

|  |    |
|--|----|
| <b>Jenkins backup and restore.....</b>                             | 62 |
| <b>Creating a Jenkins job to take periodic backup.....</b>         | 63 |
| <b>Restoring a Jenkins backup.....</b>                             | 64 |
| <b>Upgrading Jenkins.....</b>                                      | 65 |
| <b>Note .....</b>  | 65 |
| <b>Upgrading Jenkins running on the Tomcat server .....</b>        | 66 |
| <b>Upgrading standalone Jenkins master running on Ubuntu.....</b>  | 67 |
| <b>Script to upgrade Jenkins on Ubuntu.....</b>                    | 70 |
| <b>Managing Jenkins plugins .....</b>                              | 71 |
| <b>The Jenkins Plugins Manager .....</b>                           | 72 |
| <b>Installing a Jenkins plugin to take periodic backup .....</b>   | 74 |
| <b>Configuring the periodic backup plugin .....</b>                | 75 |
| <b>Note .....</b>  | 77 |
| <b>User administration.....</b>                                    | 79 |
| <b>Enabling global security on Jenkins .....</b>                   | 79 |
| <b>Note .....</b>  | 82 |
| <b>Using the Project-based Matrix Authorization Strategy .....</b> | 83 |
| <b>4. Continuous Integration Using Jenkins – Part I .....</b>      | 87 |
| <b>Jenkins Continuous Integration Design.....</b>                  | 87 |
| <b>The branching strategy .....</b>                                | 88 |
| <b>The Continuous Integration pipeline.....</b>                    | 89 |
| <b>Note .....</b>  | 92 |
| <b>Toolset for Continuous Integration .....</b>                    | 92 |
| <b>Setting up a version control system .....</b>                   | 94 |
| <b>Installing Git .....</b>  | 94 |
| <b>Installing SourceTree (a Git client) .....</b>                  | 95 |
| <b>Creating a repository inside Git .....</b>                      | 95 |
| <b>Uploading code to Git repository .....</b>                      | 95 |
| <b>Configuring branches in Git .....</b>                           | 96 |
| <b>Note .....</b>  | 97 |
| <b>Git cheat sheet .....</b>                                       | 97 |
| <b>Configuring Jenkins .....</b>                                   | 98 |

|  |     |
|--|-----|
| <b>Installing the Git plugin .....</b>   | 98  |
| <b>Note .....</b>  | 100 |
| <b>Installing and configuring JDK.....</b>   | 100 |
| <b>Note .....</b>  | 102 |
| <b>Installing and configuring Maven .....</b>  | 102 |
| <b>Installing the e-mail extension plugin .....</b>  | 103 |
| <b>The Jenkins pipeline to poll the feature branch .....</b>                                 | 103 |
| <b>Creating a Jenkins job to poll, build, and unit test code on the feature1 branch ....</b> | 104 |
| <b>Note .....</b>  | 109 |
| <b>Creating a Jenkins job to merge code to the integration branch.....</b>                   | 114 |
| <b>Creating a Jenkins job to poll, build, and unit test code on the feature2 branch ....</b> | 116 |
| <b>Creating a Jenkins job to merge code to the integration branch.....</b>                   | 117 |
| <b>5. Continuous Integration Using Jenkins – Part II.....</b>                                | 118 |
| <b>Installing SonarQube to check code quality.....</b>                                       | 118 |
| <b>Setting the Sonar environment variables .....</b>   | 119 |
| <b>Running the SonarQube application .....</b>   | 119 |
| <b>Note .....</b>  | 120 |
| <b>Creating a project inside SonarQube .....</b>   | 120 |
| <b>Installing the build breaker plugin for Sonar .....</b>                                   | 122 |
| <b>Creating quality gates.....</b>   | 123 |
| <b>Installing SonarQube Scanner .....</b>  | 125 |
| <b>Setting the Sonar Runner environment variables.....</b>                                   | 127 |
| <b>Installing Artifactory.....</b>   | 127 |
| <b>Setting the Artifactory environment variables .....</b>                                   | 128 |
| <b>Running the Artifactory application .....</b>   | 129 |
| <b>Creating a repository inside Artifactory .....</b>  | 129 |
| <b>Jenkins configuration .....</b>   | 132 |
| <b>Installing the delivery pipeline plugin .....</b>   | 132 |
| <b>Installing the SonarQube plugin .....</b>   | 134 |
| <b>Note .....</b>  | 137 |
| <b>Installing the Artifactory plugin .....</b>   | 137 |
| <b>The Jenkins pipeline to poll the integration branch.....</b>                              | 140 |

|  |            |
|--|------------|
| Creating a Jenkins job to poll, build, perform static code analysis, and integration tests ..... | 141        |
| <b>Creating a Jenkins job to upload code to Artifactory .....</b>                                | <b>150</b> |
| Note .....   | 151        |
| Note .....   | 153        |
| <b>6. Continuous Delivery Using Jenkins.....</b>   | <b>154</b> |
| What is Continuous Delivery? .....   | 154        |
| Continuous Delivery Design .....   | 156        |
| Jenkins configuration .....  | 158        |
| Configuring Jenkins slaves on the testing server .....   | 158        |
| Note .....   | 163        |
| Slave agents via SSH tunneling .....   | 164        |
| Creating a Jenkins job to deploy code on the testing server .....                                | 166        |

## 1. Concepts of Continuous Integration

Understanding the concepts of **Continuous Integration** is our prime focus in the current chapter. However, to understand Continuous Integration, it is first important to understand the prevailing software engineering practices that gave birth to it. Therefore, we will first have an overview of various software development processes, their concepts, and implications. To start with, we will first glance through the **agile software development process**. Under this topic, we will learn about the popular software development process, the waterfall model, and its advantages and disadvantages when compared to the agile model. Then, we will jump to the **Scrum framework** of software development. This will help us to answer how Continuous Integration came into existence and why it is needed. Next, we will move to the concepts and best practices of Continuous Integration and see how this helps projects to get agile. Lastly, we will talk about all the necessary methods that help us realize the concepts and best practices of Continuous Integration.

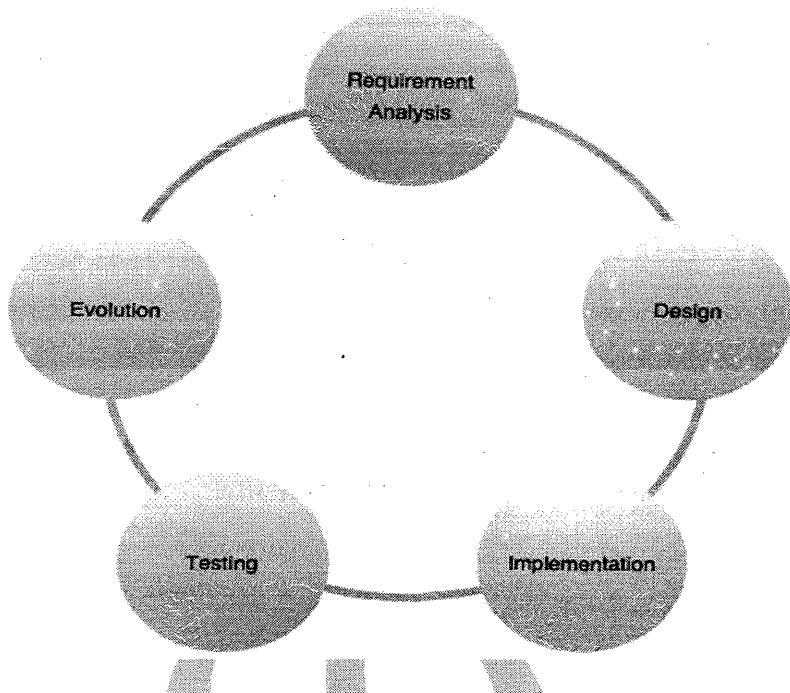
### The agile software development process

The name agile rightly suggests **quick and easy**. Agile is a collection of software development methodologies in which software is developed through collaboration among self-organized teams. Agile software development promotes adaptive planning. The principles behind agile are incremental, quick, and flexible software development.

For most of us who are not familiar with the software development process itself, let's first understand what the software development process or software development life cycle is.

### Software development life cycle

**Software development life cycle**, also sometimes referred to as **SDLC** in brief, is the process of planning, developing, testing, and deploying software. Teams follow a sequence of phases, and each phase uses the outcome of the previous phase, as shown in the following diagram:



### Continuous Integration:

Continuous Integration is a software development practice where developers frequently integrate their work with the project's **integration branch** and create a build.

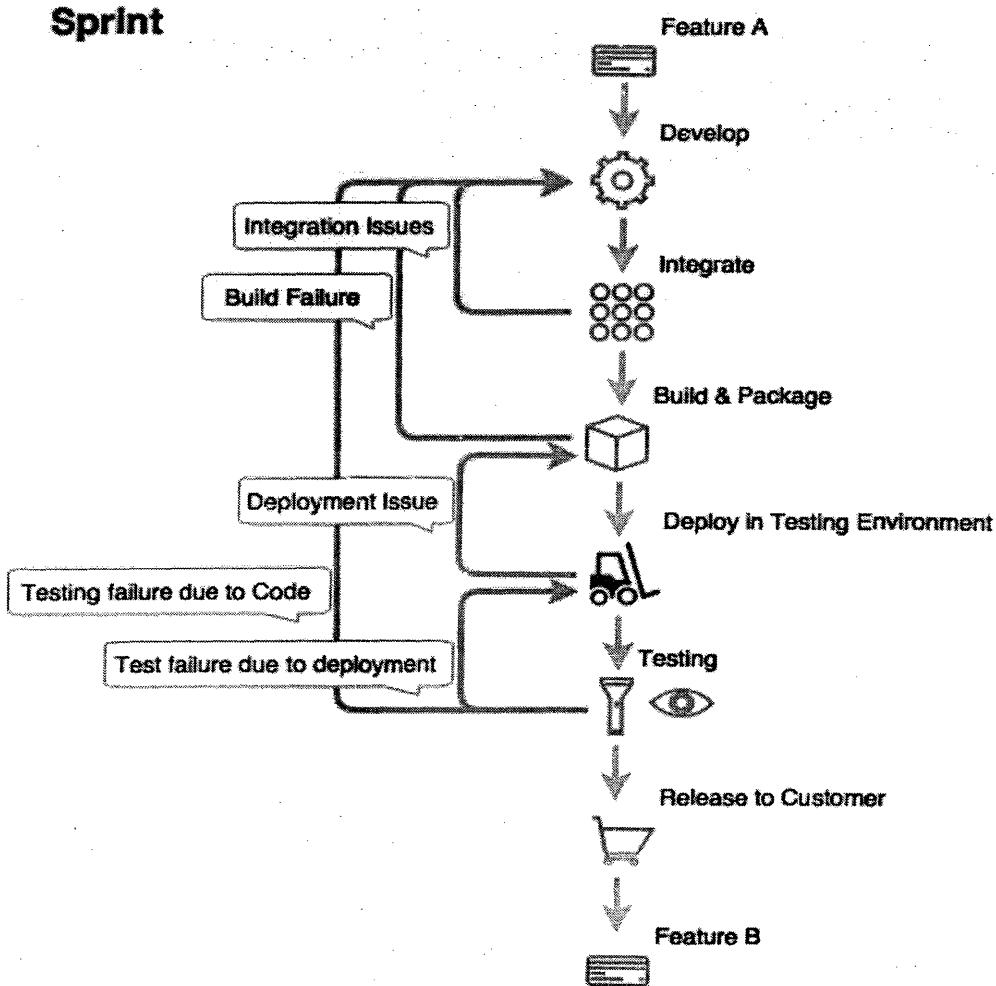
**Integration** is the act of submitting your personal work (modified code) to the common work area (the potential software solution). This is technically done by merging your personal work (personal branch) with the common work area (Integration branch). Continuous Integration is necessary to bring out issues that are encountered during the integration as early as possible.

This can be understood from the following diagram, which depicts various issues encountered during a software development lifecycle. I have considered a practical scenario wherein I have chosen the Scrum development model, and for the sake of simplicity, all the meeting phases are excluded. Out of all the issues depicted in the following diagram, the following ones are detected early when Continuous Integration is in place:

- ✓ Build failure (the one before integration)
- ✓ Integration issues
- ✓ Build failure (the one after integration)

In the event of the preceding issues, the developer has to modify the code in order to fix it. A build failure can occur either due to an improper code or due to a human error while doing a build (assuming that the tasks are done manually). An integration issue can occur if the developers do not rebase their local copy of code frequently with the code on the Integration branch.

## Sprint



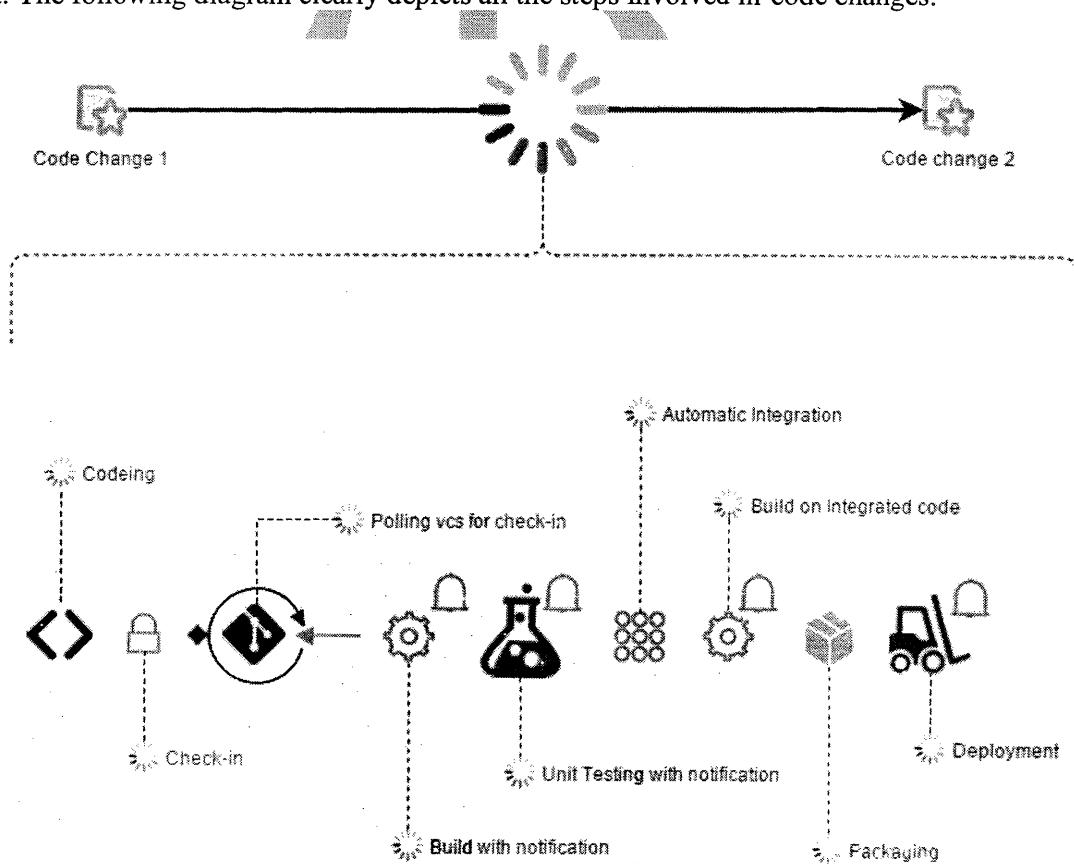
### Note

In the preceding diagram, I have considered only a single testing environment for simplicity. However, in reality, there can be as many as three to four testing environments.

### Agile runs on Continuous Integration

The agile software development process mainly focuses on faster delivery, and Continuous Integration helps it in achieving that speed. Yet, how does Continuous Integration do it? Let's understand this using a simple case.

Developing a feature may involve a lot of code changes, and between every code change, there can be a number of tasks, such as checking in the code, polling the version control system for changes, building the code, unit testing, integration, building on integrated code, packaging, and deployment. In a Continuous Integration environment, all these steps are made fast and error-free using automation. Adding notifications to it makes things even faster. The sooner the team members are aware of a build, integration, or deployment failure, the quicker they can act upon it. The following diagram clearly depicts all the steps involved in code changes:



### **How to achieve Continuous Integration**

Implementing Continuous Integration involves using various DevOps tools. Ideally, a DevOps engineer is responsible for implementing Continuous Integration. But, who is a DevOps engineer? And what is DevOps?

### **Development Operations**

DevOps stands for development operations, and the people who manage these operations are called DevOps engineers. All the following mentioned tasks fall under development operations:

- ✓ Build and release management
- ✓ Deployment management
- ✓ Version control system administration
- ✓ Software configuration management
- ✓ All sorts of automation
- ✓ Implementing continuous integration
- ✓ Implementing continuous testing
- ✓ Implementing continuous delivery
- ✓ Implementing continuous deployment
- ✓ Cloud management and virtualization

A DevOps engineer accomplishes the previously mentioned tasks using a set of tools; these tools are loosely called DevOps tools (Continuous Integration tools, agile tools, team collaboration tools, defect tracking tools, continuous delivery tools, cloud management tools, and so on).

A DevOps engineer has the capability to install and configure the DevOps tools to facilitate development operations. Hence, the name DevOps. Let's see some of the important DevOps activities pertaining to Continuous Integration.

### **Use a version control system**

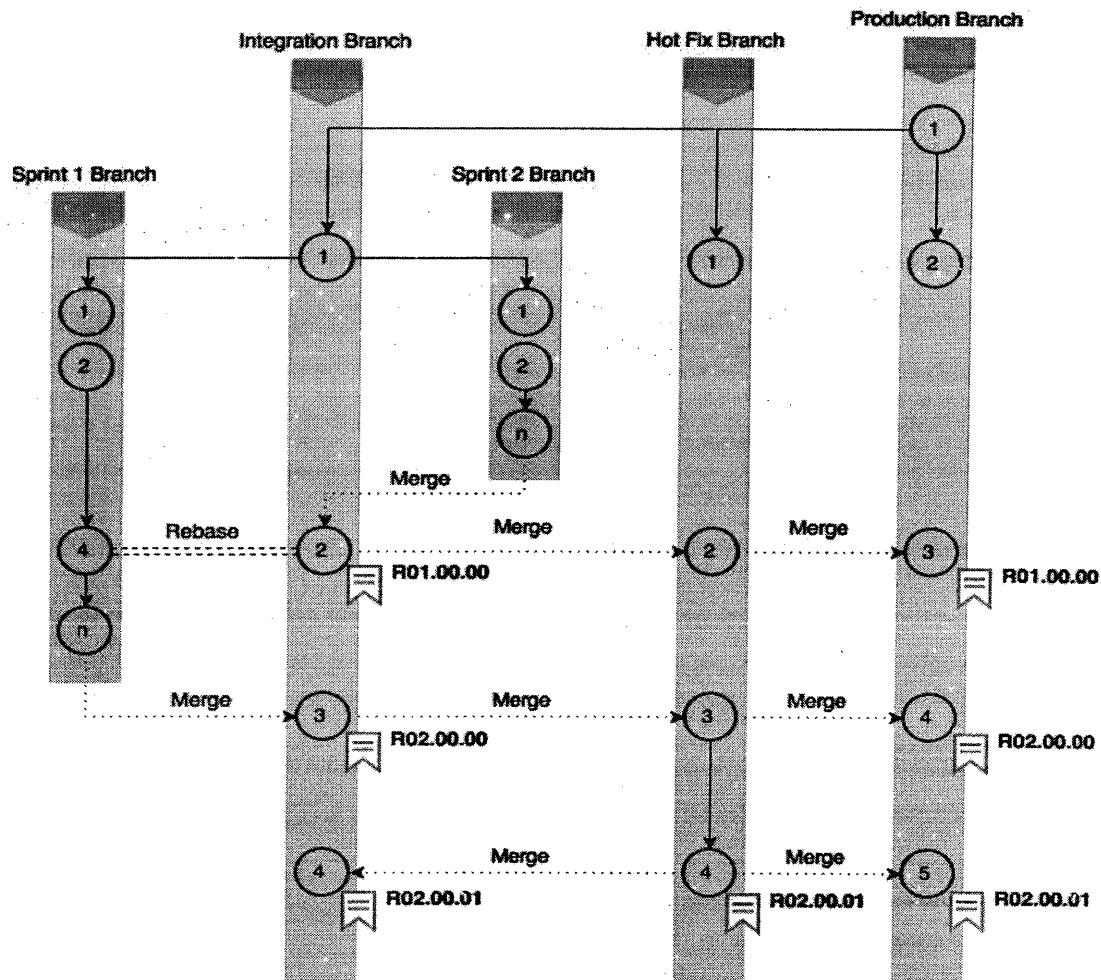
This is the most basic and the most important requirement to implement Continuous Integration. A version control system, or sometimes it's also called a **revision control system**, is a tool used to manage your code history. It can be centralized or distributed. Two of the famously centralized version control systems are SVN and IBM Rational ClearCase. In the distributed segment, we have tools such as Git. Ideally, everything that is required to build software must be version controlled. A version control tool offers many features, such as labeling, branching, and so on.

When using a version control system, keep the branching to the minimum. Few companies have only one main branch and all the development activities happening on that. Nevertheless, most companies follow some branching strategies. This is because there is always a possibility that part of a team may work on a release and others may work on another release. At other times, there is a need to support older release versions. Such scenarios always lead companies to use multiple branches.

For example, imagine a project that has an Integration branch, a release branch, a hotfix branch, and a production branch. The development team will work on the release branch. They check-out and check-in code on the release branch. There can be more than one release branch where development is running in parallel. Let's say these are sprint 1 and sprint 2.

Once sprint 2 is near completion (assuming that all the local builds on the sprint 2 branch were successful), it is merged to the Integration branch. Automated builds run when there is something checked-in on the Integration branch, and the code is then packaged and deployed in the testing environments. If the testing passes with flying colors and the business is ready to

move the release to production, then automated systems take the code and merge it with the production branch.



## Typical branching strategies

From here, the code is then deployed in production. The reason for maintaining a separate branch for production comes from the desire to maintain a neat code with less number of versions. The production branch is always in sync with the hotfix branch. Any instant fix required on the production code is developed on the hotfix branch. The hotfix changes are then merged to the production as well as the Integration branch. The moment sprint 1 is ready, it is

first rebased with the Integration branch and then merged into it. And it follows the same steps thereafter.

### **Types of version control system**

We have already seen that a version control system is a tool used to record changes made to a file or set of files over time. The advantage is that you can recall specific versions of your file or a set of files. Almost every type of file can be version controlled. It's always good to use a **Version Control System (VCS)** and almost everyone uses it nowadays. You can revert an entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover.

Looking back at the history of version control tools, we can observe that they can be divided into three categories:



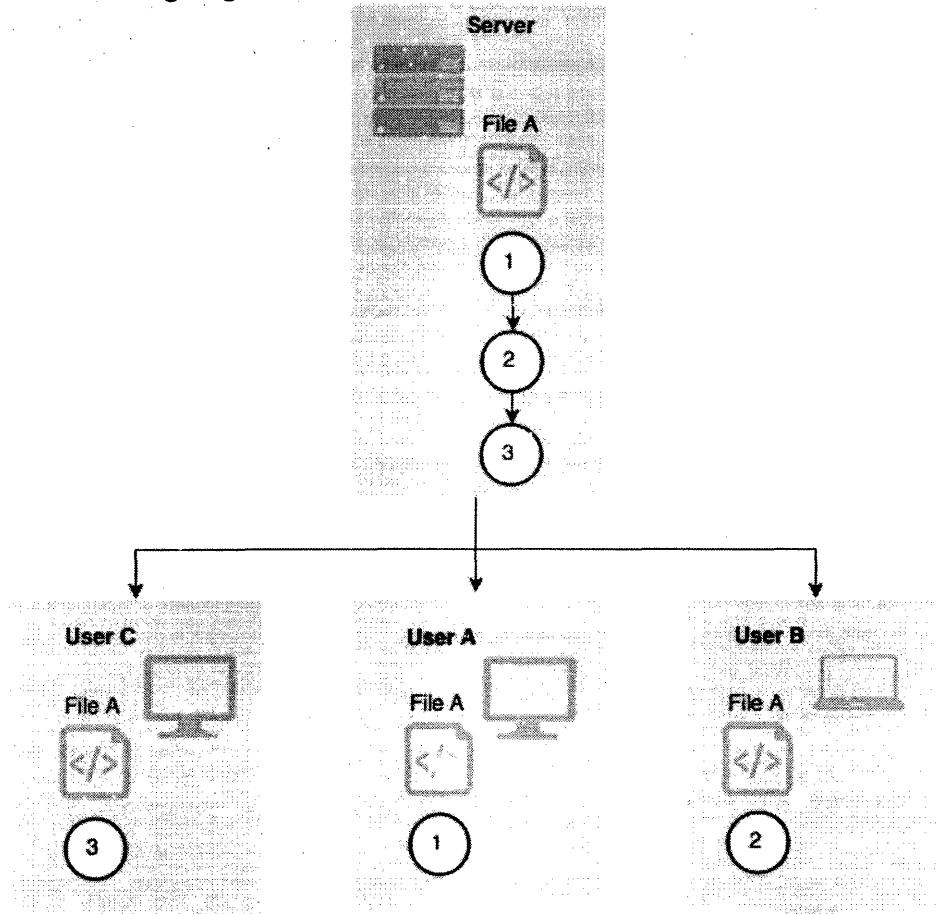
- Local version control systems
- Centralized version control systems
- Distributed version control systems

### **Centralized version control systems**

Initially, when VCS came into existence some 40 years ago, they were mostly personal, like the one that comes with Microsoft Office Word, wherein you can version control a file you are working on. The reason was that in those times software development activity was minuscule in magnitude and was mostly done by individuals. But, with the arrival of large software development teams working in collaboration, the need for a centralized VCS was sensed. Hence, came VCS tools, such as Clear Case and Perforce. Some of the advantages of a centralized VCS are as follows:

- All the code resides on a centralized server. Hence, it's easy to administrate and provides a greater degree of control.
- These new VCS also bring with them some new features, such as labeling, branching, and baselining to name a few, which help people collaborate better.
- In a centralized VCS, the developers should always be connected to the network. As a result, the VCS at any given point of time always represents the updated code.

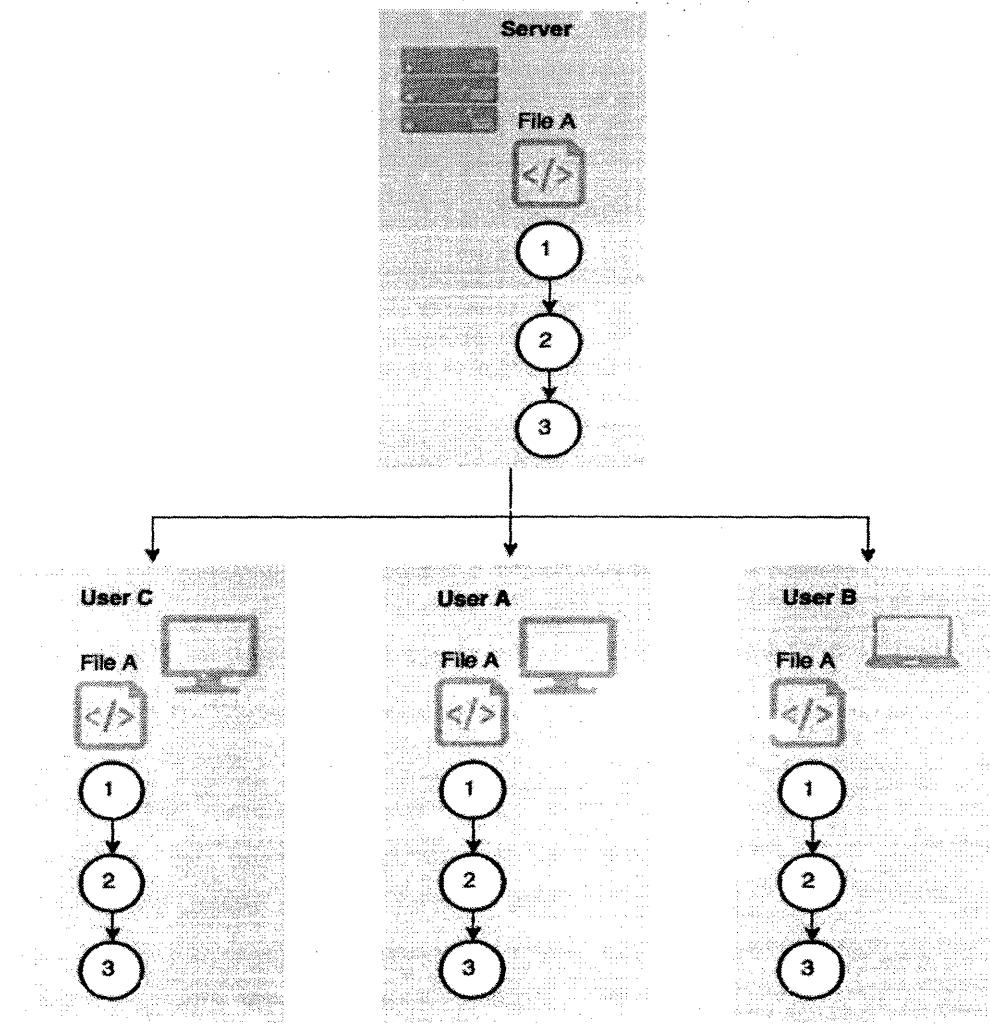
The following diagram illustrates a centralized VCS:



**Distributed version control systems**

Another type of VCS is the distributed VCS. Here, there is a central repository containing all the software solution code. Instead of creating a branch, the developers completely clone the central repository on their local machine and then create a branch out of the local clone repository. Once they are done with their work, the developer first merges their branch with the Integration branch, and then syncs the local clone repository with the central repository.

You can argue that this is a combination of a local VCS plus a central VCS. An example of a distributed VCS is Git.



### **Use repository tools**

As part of the software development life cycle, the source code is continuously built into binary artifacts using Continuous Integration. Therefore, there should be a place to store these built packages for later use. The answer is to use a repository tool. But, what is a repository tool?

A repository tool is a version control system for binary files. Do not confuse this with the version control system discussed in the previous sections. The former is responsible for versioning the source code and the latter for binary files, such as .rar, .war, .exe, .msi, and so on.

As soon as a build is created and passes all the checks, it should be uploaded to the repository tool. From there, the developers and testers can manually pick them, deploy them, and test them, or if the automated deployment is in place, then the build is automatically deployed in the respective test environment. So, what's the advantage of using a build repository?

A repository tool does the following:

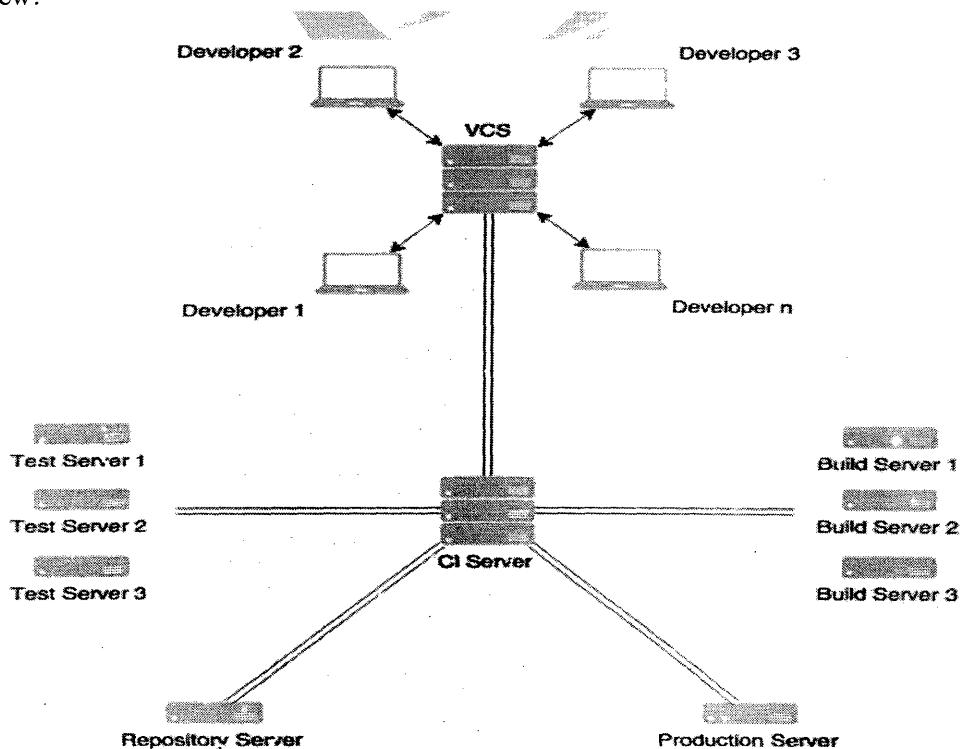
- Every time a build gets generated, it is stored in a repository tool. There are many advantages of storing the build artifacts. One of the most important advantages is that the build artifacts are located in a centralized location from where they can be accessed when needed.
- It can store third-party binary plugins, modules that are required by the build tools. Hence, the build tool need not download the plugins every time a build runs. The repository tool is connected to the online source and keeps updating the plugin repository.
- It records what, when, and who created a build package.
- It creates a staging area to manage releases better. This also helps in speeding up the Continuous Integration process.

- In a Continuous Integration environment, each build generates a package and the frequency at which the build and packaging happen is high. As a result, there is a huge pile of packages. Using a repository tool makes it possible to store all the packages in one place. In this way, developers get the liberty to choose what to promote and what not to promote in higher environments.

### **Use a Continuous Integration tool**

What is a Continuous Integration tool? It is nothing more than an orchestrator. A continuous integration tool is at the center of the Continuous Integration system and is connected to the version control system tool, build tool, repository tool, testing and production environments, quality analysis tool, test automation tool, and so on. All it does is an orchestration of all these tools, as shown in the next image.

There are many Continuous Integration tools: Jenkins, Build Forge, Bamboo, and Team city to name a few.



Basically, Continuous Integration tools consist of various pipelines. Each pipeline has its own purpose. There are pipelines used to take care of Continuous Integration. Some take care of testing, some take care of deployments, and so on. Technically, a pipeline is a flow of jobs. Each job is a set of tasks that run sequentially. Scripting is an integral part of a Continuous Integration tool that performs various kinds of tasks. The tasks may be as simple as copying a folder/file from one location to another, or it can be a complex Perl script used to monitor a machine for file modification.

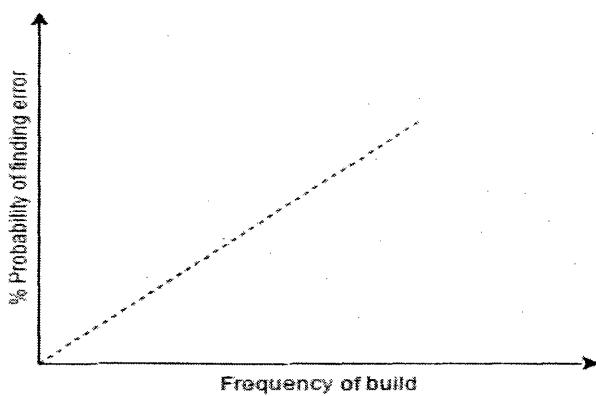
### **Creating a self-triggered build**

The next important thing is the self-triggered automated build. Build automation is simply a series of automated steps that compile the code and generate executables. The build automation can take help of build tools, such as Ant and Maven. Self-triggered automated builds are the most important parts of a Continuous Integration system. There are two main factors that call for an automated build mechanism:

- Speed
- Catching integration or code issues as early as possible

There are projects where 100 to 200 builds happen per day. In such cases, speed is an important factor. If the builds are automated, then it can save a lot of time. Things become even more interesting if the triggering of the build is made self-driven without any manual intervention. An auto-triggered build on every code change further saves time.

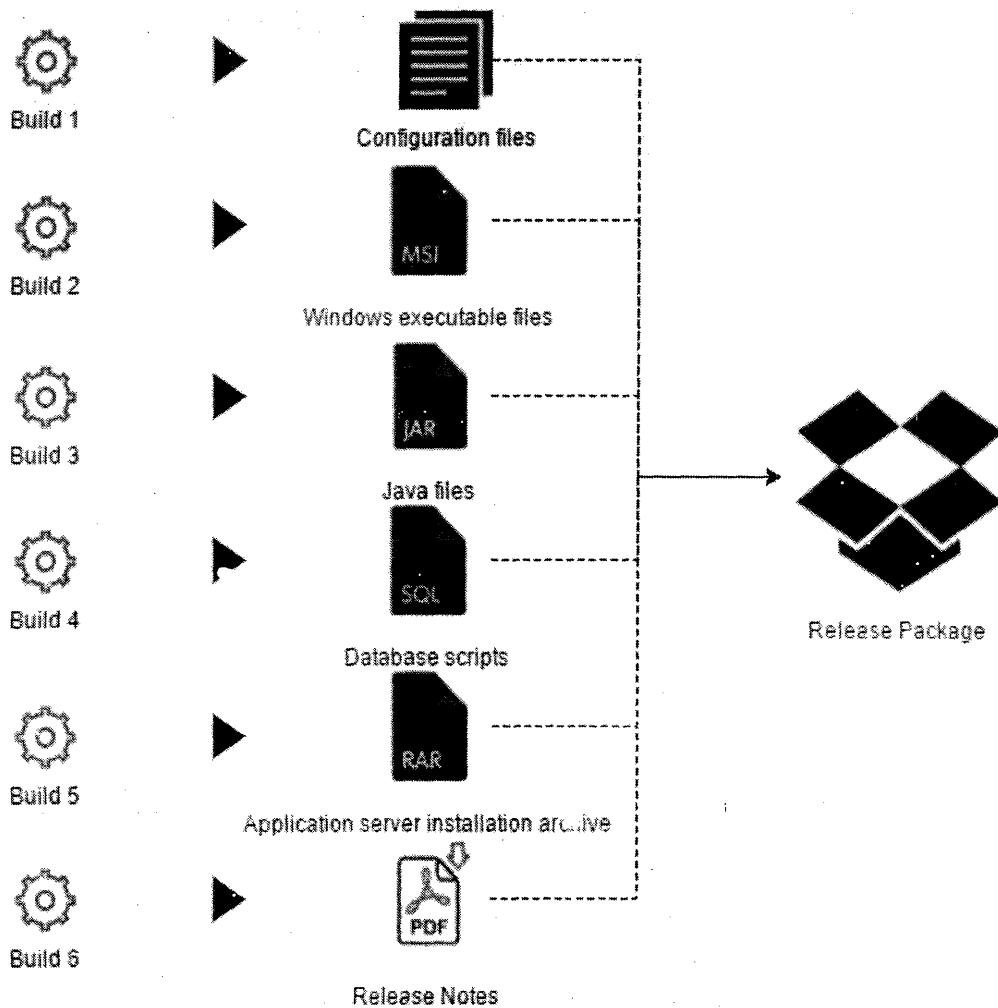
When builds are frequent and fast, the probability of finding errors (a build error, compilation error, and integration error) is also greater and faster.



### Automate the packaging

There is a possibility that a build may have many components. Let's take, for example, a build that has a .rar file as an output. Along with this, it has some Unix configuration files, release notes, some executables, and also some database changes. All these different components need to be together. The task of creating a single archive or a single media out of many components is called packaging.

This again can be automated using the Continuous Integration tools and can save a lot of time.



### **Using build tools**

IT projects can be on various platforms, such as Java, .NET, Ruby on Rails, C, and C++ to name a few. Also, in a few places, you may see a collection of technologies. No matter what, every programming language, excluding the scripting languages, has compilers that compile the code. Ant and Maven are the most common build tools used for projects based on Java. For the .NET lovers, there is MSBuild and TFS build. Coming to the Unix and Linux world, you have make and omake, and also clearmake in case you are using IBM Rational ClearCase as the version control tool. Let's see the important ones.

#### **Maven**

Maven is a build tool used mostly to compile Java code. It uses Java libraries and Maven plugins in order to compile the code. The code to be built is described using an XML file that contains information about the project being built, dependencies, and so on.

Maven can be easily integrated into Continuous Integration tools, such as Jenkins, using plugins.

#### **MSBuild**

MSBuild is a tool used to build Visual Studio projects. MSBuild is bundled with Visual Studio. MSBuild is a functional replacement for nmake. MSBuild works on project files, which have the XML syntax, similar to that of Apache Ant. Its fundamental structure and operation are similar to that of the Unix make utility. The user defines what will be the input (the various source codes), and the output (usually, a .exe or .msi). But, the utility itself decides what to do and the order in which to do it.

### **Automating the deployments**

Consider an example, where the automated packaging has produced a package that contains .war files, database scripts, and some Unix configuration files. Now, the task here is to deploy all the three artifacts into their respective environments. The .war files must be

deployed in the application server. The Unix configuration files should sit on the respective Unix machine, and lastly, the database scripts should be executed in the database server. The deployment of such packages containing multiple components is usually done manually in almost every organization that does not have automation in place. The manual deployment is slow and prone to human errors. This is where the automated deployment mechanism is helpful.

Automated deployment goes hand in hand with the automated build process. The previous scenario can be achieved using an automated build and deployment solution that builds each component in parallel, packages them, and then deploys them in parallel. Using tools such as Jenkins, this is possible. However, there are some challenges, which are as follows:

- There is a considerable amount of scripting required to orchestrate build packaging and deployment of a release containing multiple components. These scripts by themselves are huge code to maintain that require time and resources.
- In most of the cases, deployment is not as simple as placing files in a directory. For example, there are situations where the deployment activity is preceded by steps to configure the environment.

**Note**

The field of managing the configuration on multiple machines is called **configuration management**. There are tools, such as Chef and Puppet, to do this.

**Automating the testing**

Testing is an important part of a software development life cycle. In order to maintain quality software, it is necessary that the software solution goes through various test scenarios. Giving less importance to testing can result in customer dissatisfaction and a delayed product.

Since testing is a manual, time-consuming, and repetitive task, automating the testing process can significantly increase the speed of software delivery. However, automating the testing process is a bit more difficult than automating the build, release, and deployment processes. It usually takes a lot of efforts to automate nearly all the test cases used in a project. It is an activity that matures over time.

Hence, when we begin to automate the testing, we need to take a few factors into consideration. Test cases that are of great value and easy to automate must be considered first. For example, automate the testing where the steps are the same, but they run every time with different data. You can also automate the testing where a software functionality is being tested on various platforms. In addition, automate the testing that involves a software application running on different configurations.

Previously, the world was mostly dominated by the desktop applications. Automating the testing of a GUI-based system was quite difficult. This called for scripting languages where the manual mouse and keyboard entries were scripted and executed to test the GUI application. Nevertheless, today the software world is completely dominated by the web and mobile-based applications, which are easy to test through an automated approach using a test automation tool.

Once the code is built, packaged, and deployed, testing should run automatically to validate the software. Traditionally, the process followed is to have an environment for SIT, UAT, PT, and Pre-Production. First, the release goes through SIT, which stands for System Integration Test. Here, testing is performed on an integrated code to check its functionality all together. If pass, the code is deployed in the next environment, that is, UAT where it goes through a user acceptance test, and then similarly, it can lastly be deployed in PT where it goes through the performance test. Thus, in this way, the testing is prioritized.

It is not always possible to automate all of the testing. But, the idea is to automate whatever testing is possible. The previous method discussed requires the need to have many environments and also a number of automated deployments into various environments. To

avoid this, we can go for another method where there is only one environment where the build is deployed, and then, the basic tests are run and after that, long running tests are triggered manually.

### **Use static code analysis**

Static code analysis, also commonly called **white-box testing**, is a form of software testing that looks for the structural qualities of the code. For example, it reveals how robust or maintainable the code is. Static code analysis is performed without actually executing programs. It is different from the functional testing, which looks into the functional aspects of software and is dynamic.

Static code analysis is the evaluation of software's inner structures. For example, is there a piece of code used repetitively? Does the code contain lots of commented lines? How complex is the code? Using the metrics defined by a user, an analysis report can be generated that shows the code quality in terms of maintainability. It doesn't question the code functionality.

Some of the static code analysis tools, such as SonarQube come with a dashboard, which shows various metrics and statistics of each run. Usually, as part of Continuous Integration, the static code analysis is triggered every time a build runs. As discussed in the previous sections, static code analysis can also be included before a developer tries to check-in his code. Hence, code of low quality can be prevented right at the initial stage.

Static code analysis support many languages, such as Java, C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, and so on.

### **Continuous Integration benefits**

The way a software is developed always affects the business. The code quality, the design, time spent in development and planning of features, all affect the promises that a company has made to its clients.

Continuous Integration helps the developers in helping the business. While going through the previous topics, you might have already figured out the benefits of implementing Continuous Integration. However, let's see some of the benefits that Continuous Integration has to offer.

## Freedom from long integrations

When every small change in your code is built and integrated, the possibility of catching the integration errors at an early stage increases. Rather than integrating once in 6 months, as seen in the waterfall model, and then spending weeks resolving the merge issues, it is good to integrate frequently and avoid the merge hell. The Continuous Integration tool like Jenkins automatically builds and integrates your code upon check-in.

## Production-ready features

Continuous Delivery enables you to release deployable features at any point in time. From a business perspective, this is a huge advantage. The features are developed, deployed, and tested within a timeframe of 2 to 4 weeks and are ready to go live with a click of a button.

## Analyzing and reporting

How frequent are the releases? What is the success rate of builds? What is the thing that is mostly causing a build failure? Real-time data is always a must in making critical decisions. Projects are always in the need of recent data to support decisions. Usually, managers collect this information manually, which requires time and efforts. Continuous Integration tools, such as Jenkins provide the ability to see trends and make decisions. A Continuous Integration system provides the following features:

- Real-time information on the recent build status and code quality metrics.
- Since integrations occur frequently with a Continuous Integration system, the ability to notice trends in build, and overall quality becomes possible.

Continuous Integration tools, such as Jenkins provide the team members with metrics about the build health. As all the build, packaging, and deployment work is automated and tracked using a Continuous Integration tool; therefore, it is possible to generate statistics

about the health of all the respective tasks. These metrics can be the build failure rate, build success rate, the number of builds, who triggered the build, and so on.

All these trends can help project managers and the team to ensure that the project is heading in the right direction and at the right pace.

### **Catch issues faster**

This is the most important advantage of having a carefully implemented Continuous Integration system. Any integration issue or merge issue gets caught early. The Continuous Integration system has the facility to send notifications as soon as the build fails.

### **Spend more time adding features**

In the past, development teams performed the build, release, and deployments. Then, came the trend of having a separate team to handle build, release, and deployment work. Yet again that was not enough, as this model suffered from communication issues between the development team and the release team.

However, using Continuous Integration, all the build, release, and the deployment work gets automated. Therefore, now the development team need not worry about anything other than developing features. In most of the cases, even the completed testing is automated.

### **Rapid development**

From a technical perspective, Continuous Integration helps teams work more efficiently. This is because Continuous Integration works on the agile principles. Projects that use Continuous Integration follow an automatic and continuous approach while building, testing, and integrating their code. This results in a faster development.

Since everything is automated, developers spend more time developing their code and zero time on building, packaging, integrating, and deploying it. This also helps teams, which are geographically distributed, to work together. With a good software configuration management process in place, people can work on large teams. **Test Driven Development (TDD)** can further enhance the agile development by increasing its efficiency.

**"Behind every successful agile project, there is a Continuous Integration server!"**

## 2. Setting up Jenkins

### Introduction to Jenkins

Jenkins is an open source Continuous Integration tool. However, it's not limited to Continuous Integration alone. In the upcoming chapters, we will see how Jenkins can be used to achieve Continuous Delivery, Continuous Testing, and Continuous Deployment. Jenkins is supported by a large number of plugins that enhance its capability. The Jenkins tool is written in Java and so are its plugins. The tool has a minimalistic GUI that can be enhanced using specific plugins if required.

### What is Jenkins made of?

Let's have a look at the components that make up Jenkins. The Jenkins framework mainly contains jobs, builds, parameters, pipelines and plugins. Let's look at them in detail.

#### Jenkins job

At a higher level, a typical Jenkins job contains a unique name, a description, parameters, build steps, and post-build actions. This is shown in the following screenshot:

Project name

Description

Discard Old Builds 

This build is parameterized 

Disable Build (No new builds will be executed until the project is re-enabled.) 

Restrict where this project can be run 

**Advanced Project Options**

Use custom workspace 

Display Name

Keep the build logs of dependencies 

**Source Code Management**

None

Git

Subversion

**Build Triggers**

Trigger builds remotely (e.g., from scripts) 

Build after other projects are built 

Build periodically 

Poll SCM 

**Build**

Execute Windows batch command 

Command

**Post-build Actions**

E-mail Notification 

Recipients

Send e-mail for every unstable build 

Send separate e-mails to individuals who broke the build 

Trigger parameterized build on other projects 

Build Triggers

Projects to build

Trigger when build is

**Save** **Apply**

C

## Jenkins parameters

Jenkins parameters can be anything: environment variables, interactive values, pre-defined values, links, triggers, and so on. Their primary purpose is to assist the builds. They are also responsible for triggering pre-build activities and post-build activities.

## Jenkins build

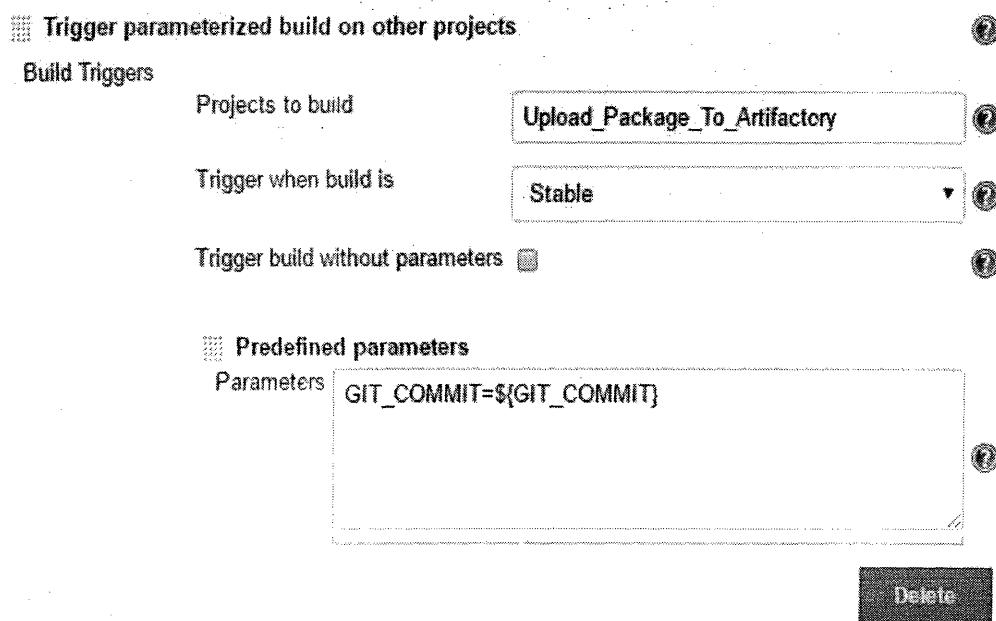
A Jenkins build (not to be confused with a software build) can be anything from a simple Windows batch command to a complex Perl script. The range is extensive, which include Shell, Perl, Ruby, and Python scripts or even Maven and Ant builds. There can be number of build steps inside a Jenkins job and all of them run in sequence. The following screenshot is an example of a Maven build followed by a Windows batch script to merge code:

The screenshot shows a Jenkins build configuration with two steps:

- Step 1: Invoke Maven 3**
  - Maven Version: Maven for Nodes
  - Root POM: payslip/pom.xml
  - Goals and options: clean test -Puat
  - Buttons: Advanced..., Delete
- Step 2: Execute Windows batch command**
  - Command: E:  
cd ProjectJenkins  
git checkout integration  
git merge feature1 --stat
  - Text: See [the list of available environment variables](#)
  - Buttons: Delete

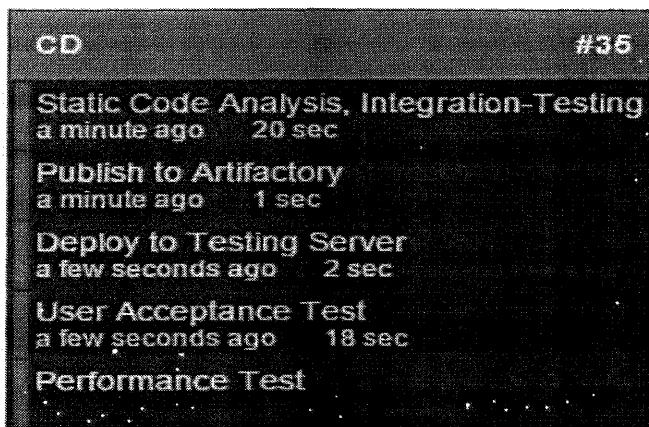
### Jenkins post-build actions

Post-build actions are parameters and settings that define the subsequent steps to be performed after a build. Some post-build actions can be configured to perform various activities depending on conditions. For example, we can have a post-build action in our current job, which in the event of a successful build starts another Jenkins job. This is shown in the following screenshot:



### Jenkins pipeline

Jenkins pipeline, in simple terms, is a group of multiple Jenkins jobs that run in sequence or in parallel or a combination of both. The following screenshot is an example of a Jenkins Continuous Delivery pipeline. There are five separate Jenkins jobs, all running one after the other.



### Note

Jenkins Pipeline is used to achieve a larger goal, like Continuous Integration or Continuous Delivery.

### Jenkins plugins

Jenkins plugins are software pieces that enhance the Jenkins' functionality. Plugins after installation, manifest in the form of either system settings or parameters inside a Jenkins job.

There is a special section inside the Jenkins master server to manage plugins. The following screenshot shows the Jenkins system configuration section. It's a setting to configure the SonarQube tool (a static code analysis tool). The configuration is available only after installing the Jenkins plugin for SonarQube named **sonar**.

SonarQube

Environment variables  Enable injection of SonarQube server configuration as build environment variables

SonarQube installations

|                            |  |
|----------------------------|--|
| Name                       | Sonar  |
| Server URL                 | <input type="text"/>                                     |
| SonarQube account login    | <input type="text"/><br>Default is http://localhost:9000 |
| SonarQube account password | <input type="password"/>                                 |

Disable   
Check to quickly disable SonarQube on all jobs.

**Advanced...**

**Delete SonarQube**

**Add SonarQube**

List of SonarQube installations



## Why use Jenkins as a Continuous Integration server?

DevOps engineers across the world have their own choice when it comes to Continuous Integration tools. Yet, Jenkins remains an undisputed champion among all. The following are some of the advantages of using Jenkins.

### It's open source

There are a number of Continuous Integration tools available in the market, such as Go, Bamboo, TeamCity, and so on. But the best thing about Jenkins is that it's free, simple yet powerful, and popular among the DevOps community.

### Community-based support

Jenkins is maintained by an open source community. The people who created the original Hudson are all working for Jenkins after the Jenkins-Hudson split.

### **Lots of plugins**

There are more than 300 plugins available for Jenkins and the list keeps increasing. Plugins are simple Maven projects. Therefore, anyone with a creative mind can create and share their plugins on the Jenkins community to serve a purpose.

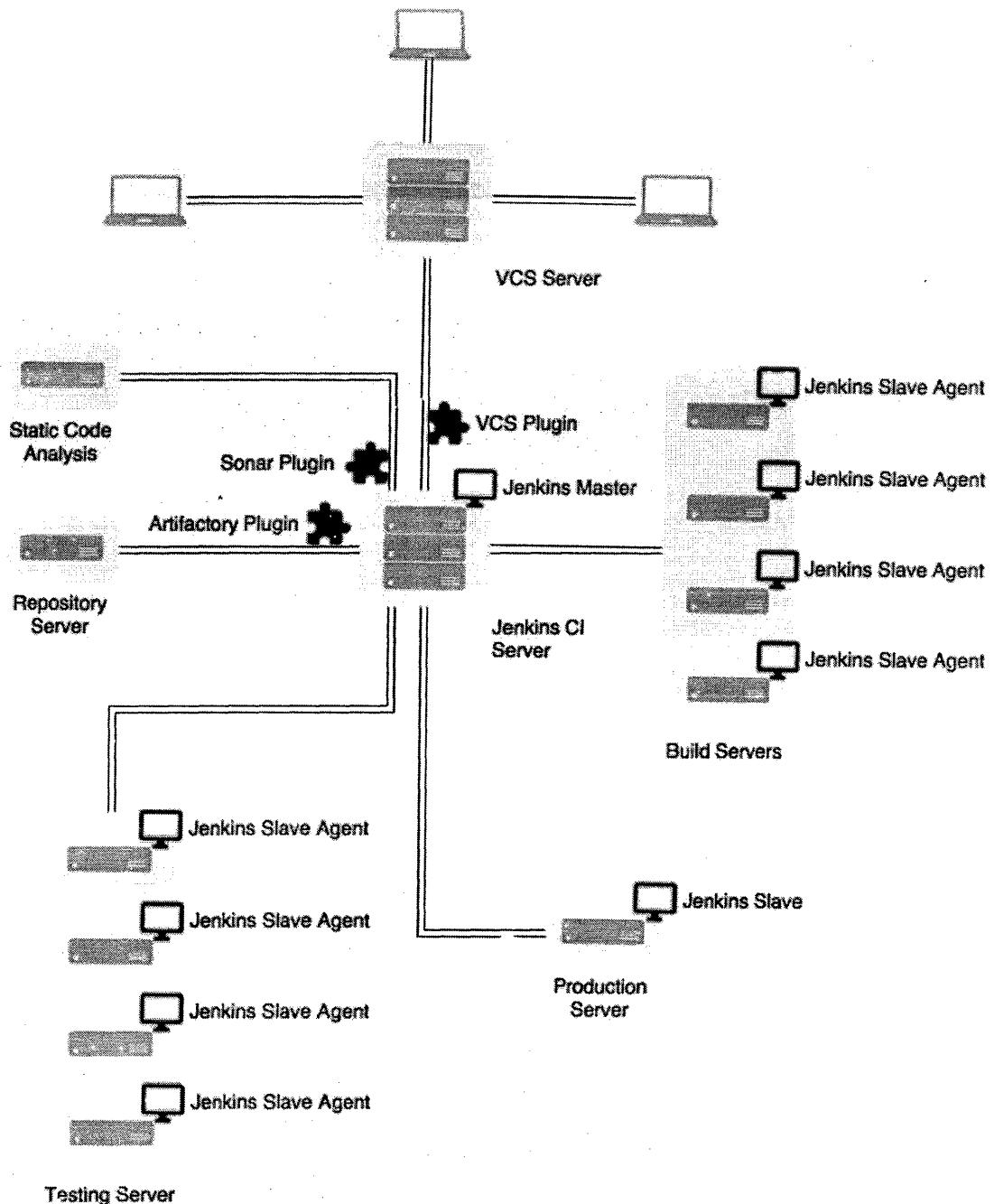
### **Jenkins has a cloud support**

There are times when the number of builds, packaging, and deployment requests are more, and other times they are less. In such scenarios, it is necessary to have a dynamic environment to perform builds. This can be achieved by integrating Jenkins with a cloud-based service such as AWS. With this set up, build environments can be created and destroyed automatically as per demand.

### **Jenkins as a centralized Continuous Integration server**



Jenkins is clearly an orchestrator. It brings all the other DevOps tools together in order to achieve Continuous Integration. This is clearly depicted in the next screenshot. We can see Jenkins communicating with the version control tool, repository tool, and static code analysis tool using plugins. Similarly, Jenkins communicates with the build servers, testing servers, and the production server using the Jenkins slave agent.



**Hardware requirements**

Answering the hardware requirements of Jenkins is quite a challenge. Ideally, a system with Java 7 or above and 1-2 GB RAM is enough to run Jenkins master server. However, there are organizations that go way up to 60+ GB RAM for their Jenkins Master Server alone.

Therefore, hardware specifications for a Jenkins master server largely depend on the organization's requirements. Nevertheless, we can make a connection between the Jenkins operations and the hardware as follows:

- The number of users accessing Jenkins master server (number of HTTP requests) will cost mostly the CPU.
- The number of Jenkins slaves connected to Jenkins master server will cost mostly the RAM.
- The number of jobs running on a Jenkins master server will cost the RAM and the disk space.
- The number of builds running on a Jenkins master server will cost the RAM and the disk space (this can be ignored if builds happen on Jenkins slave machines).

**Running Jenkins inside a container**

Jenkins can be installed as a service inside the following containers:

- Apache Geronimo 3.0
- Glassfish
- IBM WebSphere
- JBoss
- Jetty
- Jonas

- Liberty profile
- Tomcat
- WebLogic

### **Installing Jenkins as a service on the Apache Tomcat server**

Installing Jenkins as a service on the Apache Tomcat server is pretty simple. We can either choose to use Jenkins along with other services already present on the Apache Tomcat server, or we may use the Apache server solely for Jenkins.

#### **Prerequisites**

I assume that the Apache Tomcat server is installed on the machine where you intend to run Jenkins. In the following section, we will use the Apache Tomcat server 8.0. Nevertheless, Apache Tomcat server 5.0 or greater is sufficient to use Jenkins. A machine with 1 GB RAM is enough to start with. However, as the number of jobs and builds increase, so should the memory.

We also need Java running on the machine. In this section, we are using jre1.8.0\_60. While installing the Apache Tomcat server, you will be asked to install Java. Nevertheless, it is suggested that you always use the latest stable version available.

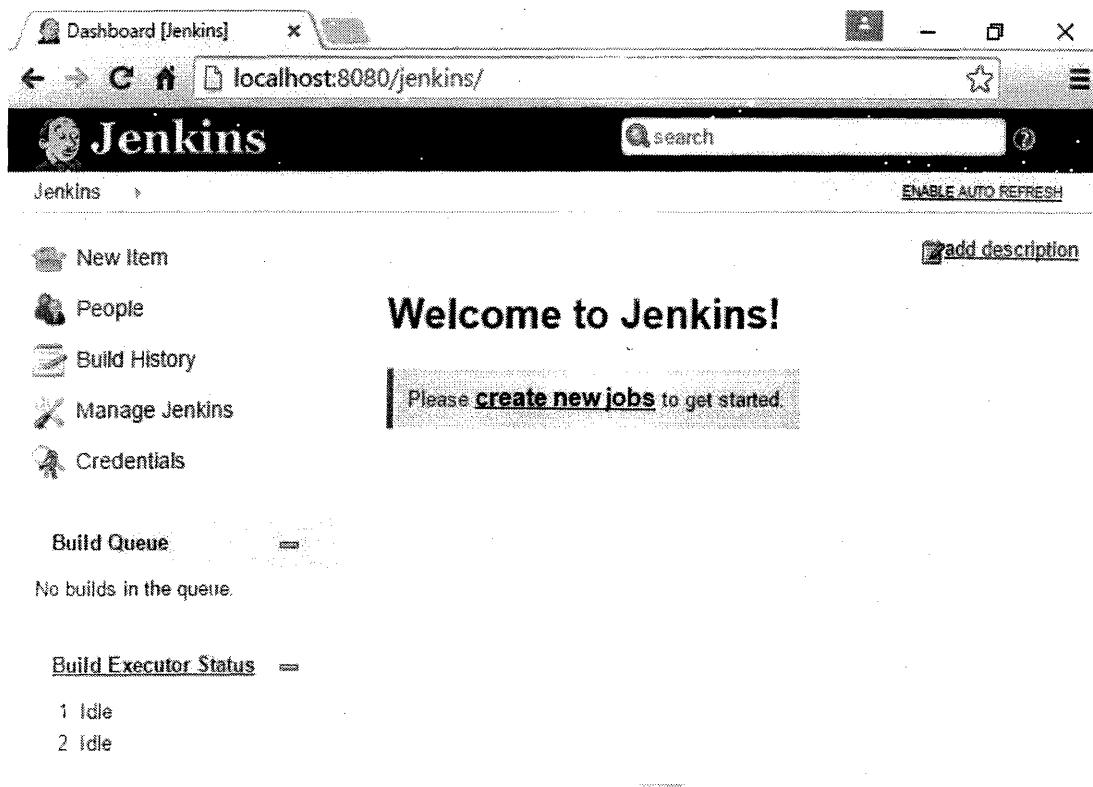
#### **Note**

The current section focuses on running Jenkins inside a container like Apache Tomcat. Therefore, the underlying OS where the Apache Tomcat server is installed can be anything. We are using Windows 10 OS in the current subtopic.

Perform the following steps for installing Jenkins inside a container:

1. Download the latest jenkins.war file from <https://jenkins.io/download/>.
2. Simply move the downloaded jenkins.war file to the webapps folder, which is present inside the installation directory of your Apache Tomcat server.

3. That's all you need to do. You can access Jenkins using the URL <http://localhost:8080/jenkins>.
4. The Jenkins Dashboard is shown in the following screenshot:



### Setting up the Jenkins home path

Before we start using Jenkins, there is one important thing to configure: the JENKINS\_HOME path. This is the location where all of the Jenkins configurations, logs, and builds are stored. Everything that you create and configure on the Jenkins dashboard is stored here.

In our case, by default, the JENKINS\_HOME variable is set to C:\Windows\System32\config\systemprofile\jenkins. We need to make it something more accessible, for example, C:\Jenkins. This can be done in two ways.

**Method 1 – configuring the context.xml file**

Context.xml is a configuration file related to the Apache Tomcat server. We can configure the JENKINS\_HOME variable inside it using the following steps:

1. Stop the Apache Tomcat server.
2. Go to C:\Program Files\Apache Software Foundation\Tomcat 8.0\conf.
3. Modify the context.xml file using the following code:

```
4. <Context>
5. <Environment name="JENKINS_HOME" value="C:\Jenkins" type="java.lang.String"/>
6. </Context>
```

7. After modifying the file, start the Apache Tomcat server.

**Method 2 – creating the JENKINS\_HOME environment variable**

We can create the JENKINS\_HOME variable using the following steps:

- 1.
2. Stop the Apache Tomcat server.
3. Now, open the Windows command prompt and run the following command:

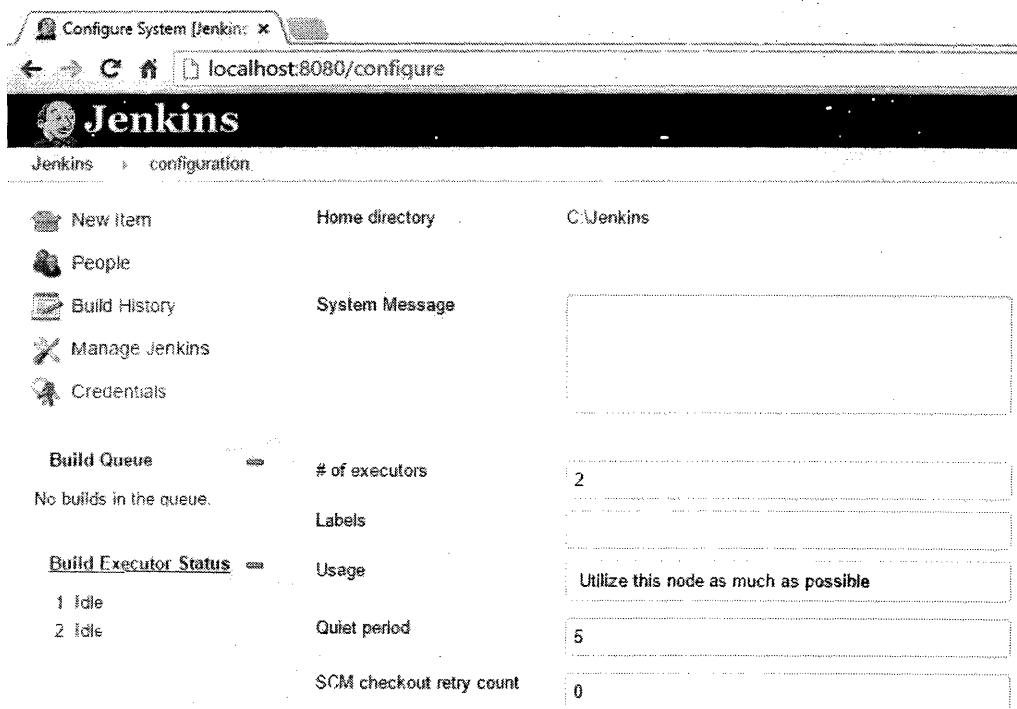
```
setx JENKINS_HOME "C:\Jenkins"
```

4. After executing the command, check the value of JENKINS\_HOME with the following command:

```
echo %JENKINS_HOME%
```
5. The output should be:

## C:\Jenkins

6. Start the Apache Tomcat server.
7. To check if the Jenkins home path is set to C:\Jenkins, open the following link: <http://localhost:8080/configure>. You should see the Home directory value set to C:\Jenkins, as shown in the following screenshot:



## Why run Jenkins inside a container?

The reason that most organizations choose to use Jenkins on a web server is the same as the reason most organizations use web servers to host their websites: better traffic management.

The following factors affect Jenkins server performance:

- Number of jobs
- Number of builds

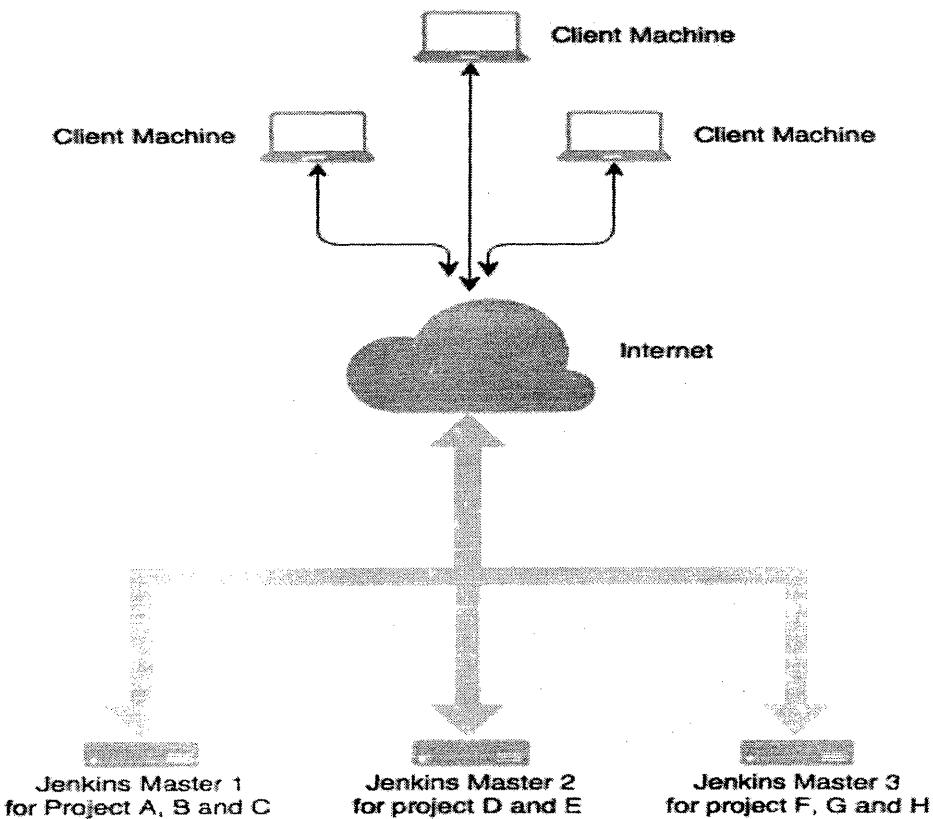
- Number of slaves
- Number of users accessing Jenkins server (number of HTTP requests)

All these factors can push organizations towards any one of the following tactics:

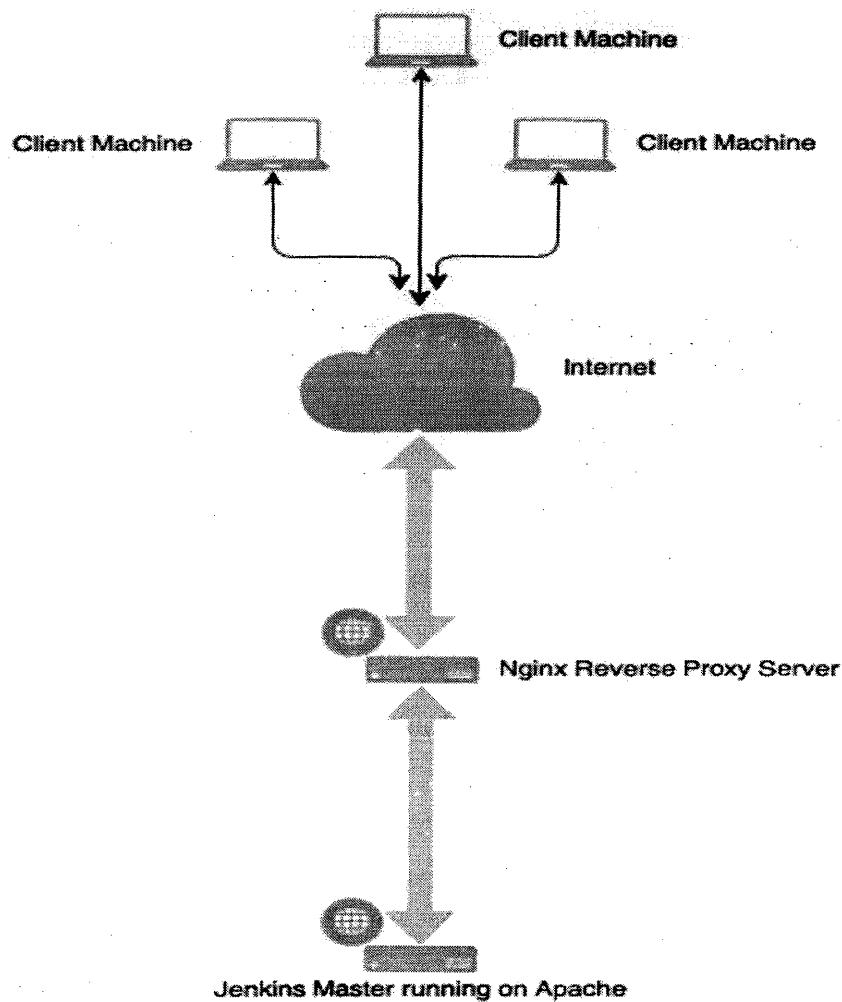
- **Approach 1:** Using multiple Jenkins masters, one each for every project
- **Approach 2:** Maintaining a single Jenkins master on a web server, with an enhanced hardware and behind a reverse proxy Server

The following table measures the merits of both tactics against few performance factors:

The following image shows **Approach 1:**



The following image demonstrates **Approach 2:**



### Running Jenkins as a standalone application

Installing Jenkins as a standalone application is simpler than installing Jenkins as a service inside a container. Jenkins is available as a standalone application on the following operating systems:

- Windows
- Ubuntu/Debian

- Red Hat/Fedora/CentOS
- Mac OS X
- openSUSE
- FreeBSD
- openBSD
- Gentoo

### Setting up Jenkins on Ubuntu

To install the latest stable version of Jenkins, perform the following steps in sequence:

1. Check for admin privileges; the installation might ask for the admin username and password.
2. Download the latest version of Jenkins using the following command:

```
 wget -q -O - http://jenkins-ci.org/debian-stable/jenkins-ci.org.key | sudo apt-key add -
 sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian-stable binary/ >
 /etc/apt/sources.list.d/jenkins.list'
```

3. To install Jenkins, issue the following commands:

```
sudo apt-get update
sudo apt-get install jenkins
```

4. Jenkins is now ready for use. By default, the Jenkins service runs on port 8080.
5. To access Jenkins, go to the following link in the web browser, <http://localhost:8080/>

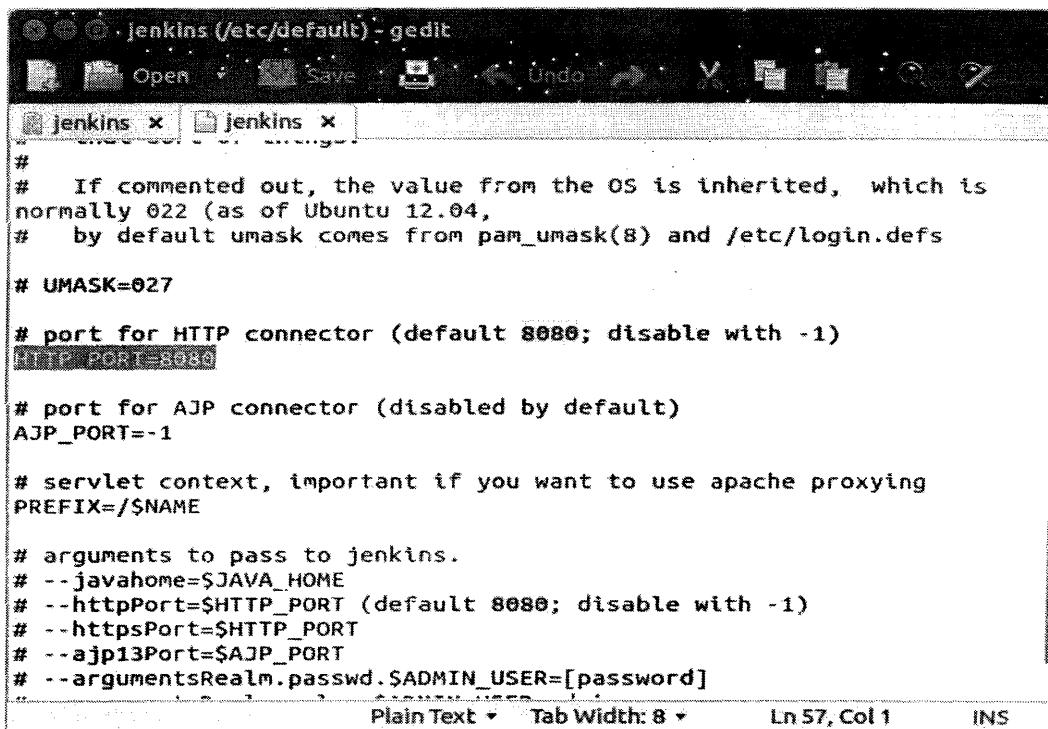
### Note

In order to troubleshoot Jenkins, access the logs present at `/var/log/jenkins/jenkins.log`.

The Jenkins service runs with the user Jenkins, which automatically gets created upon installation.

### Changing the Jenkins port on Ubuntu

- To change the Jenkins port on Ubuntu, perform the following steps:
- 1. In order to change the Jenkins port, open the `jenkins` file present inside `/etc/default/`.
- 2. As highlighted in the following screenshot, the `HTTP_PORT` variable stored the port number:



```
# If commented out, the value from the OS is inherited, which is
# normally 022 (as of Ubuntu 12.04,
# by default umask comes from pam_umask(8) and /etc/login.defs

# UMASK=027

# port for HTTP connector (default 8080; disable with -1)
#HTTP_PORT=8080

# port for AJP connector (disabled by default)
#AJP_PORT=-1

# servlet context, important if you want to use apache proxying
#PREFIX=/$NAME

# arguments to pass to jenkins.
# --javahome=$JAVA_HOME
# --httpPort=$HTTP_PORT (default 8080; disable with -1)
# --httpsPort=$HTTP_PORT
# --ajp13Port=$AJP_PORT
# --argumentsRealm.passwd.$ADMIN_USER=[password]
```

- 3. Inside the same file, there is another important thing to note, the memory heap size. Heap size is the amount of memory allocated for the Java Virtual Machine to run properly.
- 4. You can change the heap size by modifying the `JAVA_ARGS` variable as shown in the following example.
- 5. We can also change the user with which the Jenkins service runs on Ubuntu. In the following screenshot, we can see a variable `NAME` with a value `jenkins`. We can change this to any user we want.

```

Jenkins (/etc/default) - gedit
Open Save Undo Redo Cut Copy Paste Find Replace
jenkins x jenkins x
# defaults for jenkins continuous integration server
# pulled in from the init script; makes things easier.
NAME=jenkins

# location of java
JAVA=/usr/bin/java

# arguments to pass to java
JAVA_OPTS="-Djava.awt.headless=true" # Allow graphs etc. to work even
when an X server is present
#JAVA_HOME=$HOME/jdk1.6.0_20
#JAVA_OPTS="-Djava.net.preferIPv4Stack=true" # make jenkins listen on
IPv4 address

PIDFILE=/var/run/$NAME/$NAME.pid

# user and group to be invoked as (default to jenkins)
JENKINS_USER=$NAME
JENKINS_GROUP=$NAME

# location of the jenkins war file
JENKINS_WAR=/usr/share/$NAME/$NAME.war

```

Plain Text ▾ Tab Width: 8 ▾ Ln 11, Col 1 INS

## Setting up Jenkins on Fedora/Centos

In order to install Jenkins on Fedora, open the Terminal. Make sure Java is installed on the machine and JAVA\_HOMEvariable is set.

### Note

Installing Jenkins on Red Hat Linux is similar to installing Jenkins on Fedora.

### Installing the latest stable version of Jenkins

1. Check for admin privileges; the installation might ask for admin username and password.
2. Download the latest version of Jenkins using the following command:

```

sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo

```

```
sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
```

3. To install Jenkins issue the following commands:

```
sudo yum install Jenkins
```

### Note

The link <http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo> mentioned in the first command leads to the Jenkins repository for the latest stable Jenkins rpm package.

4. Once the Jenkins installation is successful, it will automatically run as a daemon service. By default Jenkins runs on the port 8080.
5. To access Jenkins, go to the following link in the web browser <http://localhost:8080/>.

### Tip

If for some reason you are unable to access Jenkins, then check the firewall setting. This is because, by default, the firewall will block the ports. To enable them, give the following commands (you might need admin privileges):

```
firewall-cmd --zone=public --add-port=8080/tcp --permanent
```

```
firewall-cmd --zone=public --add-service=http --permanent
```

```
firewall-cmd --reload
```

In order to troubleshoot Jenkins, access the logs present at `var/log/jenkins/jenkins.log`.

The Jenkins service runs with the user Jenkins which automatically gets created upon installation.

### Changing the Jenkins port on Fedora

To change the Jenkins port on Fedora, perform the following steps:

1. Open the terminal in Fedora.

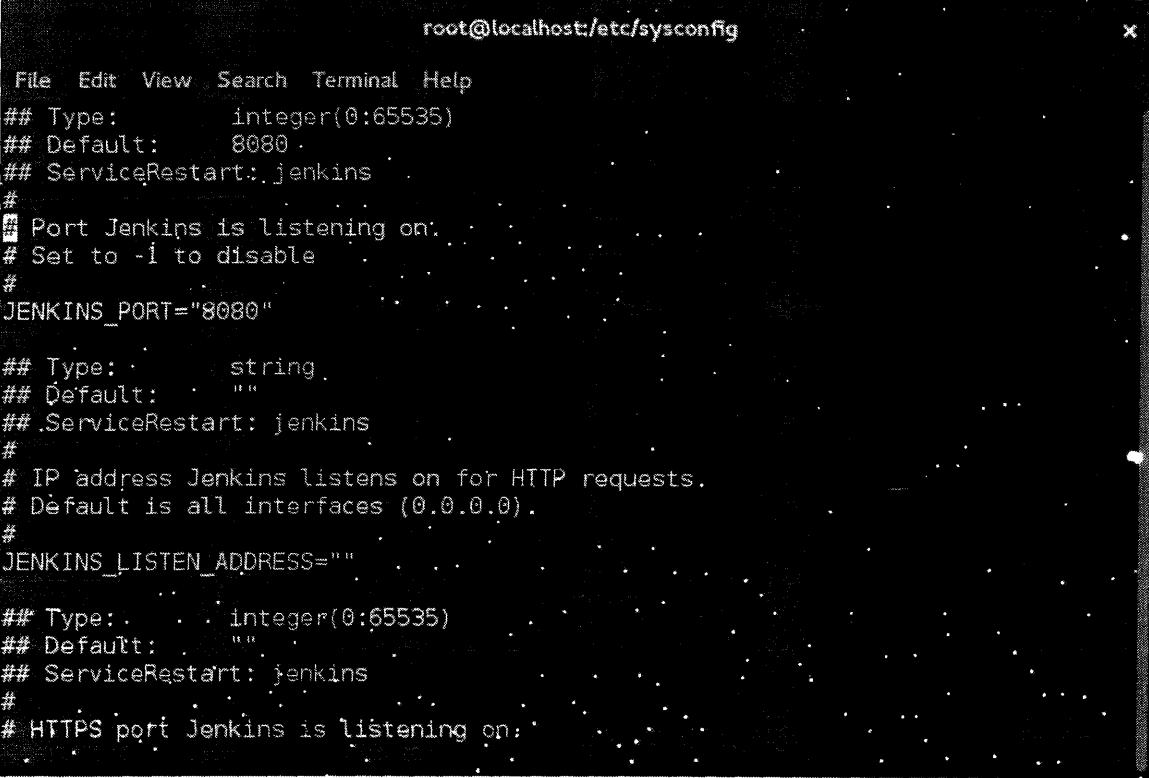
2. Switch to the admin account using the following command:

```
sudo su -
```

3. Enter the password when prompted.
4. Execute the following commands to edit the file named jenkins present at /etc/sysconfig/:

```
cd /etc/sysconfig/  
vi jenkins
```

5. Once the file is open in the terminal, move to the line where you see JENKINS\_PORT="8080", as shown in the following screenshot:



```
root@localhost:/etc/sysconfig  
File Edit View Search Terminal Help  
## Type: integer(0:65535)  
## Default: 8080  
## ServiceRestart: jenkins  
#  
# Port Jenkins is listening on:  
# Set to -i to disable  
#  
JENKINS_PORT="8080"  
  
## Type: string  
## Default: ""  
## ServiceRestart: jenkins  
#  
# IP address Jenkins listens on for HTTP requests.  
# Default is all interfaces (0.0.0.0).  
#  
JENKINS_LISTEN_ADDRESS=""  
  
## Type: integer(0:65535)  
## Default: ""  
## ServiceRestart: jenkins  
#  
# HTTPS port Jenkins is listening on:
```

**Sample use cases**

It is always good to learn from others' experiences. The following are the use cases published by some famous organizations that can give us some idea of the hardware specification.

**Netflix**

In 2012, Netflix had the following configuration:

**Hardware configuration:**

- 2x quad core x86\_64 for the Jenkins master with 26 GB RAM
  - 1 Jenkins master with 700 engineers using it
  - Elastic slaves with Amazon EC2 + 40 ad-hoc slaves in Netflix's data center
- Work load:
- 1,600 Jenkins jobs
  - 2,000 Builds per day
  - 2 TB of build data

**Yahoo!**

In 2013, Yahoo! had the following configuration:

**Hardware configuration:**

- 2 x Xeon E5645 2.40GHz, 4.80GT QPI (HT enabled, 12 cores, 24 threads) with 96 GB RAM, and 1.2 TB of disk space
- 1 Jenkins master with 1,000 engineers using it
- 48 GB max heap to JVM
- \$JENKINS\_HOME\* lives on NetApp

- 20 TB filer volume to store Jenkins job and build data
- 50 Jenkins slaves in three data centers

Workload:

- 13,000 Jenkins jobs
- 8,000 builds per day

**Note**

`$JENKINS_HOME` is the environment variable that stores the Jenkins home path. This is where all the Jenkins metadata, logs, and build data gets stored.

### **3. Configuring Jenkins**

The previous chapter was all about installing Jenkins on various platforms. In this chapter, we will see how to perform some basic Jenkins administration. We will also familiarize ourselves with some of the most common Jenkins tasks, like creating jobs, installing plugins, and performing Jenkins system configurations. We will discuss the following:

- Creating a simple Jenkins job with an overview of its components
- An overview of the Jenkins home directory
- Jenkins backup and restore
- Upgrading Jenkins
- Managing and configuring plugins
- Managing users and permissions

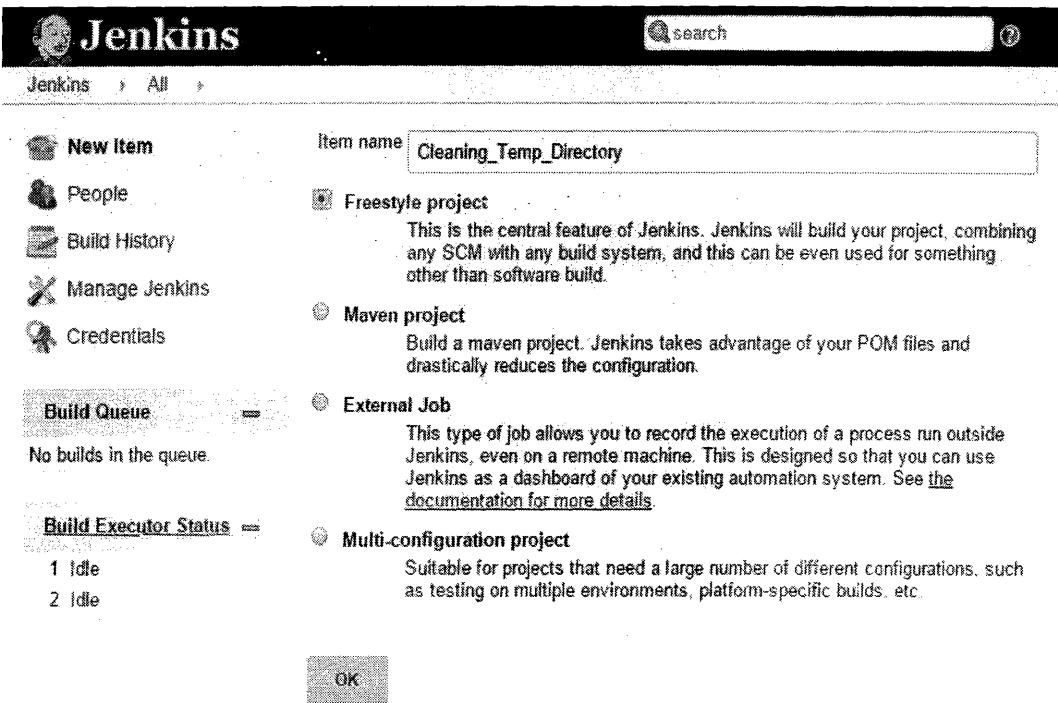
## Creating your first Jenkins job

In the current section, we will see how to create a Jenkins Job to clean up the %temp% directory on our Windows machine where the Jenkins master server is running. We will also configure it to send an e-mail notification. We will also see how Jenkins incorporates variables (Jenkins system variable and Windows system variable) while performing various tasks. The steps are as follows:

1. From the Jenkins Dashboard, click on the **New Item** link present on the left side. This is the link to create a new Jenkins job.

The screenshot shows the Jenkins dashboard at [localhost:8080/jenkins/](http://localhost:8080/jenkins/). The page title is "Dashboard [Jenkins]". On the left sidebar, there are several links: "New Item" (highlighted with a red box), "People", "Build History", "Manage Jenkins", and "Credentials". Below the sidebar, there's a "Build Queue" section with the message "No builds in the queue." and a "Build Executor Status" section showing "1 Idle" and "2 Idle". In the center, a large banner says "Welcome to Jenkins!" and "Please [create new jobs](#) to get started.". At the bottom, there are links for "Help us localize this page", "Page generated: Oct 22, 2015 4:33:50 PM", "REST API", and "Jenkins ver. 1.629".

2. Name your Jenkins job Cleaning\_Temp\_Directory in the **Item name** field.
3. Select the **Freestyle project** option that is present right below the **Item name** field.
4. Click on the **OK** button to create the Jenkins job.



5. You will be automatically redirected to the page where you can configure your Jenkins job.

### Note

The Jenkins job name contains underscores between the words. But this is not strictly necessary, as Jenkins has its own way of dealing with blank spaces. However, maintaining a particular naming standard helps in managing and comprehending Jenkins jobs better.

Below the **Item name** field, there are four options to choose from: **Freestyle project**, **Maven project**, **External Job**, and **Multi-configuration project**. These are predefined templates, each having various options that define the functionality and scope of the Jenkins job. All of them are self-explanatory.

6. The **Project name** field contains the name of our newly created Jenkins job.
7. Below that, we have the option to add some description about our Jenkins job. I added one for our Jenkins job.

|              |  |
|--------------|--|
| Project name | Cleaning_Temp_Directory  |
| Description  | Jenkins Job to clean up the temp directory on the current machine. |

[Plain text] [Preview](#)

8. Below the **Description** section, there are other options that can be ignored for now. Nevertheless, you can click on the question mark icon, present after each option to know its functionality, as shown in the following screenshot:

Discard Old Builds 

This controls the disk consumption of Jenkins by managing how long you'd like to keep records of the builds (such as console output, build artifacts, and so on.) Jenkins offers two criteria:

1. Driven by age. You can have Jenkins delete a record if it reaches a certain age (for example, 7 days old.)
2. Driven by number. You can have Jenkins make sure that it only maintains up to N build records. If a new build is started, the oldest record will be simply removed.

Jenkins also allows you to mark an individual build as 'Keep this log forever', to exclude certain important builds from being discarded automatically. The last stable and last successful build are always kept as well.

This build is parameterized 

Disable Build (No new builds will be executed until the project is re-enabled.) 

Execute concurrent builds if necessary 

9. Scrolling down further, you will see the **Advanced Project Options** section and the **Source Code Management** section. Skip them for now as we don't need them.

## Advanced Project Options

- Quiet period 
  - Retry Count 
  - Block build when upstream project is building 
  - Block build when downstream project is building 
  - Use custom workspace 
- Display Name  
- Keep the build logs of dependencies 

## Source Code Management

- None
- CVS
- CVS Projectset
- Subversion

10. On scrolling down further, you will see the **Build Triggers** option.

11. Under the **Build Triggers** section, select the **Build periodically** option and add H 23 \* \* \* inside the **Schedule** field. We would like our Jenkins job to run daily around 11:59 PM throughout the year.

## Build Triggers

- Build after other projects are built 
  - Build periodically 
- Schedule  
- Would last have run at Wednesday, 21 October, 2015 11:36:16 PM IST; would next run at Thursday, 22 October, 2015 11:36:16 PM IST. 
- Poll SCM 

## Note

The schedule format is Minute (0-59) Hour (0-23) Day (1-31) Month (1-12) Weekday (0-7). In the weekday section, 0 & 7 are Sunday.

You might ask the significance of the symbol H in place of the minute. Imagine a situation where you have more than 10 Jenkins jobs scheduled for the same time, say 59 23 \* \* \*. There is a chance Jenkins will get overloaded when all the Jenkins jobs start at once. To avoid this, we use an option H in the minute place. By doing so, Jenkins starts each job with a gap of 1 minute.

12. Moving further down brings you to the most important part of the job's configuration: the **Build** section.

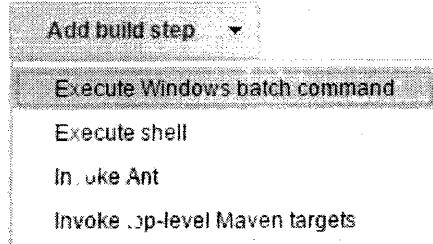
## Adding a build step



Build steps are sections inside the Jenkins jobs that contain scripts, which perform the actual task. You can run a Windows batch script or a shell script or any script for that matter. The steps are as follows:

1. Click on the **Add build step** button and select the **Execute Windows batch command** option.

### Build



2. In the **Command** field, add the following command. This build step will take us to the %temp% directory and will list its contents. The code is as follows:

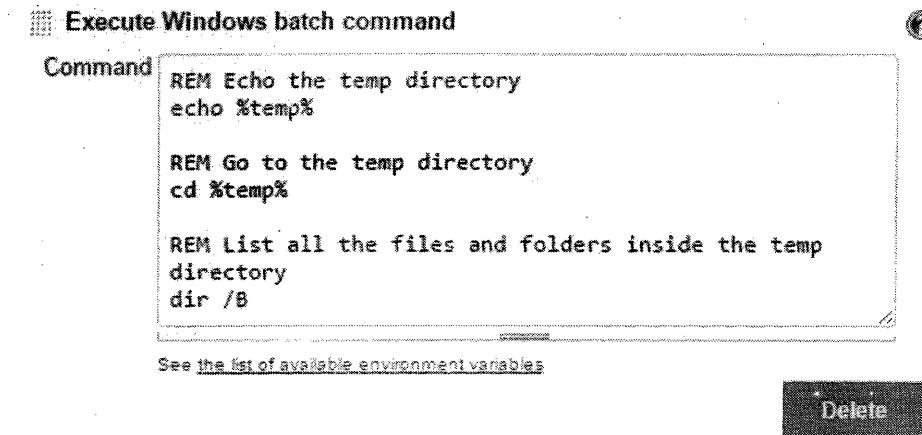
```
REM Echo the temp directory  
echo %temp%
```

```
REM Go to the temp directory  
cd %temp%
```

```
REM List all the files and folders inside the temp directory  
dir /B
```

The following screenshot displays the **Command** field in the **Execute Windows batch command** option:

Build



### Note

Instead of giving a complete path to the temp directory, I used %temp%, which is a system environment variable that stores the path to the temp directory. This is one beautiful feature of Jenkins where we can boldly use the system environment variables.

3. You can create as many builds as you want, using the **Add build step** button. Let's create one more build step that deletes everything inside the %temp% directory and then lists its content after deletion:

The following screenshot displays the **Command** field in the **Execute Windows batch command** option:

Execute Windows batch command

Command

```
REM Delete everything inside the temp directory  
del /S %temp%\*  
  
REM List all the files and folders inside the temp  
directory  
dir /B
```

See the list of available environment variables

Delete

- That's it. To summarize, the first build takes us to the %temp% directory and the second build deletes everything inside it. Both the builds list the content of the temp directory.

## Adding post-build actions



Perform the following steps to add post-build actions:

- Scroll down further and you'll come across the **Post-build Actions** option.  
**Post-build Actions**  
**Add post-build action** ▾
- Click on the **Add post-build action** button and select the **E-mail Notification** option from the menu.

Post-build Actions

- Aggregate downstream test results
- Archive the artifacts
- Build other projects
- Publish JUnit test result report
- Publish Javadoc
- Record fingerprints of files to track usage
- E-mail Notification**

Add post-build action ▾

3. In the **Recipients** field, add the list of e-mail addresses (team members), separated by a space.

## Post-build Actions

E-mail Notification

Recipients

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

Delete

Add post-build action ▾

4. There are a few options under the **E-mail Notification** section that can be ignored for now. Nevertheless, you can explore them.
5. Click on the **Save** button, present at the end of the page, to save the preceding configuration. Failing to do so will scrap the whole configuration.

## Configuring the Jenkins SMTP server

Now that we have created a Jenkins job, let's move on to configure the SMTP server without which the **E-mail Notification** wouldn't work:

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link.
2. On the **Manage Jenkins** page, click on the **Configure System** link.
3. On the configuration page, scroll down until you see the **E-mail Notification** section.

**E-mail Notification**

|   |                                    |  |
|---|------------------------------------|--|
| SMTP server   | <input type="text"/>               |  |
| Default user e-mail suffix                                  | <input type="text"/>               |  |
| <input checked="" type="checkbox"/> Use SMTP Authentication | <input type="checkbox"/>           |  |
| Use SSL   | <input type="checkbox"/>           |  |
| SMTP Port   | <input type="text"/>               |  |
| Reply-To Address  | <input type="text"/>               |  |
| Charset   | <input type="text" value="UTF-8"/> |  |

Test configuration by sending test e-mail

4. Add the **SMTP server** and **SMTP Port** details. Use authentication if applicable. Add an e-mail address in the **Reply-To-Address** field in case you want the recipient to reply to the auto-generated emails.
5. You can test the **E-mail Notification** feature using the **Test configuration by sending test e-mail** option. Add the e-mail address to receive the test e-mail and click on the **Test Configuration** button. If the configuration is correct, the recipient will receive a test e-mail.

Test configuration by sending test e-mail

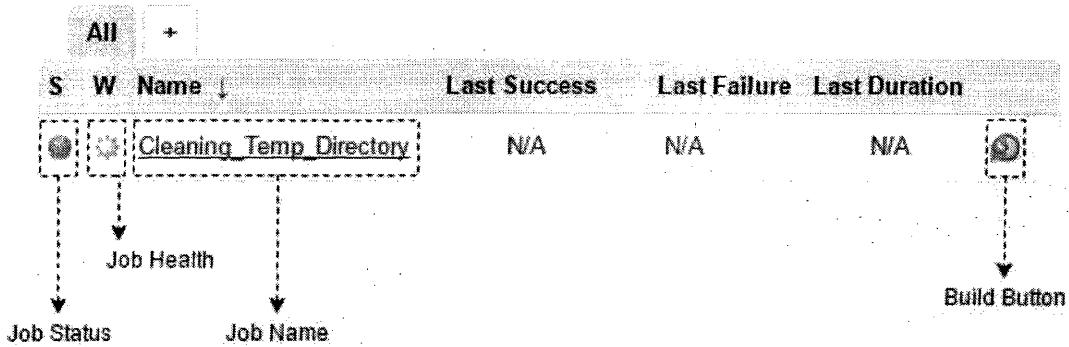
Test e-mail recipient

## Running a Jenkins job

We have successfully created a Jenkins job, now let's run it. The steps are as follows:

1. Go to the Jenkins Dashboard, either by clicking on the Jenkins logo on the top-left corner or by going to the link <http://localhost:8080/jenkins/>.

2. We should see our newly created Jenkins job **Cleaning\_Temp\_Directory**, listed on the page.



#### Note

Although our Jenkins job is scheduled to run at a specific time (anywhere between 23:00 and 23:59), clicking on the **Build** button will run it right away.

The **Job Status** icon represents the status of the most recent build. It can have the following colors that represent various states: **blue for Success**, **red for Failure**, and **gray for Disabled/Never Executed**.

The **Job Health** icon represents the success rate of a Jenkins job. **Sunny** represents 100 percent success rate, **Cloudy** represents 60 percent success rate, and **Raining** represents 40 percent success rate.

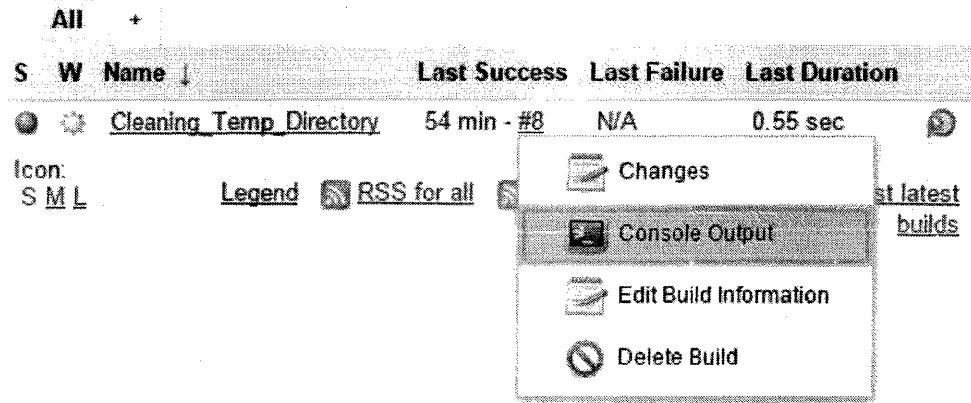
3. Click on the **Build** button to run the job. If everything is right, the job should run successfully.
4. Here's a screenshot of a successful Jenkins job. On my system, the Jenkins job took 0.55 seconds to execute. #8 represents the build number. It's 8 because I ran the Jenkins job eight times.

| All | + | S                       | W           | Name | Last Success | Last Failure | Last Duration |   |
|-----|---|-------------------------|-------------|------|--------------|--------------|---------------|---|
| ●   | ⌚ | Cleaning Temp Directory | 31 min - #8 | N/A  | 0.55 sec     | ⌚            | ⌚             | ⌚ |

## Jenkins build log

Now, let's see the build logs:

1. Hover the mouse over the build number (#8 in our case) and select **Console Output**.



2. The following screenshot is what you will see. It's the complete log of the Windows batch script.

## Console Output

```
Started by user anonymous
Building in workspace C:\Jenkins\jobs\Cleaning_Temp_Directory\workspace
[workspace] $ cmd /c call "C:\Program Files\Apache Software
Foundation\Tomcat 8.0\temp\hudson8071334469743261573.bat"

C:\Jenkins\jobs\Cleaning_Temp_Directory\workspace>REM Echo the temp
directory

C:\Jenkins\jobs\Cleaning_Temp_Directory\workspace>echo C:\WINDOWS\TEMP
C:\WINDOWS\TEMP

C:\Jenkins\jobs\Cleaning_Temp_Directory\workspace>REM Go to the temp
directory

C:\Jenkins\jobs\Cleaning_Temp_Directory\workspace>cd C:\WINDOWS\TEMP

C:\Windows\Temp>REM List all the files and folders inside the temp
directory

C:\Windows\Temp>dir /B
CProgram Files (x86)Opera32.0.1948.69opera_autoupdate.download.lock
CR_4CBB8.tmp
FAB367FF-8277-4D07-9B22-B4996BF16D49-Sigs
hsperfdata_DESKTOP-6NVBTVC$
jetty-0.0.0-8080-war--any-
jna--1137314184
Low
Microsoft Visual C++ 2010 x64 Redistributable Setup_10.0.30319
Microsoft Visual Studio Tools for Office Runtime 2010 Setup_10.0.50903
MpCmdRun.log
MPIInstrumentation
MpSigStub.log
MPTelemetrySubmit
MRT
opera autoupdate
ScheduledHeartbeat.log
SDIAG_d7e969f8-8db9-47f8-b669-59c628fe4224
```

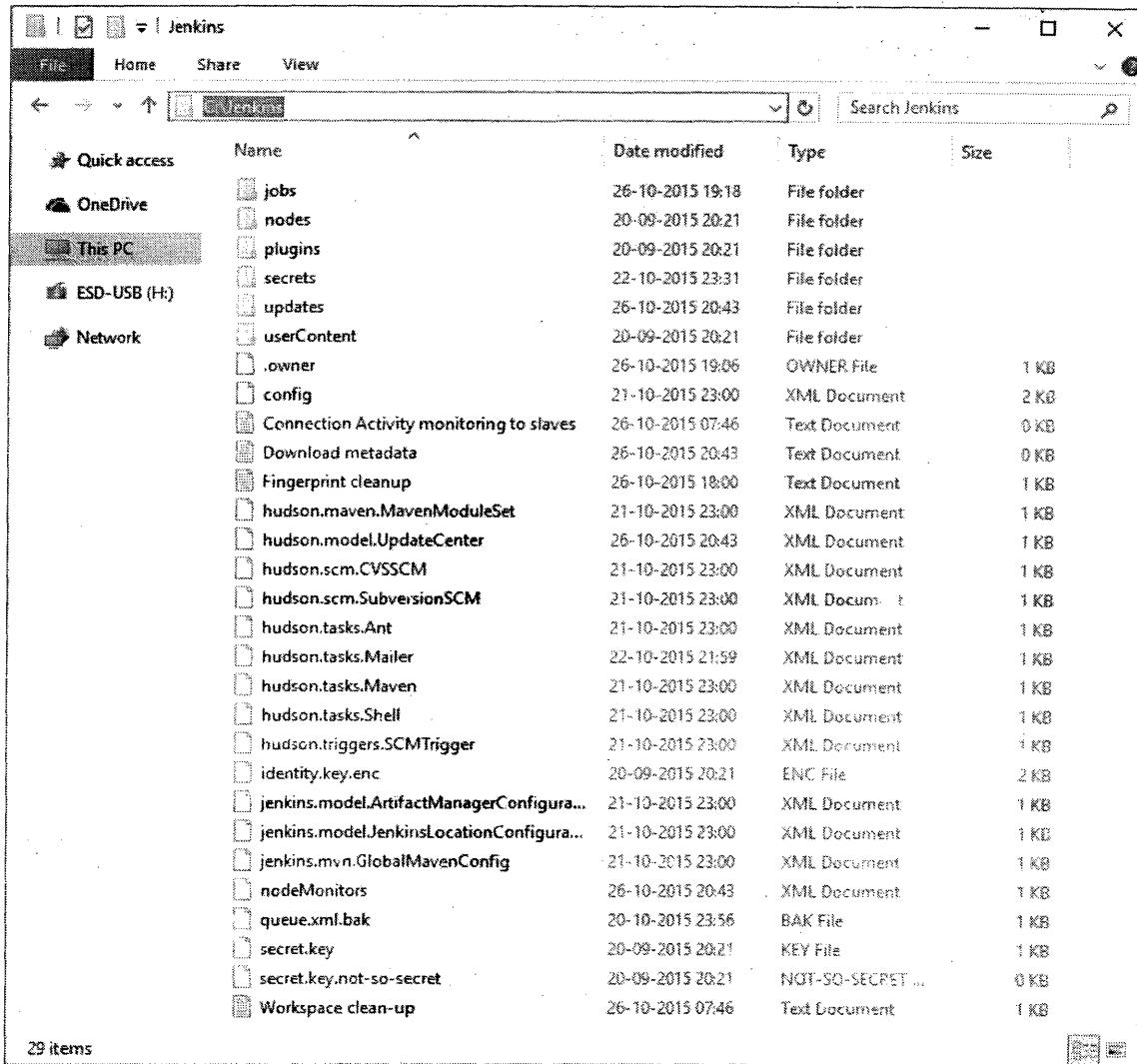
### Note

The build has run under an anonymous group; this is because we have not configured any users yet.

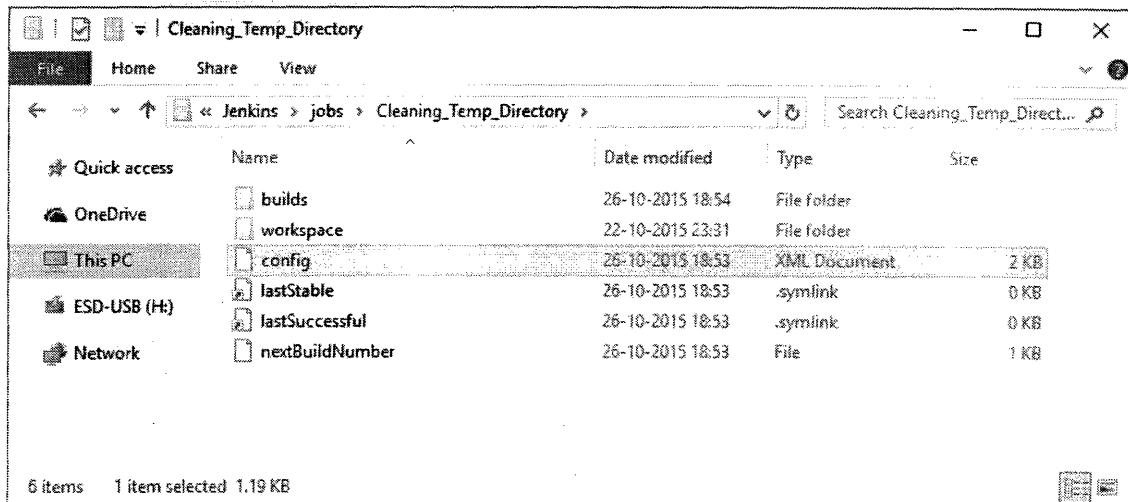
## Jenkins home directory

We saw how to create a simple Jenkins job. We also configured the SMTP server details for e-mail notifications. Now, let's see the location where all the data related to the Jenkins jobs gets stored. The steps are as follows:

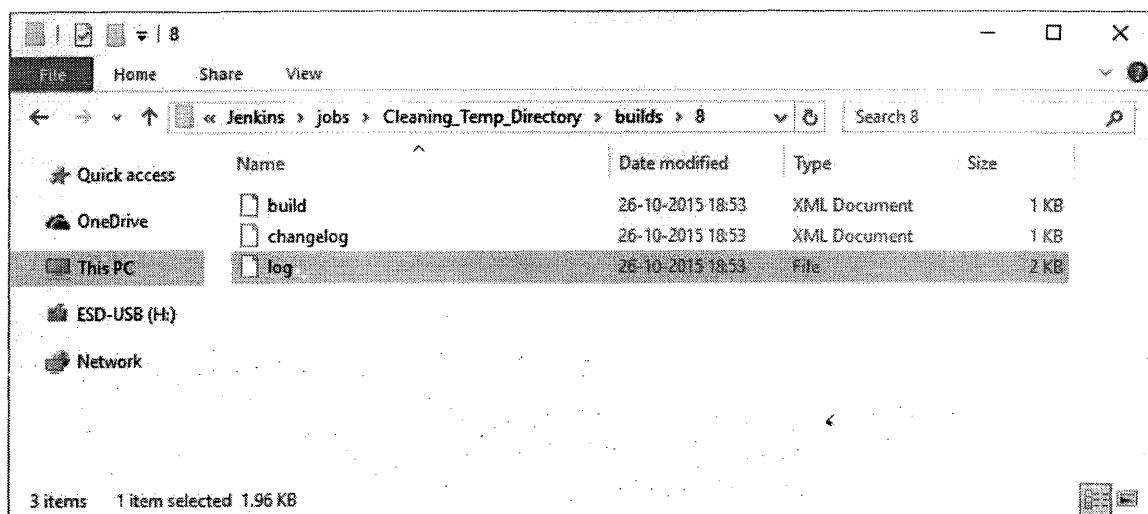
1. Go to C:\Jenkins\, our Jenkins home path. This is the place where all of the Jenkins configurations and metadata is stored, as shown in the following screenshot:



2. Now go to the folder named jobs\Cleaning\_Temp\_Directory. This is the place where all the information related to our Jenkins job is stored.
  - a. The config.xml file is an XML document that contains the Jenkins job configuration. This is something that should be backed up in case you want to restore a Jenkins job.
  - b. The workspace folder contains the output of a build. In our case, it's empty because the Jenkins job does not produce any output file or content.
  - c. The builds folder contains the log information of all the builds that have ran with respect to the respective Jenkins job.
3. This screenshot displays the config.xml file, the workspace folder, and the builds folder:



4. Now, go to the builds\8 directory, as shown in the next screenshot. The log file shown contains the same logs that we saw on the Jenkins Dashboard.



## Jenkins backup and restore

What happens if someone accidentally deletes important Jenkins configurations? Although this can be avoided using stringent user permissions, which we will see in the **User administration** section, nevertheless imagine a situation where the Jenkins server crashes or someone working on the Jenkins configuration wants to restore to a previous stable state of Jenkins. This leaves us with a few questions like, what to back up? When to back up? And how to backup?

From what we have learned so far, the entire Jenkins configuration is stored under the Jenkins home directory, which is C:\jenkins\ in our case. Everything related to Jenkins jobs like build logs, job configurations, and a workspace gets stored in the C:\jenkins\jobs folder.

Depending on the requirement, you can choose to backup only the configurations or choose to back up everything. The frequency of Jenkins backup can be anything depending on the project requirement. However, it's always good to back up Jenkins before we perform any configuration changes. Let's understand the Jenkins backup process by creating a Jenkins job.

### **Creating a Jenkins job to take periodic backup**

We will create a Jenkins job to take a complete backup of the whole Jenkins home directory. The steps are as follows:

1. You need the 7-Zip package installed on your machine. Download 7-Zip.exe from <http://www.7-zip.org/>.
2. From the Jenkins Dashboard, click on the **New Item** link.
3. Name your new Jenkins job **Jenkins\_Home\_Directory\_Backup**. Select the **Freestyle project** option and click on **OK**.
4. On the configuration page, add some description say, **Periodic Jenkins Home directory backup**.
5. Scroll down to the **Build Triggers** section and select the **Build periodically** option.
6. Add H 23 \* \* 7 in the **Schedule** section.
7. Scroll down to the **Build** section. Create a new build by selecting **Execute Windows batch command** from **Add build step**.
8. Add the following content inside the **Command** section:

```
REM Store the current date inside a variable named "DATE"
for /f %%i in ('date /t') do set DATE=%%i
REM 7-Zip command to create an archive
"C:\Program Files\7-Zip\7z.exe" a -t7z C:\Jenkins_Backup\Backup_%DATE%.7z
C:\Jenkins\*
```

9. The following screenshot displays the **Command** field in the **Execute Windows batch command** option:

## Build

The screenshot shows the Jenkins 'Build' section. Under the 'Execute Windows batch command' step, the 'Command' field contains the following script:

```
REM Store the current date inside a variable named"DATE"  
for /f %%i in ('date /t') do set DATE=%%i  
REM 7-Zip command to create an archive  
"C:\Program Files\7-Zip\7z.exe" a -t7z C:\Jenkins_Backup\Backup_%BUILD_NUMBER%_%DATE%.7z C:\Jenkins\*
```

Below the command field, there is a link 'See the list of available environment variables' and a 'Delete' button.

10. After adding the code inside the **Command** section, scroll to the end of the page and click on the **Save** button.
11. You will be taken to the jobs homepage
12. Click on the **Build Now** link to run the Jenkins job. Although it's scheduled to run every day around 23:00 hours, there is no harm in running a backup now.
13. Once you run the build, we can see its progress in the **Build History** section. Here, we can find all the builds that ran for the respective Jenkins job.
14. The build is successful once the buffering stops and the dot turns blue.
15. Once the build is complete, hover your mouse over the build number to get the menu items
16. Select the **Console Output** option. This will take you to the log page.
17. From Windows Explorer, go to the C:\Jenkins\_Backup directory. We can see that the backup archive has been created.

## Restoring a Jenkins backup

1. First, stop the Jenkins service running on the Apache Tomcat server.

2. To do this, go to the admin console at <http://localhost:8080/>.
3. Here's the Apache Tomcat server admin console:
4. From the admin console, click on the **Manager App** button.
5. You will be taken to the **Tomcat Web Application Manager** page.
6. Scroll down and under the **Applications** table, you should see the Jenkins service running along with the version number, as shown in the following screenshot:
7. Click on the **Stop** button to stop the running Jenkins instance. Once it has stopped, the Jenkins Dashboard will be inaccessible.
8. Then, simply unzip the desired backup archive into the Jenkins home directory, which is **C:\Jenkins\** in our case.
9. Once done, start the Jenkins service from the Apache Tomcat server's **Tomcat Web Application Manager** page by clicking on the **Start** button.

### Upgrading Jenkins

Jenkins has weekly releases that contain new features and bug fixes. There are also stable Jenkins releases called **Long Term Support (LTS)** releases. However, it's recommended that you always choose an LTS release for your Jenkins master server.

In this section, we will see how to upgrade Jenkins master server that is installed inside a container like Apache Tomcat and also a Jenkins standalone master server.

#### Note

It is recommended not to update Jenkins until and unless you need to. For example, upgrade Jenkins to an LTS release that contains a bug fix that you need desperately.

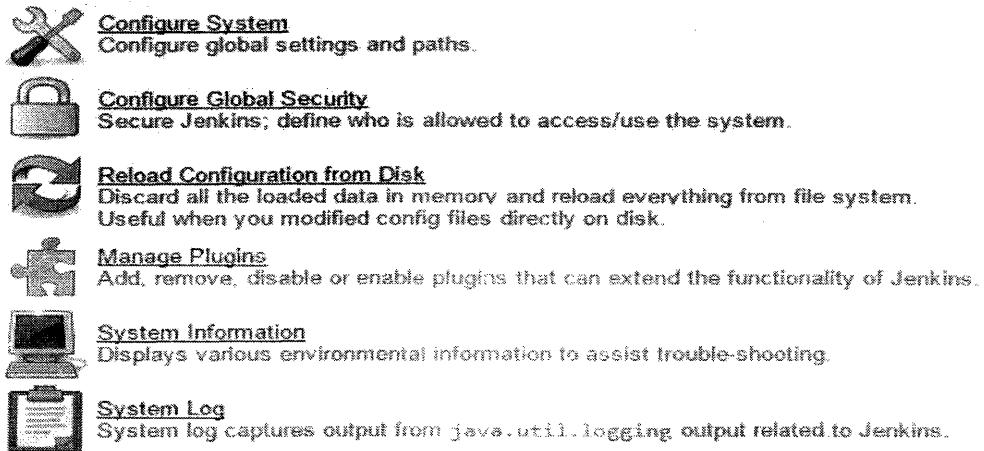
## Upgrading Jenkins running on the Tomcat server

The following are the steps to upgrade Jenkins running on the Tomcat server:

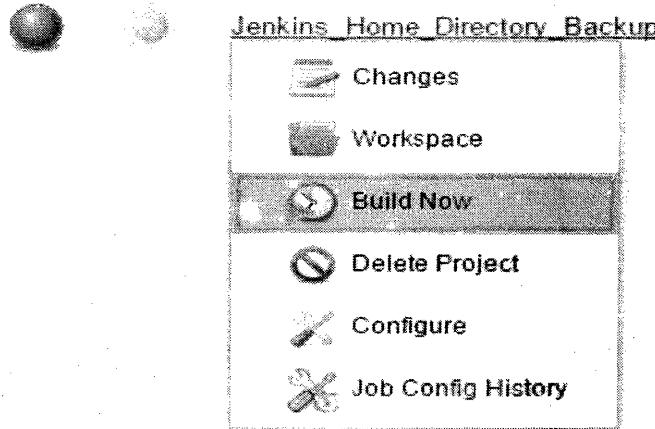
1. Download the latest jenkins.war file from <https://jenkins.io/download/>.
2. You can also download Jenkins from the **Manage Jenkins** page, which automatically list the most recent Jenkins release. However, this is not recommended.

## Manage Jenkins

 **New version of Jenkins (1.642.4) is available for download (changelog).**



3. From the Jenkins Dashboard, right-click on the Jenkins job **Jenkins\_Home\_Directory\_Backup** and select **Build Now**.



4. Our Jenkins server is running on Apache Tomcat server. Therefore, go to the location where the current jenkins.war file is running. In our case, it's C:\Program Files\Apache Software Foundation\Tomcat 8.0\webapps.
5. Stop the Jenkins service from the Apache Tomcat server admin console.
6. Now, replace the current jenkins.war file inside the webapps directory with the new jenkins.warfile that you have downloaded.
7. Start the Jenkins service from the Apache Tomcat server's **Tomcat Web Application Manager** page.
8. Go to the Jenkins Dashboard using the link <http://localhost:8080/jenkins>.
9. Check the Jenkins version on the Jenkins Dashboard.

### **Upgrading standalone Jenkins master running on Ubuntu**

#### **Upgrading to the latest stable version of Jenkins**

If you prefer to upgrade to a new stable version of Jenkins, then perform the following steps in sequence:

1. Check for admin privileges; the installation might ask for admin username and password.
2. Backup Jenkins before the upgrade.
3. Execute the following commands to update Jenkins to the latest stable version available:

```
 wget -q -O - https://jenkins-ci.org/debian-stable/jenkins-ci.org.key | sudo apt-key add -
 sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian-stable binary/ >
 /etc/apt/sources.list.d/jenkins.list'
 sudo apt-get update
 sudo apt-get install jenkins
```

### Upgrading Jenkins to a specific stable version

If you prefer to upgrade to a specific stable version of Jenkins, then perform the following steps in sequence. In the following steps, let's assume I want to update Jenkins to v1.580.3:

1. Check for admin privileges; the installation might ask for the admin username and password.
2. Backup Jenkins before the upgrade.
3. Execute the following commands to update Jenkins to the latest stable version available:

```
 wget -q -O - https://jenkins-ci.org/debian-stable/jenkins-ci.org.key | sudo apt-key add -
 sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian-stable binary/ >
 /etc/apt/sources.list.d/jenkins.list'
 sudo apt-get update
 sudo apt-get install jenkins=1.580.3
```

4. You might end up with the following error:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Version '1.580.3' for 'jenkins' was not found
```

5. In that case, run the following command to check the list of available versions:

```
apt-cache showpk
g jenkins
```

6. This will give the following output:

```
nikhil@nikhil-VirtualBox:~/Downloads$ apt-cache showpkg jenkins
Package: jenkins
Versions:
1.642.4 (/var/lib/apt/lists/pkg.jenkins-ci.org_debian-stable_binary_Packages)
Description Language:
File: /var/lib/apt/lists/pkg.jenkins-ci.org_debian-stable_binary_Packages
MD5: 483e336ea11484aaa0d84b76602263f7

1.596.3 (/var/lib/dpkg/status)
Description Language:
File: /var/lib/apt/lists/pkg.jenkins-ci.org_debian-stable_binary_Packages
MD5: 483e336ea11484aaa0d84b76602263f7

Reverse Depends:
libtap-formatter-junit-perl, jenkins
Dependencies:
1.642.4 - daemon (0 (null)) adduser (0 (null)) procps (0 (null)) psmisc (0 (null))
hudson (0 (null)) hudson:i386 (0 (null)) hudson (0 (null)) hudson:i386 (0 (null))
1.596.3 - daemon (0 (null)) adduser (0 (null)) psmisc (0 (null)) default-jre-head
) hudson (0 (null)) hudson:i386 (0 (null))
Provides:
1.642.4
1.596.3
Reverse Provides:
nikhil@nikhil-VirtualBox:~/Downloads$ sudo apt-get install jenkins=1.642.4
```

7. Notice the Jenkins version suggested; it's 1.642.4 and 1.596.3.
8. If you are ok with any of the available versions, select them and re-run the following command:

```
sudo apt-get install jenkins=1.596.3
```

9. You might get the following error:

```
nikhil@nikhil-VirtualBox:~/Downloads$ sudo apt-get install jenkins=1.596.3
Reading package lists... Done
Building dependency tree...
Reading state information... Done
jenkins is already the newest version.
You might want to run 'apt-get -f install' to correct these:
The following packages have unmet dependencies:
 jenkins : Depends: daemon but it is not going to be installed
E: Unmet dependencies.. Try 'apt-get -f install' with no packages (or specify a
nikhil@nikhil-VirtualBox:~/Downloads$ apt-get -f install
```

10. Run the following command:

**sudo apt-get -f install**

### 11. This will give the following output:

```
nikhil@nikhil-VirtualBox:~/Downloads$ sudo apt-get -f install
Reading package lists... Done
Building dependency tree...
Reading state information... Done
Correcting dependencies... Done
The following extra packages will be installed:
  daemon
the following NEW packages will be installed:
  daemon
0 upgraded, 1 newly installed, 0 to remove and 333 not upgraded.
1 not fully installed or removed.
Need to get 98.2 kB of archives.
After this operation, 287 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://in.archive.ubuntu.com/ubuntu/ trusty/universe daemon amd64 0.6.4-1 [98.2 kB]
Fetched 98.2 kB in 1s (69.9 kB/s)
Selecting previously unselected package daemon.
(Reading database ... 168557 files and directories currently installed.)
Preparing to unpack .../daemon_0.6.4-1_amd64.deb ...
Unpacking daemon (0.6.4-1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Setting up daemon (0.6.4-1) ...
Setting up jenkins (1.596.3) ...
 * Starting Jenkins Continuous Integration Server. jenkins
Processing triggers for ureadahead (0.100.0-16) ...
```

### 12. Now run the command to install Jenkins again:

**sudo apt-get install jenkins=1.596.3**

### 13. This should install Jenkins on your Ubuntu server.

## Script to upgrade Jenkins on Ubuntu

The shell script discussed in the following steps is capable of updating a standalone Jenkins master running on Ubuntu to the latest version of Jenkins available.

1. Open gedit and paste the following code inside it. Save the file as Jenkins\_Upgrade.sh.
2. Set the variables Backup\_Dir, Jenkins\_Home, and jenkinsURL accordingly.
3. Also, set the Jenkins web address accordingly:

```
#!/bin/bash

# pre-declared variables

Backup_Dir="/tmp/Jenkins_Backup"
Jenkins_Home="/usr/share/jenkins"
jenkinsURL="http://mirrors.jenkins-ci.org/war/latest/jenkins.war"

# Stopping Current Jenkins Service
sudo service jenkins stop

# Sleeping to wait for file cleanup
ping -q -c5 http://localhost:8080 >/dev/null

# clean files
sudo cp -f $Jenkins_Home/jenkins.war $Backup_Dir/jenkins.war.bak
sudo rm -rf $Jenkins_Home/jenkins.war

# Download new files
cd $Jenkins_Home
sudo wget "$jenkinsURL"

# Starting new upgraded Jenkins
sudo service jenkins start

# Sleeping to wait for service startup
ping -q -c5 http://localhost:8080 > /dev/null
```

4. Try running the shell script with a user having sudo access.

### **Managing Jenkins plugins**

Jenkins derives most of its power from plugins. As discussed in the previous chapter, every plugin that gets installed inside Jenkins manifests itself as a parameter, either inside Jenkins system configurations or inside a Jenkins job. Let's see where and how to install plugins.

In the current section, we will see how to manage plugins using the Jenkins plugins manager. We will also see how to install and configure plugins.

## The Jenkins Plugins Manager

The Jenkins Plugin Manager section is a place to install, uninstall, and upgrade Jenkins plugins. Let us understand it in detail:

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link.
2. From the **Manage Jenkins** page, click on the **Manage Plugins** link.
3. The following screenshot is what you see when you land on the **Jenkins Plugin Manager** page.

The screenshot shows the Jenkins Plugin Manager interface. At the top, there are two tabs: 'Updates' (which is selected) and 'Available'. Below these tabs is a table listing various Jenkins plugins. The table has columns for Name, Version, and Installed. A 'Filter' input field is located at the top right of the table area. At the bottom of the table, there are three buttons: 'Download now and install after restart', 'Update information obtained: 1 day 7 hr ago', and 'Check now'.

| Name                                 | Version | Installed  |
|--------------------------------------|---------|------------|
| Credentials Plugin                   | 1.24    | 1.18       |
| CVS Plug-in                          | 2.12    | 2.11       |
| Javadoc Plugin                       | 1.3     | 1.1        |
| JUnit Plugin                         | 1.9     | 1.2-beta-4 |
| Mailer Plugin                        | 1.16    | 1.11       |
| Matrix Authorization Strategy Plugin | 1.2     | 1.1        |
| Matrix Project Plugin                | 1.6     | 1.4.1      |
| Maven Integration plugin             | 2.12.1  | 2.7.1      |
| OWASP Marker Formatter Plugin        |         |            |

4. The following four tabs are displayed in the screenshot:
  - The **Updates** tab lists updates available for the plugins installed on the current Jenkins instance.
  - The **Available** tab contains the list of all the plugins available for Jenkins across the Jenkins community.

- The **Installed** tab lists all the plugins currently installed on the current Jenkins instance.
- The **Advanced** tab is used to configure Internet settings and also to update Jenkins plugins manually.

5. Let's see the **Advanced** tab in detail by clicking on it.
6. Right at the beginning, you will see a section named **HTTP Proxy Configuration**. Here, you can specify the HTTP proxy server details.
7. Provide the proxy details pertaining to your organization, or leave these fields empty if your Jenkins server is not behind a firewall.

The screenshot shows the Jenkins Advanced configuration interface. At the top, there are four tabs: Updates, Available, Installed, and Advanced, with the Advanced tab being the active one. Below the tabs, the title "HTTP Proxy Configuration" is displayed. The form contains the following fields:

- Server:** A text input field with a help icon (question mark) to its right.
- Port:** A text input field with a help icon to its right.
- User name:** A text input field with a help icon to its right.
- Password:** A text input field with a help icon to its right.
- No Proxy Host:** A text input field with a help icon to its right.
- Test URL:** A text input field with a help icon to its right.
- Validate Proxy:** A large, prominent button.
- Submit:** A dark button at the bottom left.

8. Just below the **HTTP Proxy Configuration** section, you will see the **Upload Plugin** section. It provides the facility to upload and install your own Jenkins plugin.

## **Upload Plugin**

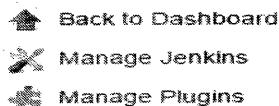
You can upload a .hpi file to install a plugin from outside the central plugin repository.

File:  No file chosen

### **Installing a Jenkins plugin to take periodic backup**

Let's try installing a plugin. In the previous sections, we saw a Jenkins job that creates a backup. Let's now install a plugin to do the same:

1. On the Jenkins **Plugin Manager** home page, go to the **Available** tab.
2. In the **Filter** field, type **Periodic Backup**.
3. Tick the checkbox beside the **Periodic Backup** plugin and click on **Install without restart**. This will download the plugin and then install it.
4. Jenkins immediately connects to the online plugin repository and starts downloading and installing the plugin, as shown in the following screenshot:



## **Installing Plugins/Upgrades**

### **Preparation**

- Checking internet connectivity
- Checking update center connectivity
- Success

### **Periodic Backup**

 Downloaded Successfully. Will be activated during the next boot

### **Periodic Backup**

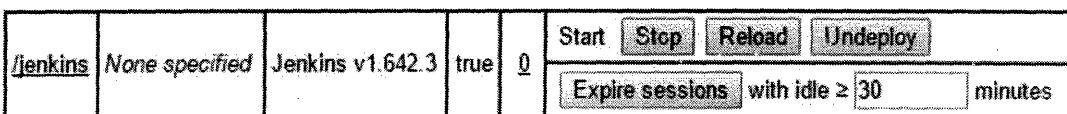
 Success

 [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

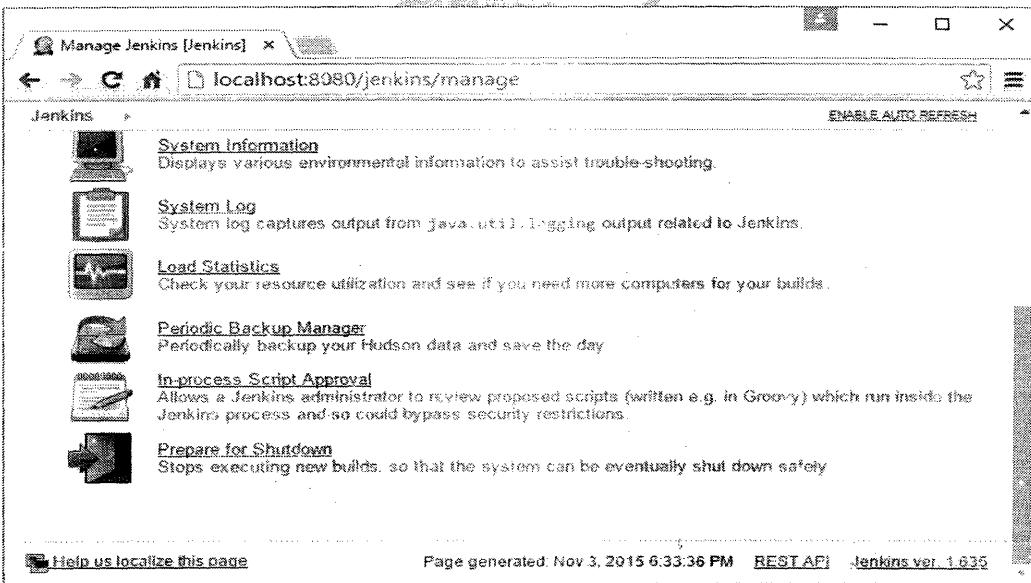
5. For the plugin to work, we need to restart the Jenkins server.
6. To restart Jenkins, go to the Apache Tomcat server home page and click on the **Manage App** button.
7. From the **Tomcat Web Application Manager** page, restart Jenkins by first clicking on the **Stop** button. Once Jenkins stops successfully, click on the **Start** button.



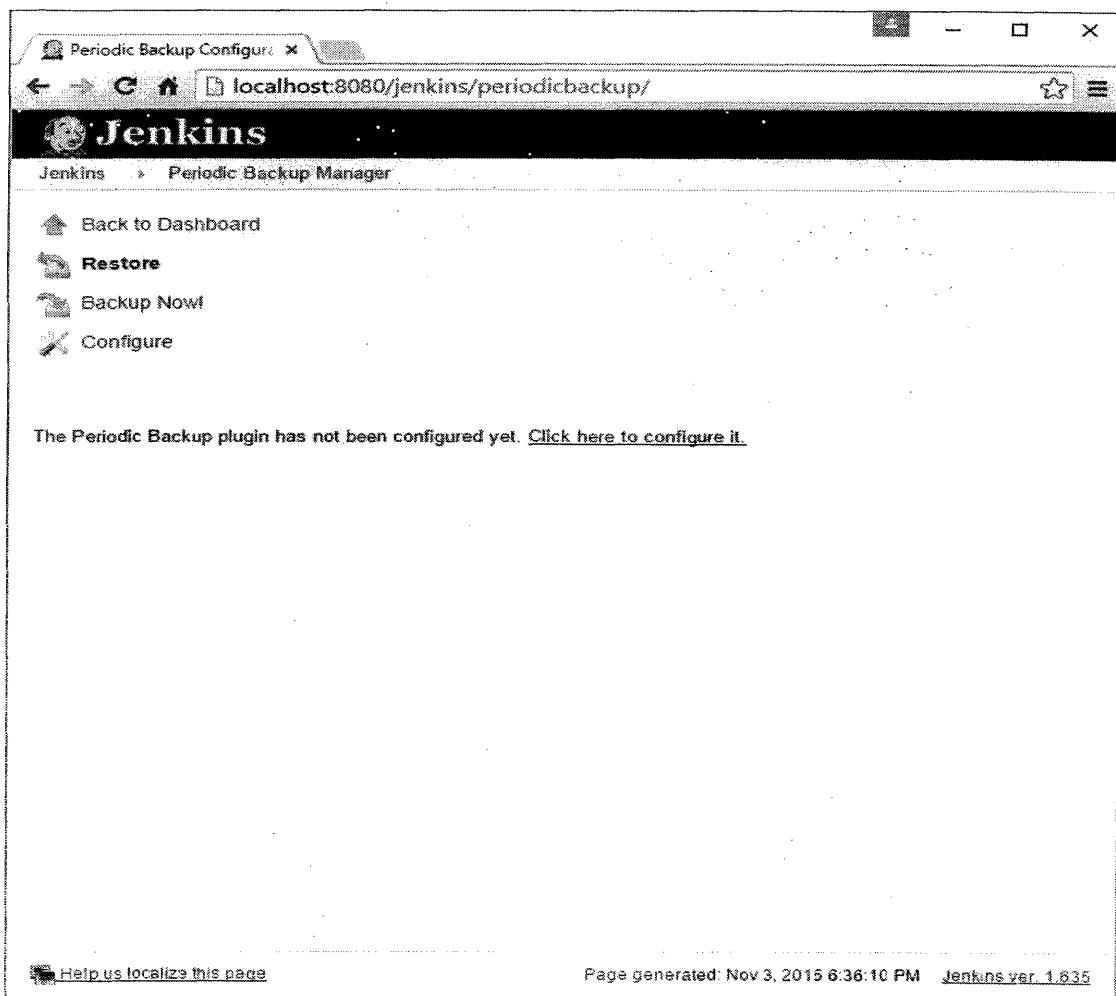
### Configuring the periodic backup plugin

We have successfully installed the periodic backup plugin. Now, let's configure it:

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link.
2. On the **Manage Jenkins** page, you will see the **Periodic Backup Manager** link.
3. Clicking on the **Periodic Backup Manager** link will take you to the **Periodic Backup Manager** page as shown in the following screenshot:



4. Clicking on **Backup Now!** creates a backup. However, it won't work presently as we have not configured the backup plugin.
5. The **Configure** link will allow you to configure the plugin.



6. Click on the **Configure** link and you will see many options to configure your backup plugin:
  - **Temporary Directory:** This is where Jenkins will temporarily expand the archive files while restoring any backup. As you can see, I used an environment variable %temp%, but you can give any path on the machine.

- **Backup schedule (cron):** This is the schedule that you want your backup to follow. I used H 23 \* \* 7, which is every Sunday anywhere between 23:00 to 23:59 hours throughout the year.
- **Maximum backups in location:** This is the total number of backups you want to store in a particular backup location. Does that mean we can have more than one backup location? Yes. We will see more on this soon.
- **Store no older than (days):** This ensures any backup in any location which is older than the number of days specified is deleted automatically.

The screenshot shows the Jenkins backup configuration interface. It includes fields for Root Directory (C:\Jenkins), Temporary Directory (%temp%), and Backup schedule (cron) (H 23 \* \* 7). A message says "This cron is OK". There is a "Validate cron syntax" button. Below this, there are fields for Maximum backups in location (5) and Store no older than (days) (30).

Root Directory: C:\Jenkins  
Temporary Directory: %temp%  
Backup schedule (cron): H 23 \* \* 7  
This cron is OK  
Validate cron syntax  
Maximum backups in location: 5  
Store no older than (days): 30

7. Scroll down to the **File Management Strategy** section. You will see the options to choose from **FullBackup** and **ConfigOnly**. Choose **FullBackup**.

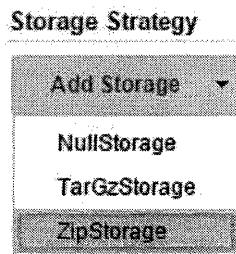
### File Management Strategy

- ConfigOnly
- FullBackup

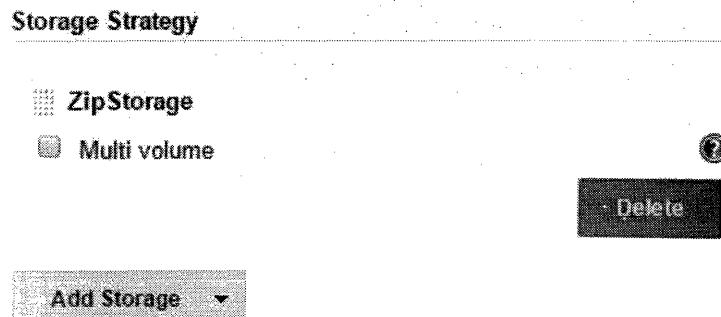
#### Note

**FullBackup** takes a backup of the whole Jenkins home directory. **ConfigOnly** takes only the backup of configurations and excludes the builds and logs.

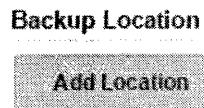
8. In the following screenshot, you will see **Storage Strategy** section. Click on it and you will have options to choose from .zip, .targz, and **NullStorage**. I chose the .zip archive.



9. Clicking on the **ZipStorage** strategy provides us an option to select the **Multi volume** zip file, that is, one huge, single zip file split into many.



10. Just below **Storage Strategy**, you can see the **Backup Location** section where you can add as many backup locations as you want.



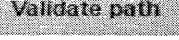
11. In my example, I added two backup locations, C:\Jenkins\_Backup and C:\Jenkins\_Backup2 respectively.

12. As you can see from the following screenshot, I enabled both the locations.

**Backup Location**

**LocalDirectory**  
Backup directory path   

Enable this location  

**LocalDirectory**  
Backup directory path   

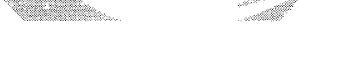
Enable this location  

**Add Location** 



13. Once done, click on the **Save** button.



### User administration



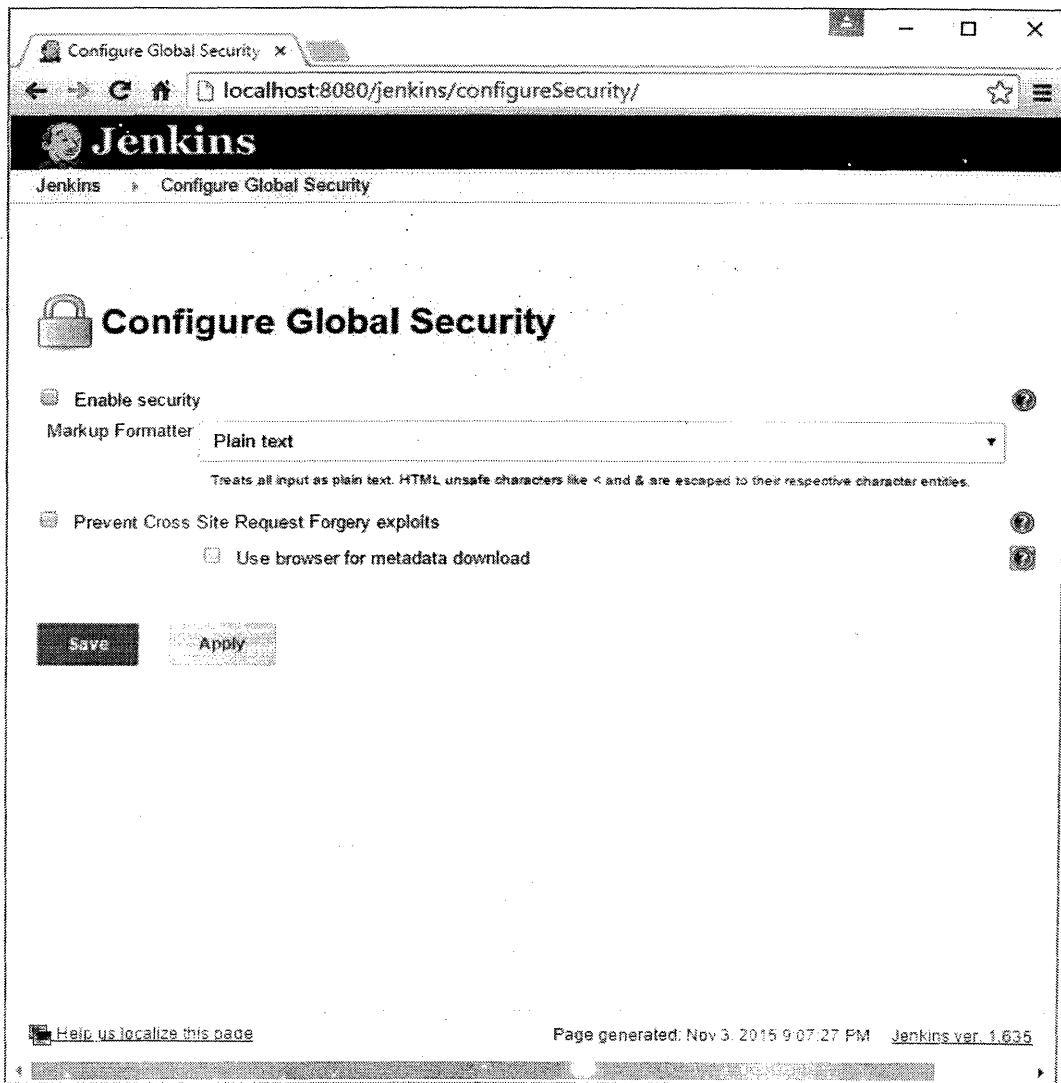
So far, all our Jenkins Jobs were running anonymously under an unidentified user. All the configurations that we did inside Jenkins were also done anonymously. But as we know, this is not how things should be. There needs to be a mechanism to manage users and define their privileges. Let's see what Jenkins has to offer in the area of user administration.

### Enabling global security on Jenkins

The **Configure Global Security** section is the place where you get various options to secure Jenkins. Let see it in detail.

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link.

2. From the **Manage Jenkins** page, click on the **Configure Global Security** link.
3. The following screenshot shows what the **Configure Global Security** page looks like:



4. Click on the **Enable security** checkbox and a new set of options will be available to configure.
5. Leave the **TCP port for JNLP slave agents** option as it is (**Random**).
6. Leave the **Disable remember me** option unchecked.



## Configure Global Security

Enable security

TCP port for JNLP slave agents  Fixed:   Random  Disable

Disable remember me

7. Go to the **Security Realm** subsection which is under the **Access Control** section. In our example, we will use the **Jenkins' own user database** option to manage users and permissions.
8. Select the **Jenkins' own user database** and you will get another option, which allows users to sign up. This is shown in the following screenshot:

### Access Control

### Security Realm

Delegate to servlet container

Jenkins' own user database

Allow users to sign up

LDAP

9. Come down to the **Authorization** section, and you will see the following options:

### Authorization

Anyone can do anything

Legacy mode

Logged-in users can do anything

Matrix-based security

Project-based Matrix Authorization Strategy

10. Choose the **Matrix-based security** option.

11. The following illustration is partial, that means there is more towards the right side.

12. To add users, enter the user names in the **User/group** to add a field, and click the **Add** button. For now, do not add any users.

**Matrix-based security**

| User/group | Overall                  |                          |                          |                          |                          |                          |                          |                          | Credentials              |                          |                          |                          |                          |                          |                          |                          |
|------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
|            | Administer               | Configure                | Update                   | Center                   | Read                     | Run Scripts              | Upload Plugins           | Create                   | Delete                   | Manage Domains           | Update View              | View                     | Administer               | Configure                | Update                   | Center                   |
| Anonymous  | <input type="checkbox"/> |

User/group to add:  **Add**

**Note**

In a **Matrix-based security** setting, all the **Users/Groups** are listed across rows and all the Jenkins tasks are listed across columns. It's a matrix of users and tasks. This matrix makes it possible to configure permissions at the task level for each user.

13. Select all the checkboxes for the **Anonymous** user. By doing this, we are giving the **Anonymous** user admin privileges.

| User/group | Overall                             |                                     |                                     |                                     |                                     |                                     |                                     |  |
|------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|--|
|            | Administer                          | Configure                           | Update                              | Center                              | Read                                | Run Scripts                         | Upload Plugins                      |  |
| Anonymous  | <input checked="" type="checkbox"/> |  |

| Credentials                         |                                     |                                     |                                     |                                     |                                     |                                     |                                     | Slave                               |                                     |                                     |                                     |                                     |                                     |  |  |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|--|--|
| Create                              | Delete                              | Manage Domains                      | Update View                         | Build                               | Configure                           | Connect                             | Create                              | Delete                              | Disconnect                          | Job                                 | Run                                 | View                                | SCM                                 |  |  |
| <input checked="" type="checkbox"/> |  |  |

| Job                                 |                                     |                                     |                                     |                                     |                                     |                                     |                                     | Run                                 |                                     |                                     |                                     |                                     |                                     |                                     |                                     | View                                |                                     |                                     |  |  |  |  |  | SCM |  |  |  |  |  |  |  |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|--|--|--|--|--|-----|--|--|--|--|--|--|--|
| Build                               | Cancel                              | Configure                           | Create                              | Delete                              | Discover                            | Read                                | Workspace                           | Delete                              | Update                              | Configure                           | Create                              | Delete                              | Read Tag                            | SCM                                 | Job                                 | Run                                 | View                                | SCM                                 |  |  |  |  |  |     |  |  |  |  |  |  |  |
| <input checked="" type="checkbox"/> |  |  |  |  |  |     |  |  |  |  |  |  |  |

14. Click on the **Save** button at the bottom of the page once done.

## Creating other users

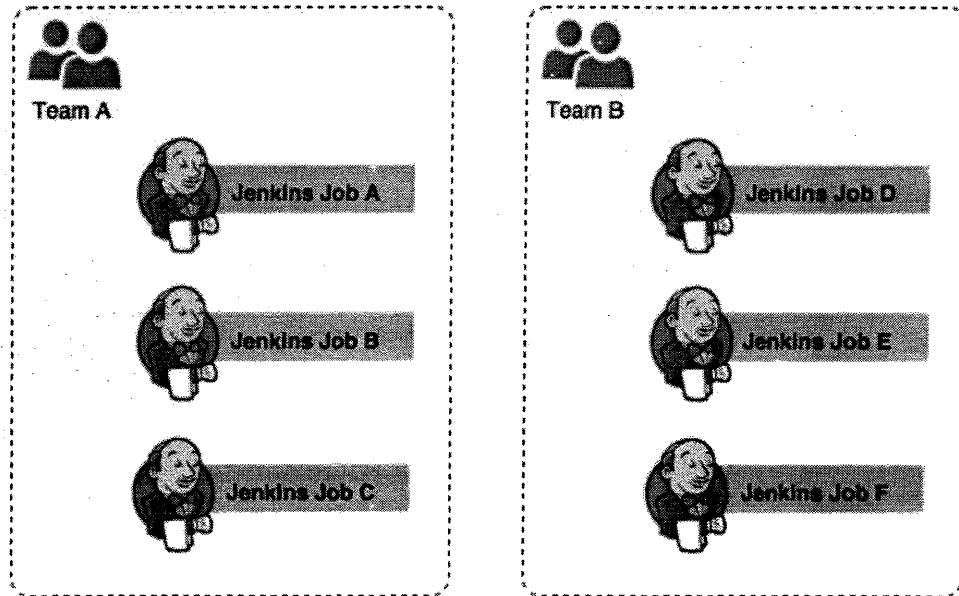
Users can always sign up and create their account in Jenkins using the **sign up** link at the top-right corner. All such users by default get all the privileges of an anonymous group.

1. You can try creating as many accounts as you want and see all those come under the anonymous category by default.
2. To see the list of Jenkins users, from the Jenkins Dashboard, click on the **People** link present at the top-left section.
3. All the users are listed on the **People** page, as shown in the following screenshot:
4. To give permissions to our newly created user, log into Jenkins as the admin user.
5. From the **Manage Jenkins** page, go to the **Configure Global Security** page.
6. On the **Configure Global Security** page, scroll down to the **Authorization** section.
7. Inside the **User/group to add** field, add the username that has signed up on the Jenkins master and click on the **Add** button. In my example, I added a user <username> that I recently created.
8. Once added, give the new user permissions to **Build**, **Cancel**, **Workspace**, and **Read** a Jenkins jobs, as shown in the following screenshot:
9. Click on the **Save** button at the end of the page to save the settings.
10. Log in as the new user and you will notice that you can only execute builds, but you cannot change the job configuration or the Jenkins system settings.

## Using the Project-based Matrix Authorization Strategy

In the previous section, we saw the **Matrix-based security** authorization feature which gave us a good amount of control over the users and permissions. However, imagine a situation where your Jenkins master server has grown to a point, where it contains multiple projects (software projects), hundreds of Jenkins jobs and many users. You

want the users to have permissions only on the jobs they use. In such a case, we need the **Project-based Matrix Authorization Strategy** feature.



Let's learn to configure the **Project-based Matrix Authorization Strategy** feature:

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link.
2. On the **Manage Jenkins** page, click on the **Configure Global Security** link.
3. Here's what our current configuration looks like:

| User/group | Overall                             |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     | Credentials                         |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                          |        |
|------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|--------------------------|--------|
|            | Administer                          | Configure                           | UpdateCenter                        | Read                                | Run                                 | Scripts                             | Upload                              | Plugins                             | Create                              | Delete                              | ManageDomains                       | UpdateView                          | Administer                          | Configure                           | UpdateCenter                        | Read                                | Run                                 | Jobs                                | Manage                              | UpdateView                          | Administer                          | Configure                           | UpdateCenter                        | Read                                | Run                                 | Jobs                     | Manage |
| admin      | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |                          |        |
| Anonymous  | <input type="checkbox"/>            | <input checked="" 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"/> |        |
| nikhil     | <input type="checkbox"/>            | <input type="checkbox"/> |        |
| Slave      |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                          |        |
| Build      | <input checked="" type="checkbox"/> |                          |        |
| Configure  | <input checked="" type="checkbox"/> |                          |        |
| Connect    | <input checked="" type="checkbox"/> |                          |        |
| Create     | <input checked="" type="checkbox"/> |                          |        |
| Delete     | <input checked="" type="checkbox"/> |                          |        |
| Disconnect | <input type="checkbox"/>            | <input type="checkbox"/> |        |
| Discover   | <input type="checkbox"/>            | <input type="checkbox"/> |        |
| Read       | <input checked="" type="checkbox"/> |                          |        |
| Run        | <input checked="" type="checkbox"/> |                          |        |
| View       |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                          |        |
| Configure  | <input checked="" type="checkbox"/> |                          |        |
| Create     | <input checked="" type="checkbox"/> |                          |        |
| Delete     | <input checked="" type="checkbox"/> |                          |        |
| Read       | <input checked="" type="checkbox"/> |                          |        |
| Tag        | <input checked="" type="checkbox"/> |                          |        |
| SCM        |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                          |        |
| Configure  | <input checked="" type="checkbox"/> |                          |        |
| Create     | <input checked="" type="checkbox"/> |                          |        |
| Delete     | <input checked="" type="checkbox"/> |                          |        |
| Read       | <input checked="" type="checkbox"/> |                          |        |
| Tag        | <input checked="" type="checkbox"/> |                          |        |

4. Select the **Project-based Matrix Authorization Strategy** option.

Authorization

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

| User/group | Overall | Administer               | Configure                | UpdateCenter             | Read                     | Run                                 | Scripts                  | Upload                   | Plugins                  |
|------------|---------|--------------------------|--------------------------|--------------------------|--------------------------|-------------------------------------|--------------------------|--------------------------|--------------------------|
| Anonymous  |         | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

| Credentials   | Slave                    |
|---------------|--------------------------|
| Create        | <input type="checkbox"/> |
| Delete        | <input type="checkbox"/> |
| ManageDomains | <input type="checkbox"/> |
| UpdateView    | <input type="checkbox"/> |
| Build         | <input type="checkbox"/> |
| Configure     | <input type="checkbox"/> |
| Connect       | <input type="checkbox"/> |
| Create        | <input type="checkbox"/> |
| Delete        | <input type="checkbox"/> |
| Disconnect    | <input type="checkbox"/> |

| Job       | Run                      |
|-----------|--------------------------|
| Build     | <input type="checkbox"/> |
| Cancel    | <input type="checkbox"/> |
| Configure | <input type="checkbox"/> |
| Create    | <input type="checkbox"/> |
| Delete    | <input type="checkbox"/> |
| Discover  | <input type="checkbox"/> |
| Read      | <input type="checkbox"/> |
| Workspace | <input type="checkbox"/> |
| Delete    | <input type="checkbox"/> |
| Update    | <input type="checkbox"/> |

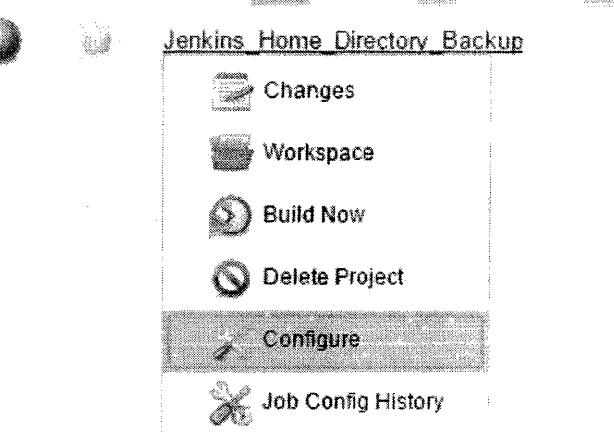
| View      | SCM                                 |
|-----------|-------------------------------------|
| Configure | <input type="checkbox"/>            |
| Create    | <input type="checkbox"/>            |
| Delete    | <input type="checkbox"/>            |
| Read      | <input type="checkbox"/>            |
| Tag       | <input checked="" type="checkbox"/> |

User/group to add:  **Add**

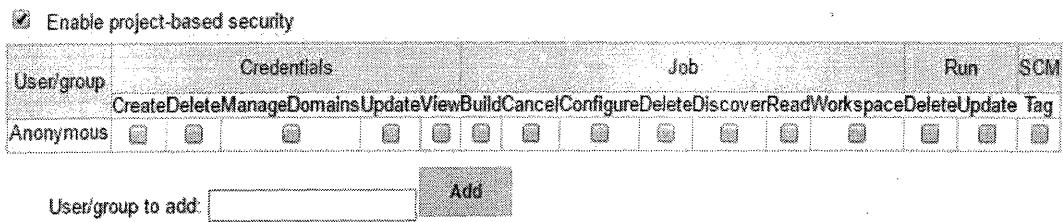
5. Inside the **User/group to add** field, add the username that has signed up on the Jenkins master and click on the **Add** button. Do not forget to add the admin user.
6. The output should look like the following screenshot:

| User/group                          | Overall                             |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     | Credentials                         |                                     |                                     |                                     |                                     |                                     |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
|                                     | Administer                          | Configure                           | UpdateCenter                        | Read                                | RunScripts                          | Upload                              | Plugins                             | Create                              | Delete                              | ManageDomains                       | Update                              | View                                |                                     |                                     |                                     |                                     |
| Anonymous                           | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            | <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"/>            |
| admin                               | <input checked="" type="checkbox"/> |
| Slave                               |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     |                                     | Job                                 |                                     |                                     |                                     |                                     |                                     |                                     |
| Build                               | <input type="checkbox"/>            | Build                               | <input type="checkbox"/>            |
| Configure                           | <input type="checkbox"/>            | Cancel                              | <input type="checkbox"/>            |
| Connect                             | <input type="checkbox"/>            | Discover                            | <input type="checkbox"/>            |
| Create                              | <input type="checkbox"/>            | Read                                | <input type="checkbox"/>            |
| Delete                              | <input type="checkbox"/>            | Workspace                           | <input type="checkbox"/>            |
| Run                                 | <input type="checkbox"/>            | Run                                 | <input type="checkbox"/>            |
| View                                | <input type="checkbox"/>            | View                                | <input type="checkbox"/>            |
| SCM                                 | <input type="checkbox"/>            | SCM                                 | <input type="checkbox"/>            |
| Delete                              | <input type="checkbox"/>            | Configure                           | <input type="checkbox"/>            |
| Update                              | <input type="checkbox"/>            | Create                              | <input type="checkbox"/>            |
| Configure                           | <input type="checkbox"/>            | Delete                              | <input type="checkbox"/>            |
| Create                              | <input type="checkbox"/>            | Discover                            | <input type="checkbox"/>            |
| Delete                              | <input type="checkbox"/>            | Read                                | <input type="checkbox"/>            |
| Read Tag                            | <input type="checkbox"/>            | Workspace                           | <input type="checkbox"/>            |
| <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> |

- Click on the **Save** button at the end of the page to save the configuration.
- From the Jenkins Dashboard, right-click on any of the Jenkins jobs and select **Configure**.

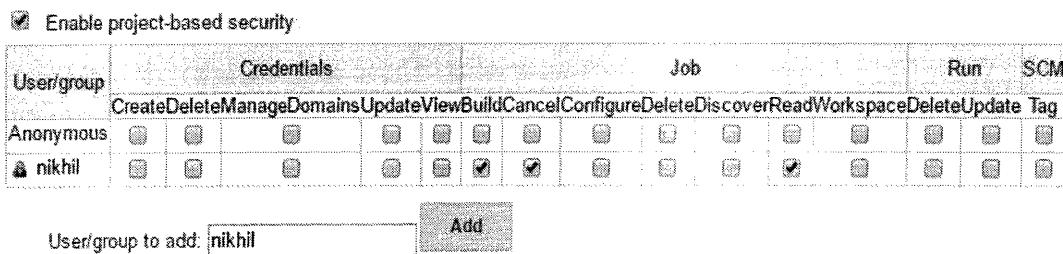


- On the job's configuration page, select the newly available option **Enable project-based security**, which is right at the beginning.



10. Now, inside the **User/group to add** field, add the username that you want to give access to the current job.

11. As shown in the following screenshot, I added a user nikhil who has the permission to build the current job.



12. Once done, click on the **Save** button at the end of the page.

#### 4. Continuous Integration Using Jenkins – Part I

##### Jenkins Continuous Integration Design

I have used a new term here: **Continuous Integration Design**. Almost every organization creates one before they even begin to explore the CI and DevOps tools. In the current section, we will go through a very general Continuous Integration Design.

Continuous Integration includes not only Jenkins or any other similar CI tool for that matter, but it also deals with how you version control your code, the branching strategy you follow, and so on. If you are feeling that we are overlapping with **software configuration management**, then you are right.

Various organizations may follow different kinds of strategies to achieve Continuous Integration. Since, it all depends on the project requirements and type.

### **The branching strategy**

It's always good to have a branching strategy. Branching helps you organize your code. It is a way to isolate your working code from the code that is under development. In our Continuous Integration Design, we will start with three types of branches:

- Master branch
- Integration branch
- Feature branch



#### **Master branch**

You can also call it the **production branch**. It holds the working copy of the code that has been delivered. The code on this branch has passed all the testing stages. No development happens on this branch.

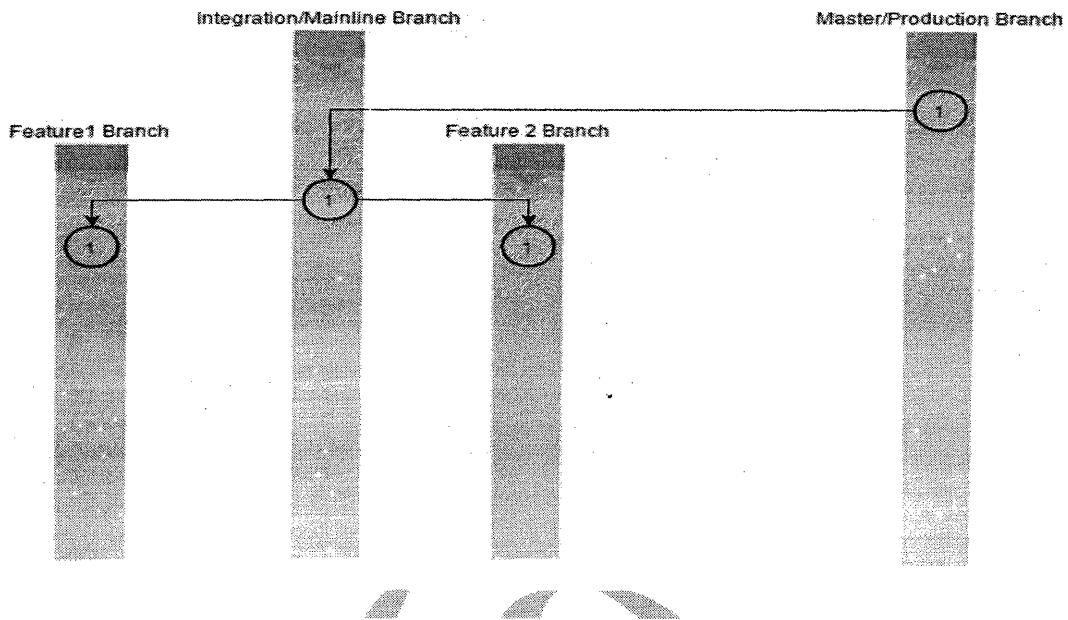
#### **Integration branch**

The integration branch is also known as the **mainline branch**. This is where all the features are integrated, built, and tested for integration issues. Again, no development happens here. However, developers can create feature branches out of the integration branch and work on them.

#### **Feature branch**

Lastly we have the feature branch. This is where the actual development takes place. We can have multiple feature branches spanning out of the integration branch.

The following image shows a typical branching strategy that we will use as part of our Continuous Integration Design. We will create two feature branches spanning out from the integration/mainline branch, which itself spans out from the master branch.



- A successful commit (code check-in) on the feature branch will go through a build and unit test phase. If the code passes these phases successfully, it is merged to the integration branch.
- A commit on the integration branch (a merge will create a commit) will go through a build, static code analysis, and integration testing phase. If the code passes these phases successfully, the resultant package is uploaded to Artifactory (binary repository).

### The Continuous Integration pipeline

We are now at the heart of the Continuous Integration Design. We will create two pipelines in Jenkins, which are as follows:

- Pipeline to poll the feature branch
- Pipeline to poll the integration branch

These two pipelines work in sequence and, as a whole, form the Continuous Integration pipeline. Their purpose is to automate the process of continuously building, testing (unit test

and integration test), and integrating the code changes. Reporting failure/success happens at every step.

Let's discuss these pipelines and their constituents in detail.

### **Jenkins pipeline to poll the feature branch**

The Jenkins pipeline to poll the feature branch is coupled with the feature branch. Whenever a developer commits something on the feature branch, the pipeline gets activated. It contains two Jenkins jobs that are as follows:

#### **Jenkins job 1**

The first Jenkins Job in the pipeline performs the following tasks:

- It polls the feature branch for changes on a regular interval
- It performs a build on the modified code
- It executes the unit tests

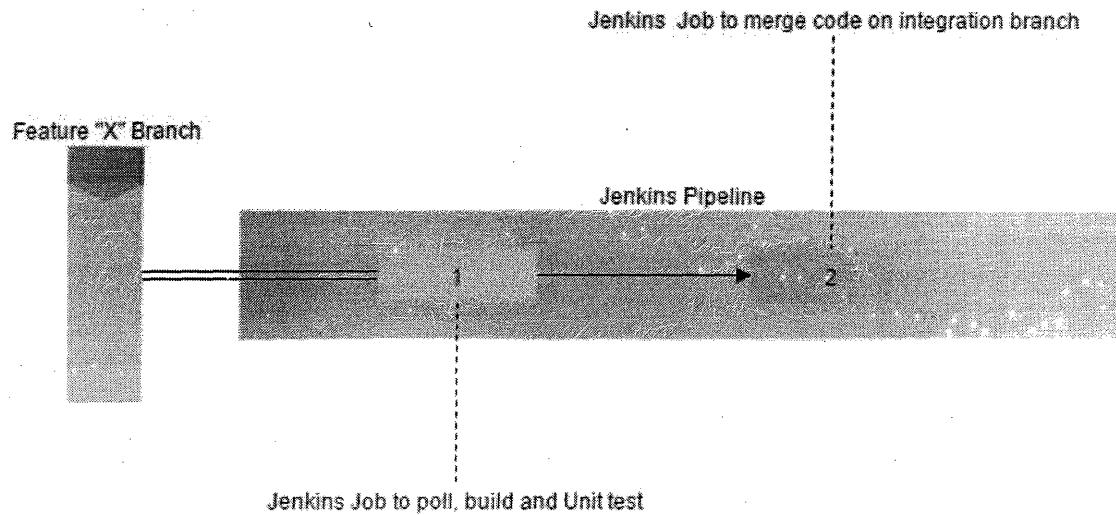
#### **Jenkins job 2**

The second Jenkins Job in the pipeline performs the following task:

It merges the successfully built and tested code onto the integration branch

If this is the first time you are seeing a Jenkins job performing automated merges, then you are not alone. The reason is such automation is mostly done across projects that are very mature in using Continuous Integration and where almost everything is automated and configured well.

The following figure depicts the pipeline to poll the feature branch:



### Jenkins pipeline to poll the integration branch

This Jenkins pipeline is coupled with the integration branch. Whenever there is a new commit on the integration branch, the pipeline gets activated. It contains two Jenkins jobs that perform the following tasks.

#### Jenkins job 1

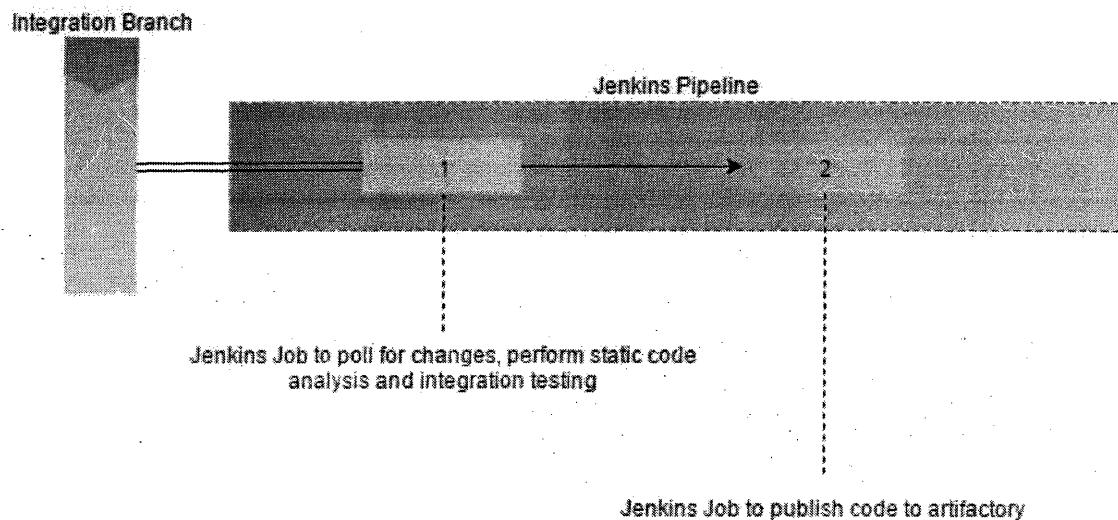
The first Jenkins job in the pipeline performs the following tasks:

- It polls the integration branch for changes at regular intervals
- Performs static code analysis on the code
- It builds and executes the integration tests

#### Jenkins job 2

The second Jenkins job in the pipeline performs the following tasks:

It uploads the built package to Artifactory/Nexus (binary code repository)



### Note

- Merge operations on the integration branch creates a new commit on it
- Each consecutive Jenkins job runs only when its preceding Jenkins job is successful
- Any success/failure event is quickly circulated among the respective teams using notifications

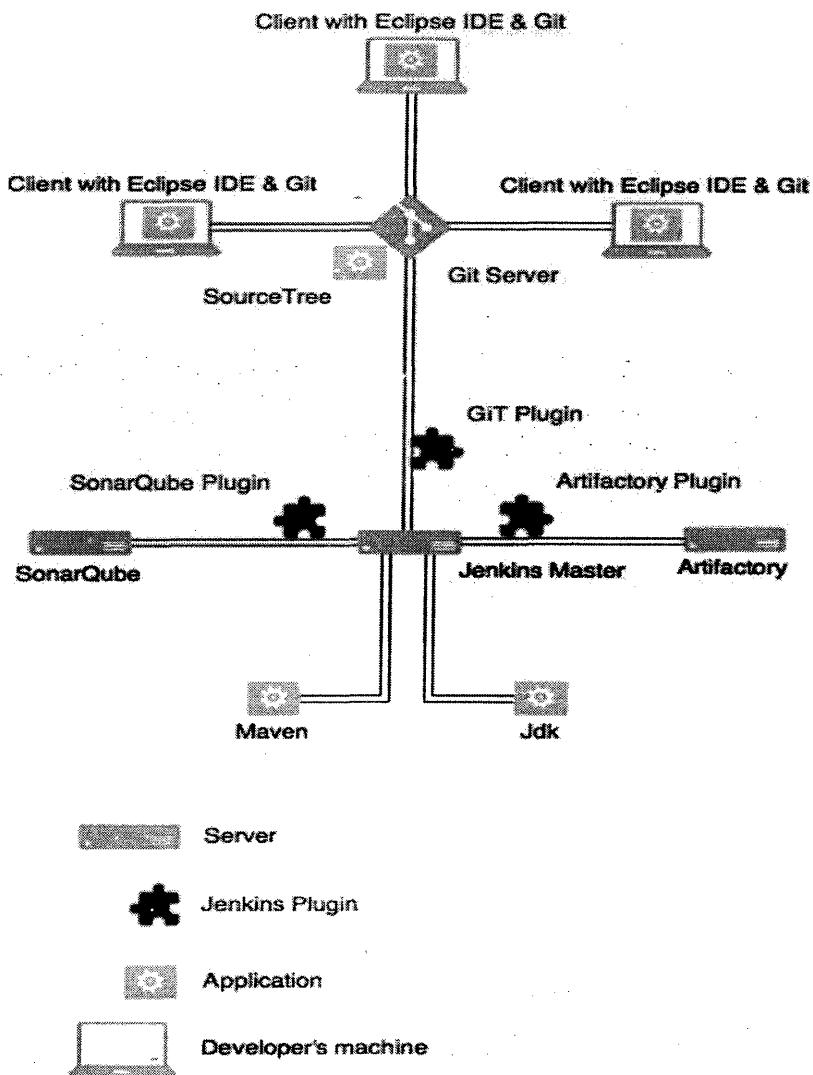
### Toolset for Continuous Integration

The example project for which we are implementing Continuous Integration is a Java-based web application. Therefore, we will see Jenkins working closely with many other tools.

| Technologies         | Description                                  |
|----------------------|--|
| Java                 | Primary programming language used for coding |
| Maven                | Build tool                                   |
| JUnit                | Unit test and integration test tools         |
| Apache Tomcat server | Servlet to host the end product              |
| Eclipse              | IDE for Java development                     |
| Jenkins              | Continuous Integration tool                  |
| Git                  | Version control system                       |
| SourceTree           | Git client                                   |
| SonarQube            | Static code analysis tool                    |

The next figure shows how Jenkins fits in as a CI server in our Continuous Integration Design, along with the other DevOps tools.

- The developers have got Eclipse IDE and Git installed on their machines. This Eclipse IDE is internally configured with the Git server. This enables the developers to clone the feature branch from the Git server onto their machines.
- The Git server is connected to the Jenkins master server using the Git plugin. This enables Jenkins to poll the Git server for changes.
- The Apache Tomcat server, which hosts the Jenkins master, has also got Maven and JDK installed on it. This enables Jenkins to build the code that has been checked in on the Git server.
- Jenkins is also connected to SonarQube server and the Artifactory server using the SonarQube plugin and the Artifactory plugin respectively.
- This enables Jenkins to perform a static code analysis on the modified code. And once the build, testing, and integration steps are successful, the resultant package is uploaded to the Artifactory for further use.



## Setting up a version control system

### Installing Git

Perform the following steps to install Git:

1. We will install Git on a Windows machine. You can download the latest Git executable from <https://git-scm.com/>

2. Begin the installation by double-clicking on the downloaded executable file.
3. Click on the **Next** button. (Until it says finish)

## Installing SourceTree (a Git client)

1. Download SourceTree from [www.sourcetreeapp.com](http://www.sourcetreeapp.com)
2. Begin the installation by double-clicking on the downloaded executable file.
3. And finish the installation

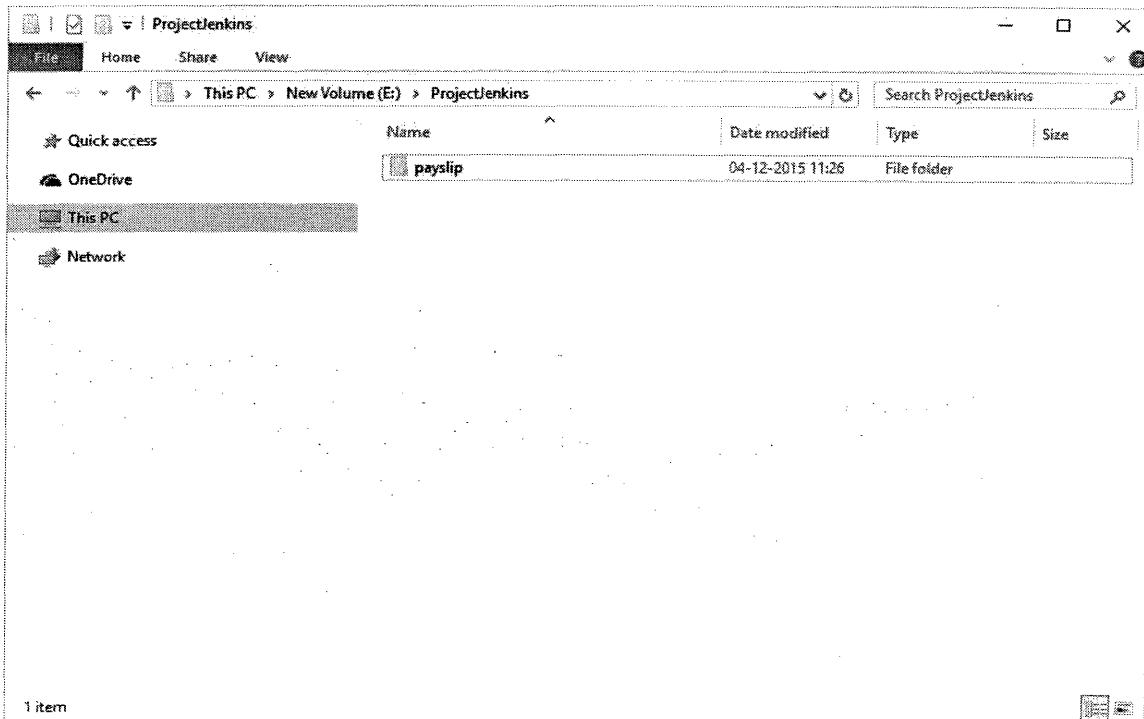
## Creating a repository inside Git

1. Open the Git bash console using the Git-bash.exe. It is present inside the directory C:\Program Files\Git\. A desktop shortcut also gets created while installing Git though.
2. Once you successfully open the Git bash console give the following command:

```
git init --shared E:\ProjectJenkins
```

## Uploading code to Git repository

1. The code can be downloaded from the following GitHub repository: <https://github.com/QualityThoughtTechnologies/ProjectJenkins>.
2. Download the payslip folder from the online repository and place it inside the Git repository's folder, as shown in the following screenshot:



3. Use the following command to add the code:

```
git add --all payslip
```

4. Now, use the following command to commit the changes to the source control:

```
git commit -m "adding files to source code" payslip
```

5. In the SourceTree dashboard, we can see the code has been added to our master branch inside the Git repository ProjectJenkins, as shown in the following screenshot:

## Configuring branches in Git

Now that we have added the code to our Git repository, let's create some branches as discussed in our CI design.

We will create an integration branch out of the master branch and two feature branches out of the integration branch. All the development will happen on the respective feature branches, and all the integration will happen on the integration branch.

The master branch will remain neat and clean and only code that has been built and tested thoroughly will reside on it.

### Using the Git commands

1. Open Git bash console and type to following command to create the integration branch:

```
cd e:/ProjectJenkins  
git branch integration
```

2. You will get the following output:

A screenshot of a Windows terminal window titled 'MINGW64/e/ProjectJenkins'. The command '\$ git branch integration' is entered and executed, resulting in the message 'Switched to branch integration'. The terminal window has a standard Windows title bar and a black background with white text.

```
MINGW64/e/ProjectJenkins  
nikhi@DESKTOP-93311SL MINGW64 /e/ProjectJenkins (master)  
$ git branch integration  
Switched to branch integration  
nikhi@DESKTOP-93311SL MINGW64 /e/ProjectJenkins (integration)  
$ |
```

3. In order to create the feature branches, first check out the integration branch with the following command:

```
git checkout integration
```

4. Then, use the following command to create the feature branches one by one:

```
git branch feature1  
git branch feature2
```

### Note

You can also see that all the branches are at the same level, which means all the branches currently have the same version of the code without any difference.

### Git cheat sheet

The following table contains the list of Git commands used in the current chapter:

| Branches                         |   |
|----------------------------------|---|
| git branch                       | List all of the branches in your repository.  |
| git branch <branch>              | Create a new branch.  |
| git checkout<br><branch>         | Create and check out a new branch named <branch>.   |
| git merge <branch>               | Merge <branch> into the current branch.   |
| Repository                       |   |
| git init <directory>             | Create empty Git repository in the specified directory.   |
| git add <directory>              | Stage all changes in <directory> for the next commit.<br>Replace <directory> with <file> to change a specific file. |
| git status                       | List which files are staged, unstaged, and untracked.   |
| git commit -m<br>"<br><message>" | Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message.            |
| Rebase                           |   |
| git rebase -i <branch>           | Interactively rebase the current branch onto another branch named <branch>  |

## Configuring Jenkins

Notification and reporting are an important part of Continuous Integration. Therefore, we need an advanced e-mail notification plugin. We will also need a plugin to make Jenkins interact with Git.

Along with these plugins, we will also need to install and configure Java and Maven inside Jenkins. This will enable Jenkins to perform builds.

### Installing the Git plugin

In order to integrate Git with Jenkins, we need to install the GIT plugin. The steps are as follows:

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link.
2. This will take you to the **Manage Jenkins** page. From here, click on the **Manage Plugins** link and go to the **Available** tab.
3. Type GIT plugin in the search box. Select **GIT plugin** from the list and click on the **Install without restart** button.
4. The download and installation starts automatically. You can see the GIT plugin has a lot of dependencies that get downloaded and installed.

## Installing Plugins/Upgrades

### Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

### Credentials Plugin

 credentials plugin is already installed. Jenkins needs to be restarted for the update to take effect

### SSH Credentials Plugin

 ssh-credentials plugin is already installed. Jenkins needs to be restarted for the update to take effect

### GIT client plugin

 Success

### SCM API Plugin

 Success

### Mailer Plugin

 mailer plugin is already installed. Jenkins needs to be restarted for the update to take effect

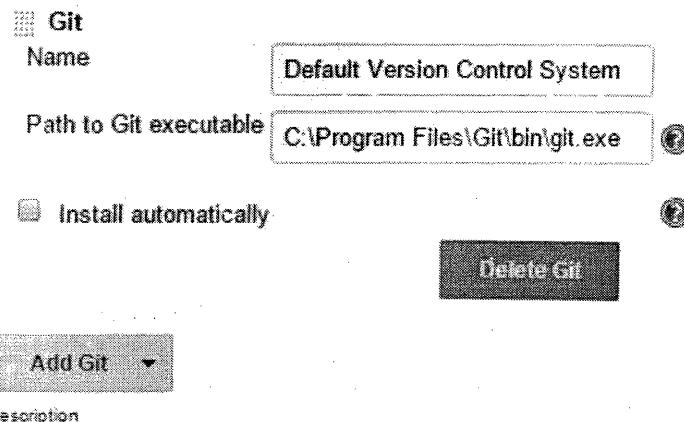
### GIT plugin

 Success

5. Upon successful installation of the GIT plugin, go to the **Configure System** link from the **Manage Jenkins** page.
6. Scroll down until you see the **Git** section and fill the blanks as shown in the following screenshot.
7. You can name your Git installation whatever you want. Point the **Path to Git executable** to the location where you have installed Git. In our example, it's C:\Program Files\Git\bin\git.exe.
8. You can add as many Git servers as you want by clicking on the **Add Git** button.

## Git

### Git installations



## Note

If you have more than one Git server to choose from, provide a different name for each Git instance.

## Installing and configuring JDK

First, download and install Java on your Jenkins server, which I guess you might have already done as part of the Apache Tomcat server installation in the previous chapter. If not, then simply download the latest Java JDK from the internet and install it.

1. After installing Java JDK, make sure to configure **JAVA\_HOME** using the following command:

Copy

```
setx JAVA_HOME "C:\Program Files\Java\jdk1.8.0_60" /M
```

2. To check the home directory of Java, use the following command:

Copy

```
echo %JAVA_HOME%
```

3. You should see the following output:

Copy

```
C:\Program Files\Java\jdk1.8.0_60
```

4. Also, add the Java executable path to the system PATH variable using the following command:

```
setx PATH "%PATH%;C:\Program Files\Java\jdk1.8.0_60\bin" /M
```

## Configuring JDK inside Jenkins

You have installed Java and configured the system variables. Now, let Jenkins know about the JDK installation:

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link.
2. On the **Manage Jenkins** page, click on the **Configure System** link.
3. Scroll down until you see the **JDK** section. Give your JDK installation a name. Also, assign the **JAVA\_HOME** value to the JDK installation path, as shown in the following screenshot:

The screenshot shows the Jenkins 'Configure System' page under the 'JDK' section. It displays a table with one row for 'JDK installations'. The row contains fields for 'Name' (set to 'JDK 1.8') and 'JAVA\_HOME' (set to 'C:\Program Files\Java\jdk1.8.0\_60'). Below the table are two buttons: 'Install automatically' (unchecked) and 'Delete JDK'. At the bottom left is a 'Add JDK' button, and at the bottom center is a link 'List of JDK installations on this system'.

|                   |           |                                   |
|-------------------|-----------|-----------------------------------|
| JDK installations | Name      | JDK 1.8                           |
|                   | JAVA_HOME | C:\Program Files\Java\jdk1.8.0_60 |

Install automatically   

List of JDK installations on this system

## Note

You can configure as many JDK instances as you want. Provide a unique name to each JDK installation.

## Installing and configuring Maven

1. Download Maven from the following link: <https://maven.apache.org/download.cgi>.
2. Extract the downloaded zip file to C:\Program Files\Apache Software Foundation\.

## Setting the Maven environment variables

1. Set the Maven M2\_HOME, M2, and MAVEN\_OPTS variables using the following commands:

```
setx M2_HOME "C:\Program Files\Apache Software Foundation\apache-maven-3.3.9" /M  
setx M2 "%M2_HOME%\bin" /M  
setx MAVEN_OPTS "-Xms256m -Xmx512m" /M
```

2. Also, add the Maven bin directory location to the system path using the following command:

```
setx PATH "%PATH%;%M2%" /M
```

3. To check if Maven has been installed properly, use the following command:

```
mvn -version
```

and output should be

```
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T22:11:47+05:30)  
Maven home: C:\Program Files\Apache Software Foundation\apache-maven-3.3.9  
Java version: 1.8.0_60, vendor: Oracle Corporation  
Java home: C:\Program Files\Java\jdk1.8.0_60\jre  
Default locale: en_IN, platform encoding: Cp1252  
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"
```

### Configuring Maven inside Jenkins

We have successfully installed Maven. Now, let us see how to connect it with Jenkins.

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link.
2. On the **Manage Jenkins** page, click on the **Configure System** link.
3. Scroll down until you see the **Maven** section.
4. Assign the **MAVEN\_HOME** field to the Maven installation directory. Name your Maven installation by giving it a unique name.

### Installing the e-mail extension plugin

The e-mail notification facility that comes with the Jenkins is not enough. We need a more advanced version of e-mail notification such as the one provided by **Email Extension** plugin.

To do this, perform the following steps:

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link. This will take you to the **Manage Jenkins** page.
2. Click on the **Manage Plugins** link and go to the **Available** tab.
3. Type email extension plugin in the search box.
4. Select **Email Extension Plugin** from the list and click on the **Install without restart** button.

### The Jenkins pipeline to poll the feature branch

In the following section, we will see how to create both the Jenkins jobs that are part of the pipeline to poll the feature branch. This pipeline contains two Jenkins jobs.

## Creating a Jenkins job to poll, build, and unit test code on the feature1 branch

The first Jenkins job from the pipeline to poll the feature branch does the following tasks:

- It polls the feature branch for changes at regular intervals
- It performs a build on the modified code
- It executes unit tests

Let's start creating the first Jenkins job. I assume you are logged in to Jenkins as an admin and have privileges to create and modify jobs. The steps are as follows:

1. From the Jenkins Dashboard, click on the **New Item** link.
2. Name your new Jenkins job **Poll\_Build\_UnitTest\_Feature1\_Branch**.
3. Select the type of job as **Freestyle project** and click on **OK** to proceed.

Item name: Poll\_Build\_UnitTest\_Feature1\_Branch

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: [empty field]

**OK**

4. Add a meaningful description about the job in the **Description** section.

### Polling version control system using Jenkins

This is a critical step where we connect Jenkins with the Version Control System. This configuration enables Jenkins to poll the correct branch inside Git and download the modified code:

1. Scroll down to the **Source Code Management** section and select the **Git** option.
2. Fill the blanks as follows:
  1. **Repository URL:** Specify the location of the Git repository. It can be a GitHub repository or a repository on a Git server. In our case it's `/e/ProjectJenkins`, as the Jenkins server and the Git server is on the same machine.
  2. Add `*/feature1` in the **Branch to build** section, since we want our Jenkins job to poll the `feature1` branch.
3. Leave rest of the fields at their default values.

Source Code Management

|                            |  |   |                                      |
|----------------------------|--|---|--------------------------------------|
| <input type="radio"/> None | <input type="radio"/> CVS  | <input checked="" type="radio"/> CVS Projectset | <input checked="" type="radio"/> Git |
| Repositories               | Repository URL   | <input type="text" value="/e/ProjectJenkins"/>  |                                      |
| Credentials                | <input type="text" value="- none -"/><br><a href="#">Add</a>               |   |                                      |
| Name                       | <input type="text"/>   |   |                                      |
| Refspec                    | <input type="text"/>   |   |                                      |
|                            |  | <a href="#">Add Repository</a>                  | <a href="#">Delete Repository</a>    |
| Branches to build          | Branch Specifier (blank for 'any') <input type="text" value="*/feature1"/> |   |                                      |
|                            |  | <a href="#">Add Branch</a>                      | <a href="#">Delete Branch</a>        |
| Repository browser         | <input type="text" value="(Auto)"/>  |   |                                      |
| Additional Behaviours      | <a href="#">Add</a>  |   |                                      |

4. Scroll down to the **Build Triggers** section.
5. Select **Poll SCM** and type **H/5 \* \* \* \*** in the **Schedule** field. We want our Jenkins job to poll the feature branch every 5 minutes. However, feel free to choose the polling duration as you wish depending on your requirements.

## Build Triggers

- Trigger builds remotely (e.g., from scripts)
- Build after other projects are built
- Build periodically
- Poll SCM

### Schedule

H/5 \* \* \* \*

Would last have run at Friday, 4 December, 2015 9:55:19 PM IST.  
would next run at Friday, 4 December, 2015 10:00:19 PM IST.

- Ignore post-commit hooks

## Compiling and unit testing the code on the feature branch

This is an important step in which we tell Jenkins to build the modified code that was downloaded from Git. We will use Maven commands to build our Java code.

1. Scroll down to the **Build** section.
2. Click on the **Add build step** button and select **Invoke top-level Maven targets** from the available options.
3. Configure the fields as shown in the following screenshot:
  1. Set **Maven Version** as Maven 3.3.9. Remember this is what we configured on the **Configure System** page in the **Maven** section. If we had configured more than one Maven, we would have a choice here.
  2. Type the following line in the **Goals** section:

```
clean verify -Dtest=VariableComponentTest -DskipITs=true javadoc:javadoc
```

3. Type `payslip/pom.xml` in the **POM** field. This tells Jenkins the location of pom.xml in the downloaded code.

#### Build

|  |   |
|--|---|
| <input checked="" type="checkbox"/> Invoke top-level Maven targets | <input type="checkbox"/>  |
| Maven Version  | Maven 3.3.9   |
| Goals  | <code>clean verify -Dtest=VariableComponentTest -DskipITs=true javadoc:javadoc</code> |
| POM  | <code>payslip/pom.xml</code>  |
| Properties   | <input type="text"/>  |
| JVM Options  | <input type="text"/>  |
| Use private Maven repository                                       | <input type="checkbox"/>  |
| Settings file  | Use default maven settings  |
| Global Settings file   | Use default maven global settings   |
| <input type="button" value="Delete"/>                              |   |

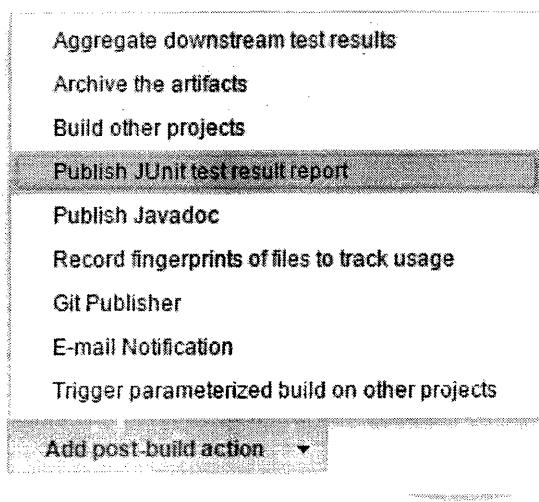
4. Let's see the Maven command inside the **Goals** field in detail:

1. The clean command will clean any old built files
2. The `-Dtest=VariableComponentTest` command will execute a unit test named `VariableComponentTest.class`
3. The `-DskipITs=true` command will skip the integration test, if any, as we do not need them to execute at this point
4. The `javadoc:javadoc` command will tell Maven to generate Java documentations

## Publishing unit test results

Publishing unit test results falls under post build actions. In this section we configure the Jenkins job to publish JUnit test results:

1. Scroll down to the **Post build Actions** section.
2. Click on the **Add post-build action** button and select **Publish JUnit test result report**, as shown in the following screenshot:



3. Under the **Test report XMLs** field, add `payslip/target/surefire-reports/*.xml`. This is the location where the unit test reports will be generated once the code has been built and unit tested.

### Post-build Actions

**Publish JUnit test result report**

**Test report XMLs**  Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/\*.xml'. Basedir of the fileset is the workspace root.

**Retain long standard output/error**

**Health report amplification factor**  1% failing tests scores as 99% health.  
5% failing tests scores as 95% health

**Delete**

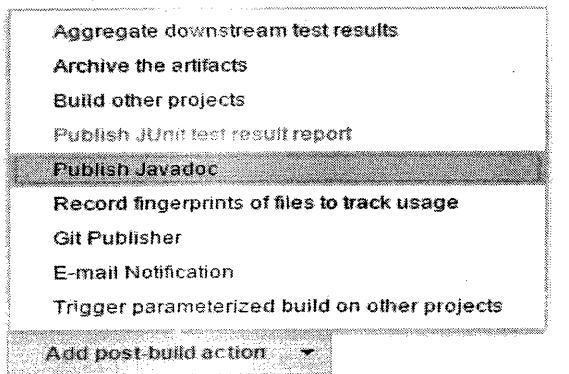
## Note

Jenkins will access all the \*.xml files present in the payslip/target/surefire-reports directory and publish the report. We will shortly see this when we run this Jenkins job.

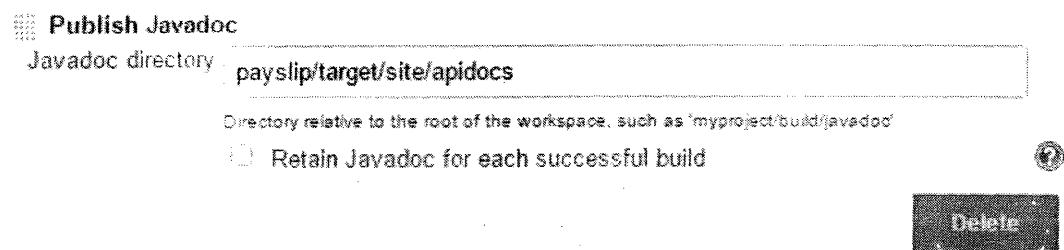
## Publishing Javadoc

The steps to publish Javadoc are:

- Once again, click on the **Add post-build action** button. This time, select **Publish Javadoc**.



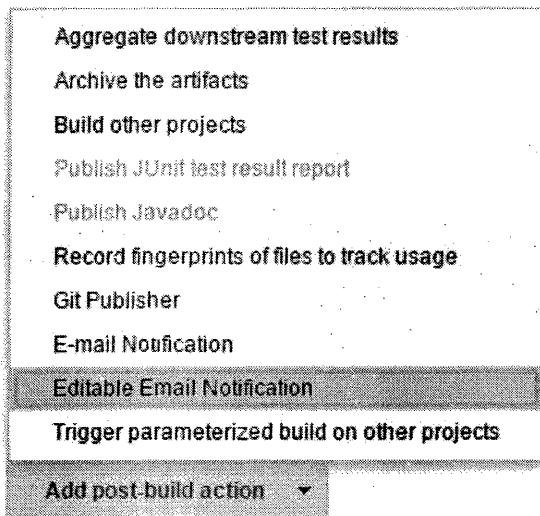
- Add the path payslip/target/site/apidocs in the **Javadoc directory** field, as shown in the following screenshot:



## Configuring advanced e-mail notification

Notification forms are an important part of CI. In this section, we will configure the Jenkins job to send e-mail notifications based on few conditions. Let's see the steps in detail:

1. Click on the **Add post-build action** button and select **Editable Email Notification**, as shown in the following screenshot:



2. Configure **Editable Email Notification** as follows:

- Under **Project Recipient List**, add the e-mail IDs separated by a comma. You can add anyone who you think should be notified for build and unit test success/failure.
  - You can add the e-mail ID of the Jenkins administrator under **Project Reply-To List**.
  - Select **Content Type** as **HTML (text/html)**.
3. Leave all the rest of the options at their default values.

Editable Email Notification  

Disable Extended Email Publisher  

Project Recipient List Allows the user to disable the publisher, while maintaining the settings.    
developer@organisation.com,manager@organisation.com

Project Reply-To List Comma-separated list of email address that should receive notifications for this project.    
admin@organisation.com

Content Type Comma-separated list of email address that should be in the Reply-To header for this project.    
HTML (text/html)

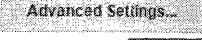
Default Subject    
\$DEFAULT SUBJECT

Default Content    
\$DEFAULT CONTENT

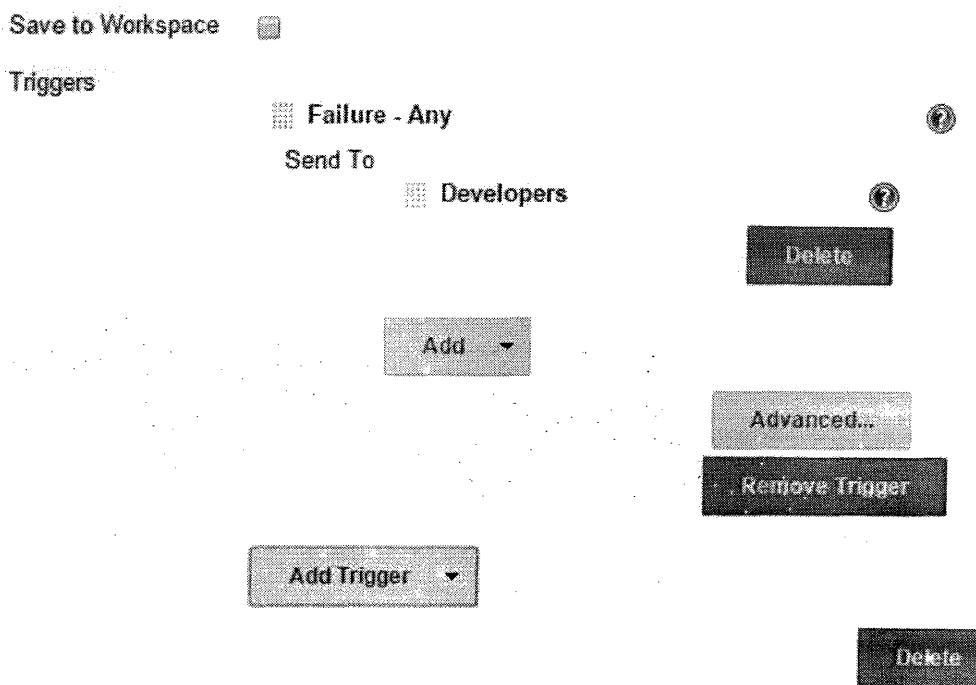
Attachments Can use wildcards like 'module/dist/\*\*.zip'. See the [includes of Ant Reset](#) for the exact format. The base directory is [the workspace](#).  

Attach Build Log    
Attach Build Log

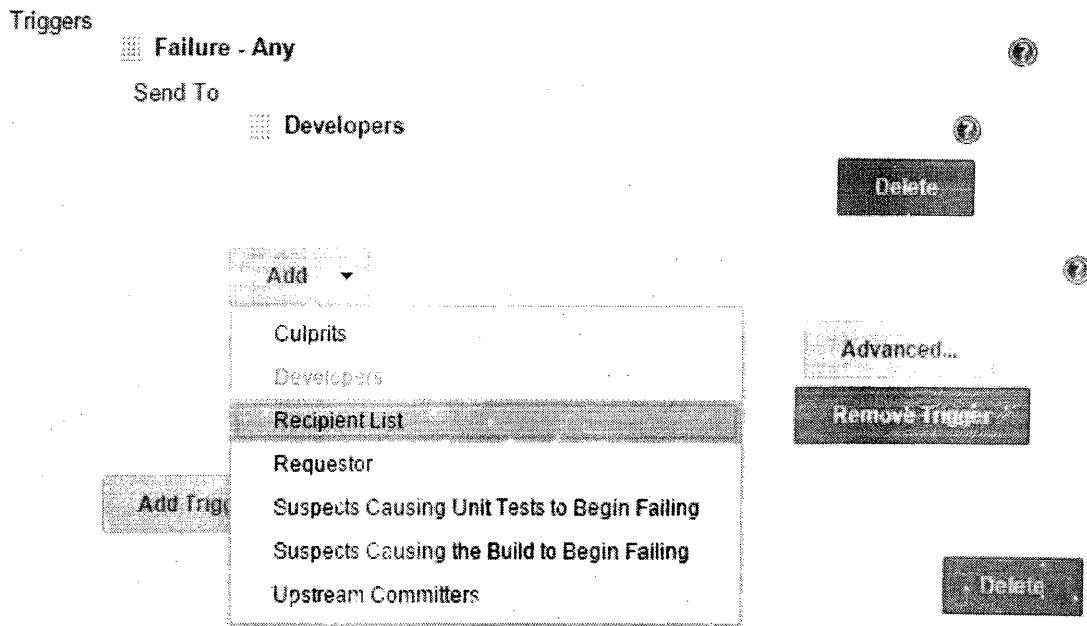
Content Token Reference  

4. Now, click on the **Advanced Settings...** button.
5. By default, there is a trigger named **Failure – Any** that sends an e-mail notification in the event of a failure (any kind of failure).
6. By default, the **Send To** option is set to **Developers**.

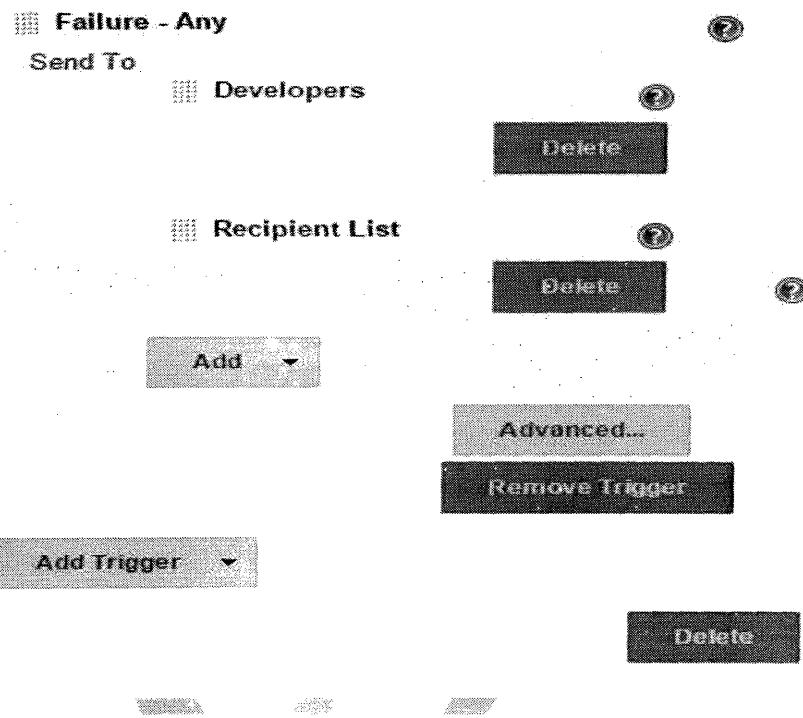


7. But we don't want that; we have already defined whom to send e-mails to. Therefore, click on the **Add** button and select the **Recipient List** option, as shown in the following screenshot:



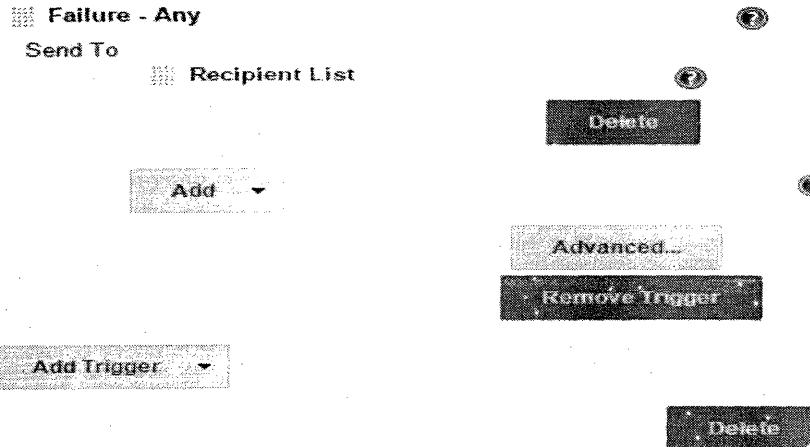
8. The result will look something like the following screenshot:

Triggers



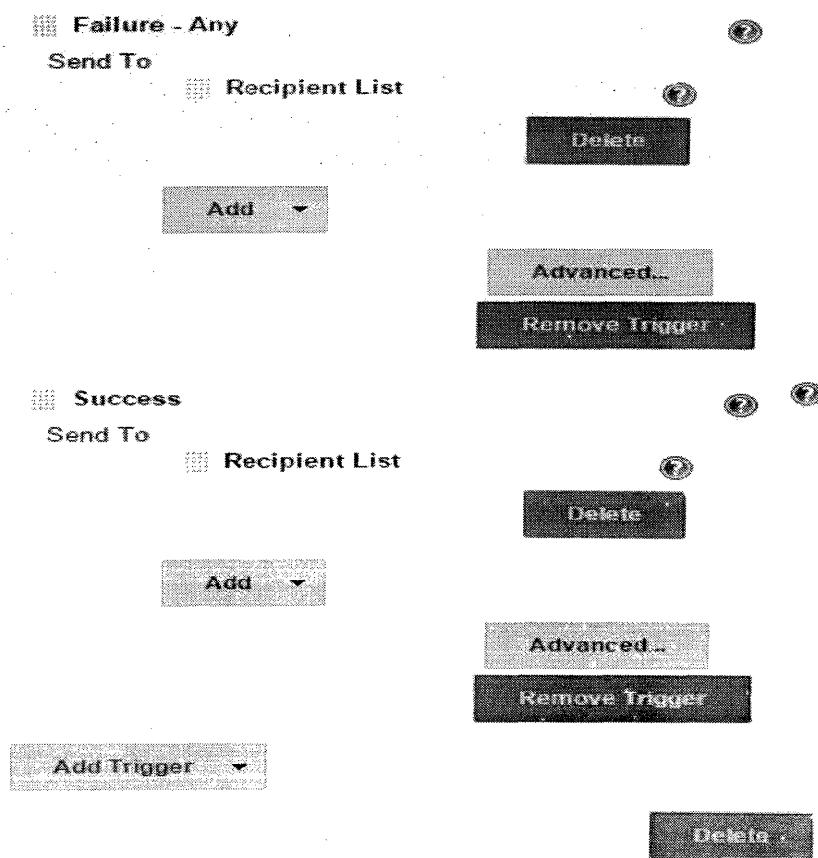
9. Delete **Developers** from the **Send To** section by clicking on the **Delete** button adjacent to it. The result should look something like the following screenshot:

Triggers



10. Let's add another trigger to send an e-mail when the job is successful.

11. Click on the **Add Trigger** button and select the **Success** option, as shown in the following screenshot:
12. Configure this new success trigger in a similar fashion, by removing **Developers** and adding **Recipient List** under the **Send To** section. Finally, everything should look like the following screenshot:

**Triggers**

### **Creating a Jenkins job to merge code to the integration branch**

The second Jenkins job in the pipeline performs the task of merging the successfully built and tested code into the integration branch:

1. I assume you are logged in to Jenkins as an admin and have privileges to create and modify jobs.
2. From the Jenkins Dashboard, click on the **New Item**.
3. Name your new Jenkins job **Merge\_Feature1\_Into\_Integration\_Branch**.
4. Select the type of job as **Freestyle project** and click on **OK** to proceed.
5. Add a meaningful description of the job in the **Description** section.

#### **Using the build trigger option to connect two or more Jenkins jobs**

This is an important section wherein we will connect two or more Jenkins jobs to form a Jenkins pipeline to achieve a particular goal:

1. Scroll down to the **Build Triggers** section and select the **Build after other projects are built** option.
2. Under the **Projects to watch** field, add **Poll\_Build\_UnitTest\_Feature1\_Branch**.
3. Select the **Trigger only if the build is stable** option.
4. Scroll down to the **Build** section. From the **Add build step** dropdown, select **Execute Windows batch command**.
5. Add the following code into the **Command** section:

**Build**

**Execute Windows batch command**

**Command**

```
E:  
cd ProjectJenkins  
git checkout integration  
git merge feature1 --stat
```

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

**Delete**

6. Configure advanced e-mail notifications exactly the same way as mentioned earlier.
7. Save the Jenkins job by clicking on the **Save** button.

### **Creating a Jenkins job to poll, build, and unit test code on the feature2 branch**

Since we have the two feature branches in place, we need to create a Jenkins Job to poll, build, and unit test code on the **feature2** branch. We will do this by cloning the already existing Jenkins job that polls the **feature1** branch. The steps are as follows:

1. From the Jenkins Dashboard, click on **New Item**.
2. Name your new Jenkins job **Poll\_Build\_UnitTest\_Feature2\_Branch**.
3. Select the type of job as **Copy existing Item** and type **Poll\_Build\_UnitTest\_Feature1\_Branch** in the **Copy from** field.
4. Click on **OK** to proceed.
5. Scroll down to the **Source Code Management** section. You will find everything prefilled, as this is a copy of the Jenkins job **Poll\_Build\_UnitTest\_Feature1\_Branch**.
6. Change the **Branch to build** section from `*/feature1` to `*/feature2`, since we want our Jenkins job to poll the **feature2** branch.
7. Scroll down to the **Build** section. Modify the **Goals** field. Replace the existing one with `clean verify -Dtest=TaxComponentTest -DskipITs=true javadoc:javadoc`.
8. Leave everything as it is.
9. Scroll down to the **Editable Email Notification** section and you can change the **Project Recipient List** values if you want to.

**Creating a Jenkins job to merge code to the integration branch**

Similarly, we need to create another Jenkins job that will merge the successfully built and unit tested code on the feature1 branch into the integration branch. And, we will do this by cloning the already existing Jenkins job that merges the successfully build and unit tested code from feature1 branch into the Integration branch. The steps are as follows:

1. From the Jenkins Dashboard, click on **New Item**.
2. Name your new Jenkins job **Merge\_Feature2\_Into\_Integration\_Branch**. Alternatively, use any name that makes sense.
3. Select the type of job as **Copy existing Item** and type **Merge\_Feature1\_Into\_Integration\_Branch** in the **Copy from** field.
4. Click on **OK** to proceed.
5. Scroll down to the **Build Triggers** section and select the **Build after other projects are built** option.
6. Under the **Projects** to **watch** field, replace **Poll\_Build\_UnitTest\_Feature1\_Branch** with **Poll\_Build\_UnitTest\_Feature2\_Branch**.
7. Scroll down to the **Build** section. Replace the existing code with the following:

**Build**

**Execute Windows batch command**

```
Command: E:\  
          cd ProjectJenkins  
          git checkout integration  
          git merge feature2 --stat
```

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

**Delete**

**Add build step ▾**

8. Leave everything as it is.
9. Scroll down to the **Editable Email Notification** section. You can change the **Project Recipient List** values if you want to.

## **5. Continuous Integration Using Jenkins – Part II**

Here, we will cover the following topics:

- Installing SonarQube
- Installing SonarQube Scanner
- Installing Artifactory (binary repository)
- Installing and configuring Jenkins plugin for SonarQube and Artifactory
- Creating the Jenkins pipeline to poll the integration branch

### **Installing SonarQube to check code quality**

Apart from integrating code in a continuous way, CI pipelines also include tasks that perform Continuous Inspection—inspecting code for its quality in a continuous approach.

Continuous Inspection deals with inspecting and avoiding code that is of poor quality. Tools such as SonarQube help us to achieve this. Every time a code gets checked in (committed), it is analyzed. This analysis is based on some rules defined by the code analysis tool. If the code passes the error threshold, it's allowed to move to the next step in its life cycle. If it crosses the error threshold, it's dropped.

Some organizations prefer checking the code for its quality right when the developer tries to check in the code. If the analysis is good, the code is allowed to be checked in, or else the check in is canceled and the developer needs to work on the code again.

SonarQube is a code quality management tool that allows teams to manage, track, and improve the quality of their source code. It is a web-based application that contains rules, alerts, and thresholds, all of which can be configured. It covers the seven types of code quality parameters: architecture and design, duplications, unit tests, complexity, potential bugs, coding rules, and comments.

SonarQube is an open source tool that supports almost all popular programming languages with the help of plugins. SonarQube can also be integrated with a CI tool such as Jenkins to perform Continuous Inspection

let's see how to install SonarQube. We will install SonarQube 5.1.2 on Windows 10 with the following steps:

1. To do this, download SonarQube 5.1.2 from <http://www.sonarqube.org/downloads/>
2. Once you have successfully downloaded the SonarQube 5.1.2 archive, extract it to C:\Program Files\. I have extracted it to C:\Program Files\sonarqube-5.1.2.

### **Setting the Sonar environment variables**

Perform the following steps to set the %SONAR\_HOME% environment variable:

Set the %SONAR\_HOME% environment variable to the installation directory which, in our example, is C:\Program Files\sonarqube-5.1.2. Use the following command:

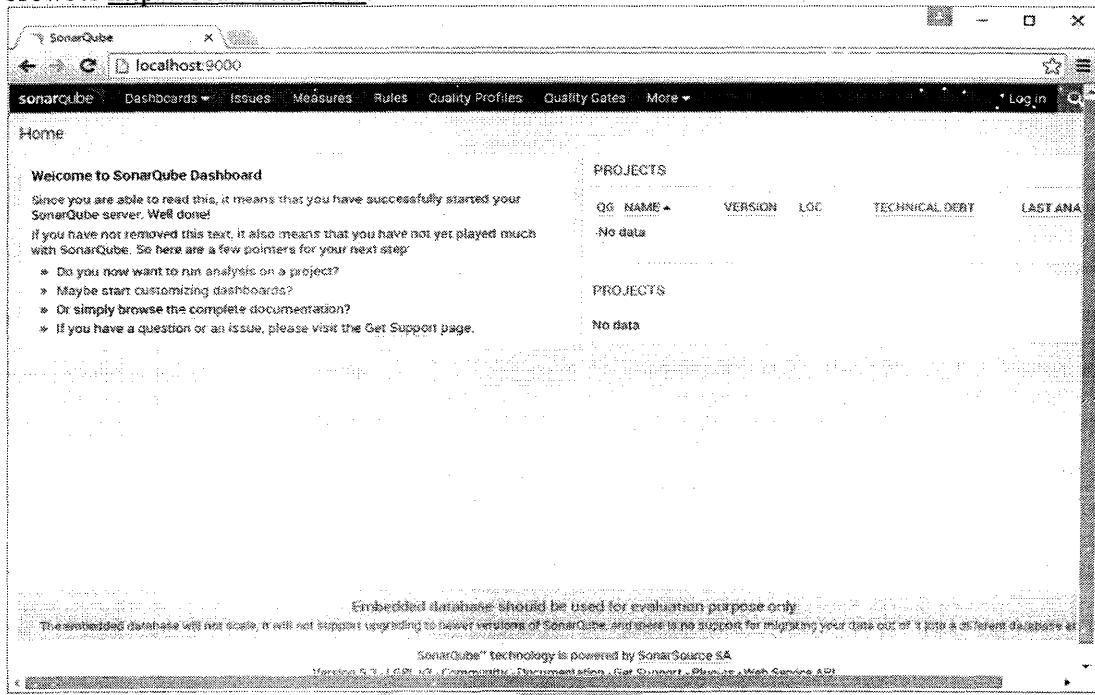
```
setx SONAR_HOME "C:\Program Files\sonarqube-5.1.2" /M
```

### **Running the SonarQube application**

1. Use the following commands to go to the directory where the scripts to install and start SonarQube are present:

```
cd %SONAR_HOME%\bin\windows-x86-64
```

2. To install SonarQube, run the `InstallNTService.bat` script:
3. To start SonarQube, run the `StartNTService.bat` script:
4. To access SonarQube, type the following link in your favorite web browser <http://localhost:9000/>.



## Note

Right now, there are no user accounts configured in SonarQube. However, by default, there is an admin account with the username `admin` and the password `admin`.

## Creating a project inside SonarQube

To create the project in SonarQube, use the following steps:

1. Log in as an admin by clicking on the **Log in** link at the top-right corner on the Sonar Dashboard. You will see some more items in the menu bar, as shown in the following screenshot:

Welcome to SonarQube Dashboard

Since you are able to read this, it means that you have successfully started your SonarQube server. Well done!

If you have not removed this text, it also means that you have not yet played much with SonarQube. So here are a few pointers for your next step:

- » Do you now want to run analysis on a project?
- » Maybe start customizing dashboards?
- » Or simply browse the complete documentation?
- » If you have a question or an issue, please visit the Get Support page.

**MY FAVOURITES**

| QG NAME | LAST ANALYSIS |
|---------|---------------|
| No data |               |

SonarQube™ technology is powered by SonarSource SA  
Version 5.1.2 - LGPL v3 - Community - Documentation - Get Support - Plugins - Web Service API

2. Click on the **Settings** link on the menu bar.
3. On the **Settings** page, click on **System** and select the **Provisioning** option, as shown in the following screenshot:

**Settings**

Configuration ▾ Security ▾ System ▾

**General Settings**  
Edit global settings for this SonarQube instance

**CATEGORY**

- Build Breaker
- Exclusions
- General
- Java
- Licenses
- SCM
- Security
- Technical Debt

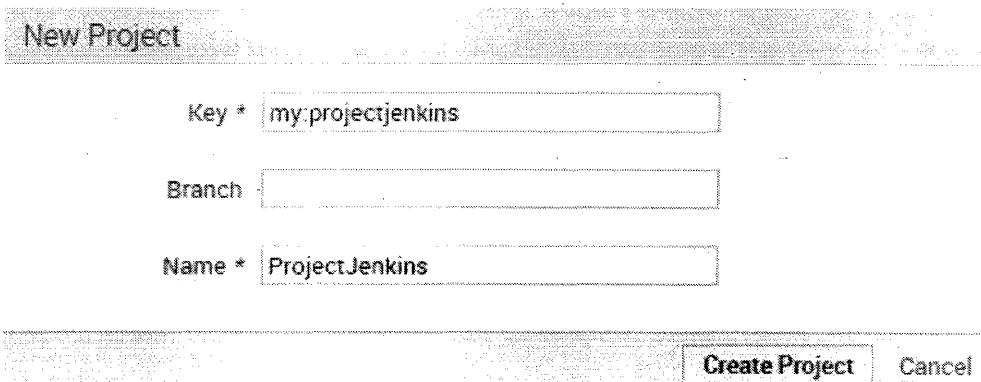
**Provisioning**

Comma-separated list of plugin keys. Those plugins will be used for analyses.  
Key: sonar.preview.includePlugins

Default: buildstability,devcockpit,pdfreport,report,views,jira,buildbreakers,sonarqube,sonarqube-issues,sonarqube-security,sonarqube-technical-debt,sonarqube-build-breaker,sonarqube-exclusions,sonarqube-languages,sonarqube-licenses,sonarqube-scm,sonarqube-security,sonarqube-technical-debt

Comma-separated list of plugin keys. Those plugins will not be used for analyses.

4. On the **Provisioning** page, click on the **Create** link present at the right corner to create a new project.
5. A pop-up window will appear asking for **Key**, **Branch**, and **Name** values. Fill the blanks as shown in the following screenshot and click on the **Create Project** button.



6. That's it. We have successfully created a project inside SonarQube.

## Installing the build breaker plugin for Sonar

The build breaker plugin is available for SonarQube. It's exclusively a SonarQube plugin and not a Jenkins plugin. This plugin allows the Continuous Integration system (Jenkins) to forcefully fail a Jenkins build if a quality gate condition is not satisfied. To install the build breaker plugin, follow these steps:

1. Download the build breaker plugin from the following link: <http://update.sonarsource.org/plugins/buildbreaker-confluence.html>.
2. Place the downloaded sonar-build-breaker-plugin-1.1.rar file in the following location: C:\Program Files\sonarqube-5.1.2\extensions\plugins.
3. We need to restart SonarQube service. To do so, type services.msc in Windows Run.
4. From the Services window, look for a service named **SonarQube**. Right-click on it and select **Restart**.

5. After a successful restart, go to the SonarQube dashboard and log in as admin. Click on the **Settings** link from the menu options.
6. On the **Settings** page, you will find the **Build Breaker** option under the **CATEGORY** sidebar as shown in the following screenshot. Do not configure anything.

## Creating quality gates

For the build breaker plugin to work, we need to create a **quality gate**. It's a rule with some conditions. When a Jenkins job that performs a static code analysis is running, it will execute the **quality profiles** and the **quality gate**. If the quality gate check passes successfully, then the Jenkins job continues. If it fails, then the Jenkins job is aborted. Nevertheless, the static code analysis still happens. To create a quality gate, perform the following steps:

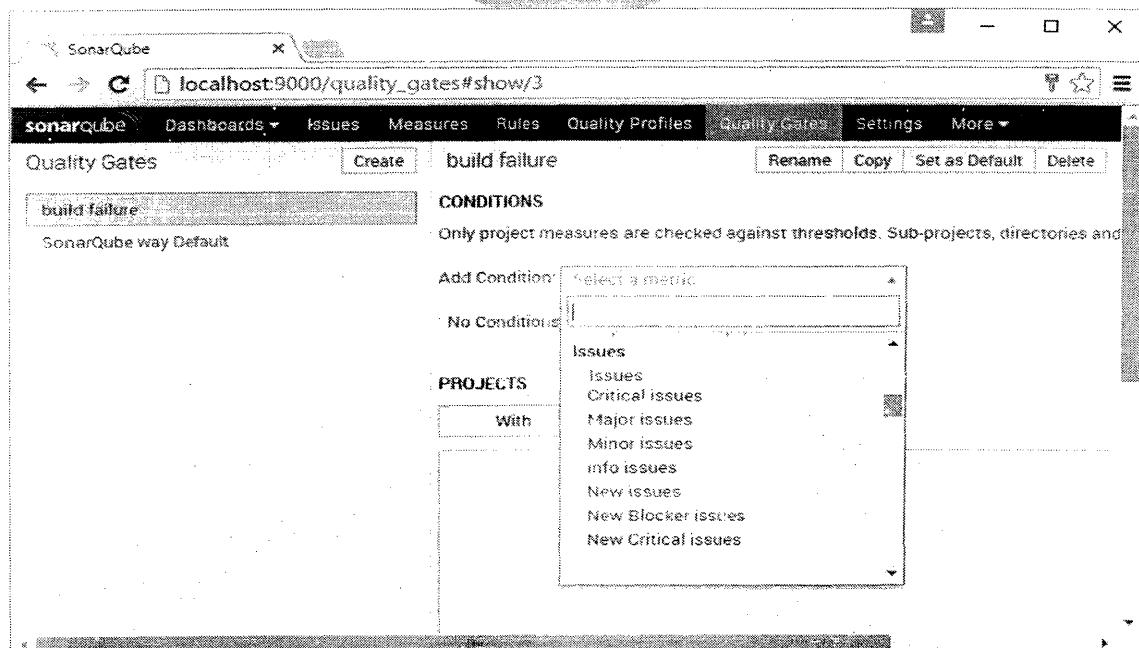
1. Click on the **Quality Gates** link on the menu. By default, we have a quality gate named **SonarQube way**.
2. Click on the **Create** button to create a new quality gate.

| Condition            | Value | Operator        |
|----------------------|-------|-----------------|
| Blocker issues       | Value | is greater than |
| Critical issues      | Value | is greater than |
| Coverage on new code | Value | is less than    |
| Open issues          | Value | is greater than |
| Reopened issues      | Value | is greater than |
| Skipped unit tests   | Value | is greater than |
| Unit test errors     | Value | is greater than |
| Unit test failures   | Value | is greater than |

3. In the pop-up window, add the name that you wish to give to your new quality gate in the **Name** field. In our example, I have used build failure.
4. Once done, click on the **Create** button.



5. You will see a new quality gate named build failure on the left-hand side of the page.
6. Now, click on the build failure quality gate and you will see a few settings on the right-hand side of the page.
7. Set the build failure quality gate as the default quality gate by clicking on the **Set as Default** button.
8. Now, in the **Add Condition** field, choose a condition named **Major issues** from the drop-down menu, as shown in the following screenshot:



9. Now let's configure our condition. If the number of **Major issues** is greater than six, the build should fail; and if it is between five and six, it should be a warning. To achieve this, set the condition parameters as shown here:

### CONDITIONS

Only project measures are checked against thresholds. Sub-projects, directories and files are ignored. More

Add Condition: Select a metric

Major issues Value is greater than 5 6 Add Cancel

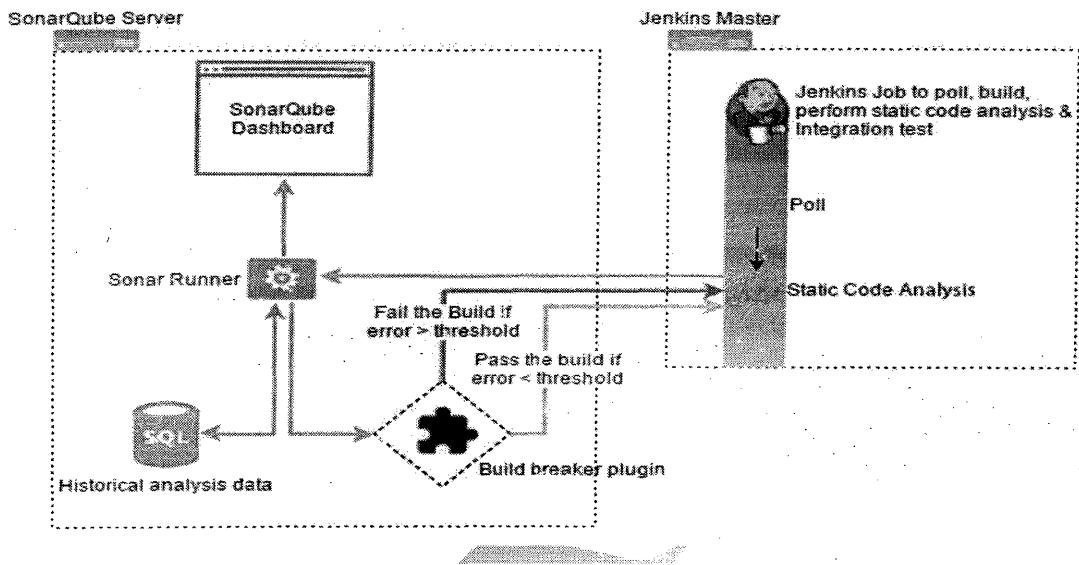
### Installing SonarQube Scanner

SonarQube Scanner, also called **SonarQube Runner**, is an important application that actually performs the code analysis. SonarQube Scanner scans the code for its quality, based on some predefined rules. It then helps the SonarQube web application to display this analysis along with other metrics.

The following image clearly depicts how SonarQube Server, SonarQube Scanner, and the build breaker plugin work together with Jenkins.

SonarQube Scanner is invoked through Jenkins to perform the code analysis. The code analysis is presented on the SonarQube dashboard and also passed to the build breaker plugin.

There are conditions defined inside the quality gates. If the analysis passes these conditions, then the Jenkins job is signed to proceed. However, if the analysis fails the condition, then the build breaker plugin terminates the Jenkins job.



Follow these steps to install SonarQube Scanner:

1. Download the SonarQube Scanner (that is, SonarQube Runner) from the link <http://docs.sonarqube.org/display/SONAR/Analyzing+with+SonarQube+Scanner>.
2. The link keeps updating, so just look for SonarQube Scanner on Google if you don't find it.

**Installing and Configuring SonarQube Scanner**

Created by David Rascón on Nov 02, 2015

|                            |   |
|----------------------------|---|
| Name                       | SonarQube Scanner   |
| Latest version             | 2.4 (28 Apr 2014)   |
| Requires SonarQube version | 4.5.1 or higher   |
| Download                   | <a href="http://repo1.maven.org/maven2/org/codehaus/sonar/runn...">http://repo1.maven.org/maven2/org/codehaus/sonar/runn.../sonar-runner-deb/2.4/sonar-runner-deb-2.4.zip</a> |
| License                    | GNU LGPL 3  |
| Developers                 | Julien Henry  |
| Issue tracker              | <a href="http://jira.sonarsource.com/browse/SONARUNNER">http://jira.sonarsource.com/browse/SONARUNNER</a>   |
| Sources                    | <a href="https://github.com/SonarSource/sonar-runner">https://github.com/SonarSource/sonar-runner</a>   |

**Features**

The SonarQube Scanner is recommended as the default launcher to analyze a project with SonarQube.

3. Extract the downloaded file to C:\Program Files\ . I have extracted it to C:\Program Files\sonar-runner-2.4.
4. That's it, SonarQube Runner is installed.

### Setting the Sonar Runner environment variables

Perform the following steps to set the %SONAR\_RUNNER\_HOME% environment variable:

Set the %SONAR\_RUNNER\_HOME% environment variable to the installation directory of SonarQube Runner by giving the following command:

```
setx SONAR_RUNNER_HOME "C:\Program Files\sonar-runner-2.4" /M
```

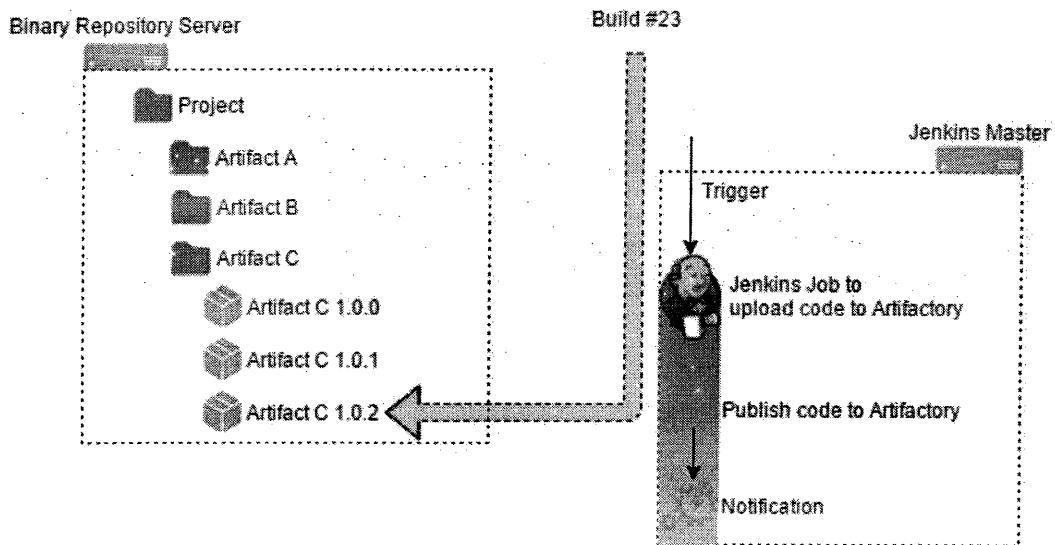
### Installing Artifactory

Continuous Integration results in frequent builds and packages. Hence, there is a need for a mechanism to store all this binary code (builds, packages, third-party plugins, and so on) in a system akin to a version control system.

Since, version control systems such as Git, TFS, and SVN store code and not binary files, we need a binary repository tool. A **binary repository** tool such as Artifactory or Nexus that is tightly integrated with Jenkins provides the following advantages:

- Tracking builds (Who triggers a build? What version of code in the VCS was build?)
- Dependencies
- Deployment history

The following image depicts how a binary repository tool such as Artifactory works with Jenkins to store build artifacts. In the coming sections, we will see how to achieve this by creating a Jenkins job to upload code to Artifactory.



we will use Artifactory to store our builds. Artifactory is a tool used to version control binaries. The binaries can be anything from built code, packages, executables, Maven plugins, and so on. We will install Artifactory on Windows 10. The steps are as follows:

1. Download the latest stable version of Artifactory from <https://www.jfrog.com/open-source/>. Download the **ZIP** archive.
2. Extract the downloaded file to C:\Program Files\. I have extracted it to C:\Program Files\artifactory-oss-4.3.2.

### Setting the Artifactory environment variables

Perform the following steps to set the %ARTIFACTORY\_HOME% environment variable:

```
setx ARTIFACTORY_HOME "C:\Program Files\artifactory-oss-4.3.2" /M
```

### Running the Artifactory application

To run Artifactory, open **Command Prompt** using admin privileges. Otherwise, this doesn't work.

Go to the location where the script to run Artifactory is present:

```
cd %ARTIFACTORY_HOME%\bin
```

Execute the installService.bat script:

This will open up a new **Command Prompt** window that will install Artifactory as a windows service.

To start Artifactory, open **Command Prompt** using admin privileges and use the following command:

```
sc start Artifactory
```



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>sc start Artifactory
SERVICE_NAME: Artifactory
    TYPE               : 10  WIN32_OWN_PROCESS
    STATE              : 2   START_PENDING
                           (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
    WIN32_EXIT_CODE    : 0   (0x0)
    SERVICE_EXIT_CODE : 0   (0x0)
    CHECKPOINT        : 0x0
    WAIT_HINT         : 0x7d0
    PID                : 5192
    FLAGS              :
C:\WINDOWS\system32>
```

Access Artifactory using the following link: <http://localhost:8081/artifactory/>.

### Creating a repository inside Artifactory

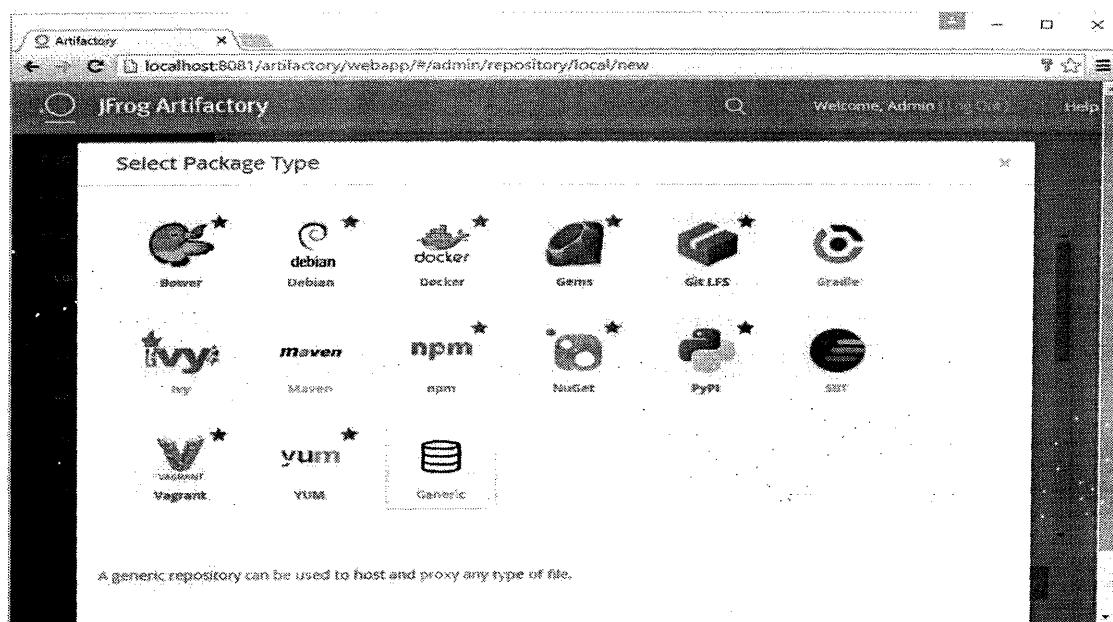
We will now create a repository inside Artifactory to store our package. The steps are as follows:

1. Log in to Artifactory using the **admin** account.
2. On the menu on the left-hand side, click on **Repositories** and then select **Local**. You will see a list of repositories that are present by default.
3. Click on the New button with a plus symbol, which is present on the right-hand side of the page.

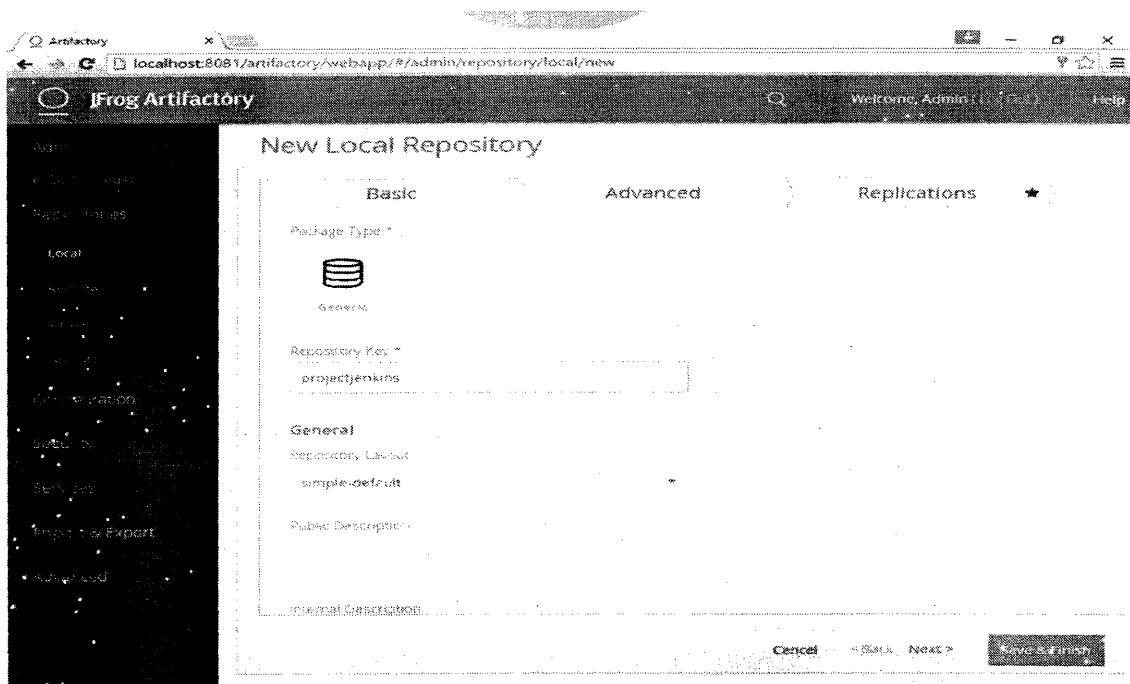
The screenshot shows the JFrog Artifactory interface. The left sidebar has a tree view with 'Admin', 'Artifacts', 'Local', 'Remote', 'Virtual', and 'Configuration' expanded. Under 'Local', 'Repositories' is selected. The main content area is titled 'Local Repositories' and shows '6 Repositories'. A table lists the repositories with columns for 'Repository Key', 'Type', 'Recalculat...', and 'Replicat...'. The repositories listed are: ext-release-local, ext-snapshot-local, libs-release-local, libs-snapshot-local, plugins-release-local, and plugins-snapshot-local. All are of type 'Maven'. There are 'Edit' and 'Delete' icons next to each row. A 'New' button is located in the top right corner of the main content area.

| Repository Key         | Type  | Recalculat... | Replicat... |
|------------------------|-------|---------------|-------------|
| ext-release-local      | Maven |               |             |
| ext-snapshot-local     | Maven |               |             |
| libs-release-local     | Maven |               |             |
| libs-snapshot-local    | Maven |               |             |
| plugins-release-local  | Maven |               |             |
| plugins-snapshot-local | Maven |               |             |

4. In the window that pops-up, select the package type as **Generic**.



5. Give a name in the **Repository Key \*** field. In our example I have used projectjenkins.
6. Leave the rest of the fields at their default values and click on the **Save & Finish** button.



7. As you can see in the following screenshot, there is a new repository named projectjenkins.

The screenshot shows the JFrog Artifactory interface. On the left, a sidebar menu includes 'Admin', 'Back to Main', 'Repositories' (selected), 'Local', 'Remote', 'Virtual', 'Caches', 'Configuration', 'SECURITY', 'Services', and 'Import & Export'. The main content area is titled 'Local Repositories' and shows a table of '7 Repositories'. The table has columns for 'Repository Key', 'Type', 'Recalculat...', and 'Replication...'. The repositories listed are: ext-release-local, ext-snapshot-local, libs-release-local, libs-snapshot-local, plugins-release-local, plugins-snapshot-local, and projectjenkins. The 'projectjenkins' entry is highlighted with a red border. A 'New' button is visible in the top right corner of the table header.

### Jenkins configuration

In the previous sections, we saw how to install and configure Artifactory and SonarQube along with SonarQube Runner. For these tools to work in collaboration with Jenkins, we need to install their respective Jenkins plugins.

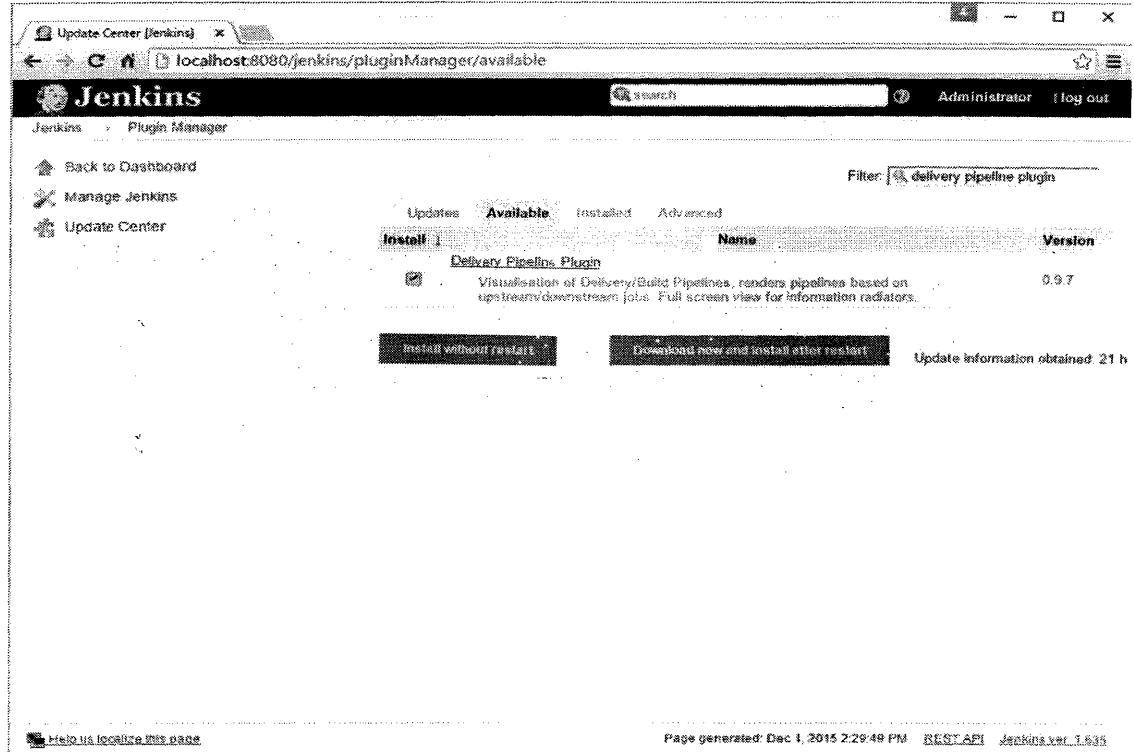
Also, we will see the installation of a special plugin named **delivery pipeline plugin**, which is used to give a visual touch to our Continuous Integration pipeline.

### Installing the delivery pipeline plugin

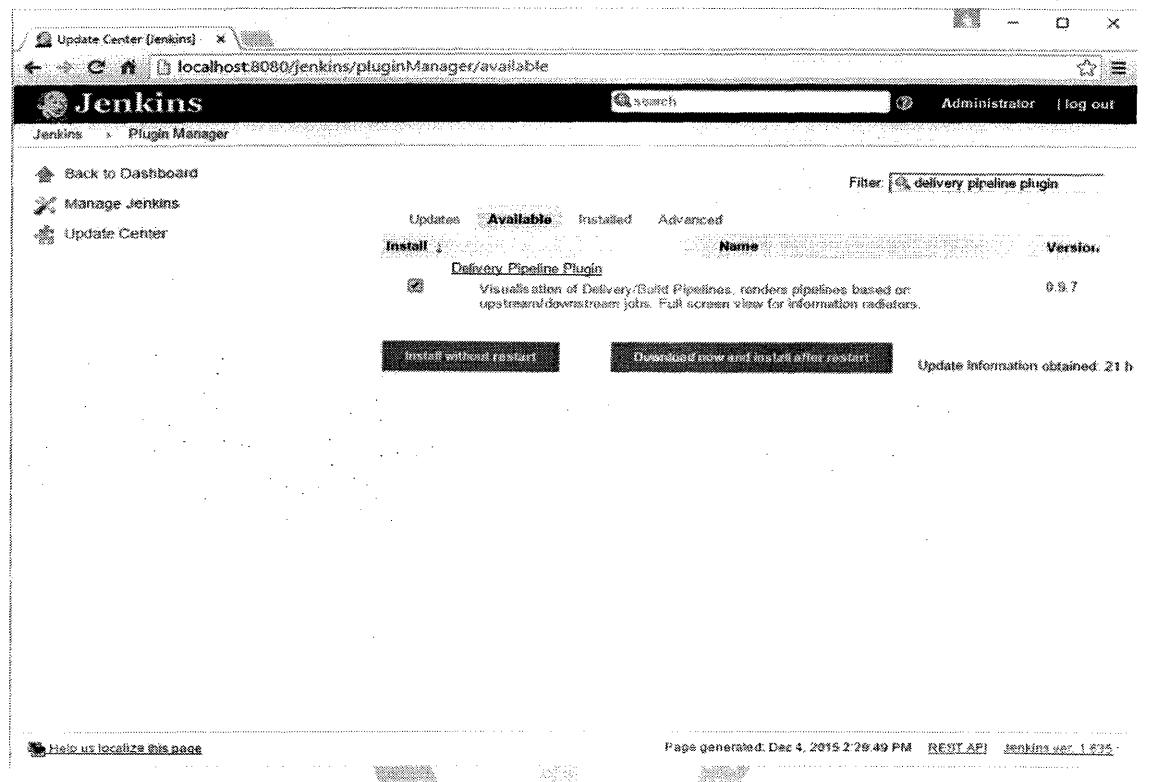
To install the delivery pipeline plugin, perform the following steps:

1. On the Jenkins Dashboard, click on the **Manage Jenkins** link. This will take you to the Manage Jenkins page.
2. Click on the **Manage Plugins** link and go to the **Available** tab.
3. Type **delivery pipeline plugin** in the search box.

4. Select **Delivery Pipeline Plugin** from the list and click on the **Install without restart** button.



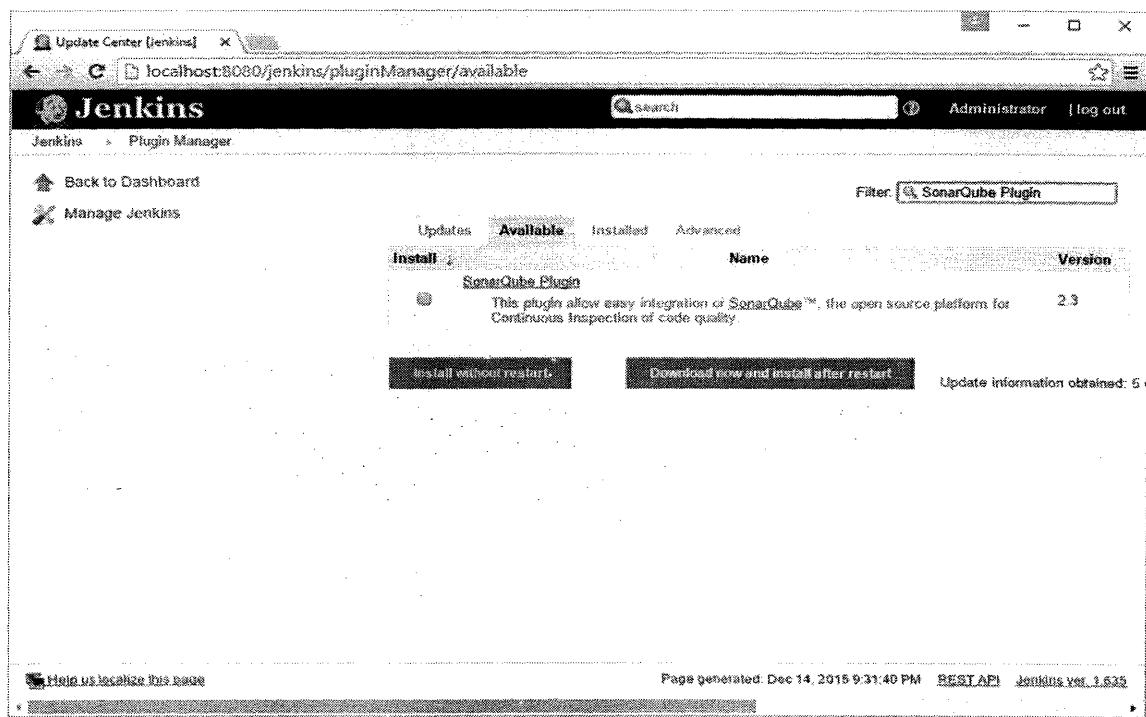
5. The download and installation of the plugin starts automatically. You can see **Delivery Pipeline Plugin** has some dependencies that get downloaded and installed.



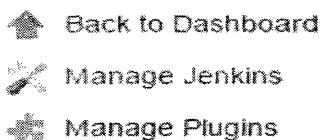
## Installing the SonarQube plugin

To install the SonarQube plugin, perform the following steps:

1. From the Jenkins Dashboard, click on the **Manage Jenkins** link. This will take you to the **Manage Jenkins** page.
2. Click on the **Manage Plugins** link and go to the **Available** tab.
3. Type SonarQube plugin in the search box. Select **SonarQube Plugin** from the list and click on the **Install without restart** button.



4. As it can be seen in the next screenshot, the plugin is installed immediately:



## Installing Plugins/Upgrades

### Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

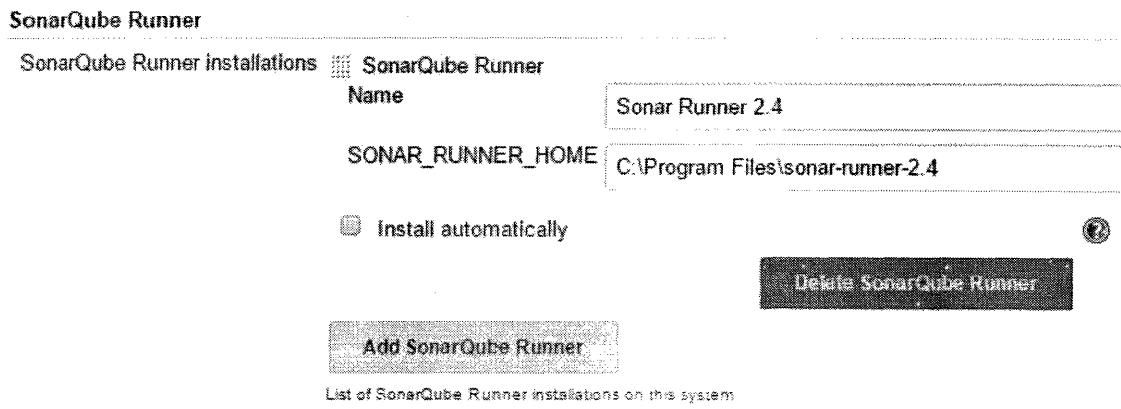
### SonarQube Plugin



➡ [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

5. Upon successful installation of the SonarQube Plugin, go to the **Configure System** link on the **Manage Jenkins** page.
6. Scroll down until you see the **SonarQube Runner** section and fill in the blanks as shown here:
  - You can name your SonarQube Runner installation using the **Name** field.
  - Set the **SONAR\_RUNNER\_HOME** value to the location where you have installed SonarQube Runner. In our example, it's `C:\Program Files\sonar-runner-2.4`.



7. Now, scroll down until you see the **SonarQube** section and fill in the blanks as shown here:
  - Name your SonarQube installation using the **Name** field.
  - Provide the **Server URL** field for the SonarQube. In our example, it's `http://localhost:9000`.

SonarQube

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

SonarQube installations

|                            |  |
|----------------------------|--|
| Name                       | Sonar  |
| Server URL                 | http://localhost:9000  |
| SonarQube account login    | Default is http://localhost:9000   |
| SonarQube account password | SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. |
| Disable                    | <input checked="" type="checkbox"/><br>Check to quickly disable SonarQube on all jobs.   |

**Advanced...**

**Delete SonarQube**

**Add SonarQube**

List of SonarQube installations

- Save the configuration by clicking on the **Save** button at the bottom of the screen.

### Note

You can add as many SonarQube instances as you want by clicking on the **Add SonarQube** button. Although not necessary, if you do, provide each SonarQube installation a different name.

### Installing the Artifactory plugin

To install the Artifactory plugin, perform the following steps:

- From the Jenkins Dashboard, click on the **Manage Jenkins** link. This will take you to the **Manage Jenkins** page.
- Click on the **Manage Plugins** link and go to the **Available** tab.
- Type Artifactory Plugin in the search box. Select **Artifactory Plugin** from the list and click on the **Install without restart** button.

The screenshot shows the Jenkins Plugin Manager interface. The URL is `localhost:8080/jenkins/pluginManager/available`. The 'Available' tab is selected. A search bar at the top right contains the text 'Artifactory Plugin'. A single plugin entry is shown:

| Name               | Version |
|--------------------|---------|
| Artifactory Plugin | 2.4.6   |

The description below the table states: "This plugin allows deploying Maven 2, Maven 3, Ivy and Gradle artifacts and build info to the Artifactory artifacts manager." Below the table are three buttons: "Install without restart", "Download now and install after restart", and "Update information obtained: 1".

4. The download and installation of the plugin starts automatically. You can see the Artifactory Plugin has some dependencies that get downloaded and installed.

The screenshot shows the Jenkins Update Center interface. The URL is `localhost:8080/jenkins/updateCenter/`. The 'Available' tab is selected. A search bar at the top right contains the text 'Artifactory'. The results section shows the 'Artifactory Plugin' listed under 'Available' with a status of 'Success'.

**Installing Plugins/Upgrades**

| Preparation                           | Success |
|---------------------------------------|---------|
| • Checking internet connectivity      | Success |
| • Checking update center connectivity | Success |
| • Success                             | Success |

**MapDB API Plugin** Success  
**Subversion Plugin** subversion plugin is already installed. Jenkins needs to be restarted for the update to take effect  
**Artifactory 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

5. Upon successful installation of the Artifactory Plugin, go to the **Configure System** link on the **Manage Jenkins** page.

6. Scroll down until you see the **Artifactory** section and fill in the blanks as shown here:

1. Provide the **URL** field as the default Artifactory URL configured at the time of installation. In our example, it is <http://localhost:8081/artifactory>.

2. In the **Default Deployer Credentials** field, provide the values for **Username** and **Password**.

**Artifactory**

Artifactory servers  Use the Credentials Plugin

Artifactory

URL

<http://localhost:8081/artifactory>



#### Default Deployer Credentials

Username

admin



Password

.....



**Test Connection**

Use Different Resolver Credentials

**Delete**

**Advanced...**

**Add**

List of Artifactory servers that projects will want to deploy artifacts and build info to

7. That's it. You can test the connection by clicking on the **Test Connection** button. You should see your Artifactory version displayed, as shown in the following screenshot:

**Artifactory**

Artifactory servers  Use the Credentials Plugin

 Artifactory

URL

**Default Deployer Credentials**

Username

Password

Found Artifactory 4.3.2

Use Different Resolver Credentials

 Add

List of Artifactory servers that projects will want to deploy artifacts and build info to

- Save the configuration by clicking on the **Save** button at the bottom of the screen.

### The Jenkins pipeline to poll the integration branch

This is the second pipeline of the two, both of which are part of the CI pipeline structure discussed in the previous chapter. This pipeline contains two Jenkins jobs. The first Jenkins job does the following tasks:

- It polls the integration branch for changes at regular intervals
- It executes the static code analysis
- It performs a build on the modified code
- It executes the integration tests

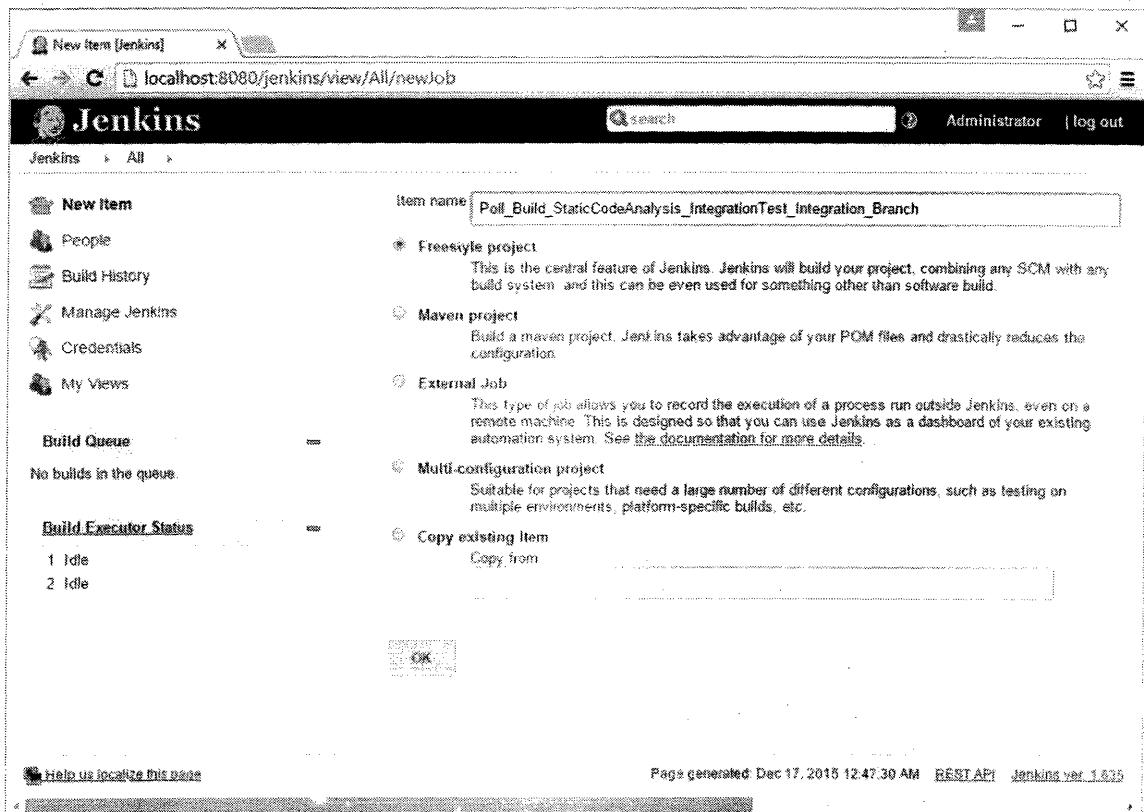
## Creating a Jenkins job to poll, build, perform static code analysis, and integration tests

I assume you are logged in to Jenkins as an admin and have privileges to create and modify jobs. From the Jenkins Dashboard, follow these steps:

1. Click on **New Item**.

2. Name your new Jenkins job `Poll_Build_StaticCodeAnalysis_IntegrationTest_Integration_Branch`.

3. Set the type of job as **Freestyle project** and click on **OK** to proceed.



### Polling the version control system for changes using Jenkins

This is a critical step in which we connect Jenkins with the version control system. This configuration enables Jenkins to poll the correct branch inside Git and download the modified code.

1. Scroll down to the **Source Code Management** section.

2. Select **Git** and fill in the blanks as follows:

- Specify **Repository URL** as the location of the Git repository. It can be a GitHub repository or a repository on a Git server. In our case, it's `/e/ProjectJenkins` because the Jenkins server and the Git server is on the same machine.
- Add `*/integration` in the **Branch to build** section, since we want our Jenkins job to poll integration branch. Leave rest of the fields as they are.

Source Code Management

|                            |   |   |  |
|----------------------------|---|---|--|
| <input type="radio"/> None | <input type="radio"/> CVS                                     | <input type="radio"/> CVS Projectset  | <input checked="" type="radio"/> Git   |
| Repositories               | Repository URL <input type="text" value="/e/ProjectJenkins"/> |   |  |
| Credentials                | <input type="button" value="- none -"/>                       | <input type="button" value="Add"/>  | <input type="button" value="Advanced..."/>   |
| Branches to build          |   | Branch Specifier (blank for 'any') <input type="text" value="*/integration"/> | <input type="button" value="Add Repository"/> <input type="button" value="Delete Repository"/> |
| Repository browser         |   | <input type="button" value="Auto"/>   | <input type="button" value="Add Branch"/> <input type="button" value="Delete Branch"/>         |

3. Scroll down to the **Build Triggers** section.

4. We want our Jenkins job to poll the feature branch every 5 minutes. Nevertheless, you are free to choose the polling duration that you wish depending on your requirements. Therefore, select the **Poll SCM** checkbox and add `H/5 * * * *` in the **Schedule** field.

## Build Triggers

- Trigger builds remotely (e.g., from scripts)
- Build after other projects are built
- Build periodically
- Poll SCM

Schedule

H/5 \* \* \* \*

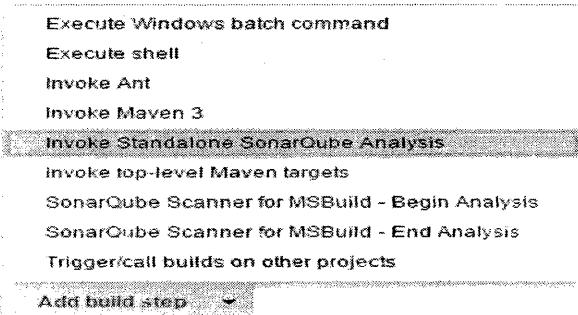
Would last have run at Thursday, 17 December, 2015 12:50:07 AM IST;  
would next run at Thursday, 17 December, 2015 12:55:07 AM IST.

Ignore post-commit hooks

## Creating a build step to perform static analysis

The following configuration tell Jenkins to perform a static code analysis on the downloaded code:

1. Scroll down to the Build section and click on the **Add build step** button. Select **Invoke Standalone SonarQube Analysis**.



2. Leave all the fields empty except the **JDK** field. Choose the appropriate version from the menu. In our example, it's **JDK 1.8**.

## Build

**Invoke Standalone SonarQube Analysis**

Task to run

JDK

JDK to be used for this sonar analysis

Path to project properties

Analysis properties

Additional arguments

JVM Options

## Creating a build step to build and integration test code

After successfully completing the static code analysis using SonarQube, the next step is to build the code and perform integration testing:

1. Click on the **Add build step** button again. Select **Invoke top-level Maven targets**.

## Build

- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke top-level Maven targets**
- Trigger/call builds on other projects

2. We will be present with the following options:

- Set the **Maven Version** field as **Maven 3.3.9**. Remember, this is what we configured on the **Configure System** page in the **Maven** section. If we had configured more than one Maven, we would have a choice here.
- Add the following line to the **Goals** section:

```
clean verify -Dsurefire.skip=true javadoc:javadoc
```

- Type `payslip/pom.xml` in the **POM** field. This tells Jenkins the location of the pom.xml file in the downloaded code.
3. The following screenshot displays the **Invoke top-level Maven targets** window and the mentioned fields:

The screenshot shows the 'Invoke top-level Maven targets' configuration window. It includes fields for Maven Version (set to Maven 3.3.9), Goals (containing the command `mvn clean verify -Dsurefire.skip=true javadoc:javadoc`), POM (containing the path `payslip/pom.xml`), Properties (empty), JVM Options (empty), Use private Maven repository (disabled), Settings file (set to 'Use default maven settings'), Global Settings file (set to 'Use default maven global settings'), and a Delete button at the bottom right.

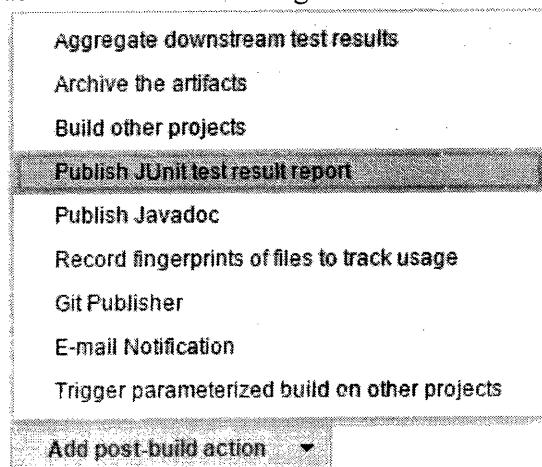
|                              |  |
|------------------------------|--|
| Maven Version                | Maven 3.3.9  |
| Goals                        | <code>mvn clean verify -Dsurefire.skip=true javadoc:javadoc</code> |
| POM                          | <code>payslip/pom.xml</code>                                       |
| Properties                   |  |
| JVM Options                  |  |
| Use private Maven repository | (disabled)   |
| Settings file                | Use default maven settings   |
| Global Settings file         | Use default maven global settings                                  |
| Delete                       |  |

4. Let's see the Maven command inside the **Goals** field in detail:

- clean will clean any old built files
- -Dsurefire.skip=true will execute the integration test
- javadoc:javadoc will tell Maven to generate Java documentation

5. Scroll down to the **Post build Actions** section.

6. Click on the **Add post-build action** button and select **Publish JUnit test result report**, as shown in the following screenshot:



7. Under the **Test report XMLs** field, type `payslip/target/surefire-reports/*.xml`.

**Post-build Actions**

**Publish JUnit test result report**

Test report XMLs

`payslip/target/surefire-reports/*.xml`

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'mvnProject/target/test-reports/\*.xml'. Basedir of the fileset is the workspace root.

**Retain long standard output/error**

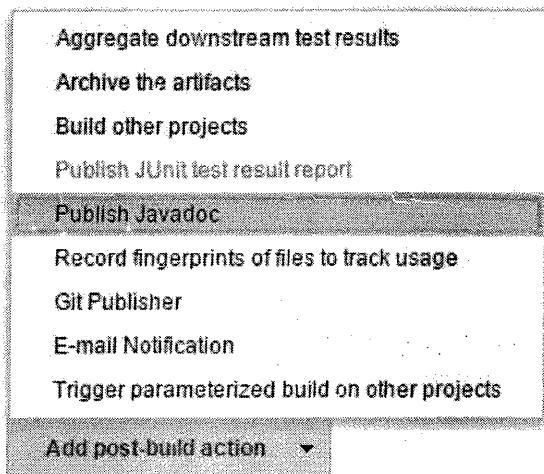
Health report amplification factor

`1.0`

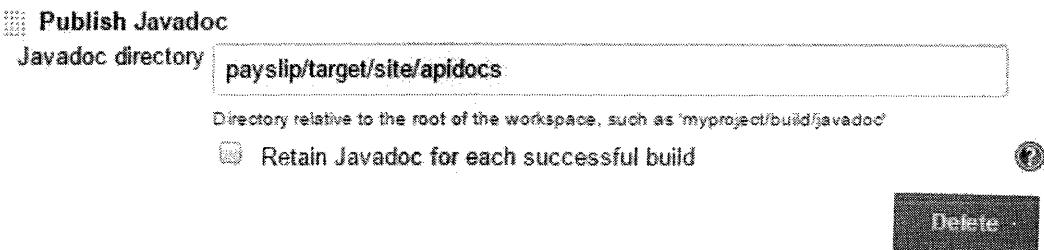
1% failing tests scores as 99% health. 5% failing tests scores as 95% health

**Delete**

8. Next, click on the **Add post-build action** button. This time, select **Publish Javadoc**.



9. Type the path `payslip/target/site/apidocs` under the **Javadoc directory** field.

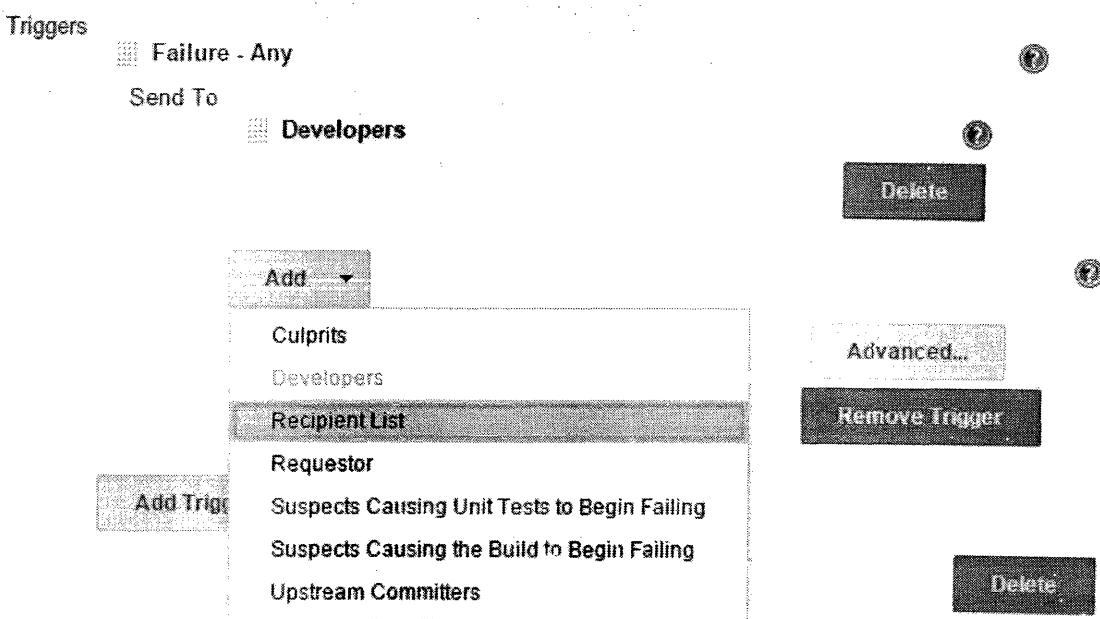


## Configuring advanced e-mail notifications

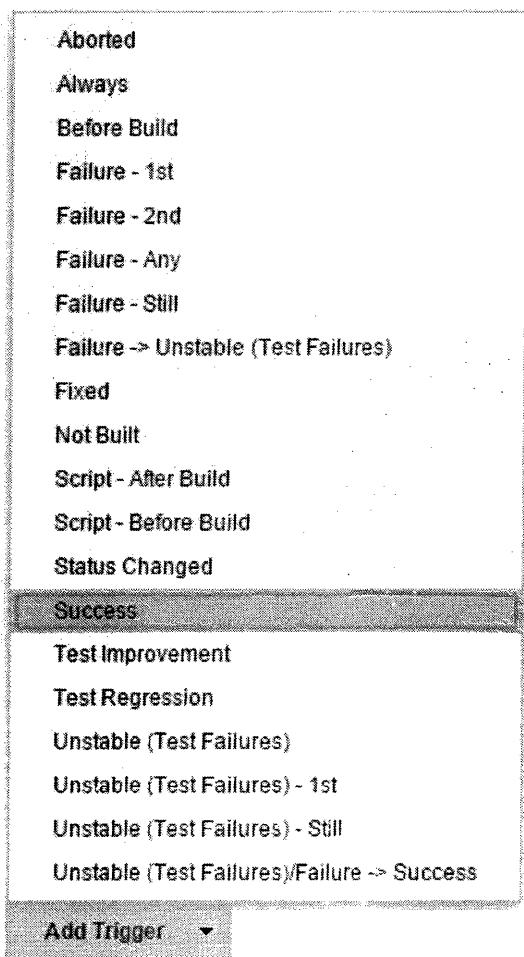
Notification forms are an important part of CI. In this section, we will configure the Jenkins job to send e-mail notifications based on few conditions. Let's see the steps in detail:

1. Click on the **Add post-build action** button and select **Editable Email Notification**.
2. Configure **Editable Email Notification** as shown here:
  - Under **Project Recipient List**, add the e-mail IDs separated by commas. You can add anyone whom you think should be notified for build and unit test success/failure.
  - You can add the e-mail ID of the Jenkins administrator under **Project Reply-To List**.

- Set Content Type as HTML (text/html).
3. Leave all the rest of the options at their default values.
  4. Now, click on the **Advanced Settings...** button.
  5. By default, there is a trigger named **Failure – Any** that sends e-mail notifications in the event of failure (any kind of failure).
  6. By default, the **Send To** option is set to **Developers**.
  7. But we don't want that, we have already defined whom to send e-mails to. Therefore, click on the **Add** button and select the **Recipient List** option, as shown in the following screenshot:



8. Let's add another trigger to send an e-mail when the job is successful.
9. Click on the **Add Trigger** button and select the **Success** option.



10. Configure this new success trigger in the similar fashion by removing **Developers** and adding **Recipient List** under the **Send To** section. Finally, everything should look like this:

**Triggers****Failure - Any****Send To****Recipient List****Delete****Add ▾****Advanced...****Remove Trigger****Success****Send To****Recipient List****Delete****Add ▾****Advanced...****Remove Trigger****Add Trigger ▾****Delete**

11. Save the Jenkins job by clicking on the **Save** button.

**Creating a Jenkins job to upload code to Artifactory**

The second Jenkins job in the pipeline uploads the build package to Artifactory (binary code repository). From the Jenkins Dashboard:

1. Click on **New Item**.

2. Name your new Jenkins job **Upload\_Package\_To\_Artifactory**.
3. Select the type of job as **Freestyle project** and click on **OK** to proceed.
4. Scroll down to the **Build Triggers** section and select the **Build after other projects are built** option.
5. Under **Projects** to **watch field**, type **Poll\_Build\_StaticCodeAnalysis\_IntegrationTest\_Integration\_Branch**.
6. Select the **Trigger only if build is stable** option.

### Note

In this way, we are telling Jenkins to initiate the current Jenkins job **Upload\_Package\_To\_Artifactory** only after the **Poll\_Build\_StaticCodeAnalysis\_IntegrationTest\_Integration\_Branch** job has completed successfully.

### Configuring the Jenkins job to upload code to Artifactory

The following configuration will tell Jenkins to look for a potential .war file under the Jenkins job's workspace to upload it to Artifactory:

1. Scroll down further until you see the **Build Environment** section. Check the **Generic-Artifactory Integration** option. Doing so will display a lot of options for Artifactory. Fill them in as follows:
  - **Artifactory deployment server** is your Artifactory web link. In our case, it is <http://localhost:8081/artifactory>.
  - Next is the **Target Repository** field. Select **projectjenkins** from the drop-down menu. You will notice that all the repositories present inside Artifactory will be listed here.
  - To refresh the list, click on the **Refresh Repositories** button.

- Add `**/*.war=>${BUILD_NUMBER}` to the **Published Artifacts** field.
- Leave rest of the fields at their default values.

#### Build Environment

- Ant/Ivy-Artifactory Integration
- Create Delivery Pipeline version
- Generic-Artifactory Integration

#### Artifactory Configuration

##### Deployment Details

Artifactory deployment server

Target Repository   Items refreshed successfully

- Override default credentials

##### Published Artifacts

`**/*.war=>${BUILD_NUMBER}`

##### Deployment properties

#### Resolution Details

Artifactory resolver server

- Override default credentials

Resolved Artifacts (requires Artifactory Pro)

## 2. Let's see what the **Published Artifacts** field means.

- `**/*.war` tells Jenkins to search for and pick a WAR file anywhere inside the current workspace
- `${BUILD_NUMBER}` is a Jenkins variable that stores the current build number

- Finally, `**/*.war=>${BUILD_NUMBER}` means search and pick any .war file present inside the workspace, and upload it to Artifactory with the current build number as its label
3. Scroll down to the **Build** section and add a build step to **Execute Windows batch command**.
  4. Add the following code into the **Command** section:

```
COPY  
/YC:\Jenkins\jobs\Poll_Build_StaticCodeAnalysis_IntegrationTest_Integration_Branch  
\workspace\payslip\target\payslip-0.0.1.war %WORKSPACE%\payslip-0.0.1.war
```

## Build

### Execute Windows batch command

#### Command

```
COPY /Y  
C:\Jenkins\jobs\Poll_Build_StaticCodeAnalysis_IntegrationTest_Integration_Branch  
\workspace\payslip\target\payslip-0.0.1.war %WORKSPACE%\payslip-0.0.1.war
```

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

[Delete](#)

## Note

This simply copies the payslip-0.0.1.war package file generated in the previous Jenkins job from its respective workspace to the current job's workspace. This build step happens first and then the upload to Artifactory takes place.

5. Configure advanced e-mail notifications exactly the same way as mentioned earlier.
6. Save the Jenkins job by clicking on the **Save** button.

## **6. Continuous Delivery Using Jenkins**

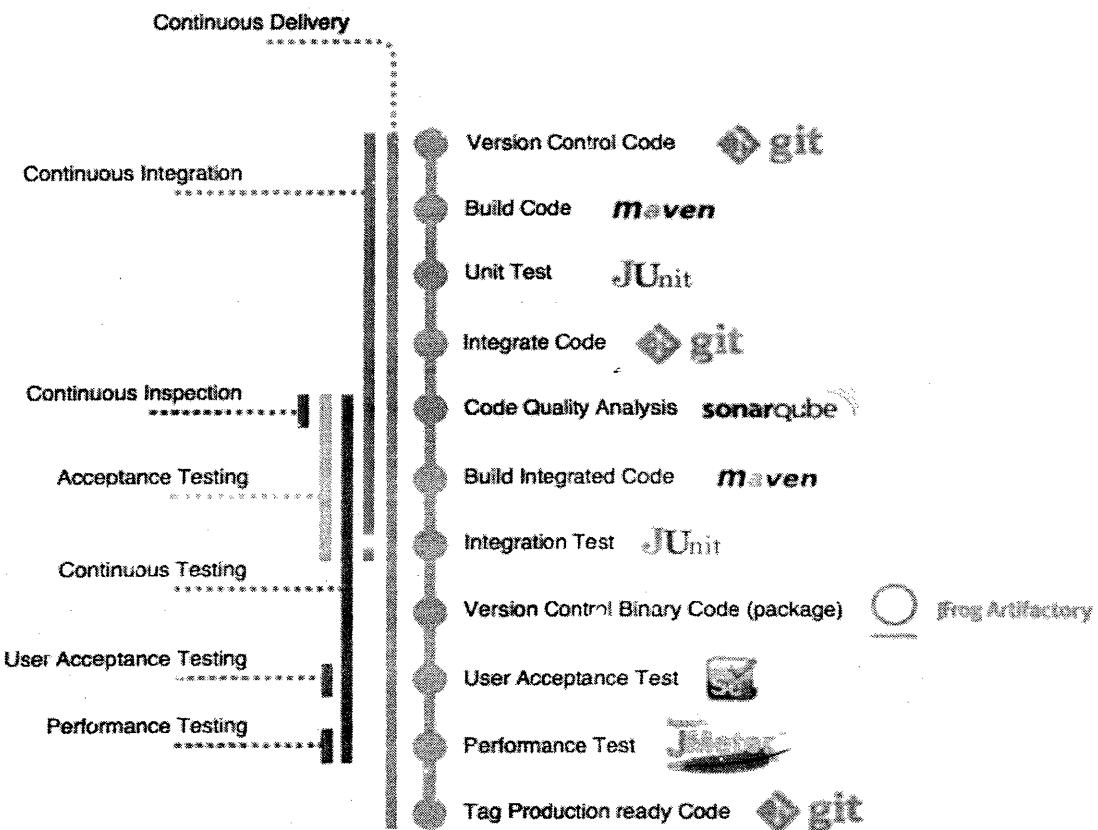
### **What is Continuous Delivery?**

Continuous Delivery is the software engineering practice wherein production-ready features are produced in a continuous manner.

When we say production-ready features, we mean only those features that have passed the following check points:

- Unit testing
- Integration
- Static code analysis (code quality)
- Integration testing
- System integration testing
- User acceptance testing
- Performance testing
- End-to-end testing
- However, the list is not complete. You can incorporate as many types of testing as you want to certify that the code is production ready.
- From the preceding list, the first four check points are covered as part of the Continuous Integration Design discussed in the previous chapter. This Continuous Integration Design, when combined with deployments (not listed here) and with all sorts of automated testing can be safely called Continuous Delivery.

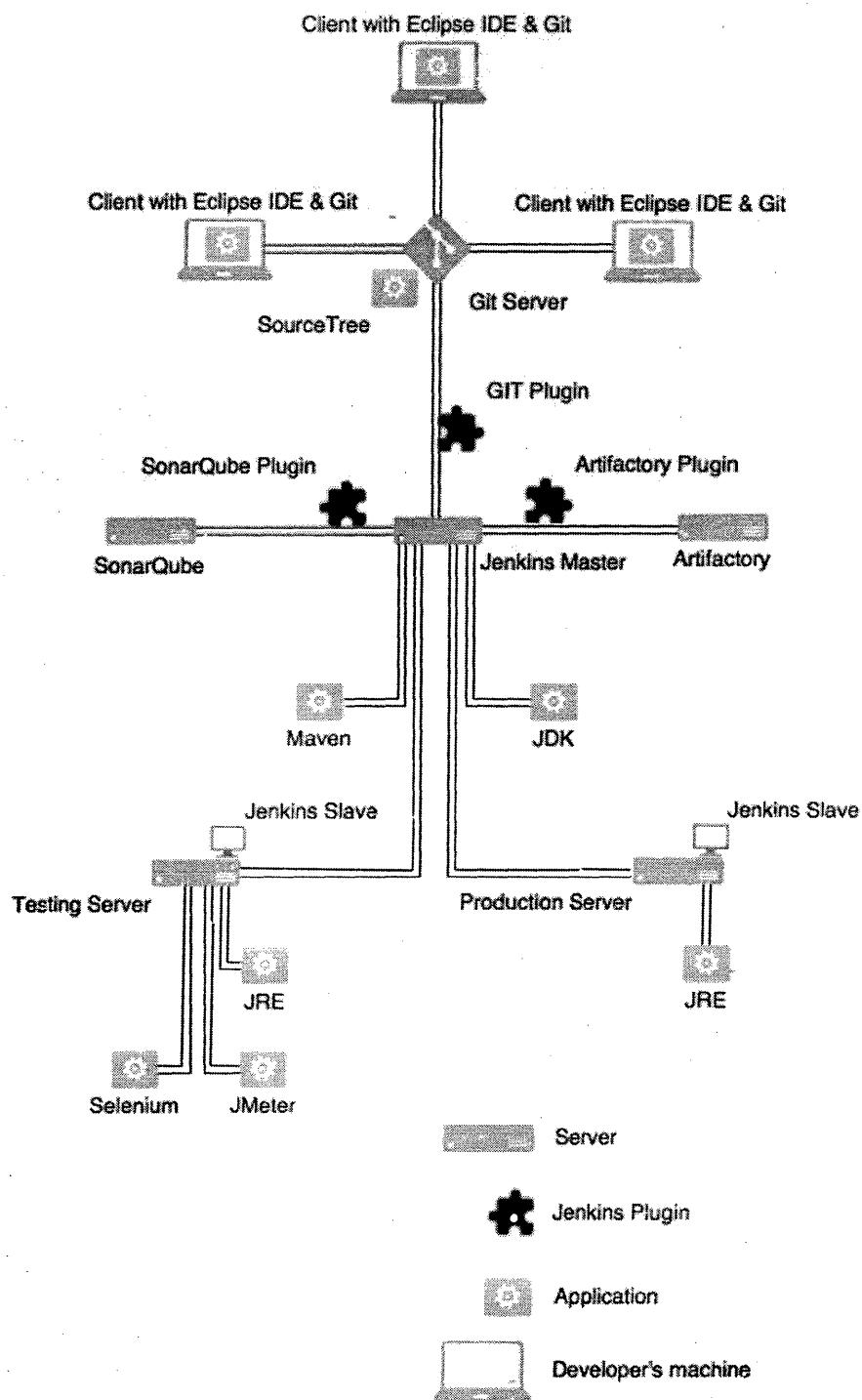
- In other words, Continuous Delivery is an extension of the Continuous Integration methodology to the deployment and testing phases of a **Software Development Life Cycle (SDLC)**. Testing in itself is a vast area.
- In any organization, big or small, the previously mentioned testing is either performed on a single environment or on multiple environments. If there are multiple testing environments, then there is a need to deploy the package in all those testing environments. Therefore, deployment activities are also part of Continuous Delivery.
- The next figure will help us understand the various terminologies that were discussed just now. The various steps a software code goes through, from its inception to its utilization (development to production) are listed in the following figure. Each step has a tool associated with it, and each one is part of a methodology:



**Continuous Delivery Design**

The next figure demonstrates how Jenkins fits in as a CD server in our Continuous Delivery Design, along with the other DevOps tools:

- The developers have the Eclipse IDE and Git installed on their machines. This Eclipse IDE is internally configured with the Git server. This enables the developers to clone the feature branch from the Git server on their machines.
- The Git server is connected to the Jenkins master server using the Git plugin. This enables Jenkins to poll the Git server for changes.
- The Apache Tomcat server, which hosts the Jenkins master, also has Maven and JDK installed on it. This enables Jenkins to build the code that has been checked-in on the Git Server.
- Jenkins is also connected to the SonarQube server and the Artifactory server using the SonarQube plugin and the Artifactory plugin, respectively.
- This enables Jenkins to perform a static code analysis of the modified code. Once all the build, quality analysis, and integration testing is successful, the resultant package is uploaded to the Artifactory for further use.
- The package also gets deployed on a testing server that contains testing tools such as JMeter, TestNG, and Selenium. Jenkins, in collaboration with the testing tools, will perform user acceptance tests and performance tests on the code.

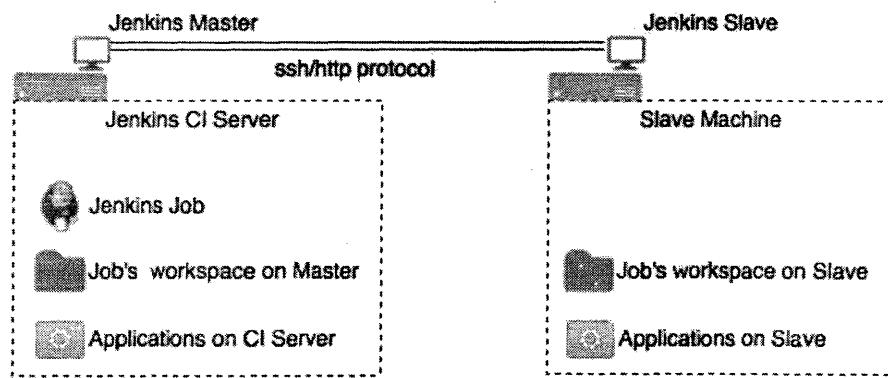


## Jenkins configuration

### Configuring Jenkins slaves on the testing server

In the previous section, we saw how to configure the testing server. Now, we will see how to configure the Jenkins slave to run on the testing server. In this way, the Jenkins master will be able to communicate and run Jenkins jobs on the slave. Follow the next few steps to set up the Jenkins slave:

**Jenkins Master-Slave Architecture**



**Timeline of a Jenkins Job during execution**

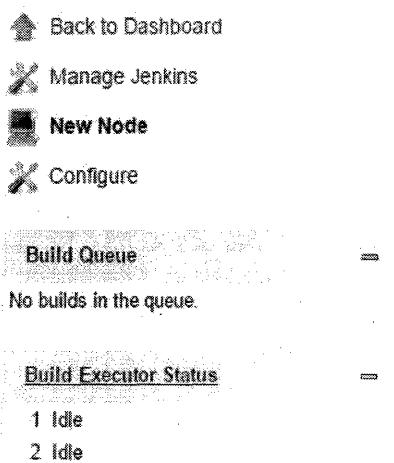
- Jenkins Job triggers from Jenkins Master
- Artifacts if any, are copied to Jenkins workspace on Slave
- Build steps run on the Slave machine
- While the build runs on the Slave machine, it might use applications present on the Slave machines or the Jenkins CI Server. It can also call the application present elsewhere
- Post build action are performed either on the Slave machine or on the Jenkins CI Server
- Logs are stored on the Job's workspace on the Master

1. Log in to the testing server and open the Jenkins dashboard from the browser using the following link: <http://<ip address>:8080/jenkins/>. Remember, you are accessing the Jenkins master from the testing server. <ip address> is the IP of your Jenkins server.
2. From the Jenkins dashboard, click on **Manage Jenkins**. This will take you to the **Manage Jenkins** page. Make sure you have logged in as an Admin in Jenkins.
3. Click on the **Manage Nodes** link. In the following screenshot, we can see that the master node (which is the Jenkins server) is listed:

The screenshot shows the Jenkins Manage Nodes interface. On the left, there's a sidebar with links: Back to Dashboard, Manage Jenkins, New Node, and Configure. The main area has two sections: 'Build Queue' (No builds in the queue) and 'Build Executor Status'. The 'Build Executor Status' section shows 1 Idle and 2 Idle executors. A table lists nodes: 'master' (Windows 10 (amd64), In sync, 289.89 GB free disk space, 4.92 GB free swap space, 289.89 GB free temp space, 0ms response time) and 'Data obtained' (3 min 30 sec, 3 min 29 sec). A 'Refresh status' button is at the bottom right.

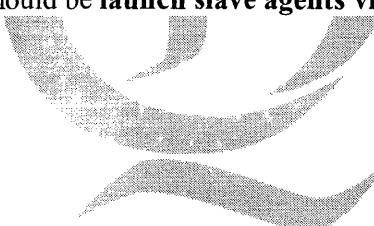
| S | Name          | Architecture       | Clock Difference | Free Disk Space | Free Swap Space | Free Temp Space | Response Time |
|---|---------------|--------------------|------------------|-----------------|-----------------|-----------------|---------------|
| 1 | master        | Windows 10 (amd64) | In sync          | 289.89 GB       | 4.92 GB         | 289.89 GB       | 0ms           |
| 2 | Data obtained |                    | 3 min 30 sec     | 3 min 29 sec    | 3 min 29 sec    | 3 min 29 sec    | 3 min 29 sec  |

4. Click on the **New Node** button on the left-hand side panel. Name the new node Testing\_Server and select the option **Dumb Slave**. Click on the **OK** button to proceed:



The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links: Back to Dashboard, Manage Jenkins, New Node (highlighted with a red box), and Configure. Below that are sections for Build Queue (No builds in the queue) and Build Executor Status (1 Idle, 2 Idle). On the right, a configuration dialog is open for adding a new node. It has a 'Node name' field containing 'Testing\_Server'. There are two options: 'Dumb Slave' (selected) and 'VirtualBox Slave'. The 'Dumb Slave' section contains a detailed description: 'Adds a plain, dumb slave to Jenkins. This is called "dumb" because Jenkins doesn't provide higher level of integration with these slaves, such as dynamic provisioning. Select this type if no other slave types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.' A large 'OK' button is at the bottom of the dialog.

5. Add some description, as shown in the next screenshot. The **Remote root directory** value should be the local user account on the testing server. It should be /home/<user>. The **Labels** filed is extremely important, so add Testing as the value.
6. The **Launch Method** should be **launch slave agents via Java Web Start**:



Back to Dashboard

Manage Jenkins

New Node

Configure

**Build Queue**  
No builds in the queue.

**Build Executor Status**  
1 Idle  
2 Idle

|                           |   |
|---------------------------|---|
| Name                      | Testing_Server                              |
| Description               | Jenkins slave to be on testing server       |
| # of executors            | 1   |
| Remote root directory     | /home/nikhil                                |
| Labels                    | Testing                                     |
| Usage                     | Utilize this node as much as possible       |
| Launch method             | Launch slave agents via Java Web Start      |
| Tunnel connection through |   |
| JVM options               |   |
| Availability              | Keep this slave on-line as much as possible |

**Node Properties**

Environment variables

Tool Locations

**Save**

7. Click on the **Save** button. As you can see from the following screenshot, the Jenkins node on the testing server is configured but it's not running yet:

The screenshot shows the Jenkins dashboard. At the top, there are links: Back to Dashboard, Manage Jenkins, New Node, and Configure. Below these are two main sections: 'Build Queue' (No builds in the queue) and 'Build Executor Status'. The 'Build Executor Status' section lists two nodes: 'master' (Windows 10 (amd64), 1 idle, 2 idle) and 'Testing\_Server' (offline). A table provides detailed status for each node, including response time (0ms for master, Time out for last 1 try for Testing\_Server), architecture, clock difference, free disk space, free swap space, free temp space, and data obtained (41 sec for both). A 'Refresh status' button is at the bottom.

| S | Name           | Architecture       | Clock Difference | Free Disk Space | Free Swap Space | Free Temp Space | Response Time           |
|---|----------------|--------------------|------------------|-----------------|-----------------|-----------------|-------------------------|
| 1 | master         | Windows 10 (amd64) | In sync          | 289.89 GB       | 4.81 GB         | 289.89 GB       | 0ms                     |
| 2 | Testing_Server |                    | N/A              | N/A             | N/A             | N/A             | Time out for last 1 try |

- Click on the Testing\_Server link from the list of nodes. You will see something like this:

This screenshot shows the configuration page for the 'Testing\_Server' Jenkins slave. It includes a 'Launch' button, instructions for connecting via browser or command line, a Java command for connecting, and a note that it was created by 'Administrator'.

Connect slave to Jenkins one of these ways:

- Launch Launch agent from browser on slave
- Run from slave command line:  
java -jar slave.jar -jnlpUrl http://192.168.1.101:8080/jenkins/computer/Testing\_Server/slave-a\_b\_st.jnlp -secret 916d8164f7ccc1b6fb4521d0c9523ee3b9933328f4cc9cd5e75b4cd65f139f7

Created by Administrator

## Labels

Testing

## Projects tied to Testing\_Server

None

- You can either click on the orange **Launch** button, or you can execute the long command mentioned below it from the terminal.
- If you choose the latter option, then download the slave.jar file mentioned in the command by clicking on it. It will be downloaded to /home/<user>/Downloads/.

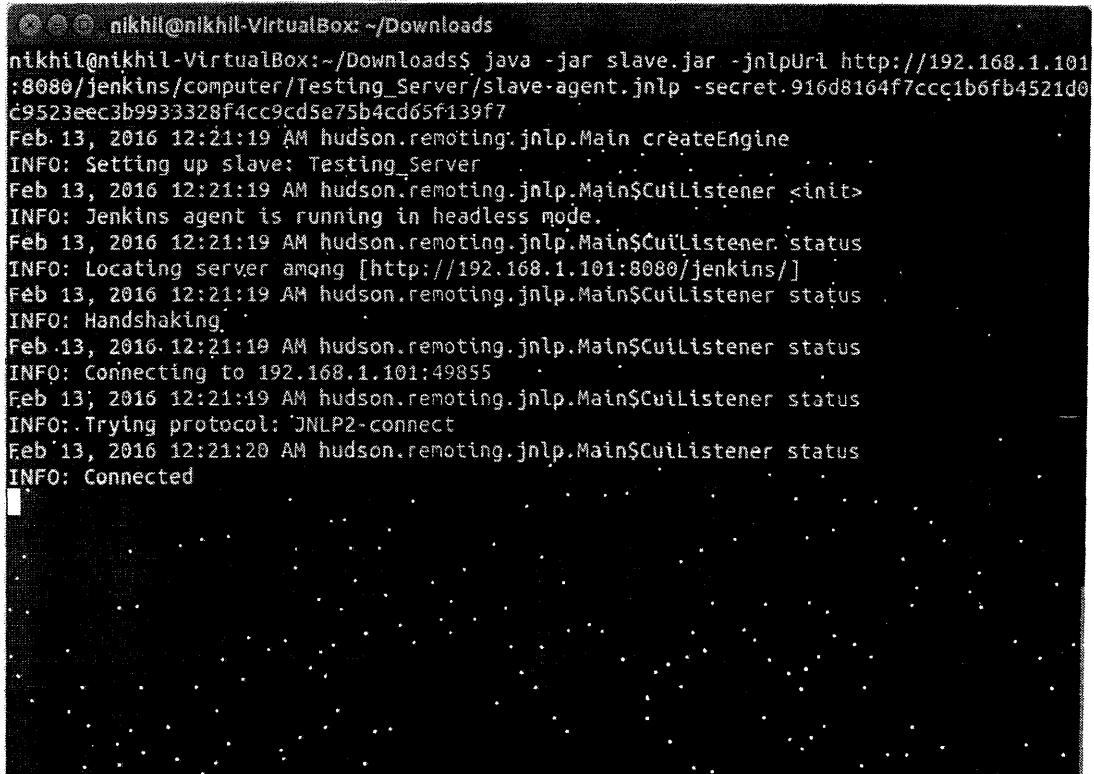
11. Execute the following commands in sequence:

**cd Downloads**

```
java -jar slave.jar -jnlpUrl  
http://192.168.1.101:8080/jenkins/computer/Testing_Server/slave-agent.jnlp -secret  
916d8164f7ccc1b6fb4521d0c9523eec3b9933328f4cc9cd5e75b4cd65f139f7
```

### Note

The preceding command is machine specific. Do not copy and paste and execute the same. Execute the command that appears on your screen.



```
nikhil@nikhil-VirtualBox:~/Downloads$ java -jar slave.jar -jnlpUrl http://192.168.1.101:8080/jenkins/computer/Testing_Server/slave-agent.jnlp -secret.916d8164f7ccc1b6fb4521d0c9523eec3b9933328f4cc9cd5e75b4cd65f139f7  
Feb 13, 2016 12:21:19 AM hudson.remoting.jnlp.Main createEngine  
INFO: Setting up slave: Testing_Server  
Feb 13, 2016 12:21:19 AM hudson.remoting.jnlp.Main$CuilListener <init>  
INFO: Jenkins agent is running in headless mode.  
Feb 13, 2016 12:21:19 AM hudson.remoting.jnlp.Main$CuilListener status  
INFO: Locating server among [http://192.168.1.101:8080/jenkins/]  
Feb 13, 2016 12:21:19 AM hudson.remoting.jnlp.Main$CuilListener status  
INFO: Handshaking.  
Feb 13, 2016 12:21:19 AM hudson.remoting.jnlp.Main$CuilListener status  
INFO: Connecting to 192.168.1.101:49855  
Feb 13, 2016 12:21:19 AM hudson.remoting.jnlp.Main$CuilListener status  
INFO: Trying protocol: JNLP2-connect  
Feb 13, 2016 12:21:20 AM hudson.remoting.jnlp.Main$CuilListener status  
INFO: Connected
```

12. The node on testing server is up and running, as shown in the following screenshot:

Back to Dashboard

Manage Jenkins

New Node

Configure

## Build Queue

No builds in the queue.

## Build Executor Status

### master

1 Idle

2 Idle

### Testing\_Server

1 Idle

| S             | Name           | Architecture       | Clock Difference | Free Disk Space | Free Swap Space | Free Temp Space | Response Time |
|---------------|----------------|--------------------|------------------|-----------------|-----------------|-----------------|---------------|
|               | master         | Windows 10 (amd64) | In sync          | 289.87 GB       | 4.54 GB         | 289.87 GB       | 0ms           |
|               | Testing_Server | Linux (amd64)      | 1.3 sec ahead    | 24.31 GB        | 2.00 GB         | 24.31 GB        | 3515ms        |
| Data obtained | 8 min 8 sec    | 8 min 7 sec        | 8 min 7 sec      | 8 min 7 sec     | 8 min 7 sec     | 8 min 7 sec     | 8 min 7 sec   |

Refresh status



## Slave agents via SSH tunneling

The widely preferred approach for Jenkins slave nodes on Linux, Unix, and OS X hosts is to leverage SSH tunneling. This launch method starts by sending commands over an SSH connection, which downloads the slave.jar and launches the slave agent on the host. For the installation process to work, Java 1.7 or later must be installed; the slave host needs to be reachable from the master, and the account specified in Jenkins will need to have SSH logon rights for the target machine.

The SSH launch method provides a number of valuable features that make this an attractive option when connecting Jenkins slave agents to the master. These benefits include:

- More reliable connectivity and stability
- Encrypted communications
- Auto restart and reconnect functionality

- No need for slave services or `init.d` scripts

To use the SSH launch method, select **Launch slave agents on Unix machines via SSH**, SSH authorized user credentials, Host IP address, and click the SAVE button to create the new slave node. Once the slave has been saved, Jenkins will automatically attempt to connect to the slave and install the slave agent using SSH and the credentials provided.

The screenshot shows the configuration for a new Jenkins slave node named "SSH Slave Node1". The fields are as follows:

|                       |  |
|-----------------------|--|
| Name                  | SSH Slave Node1                              |
| Description           |  |
| # of executors        | 1  |
| Remote root directory | /var/lib/jenkins                             |
| Labels                |  |
| Usage                 | Utilize this node as much as possible        |
| Launch method         | Launch slave agents on Unix machines via SSH |
| Host                  | 10.10.10.136                                 |
| Credentials           | exeterstudios/******** (exeterstudios.com)   |

At the bottom right of the dialog box, there is an "Advanced..." button.

When configuring a new SSH slave node, the best approach for configuring authentication is to use the Jenkins credential management system. This will store the login and password information for the SSH slaves in Jenkins directly. The Jenkins credential management system allows the Jenkins administrator to manage credentials and later reuse them when executing jobs, connecting SSH slave agents, and connecting Jenkins to third-party services. To add usernames and passwords to the credentials manager, navigate to the credential management system and select add credentials in the UI:

### [Manage Jenkins](#) | [Manage Credentials](#) | [Add credentials](#)

Once any credentials have been added, they will appear as available credentials in the SSH Host **Credentials** dropdown

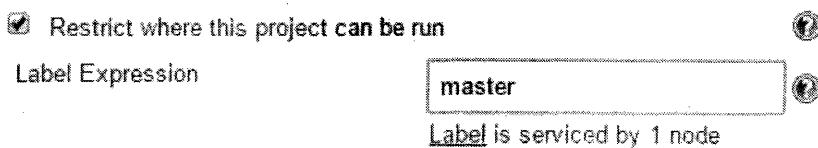
Upon saving the configuration for an SSH slave node, Jenkins will immediately attempt to connect to and install the slave agent service on the target host.

Detailed logs related to the connection can be viewed by clicking the  Log button on the left-hand side of the slave node status screen. If everything was successful, the logs will contain text similar to the following:

```
JNLP agent connected from /127.0.0.1  
<===[JENKINS REMOTING CAPACITY]==>Slave.jar version: 2.49  
This is a Unix slave  
Slave successfully connected and online.
```

### Modifying the project to run on selected node

1. From the Jenkins dashboard, begin by clicking on any existing Jenkins job.
2. Click on the **Configure** link present on the left-hand side panel.
3. Scroll down until you see the **Advanced Project Options** section.
4. From the options, choose **Restrict where this project can be run** and add master as the value for the **Label Expression** field, as shown in the following screenshot:



### Creating a Jenkins job to deploy code on the testing server

- It deploys packages to the testing server using the `BUILD_NUMBER` variable

- It passes the **GIT\_COMMIT** and **BUILD\_NUMBER** variable to the Jenkins job that performs the user acceptance test

Follow the next few steps to create it:

1. From the Jenkins dashboard, click on **New Item**.
2. Name your new Jenkins job **Deploy\_Artifact\_To\_Testing\_Server**.
3. Select the type of job as **Multi-configuration project** and click on **OK** to proceed.
4. Scroll down until you see **Advanced Project Options**. Select **Restrict where this project can be run**.
5. Add Testing as the value for **Label Expression**.
6. Scroll down to the **Build** section.
7. Click on the **Add build step** button and choose the option **Execute shell**.
8. Add the following code in the **Command** field:

➤ The first line of the command downloads the respective package from Artifactory to the Jenkins workspace:

```
wget
```

```
http://192.168.1.101:8081/artifactory/projectjenkins/$BUILD_NUMBER/payslip-0.0.1.war
```

➤ The second line of command deploys the downloaded package to the Apache Tomcat server's webapps directory:

```
mv payslip-0.0.1
```

```
.war /opt/tomcat/webapps/payslip-0.0.1.war -f
```

### Build

#### Execute shell

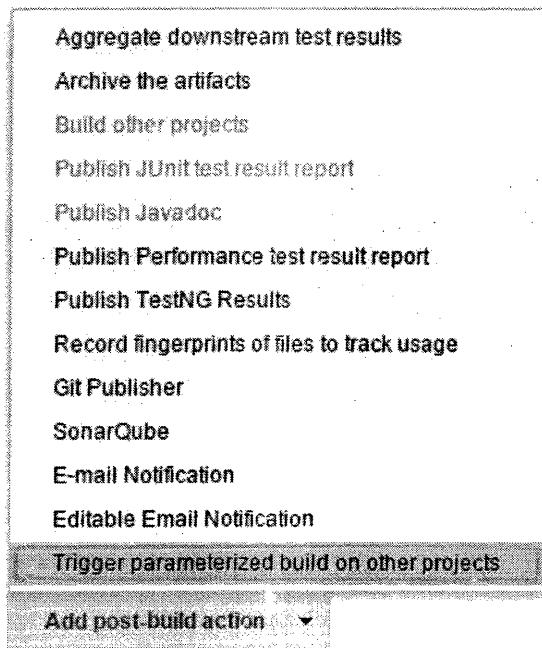
##### Command

```
wget http://192.168.1.101:8081/artifactory/projectjenkins/$BUILD_NUMBER/payslip-0.0.1.war  
mv payslip-0.0.1.war /opt/tomcat/webapps/payslip-0.0.1.war -f
```

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

[Delete](#)

9. Scroll down to the **Post-build Actions** section. Click on the **Add post-build action** button. From the drop-down list, choose the option **Trigger parameterized build on the other projects**:

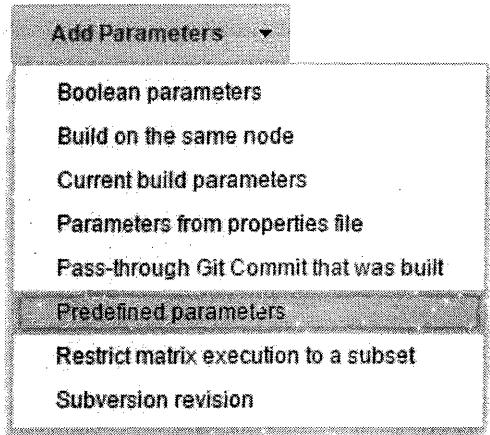


10. Add the values as shown in the screenshot:

The screenshot shows the configuration for a build trigger:

- Build Triggers**:
  - Projects to build: User\_Acceptance\_Test
  - Trigger when build is: Stable
  - Trigger build without parameters
- Buttons:
  - Add Parameters
  - Add trigger...
  - Delete

11. Along with triggering the build, we would also like to pass some predefined parameters to it. Click on the Add Parameters button and select Predefined parameters:



12. Add the following values:

The screenshot shows the Jenkins job configuration interface. Under 'Build Triggers', there is a section for 'Trigger parameterized build on other projects'. It includes fields for 'Projects to build' (set to 'User\_Acceptance\_Test'), 'Trigger when build is' (set to 'Stable'), and 'Trigger build without parameters'. Below this, under 'Predefined parameters', there is a 'Parameters' field containing the values 'BUILD\_NUMBER=\${BUILD\_NUMBER}' and 'GIT\_COMMIT=\${GIT\_COMMIT}'. At the bottom right of the configuration area, there is a 'Delete' button. To the left of the configuration area, there is an 'Add Parameters' button and an 'Add trigger...' button. There is also another 'Delete' button at the bottom right of the page.

13. Save the Jenkins job by clicking on the Save button.



# CHEF COOKBOOK DEVELOPMENT

## Contents

|   |    |
|---|----|
| 1. Chef Infrastructure.....                                   | 4  |
| Introduction .....  | 4  |
| Using version control .....                                   | 4  |
| Note .....  | 4  |
| Getting ready .....   | 5  |
| Installing the Chef Development Kit on your workstation ..... | 5  |
| How to do it... .....   | 5  |
| Using the hosted Chef platform .....                          | 6  |
| Getting ready .....   | 6  |
| How to do it... .....   | 6  |
| How it works... .....   | 8  |
| There's more... .....   | 8  |
| Note .....  | 8  |
| Managing virtual machines with Vagrant.....                   | 8  |
| Tip .....   | 9  |
| Getting ready .....   | 9  |
| How to do it... .....   | 9  |
| How it works... .....   | 11 |
| There's more... .....   | 13 |
| Creating and using cookbooks.....                             | 13 |
| Getting ready .....   | 13 |
| How to do it... .....   | 14 |
| How it works... .....   | 15 |
| There's more... .....   | 15 |
| Inspecting files on your Chef server with knife .....         | 16 |
| Getting ready .....   | 16 |
| How to do it... .....   | 17 |
| How it works... .....   | 18 |
| There's more... .....   | 18 |
| Defining cookbook dependencies.....                           | 18 |
| Getting ready .....   | 19 |
| How to do it... .....   | 19 |

|  |    |
|--|----|
| <b>How it works...</b>                               | 19 |
| <b>There's more...</b>                               | 20 |
| <b>Tip</b>   | 20 |
| <b>Managing cookbook dependencies with Berkshelf</b> | 20 |
| <b>Getting ready</b>                                 | 21 |
| <b>How to do it</b>                                  | 21 |
| <b>How it works</b>                                  | 22 |
| <b>Note</b>  | 23 |
| <b>There's more</b>                                  | 24 |
| <b>See also</b>                                      | 25 |
| <b>Using custom knife plugins</b>                    | 25 |
| <b>Getting ready</b>                                 | 26 |
| <b>How to do it</b>                                  | 26 |
| <b>How it works</b>                                  | 27 |
| <b>There's more</b>                                  | 27 |
| <b>Deleting a node from the Chef server</b>          | 28 |
| <b>Getting ready</b>                                 | 28 |
| <b>How to do it</b>                                  | 28 |
| <b>How it works</b>                                  | 28 |
| <b>There's more</b>                                  | 29 |
| <b>Developing recipes with local mode</b>            | 29 |
| <b>Getting ready</b>                                 | 29 |
| <b>How to do it</b>                                  | 30 |
| <b>How it works</b>                                  | 31 |
| <b>There's more</b>                                  | 31 |
| <b>Using roles</b>                                   | 32 |
| <b>Getting ready</b>                                 | 32 |
| <b>How to do it</b>                                  | 32 |
| <b>How it works</b>                                  | 33 |
| <b>Using environments</b>                            | 34 |
| <b>Getting ready</b>                                 | 34 |
| <b>How to do it</b>                                  | 34 |
| <b>How it works</b>                                  | 36 |
| <b>There's more</b>                                  | 37 |

|  |           |
|--|-----------|
| <b>Freezing cookbooks .....</b>  | <b>38</b> |
| <b>Note .....</b>  | <b>38</b> |
| <b>Getting ready .....</b>   | <b>38</b> |
| <b>Make sure you have at least one cookbook (I'll use the ntp cookbook) registered with your Chef server. ....</b> | <b>38</b> |
| <b>How to do it....</b>  | <b>38</b> |
| <b>How it works...</b>   | <b>39</b> |
| <b>There's more...</b>   | <b>39</b> |
| <b>Running the Chef client as a daemon .....</b>   | <b>39</b> |
| <b>Getting ready .....</b>   | <b>40</b> |
| <b>How to do it... ....</b>  | <b>40</b> |
| <b>How it works...</b>   | <b>40</b> |
| <b>Tip .....</b>   | <b>40</b> |
| <b>There's more...</b>   | <b>40</b> |
| <b>Note .....</b>  | <b>41</b> |
| <b>2. Evaluating and Troubleshooting Cookbooks and Chef Runs.....</b>  | <b>41</b> |
| <b>Introduction .....</b>  | <b>41</b> |
| <b>Testing your Chef cookbooks with cookstyle and Rubocop.....</b>   | <b>41</b> |
| <b>How to do it....</b>  | <b>42</b> |
| <b>Carry out the following steps to test your cookbook; run cookstyle on the ntp cookbook: ....</b>                | <b>42</b> |
| <b>How it works...</b>   | <b>42</b> |
| <b>There's more...</b>   | <b>42</b> |
| <b>See also.....</b>   | <b>42</b> |
| <b>Flagging problems in your Chef cookbooks with Foodcritic .....</b>  | <b>43</b> |
| <b>Getting ready .....</b>   | <b>43</b> |
| <b>How to do it... ....</b>  | <b>43</b> |
| <b>How it works...</b>   | <b>44</b> |
| <b>There's more...</b>   | <b>45</b> |
| <b>See also.....</b>   | <b>45</b> |

## 1. Chef Infrastructure

### Introduction

Let's talk about some important terms used in the Chef universe.

A **cookbook** is a collection of all the components needed to change something on a server. Things such as installing MySQL or configuring SSH can be done by cookbooks. The most important parts of cookbooks are recipes, which tell Chef which resources you want to configure on your host.

You need to deploy cookbooks to the nodes that you want to change. Chef offers multiple methods for this task. Most probably, you'll use a central **Chef server**. You can either run your own server or sign up for **hosted Chef**.

The Chef server is the central registry, where each node needs to be registered. The Chef server distributes the cookbooks you uploaded to it, to your nodes.

**Knife** is Chef's command-line tool to interact with the Chef server. You run it on your local workstation and use it to upload cookbooks and manage other aspects of Chef.

On your nodes, you need to install **Chef Client**—the program that runs on your nodes, retrieving cookbooks from the Chef server and executing them on the node.

### Using version control

A **version control system (VCS)** helps you stay sane when dealing with important files and collaborating on them.

Using version control is a fundamental part of any infrastructure automation. There are multiple solutions to manage source version control, including Git, SVN, Mercurial, and Perforce. Due to its popularity among the Chef community, we will be using Git. However, you could easily use any other version control system with Chef.

#### Note

Don't even think about building your infrastructure as code without using a version control system to manage it!

### Getting ready

You'll need Git installed on your local workstation. Either use your operating system's package manager (such as Apt on Ubuntu or Homebrew on OS X, or simply download the installer from [www.git-scm.org](http://www.git-scm.org).

Git is a distributed version control system. This means that you don't necessarily need a central host to store your repositories. However, in practice, using GitHub as your central repository has proven to be very helpful. In this book, I'll assume that you're using GitHub. Therefore, you need to go to [www.github.com](http://www.github.com) and create an account (which is free) to follow the instructions given in this book. Make sure that you upload your Secure Shell (SSH) key by following the instructions at <https://help.github.com/articles/generating-ssh-keys>, so that you're able to use the SSH protocol to interact with your GitHub account.

As soon as you have created your GitHub account, you should create your repository by visiting <https://github.com> and using chef-repo as the repository name.

Walk through the basic steps using GitHub at <https://help.github.com/categories/bootcamp>

### Installing the Chef Development Kit on your workstation

If you want to use Chef, you'll need to install the **Chef Development Kit (DK)** on your local workstation first. You'll have to develop your configurations locally and use Chef to distribute them to your Chef server.

Chef provides a fully packaged version, which does not have any external prerequisites. This fully packaged Chef is called the **omnibus installer**. We'll see how to use it in this section.

### How to do it...

To install the Chef development kit:

1. Visit this page: <https://downloads.chef.io/chefdk/>. The Chef development kit supports macOS, Red Hat Enterprise Linux, Ubuntu, and Microsoft Windows.

2. Select a platform, and then a package. (chef-docs uses the macOS setup within the documentation.)
3. Click the download button.
4. Follow the steps in the installer and install the Chef development kit to your machine. The Chef development kit is installed to /opt/chefdk/ on UNIX and Linux systems.
5. When finished, open a command window and enter the following:

```
$ chef verify
```

### **Using the hosted Chef platform**

If you want to get started with Chef right away (without the need to install your own Chef server) or want a third party to give you a **Service Level Agreement (SLA)** for your Chef server, you can sign up for hosted Chef by Chef Software, Inc. Chef Software, Inc. operates Chef as a cloud service. It's quick to set up and gives you full control, using users and groups to control access permissions to your Chef setup. We'll configure **knife**, Chef's command-line tool, to interact with hosted Chef, so that you can start managing your nodes.

#### **Getting ready**

Before being able to use hosted Chef, you need to sign up for the service. There is a free account for up to five nodes.

Visit <http://manage.chef.io/signup> and register for a free account.

After registering your account, it is time to prepare your organization to be used with your chef-repo repository.

#### **How to do it...**

Download Chef-starter kit from hosted chef or Carry out the following steps to interact with the hosted Chef:

1. Create the configuration directory for your Chef client on your local workstation:

```
mma@laptop:~ $ cd ~/chef-repo  
mkdir .chef
```

2. Generate the knife config and put the downloaded knife.rb into the .chef directory inside your chef-repo directory. Make sure you have your user's private key saved as .chef/<YOUR USERNAME>.pem, (in my case it is .chef/webops.pem). If needed, you can reset it at <https://id.chef.io/id/profile>. Replace webops with the username you chose for hosted Chef, and awo with the short name you chose for your organization in your knife.rb file:

```
current_dir = File.dirname(__FILE__)  
log_level:info  
log_location STDOUT  
node_name "webops"  
client_key "#{current_dir}/webops.pem"  
chef_server_url "https://api.chef.io/organizations/awo"  
cache_type 'BasicFile'  
cache_options( :path => "#{ENV['HOME']}/.chef/checksums" )  
cookbook_path ["#{current_dir}/../cookbooks"]  
Note:
```

You should add the following code to your .gitignore file inside chef-repo to avoid your credentials ending up in your Git repository

#### **.chef/\*.pem**

3. Use knife to verify that you can connect to your hosted Chef organization. It should not have any clients, so far:

Copy

```
mma@laptop:~/chef-repo $ knife client list
```

**How it works...**

The following line of code in your knife.rb file tells knife where to find your user's private key. It is used to authenticate you with the Chef server:

```
client_key      "#{current_dir}/webops.pem"
```

Also, the following line of code in your knife.rb file tells knife that you are using hosted Chef. You will find your organization name as the last part of the URL:

```
chef_server_url "https://api.chef.io/organizations/awo"
```

Using the knife.rb file and your user's key, you can now connect to your organization hosted by Chef Software, Inc.

**There's more...**

This setup is good for you if you do not want to worry about running, scaling, and updating your own Chef server and if you're happy with saving all your configuration data in the Cloud (under the control of Chef Software, Inc.).

**Note**

If you need to have all your configuration data within your own network boundaries, you can install Chef server on-premises by choosing ON-PREMISES CHEF at <https://www.chef.io/chef/choose-your-version/> or install the Open-Source version of Chef server directly from GitHub at <https://github.com/chef/chef>.

**Managing virtual machines with Vagrant**

Vagrant is a command-line tool that provides you with a configurable, reproducible, and portable development environment using VMs. It lets you define and use preconfigured disk images to create new VMs from. Also, you can configure Vagrant to use provisioners such as Shell scripts, Puppet, or Chef to bring your VM into the desired state.

**Tip**

Chef comes with Test Kitchen, which enables you to test your cookbooks in Vagrant without you needing to setup anything manually.

You only need to follow this section if you want to learn how to use Vagrant and Chef in more advanced cases.

In this section, we will see how to use Vagrant to manage VMs using VirtualBox and Chef client as the provisioner.

**Getting ready**

1. Download and install VirtualBox at <https://www.virtualbox.org/wiki/Downloads>.
2. Download and install Vagrant at <https://www.vagrantup.com/downloads.html>.
3. Install the Omnibus Vagrant plugin to enable Vagrant to install the Chef client on your VM by running the following command:

```
mma@laptop:~/chef-repo $ vagrant plugin install vagrant-omnibus  
Installing the 'vagrant-omnibus' plugin. This can take a few minutes...  
Installed the plugin 'vagrant-omnibus (1.5.0)'!
```

**How to do it...**

Let's create and boot a virtual node by using Vagrant:

1. Visit <https://github.com/chef/bento> and choose a Vagrant box to base your VMs on. We'll use the amd64 image of ubuntu-16.04 in this example.
2. The URL of that box is [http://opscode-vm-bento.s3.amazonaws.com/vagrant/virtualbox/opscode\\_ubuntu-16.04\\_chef-provisionerless.box](http://opscode-vm-bento.s3.amazonaws.com/vagrant/virtualbox/opscode_ubuntu-16.04_chef-provisionerless.box).

3. Create a new Vagrantfile. Make sure that you replace <YOUR-ORG> with the name of your organization on the Chef server. Use the name and URL of the box file you noted down in the first step as config.vm.box and config.vm.box\_url:

Copy

```
mma@laptop:~/chef-repo $ sublVagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "opscode-ubuntu-16.04"
  config.vm.box_url = "http://opscode-vm-bento.s3.amazonaws.com/vagrant/virtualbox/opscode_ubuntu-16.04_chef-provisionerless.box"
  config.omnibus.chef_version= :latest

  config.vm.provision :chef_client do |chef|
    chef.provisioning_path = "/etc/chef"
    chef.chef_server_url = "https://api.chef.io/organizations/<YOUR_ORG>"
    chef.validation_key_path = ".chef/<YOUR_USER>.pem"
    chef.validation_client_name = "<YOUR_USER>"
    chef.node_name = "server"
  end
end
```

4. Create your virtual node using Vagrant:

Copy

```
mma@laptop:~/chef-repo $ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==>default: Box 'opscode-ubuntu-16.04' could not be found. Attempting to find
and install...
...TRUNCATED OUTPUT...
==>default: Importing base box 'opscode-ubuntu-16.04'...
...TRUNCATED OUTPUT...
==>default: Installing Chef latest Omnibus package...
...TRUNCATED OUTPUT...
==>default: Running chef-client...
==>default: Starting Chef Client, version 12.14.89
...TRUNCATED OUTPUT...
```

5. Log in to your virtual node using SSH:

Copy

```
mma@laptop:~/chef-repo $ vagrant ssh  
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-31-generic x86_64)  
...TRUNCATED OUTPUT...  
vagrant@server:~$
```

6. Log out of your virtual node:

Copy

```
vagrant@server:~$ exit  
logout  
Connection to 127.0.0.1 closed.  
mma@laptop:~/chef-repo $
```

7. Validate that the Chef server knows your new virtual machine as a client called **server**:

Copy

```
mma@laptop:~/chef-repo $ knife client list  
awo-validator  
server
```

8. Go to <https://manage.chef.io/organizations/<YOUR ORGANIZATION>/nodes> and validate that your new virtual machine shows up as a registered node:

#### How it works...

The Vagrantfile is written in a Ruby **Domain Specific Language (DSL)** to configure the Vagrant virtual machines. We want to boot a simple Ubuntu VM. Let's go through the Vagrantfile step by step.

First, we create a config object. Vagrant will use this config object to configure the VM:

```
Vagrant.configure("2") do |config|  
  ...  
end
```

Inside the config block, we tell Vagrant which VM image to use, in order to boot the node:

```
config.vm.box = "opscode-ubuntu-16.04"
config.vm.box_url = "http://opscode-vm-bento.s3.amazonaws.com/vagrant/virtualbox/opscode_ubuntu-16.04_chef-provisionerless.box"
```

We want to boot our VM using a so-called Bento Box, provided by Chef. We use Ubuntu Version 16.04 here.

As we want our VM to have the Chef client installed, we tell the omnibus vagrant plugin to use the latest version of the Chef client:

```
config.omnibus.chef_version= :latest
```

After selecting the VM image to boot, we configure how to provision the box by using Chef. The Chef configuration happens in a nested Ruby block:

```
config.vm.provision :chef_client do |chef|
  ...
end
```

Inside this chef block, we need to instruct Vagrant on how to hook up our virtual node to the Chef server. First, we need to tell Vagrant where to store all the Chef stuff on your node:

```
chef.provisioning_path = "/etc/chef"
```

Vagrant needs to know the API endpoint of your Chef server. If you use hosted Chef, it is [https://api.chef.io/organizations/<YOUR\\_ORG>](https://api.chef.io/organizations/<YOUR_ORG>). You need to replace <YOUR\_ORG> with the name of the organization that you created in your account on hosted Chef. If you are using your own Chef server, change the URL accordingly:

```
chef.chef_server_url = "https://api.chef.io/organizations/<YOUR_ORG>"
```

While creating your user on hosted Chef, you must have downloaded your private key. Tell Vagrant where to find this file:

```
chef.validation_key_path = ".chef/<YOUR_USER>.pem"
```

Also, you need to tell Vagrant which client it should use to validate itself against in the Chef server:

```
chef.validation_client_name = "<YOUR_USER>"
```

Finally, you should tell Vagrant how to name your node:

```
chef.node_name = "server"
```

After configuring your Vagrantfile, all you need to do is run the basic Vagrant commands such as `vagrant up`, `vagrant provision`, and `vagrant ssh`. To stop your VM, just run the `vagrant halt` command.

### **There's more...**

If you want to start from scratch again, you will have to destroy your VM and delete both the client and the node from your Chef server by running the following command:

```
mma@laptop:~/chef-repo $ vagrant destroy  
mma@laptop:~/chef-repo $ knife node delete server -y && knife client delete server -y
```

### **Creating and using cookbooks**

Cookbooks are an essential part of Chef. Basically, you describe the configurations you want to apply to your nodes in cookbooks. You can create them using the Chef executable installed by the Chef DK.

In this section, we'll create and apply a simple cookbook using the `chef` and `knife` command-line tools.

#### **Getting ready**

Make sure you have Chef DK installed and a node available for testing. Check out the installation instructions at <http://learn.chef.io> if you need help here.

Edit your `knife.rb` file (usually found in the hidden `.chef` directory) and add the following three lines to it, filling in your own values:

```
cookbook_copyright "your company"  
cookbook_license "apachev2"  
cookbook_email "your email address"
```

Chef will use the preceding values as the defaults whenever you create a new cookbook. We assume that you have a node called `server` registered with your Chef server, as described in the **Managing virtual machines with Vagrant** section in this chapter.

#### **How to do it...**

Carry out the following steps to create and use cookbooks:

1. Create a cookbook named `my_cookbook` by running the following command:

```
mma@laptop:~/chef-repo $ chef generate cookbook cookbooks/my_cookbook  
Generating cookbook my_cookbook  
- Ensuring correct cookbook file content  
- Ensuring delivery configuration  
- Ensuring correct delivery build cookbook content
```

Your cookbook is ready. Type `cd cookbooks/my\_cookbook` to enter it.  
...TRUNCATED OUTPUT...

2. Upload your new cookbook on the Chef server:

```
mma@laptop:~/chef-repo $ knife cookbook upload my_cookbook  
Uploading my_cookbook [0.1.0]  
Uploaded 1 cookbook.
```

3. Add the cookbook to your node's run list. In this example, the name of the node is `server`:

```
mma@laptop:~/chef-repo $ knife node run_list add server 'recipe[my_cookbook]'  
server:  
run_list: recipe[my_cookbook]
```

4. Run the Chef client on your node:

```
user@server:~$ sudo chef-client
```

### How it works...

The chef executable helps you to manage your local Chef Development environment. We used it here to generate the cookbook.

Knife is the command-line interface for the Chef server. It uses the RESTful API exposed by the Chef server to do its work and helps you to interact with the Chef server.

The knife command supports a host of commands structured as follows:

```
knife<subject><command>
```

The <subject> used in this section is either cookbook or node. The commands we use are upload for the cookbook, and run\_list add for the node.

### There's more...

Before uploading your cookbook to the Chef server, it's a good idea to run it in Test Kitchen first. Test Kitchen will spin up a virtual machine, execute your cookbook, and destroy the virtual machine again. That way you can evaluate what your cookbook does before you upload it to the Chef server and run it on real nodes.

To run your cookbook with Test Kitchen on an Ubuntu 16.04 virtual machine, execute the following steps:

1. Create a configuration file for Test Kitchen for executing the default recipe of my cookbook:

```
mma@laptop:~/chef-repo $ subl .kitchen.yml
```

```
---
```

```
driver:
```

```
name: vagrant
```

```
provisioner:  
  name: chef_zero  
  
platforms:  
  - name: ubuntu-16.04  
  
suites:  
  - name: default  
run_list:  
  - recipe[my_cookbook::default]  
attributes:
```

2. Run kitchen test to execute the default recipe of my\_cookbook:

```
mma@laptop:~/chef-repo $ kitchen test  
----> Starting Kitchen (v1.13.2)  
...TRUNCATED OUTPUT...  
----> Kitchen is finished. (0m45.42s)
```

### Inspecting files on your Chef server with knife

Sometimes, you may want to peek into the files stored on your Chef server. You might not be sure about an implementation detail of the specific cookbook version currently installed on your Chef server, and need to look it up. Knife can help you out by letting you show various aspects of the files stored on your Chef server.

#### Getting ready

Install the iptables community cookbook by executing the following command:

```
mma@laptop:~/chef-repo $ knife cookbook site install iptables  
Installing iptables to /Users/mma/work/chef-repo/cookbooks  
...TRUNCATED OUTPUT...
```

Take a look at the following error:

```
ERROR: IOError: Cannot open or read ..../chef-repo/cookbooks/iptables/metadata.rb!
```

If you get the preceding error, your cookbook only has a `metadata.json` file. Make sure that you delete it and create a valid `metadata.rb` file instead.

Upload the iptables cookbook on your Chef server by executing the following command:

```
mma@laptop:~/chef-repo $ knife cookbook upload iptables --include-dependencies
Uploading iptables      [3.0.0]
Upgrading compat_resource [12.14.7]
Uploaded 2 cookbooks.
```

### How to do it...

Let's find out how knife can help you to look into a cookbook stored in your Chef server:

1. First, you want to find out the current version of the cookbook you're interested in. In our case, we're interested in the iptables cookbook:

```
mma@laptop:~/work/chef_helpster $ knife cookbook show iptables
iptables 3.0.0 0.14.1
```

2. Then, you can look up the definitions of the iptables cookbook, using the version number that you found in the previous step:

```
mma@laptop:~/chef-repo $ knife cookbook show iptables 0.14.1 definitions
checksum: 45c0b77ff10d7177627694827ce47340
name: iptables_rule.rb
path: definitions/iptables_rule.rb
specificity: default
url: https://s3-external-1.amazonaws.com:443/opscode-platform...
```

3. Now, you can even show the contents of the `iptables_rule.rb` definition file, as stored on the server:

```
mma@laptop:~/chef-repo $ knife cookbook show iptables 0.14.1 definitions
iptables_rule.rb
#
# Cookbook Name::iptables
# Definition::iptables_rule
#
define :iptables_rule, :enable => true, :source => nil, :variables => {}, :cookbook => nil do
```

**...TRUNCATED OUTPUT...**  
end

### How it works...

The `knife cookbook show` subcommand helps you understand what exactly is stored on the Chef server. It lets you drill down into specific sections of your cookbooks and see the exact content of the files stored in your Chef server.

### There's more...

You can pass patterns to the `knife show` command to tell it exactly what you want to see. Showing the attributes defined by the cookbook can be done as follows:

```
mma@laptop:~/work/chef_helpster $ knife show cookbooks/iptables/attributes/*  
cookbooks/iptables/attributes/default.rb:  
#  
# Cookbook Name::iptables  
# Attribute:: default  
...TRUNCATED OUTPUT...
```

To find some more examples on `knife show`, visit [https://docs.chef.io/knife\\_show.html](https://docs.chef.io/knife_show.html)

### Defining cookbook dependencies

Quite often, you might want to use features of other cookbooks in your own cookbooks. For example, if you want to make sure that all packages required for compiling software written in C are installed, you might want to include the `build-essential` cookbook, which does just that. The Chef server needs to know about such dependencies in your cookbooks. You declare them in a cookbook's metadata.

**Getting ready**

Make sure you have a cookbook named `my_cookbook`, and the `run_list` of your node includes `my_cookbook`, as described in the **Creating and using cookbooks** in this chapter.

**How to do it...**

Edit the metadata of your cookbook in the file `cookbooks/my_cookbook/metadata.rb` to add a dependency to the `build-essential` cookbook:

```
mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/metadata.rb
...
depends 'build-essential', '>= 7.0.3'
```

**How it works...**

If you want to use a feature of another cookbook inside your cookbook, you will need to include the other cookbook in your recipe using the `include_recipe` directive:

```
include_recipe 'build-essential'
```

To tell the Chef server that your cookbook requires the `build-essential` cookbook, you need to declare that dependency in the `metadata.rb` file. If you uploaded all the dependencies to your Chef server either using `knife cookbook upload my_cookbook --include-dependencies` or `berks install` and `berks upload`, as described in the **Managing cookbook dependencies with Berkshelf** recipe in this chapter, the Chef server will then send all the required cookbooks to the node.

The `depends` function call tells the Chef server that your cookbook depends on a version greater than or equal to 7.0.3 of the `build-essential` cookbook.

You may use any of these version constraints with `depends` calls:

- < (less than)
- <= (less than or equal to)
- = (equal to)
- >= (greater than or equal to)
- ~> (approximately greater than)
- > (greater than)

**There's more...**

If you include another recipe inside your recipe, without declaring the cookbook dependency in your `metadata.rb` file, Foodcritic will warn you:

```
mma@laptop:~/chef-repo $ foodcritic cookbooks/my_cookbook
FC007: Ensure recipe dependencies are reflected in cookbook metadata:
cookbooks/my_cookbook/recipes/default.rb:9
```

Foodcritic will just return an empty line if it doesn't find any issues.

**Managing cookbook dependencies with Berkshelf**

It's a pain to manually ensure that you have installed all the cookbooks that another cookbook depends on. You must download each and every one of them manually only to find out that, with each downloaded cookbook, you inherit another set of dependent cookbooks.

And even if you use knife cookbook site install, which installs all the dependencies locally for you, your cookbook directory and your repository get cluttered with all those cookbooks. Usually, you don't really care about all those cookbooks and don't want to see or manage them.

This is where Berkshelf comes into play. It works like Bundler for Ruby gems, managing cookbook dependencies for you. Berkshelf downloads all the dependencies you defined recursively and helps you to upload all cookbooks to your Chef server.

Instead of polluting your Chef repository, it stores all the cookbooks in a central location. You just commit your Berkshelf dependency file (called **Berksfile**) to your repository, and every colleague or build server can download and install all those dependent cookbooks based on it.

Let's see how to use Berkshelf to manage the dependencies of your cookbook.

### **Getting ready**

Make sure you have a cookbook named `my_cookbook` and the `run_list` of your node includes `my_cookbook`, as described in the **Creating and using cookbooks**

### **How to do it...**

Berkshelf helps you to keep those utility cookbooks out of your Chef repository. This makes it much easier to maintain the important cookbooks.

Let's see how to write a cookbook by running a bunch of utility recipes and manage the required cookbooks with Berkshelf:

1. Edit your cookbook's metadata:

```
mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/metadata.rb
...
depends "chef-client"
depends "apt"
depends "ntp"
```

2. Edit your cookbook's default recipe:

```
mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
...
include_recipe "chef-client"
include_recipe "apt"
```

```
include_recipe "ntp"
```

3. Run Berkshelf to install all the required cookbooks:

```
mma@laptop:~/chef-repo $ cd cookbooks/my_cookbook
mma@laptop:~/chef-repo/cookbooks/my_cookbook $ berks install
Resolving cookbook dependencies...
Fetching 'my_cookbook' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Installing apt (4.0.2)
...TRUNCATED OUTPUT...
```

4. Upload all the cookbooks on the Chef server:

```
mma@laptop:~/chef-repo/cookbooks/my_cookbook $ berks upload
Using my_cookbook (0.1.0)
...TRUNCATED OUTPUT...
Uploaded windows (2.0.2) to: 'https://api.opscode.com:443/organizations/awo'
```

### How it works...

Berkshelf comes with the Chef DK.

We edit our cookbook and tell it to use a few basic cookbooks.

Instead of making us manually install all the cookbooks using knife cookbook site install, Chef generates a Berksfile, besides the metadata.rb file.

The Berksfile is simple. It tells Berkshelf to use the Chef supermarket as the default source for all cookbooks:

```
source "https://supermarket.chef.io"
```

And the Berksfile tells Berkshelf to read the metadata.rb file to find all the required cookbooks. This is the simplest way when working inside a single cookbook. Please see the following **There's more...** section to find an example of a more advanced usage of the Berksfile.

After telling Berkshelf where to find all the required cookbook names, we use it to install all those cookbooks:

**berks install**

Berkshelf stores cookbooks in `~/berkshelf/cookbooks`, by default. This keeps your Chef repository clutter-free. Instead of having to manage all the required cookbooks inside your own Chef repository, Berkshelf takes care of them. You simply need to check in Berksfile with your cookbook, and everyone using your cookbook can download all the required cookbooks by using Berkshelf.

To make sure that there's no mix-up with different cookbook versions when sharing your cookbook, Berkshelf creates a file called `Berksfile.lock` alongside Berksfile.

**Note**

`Don't commit the Berksfile.lock to version control. If you use berks generate it will auto populate the .gitignore for you. Otherwise, you need to add Berksfile.lock to your .gitignore manually.`

Here, you'll find the exact versions of all the cookbooks that Berkshelf installed:

**DEPENDENCIES**`my_cookbook``path: .``metadata: true`**GRAPH**`apt (4.0.2)``compat_resource (>= 12.10)``chef-client (6.0.0)``cron (>= 1.7.0)``logrotate (>= 1.9.0)``windows (>= 1.42.0)``compat_resource (12.14.7)``cron (2.0.0)``logrotate (2.1.0)``compat_resource (>= 0.0.0)`

```
my_cookbook (0.1.1)
apt (>= 0.0.0)
chef-client (>= 0.0.0)
ntp (>= 0.0.0)
ntp (3.2.0)
windows (2.0.2)
```

Berkshelf will only use the exact versions specified in the Berksfile.lock file, if it finds this file.

Finally, we use Berkshelf to upload all the required cookbooks to the Chef server:

```
berks upload
```

**There's more...**

Berkshelf integrates tightly with Vagrant via the vagrant-berkshelf plugin. You can set up Berkshelf and Vagrant in such a way that Berkshelf installs and uploads all the required cookbooks on your Chef server whenever you execute vagrant up or vagrant provision. You'll save all the work of running berks install and berks upload manually before creating your node with Vagrant.

Let's see how you can integrate Berkshelf and Vagrant:

1. First, you need to install the Berkshelf plugin for Vagrant:

```
mma@laptop:~/work/chef-repo $ vagrant plugin install vagrant-berkshelf
Installing the 'vagrant-berkshelf' plugin. This can take a few minutes...
Installed the plugin 'vagrant-berkshelf (5.0.0)'!
```

2. Then, you need to tell Vagrant that you want to use the plugin. You do this by enabling the plugin in Vagrantfile:

```
mma@laptop:~/work/chef-repo $ sublVagrantfile
config.berkshelf.enabled = true
```

3. Then, you need a Berksfile in the root directory of your Chef repository to tell Berkshelf which cookbooks to install on each Vagrant run:

```
mma@laptop:~/work/chef-repo $ sublBerksfile  
source 'https://supermarket.chef.io'  
  
cookbook 'my_cookbook', path: 'cookbooks/my_cookbook'
```

4. Eventually, you can start your VM using Vagrant. Berkshelf will first download and then install all the required cookbooks in the Berkshelf, and upload them to the Chef server. Only after all the cookbooks are made available on the Chef server by Berkshelf will Vagrant go on:

```
mma@mma-mbp:~/work/chef-repo $ vagrant up  
Bringing machine 'server' up with 'virtualbox' provider...  
  
==>default: Updating Vagrant's Berkshelf...  
==>default: Resolving cookbook dependencies...  
==>default: Fetching 'my_cookbook' from source at cookbooks/my_cookbook  
==>default: Fetching cookbook index from https://supermarket.chef.io...  
...TRUNCATED OUTPUT...
```

5. This way, using Berkshelf together with Vagrant saves a lot of manual steps and gets faster cycle times for your cookbook development. if you're using your manual Vagrant setup instead of Test Kitchen.

#### See also

- For the full documentation for Berkshelf, please visit <http://berkshelf.com/>
- Please find the Berkshelf source code at <https://github.com/berkshelf/berkshelf>
- Please find the Vagrant Berkshelf plugin source code at <https://github.com/berkshelf/vagrant-berkshelf>

#### Using custom knife plugins

Knife comes with a set of commands out-of-the-box. The built-in commands deal with the basic elements of Chef-like cookbooks, roles, data bags, and so on. However, it would be nice to use knife for more than just the basic stuff. Fortunately, knife comes with a plugin API and

there are already a host of useful knife plugins built by the makers of Chef and the Chef community.

### Getting ready

Make sure you have an account at **Amazon Web Services (AWS)** if you want to follow along and try out the `knife-ec2` plugin. There are knife plugins available for most Cloud providers. Go through the **There's more...** section of this recipe for a list.

### How to do it...

Let's see which knife plugins are available, and try to use one to manage Amazon EC2 instances:

1. List the knife plugins that are shipped as Ruby gems using the chef command-line tool:

```
mma@laptop:~/chef-repo $ chef gem search -r knife
*** REMOTE GEMS ***
...TRUNCATED OUTPUT...
```

```
knife-azure (1.6.0)
...TRUNCATED OUTPUT...
knife-ec2 (0.13.0)
...TRUNCATED OUTPUT...
```

2. Install the EC2 plugin to manage servers in the Amazon AWS Cloud:

```
mma@laptop:~/chef-repo $ chef gem install knife-ec2
Building native extensions. This could take a while...
...TRUNCATED OUTPUT...
Fetching: knife-ec2-0.13.0.gem (100%)
Successfully installed knife-ec2-0.13.0
...TRUNCATED OUTPUT...
```

**6 gems installed**

3. List all the available instance types in AWS using the knife ec2 plugin. Please use your own AWS credentials instead of XXX and YYYYYY:

```
mma@laptop:~/chef-repo $ knife ec2 flavor list --aws-access-key-id XXX --aws-secret-access-key YYYYYY
ID          Name           Arch   RAM   Disk   Cores
c1.medium   High-CPU Medium    32-bit 1740.8 350 GB  5
...TRUNCATED OUTPUT...
m2.xlarge   High-Memory Extra Large 64-bit 17510. 420 GB  6.5
t1.micro    Micro Instance     0-bit   613   0 GB   2
```

**How it works...**

Knife looks for plugins in various places.

First, it looks into the .chef directory, which is located inside your current Chef repository, to find plugins specific to this repository:

```
./.chef/plugins/knife/
```

Then, it looks into the .chef directory, which is located in your home directory, to find plugins that you want to use in all your Chef repositories:

```
~/.chef/plugins/knife/
```

Finally, it looks for installed gems. Knife will load all the code from any chef/knife/ directory found in your installed Ruby gems. This is the most common way of using plugins developed by Chef or the Chef community.

**There's more...**

There are hundreds of knife plugins, including plugins for most of the major Cloud providers, as well as the major virtualization technologies, such as VMWare, vSphere, and OpenStack, among others.

### **Deleting a node from the Chef server**

Every node managed by a Chef server has a corresponding client object on the Chef server. Running the Chef client on your node uses the client object to authenticate itself against the Chef server on each run.

Additionally, to register a client, a node object is created on the Chef server. The node object is the main data structure, which you can use to query node data inside your recipes.

#### **Getting ready**

Make sure you have at least one node registered on your Chef server that is safe to remove.

#### **How to do it...**

Let's delete the node and client object to completely remove a node from the Chef server.

1. Delete the node object:

```
mma@laptop:~/chef-repo $ knife node delete my_node  
Do you really want to delete my_node? (Y/N) y  
Deleted node[my_node]
```

2. Delete the client object:

```
mma@laptop:~/chef-repo $ knife client delete my_node  
Do you really want to delete my_node? (Y/N) y  
Deleted client[my_node]
```

#### **How it works...**

To keep your Chef server clean, it's important to not only manage your node objects but to also take care of your client objects, as well.

Knife connects to the Chef server and deletes the node object with a given name, using the Chef server RESTful API.

The same happens while deleting the client object on the Chef server.

After deleting both objects, your node is totally removed from the Chef server. Now you can reuse the same node name with a new box or virtual machine.

### **There's more...**

Having to issue two commands is a bit tedious and error-prone. To simplify things, you can use a knife plugin called playground.

1. Run the chef command-line tool to install the knife plugin:

```
mma@laptop:~/chef-repo $ chef gem install knife-playground  
...TRUNCATED OUTPUT...  
Installing knife-playground (0.2.2)
```

2. Run the knife pgclientnode delete subcommand:

```
mma@laptop:~/chef-repo $ knife pgclientnode delete my_node  
Deleting CLIENT my_node...  
Do you really want to delete my_node? (Y/N) y  
Deleted client[my_node]  
Deleting NODE my_node...  
Do you really want to delete my_node? (Y/N) y  
Deleted node[my_node]
```

### **Developing recipes with local mode**

If running your own Chef server seems like overkill and you're not comfortable with using the hosted Chef, you can use local mode to execute cookbooks.

#### **Getting ready**

1. Create a cookbook named my\_cookbook by running the following command:

```
mma@laptop:~/chef-repo $ chef generate cookbook cookbooks/my_cookbook
Compiling Cookbooks...
Recipe: code_generator::cookbook
...TRUNCATED OUTPUT...
```

2. Edit the default recipe of my\_cookbook so that it creates a temporary file:

```
mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
file "/tmp/local_mode.txt" do
  content "created by chef client local mode"
  action :create
end
```

### How to do it...

Let's run my\_cookbook on your local workstation using the Chef client's local mode:

1. Run the Chef client locally with my\_cookbook in the run list:

```
mma@laptop:~/chef-repo $ chef-client --local-mode -o my_cookbook
[2016-10-03T20:37:02+02:00] INFO: Started chef-zero at chefzero://localhost:8889
with repository at /Users/matthias.marschall/chef-repo
  One version per cookbook

[2016-10-03T20:37:02+02:00] INFO: Forking chef instance to converge...
Starting Chef Client, version 12.14.89
[2016-10-03T20:37:02+02:00] INFO: *** Chef 12.14.89 ***
[2016-10-03T20:37:02+02:00] INFO: Platform: x86_64-darwin13
...TRUNCATED OUTPUT...
Chef Client finished, 1/1 resources updated in 04 seconds
```

2. Validate that the Chef client run creates the desired temporary file on your local workstation:

```
mma@laptop:~/chef-repo $ cat /tmp/local_mode.txt
created by chef client local mode
```

**How it works...**

The `--local-mode` (short form: `-z`) parameter switches the Chef client into local mode. Local mode uses `chef-zero`—a simple, in-memory version of the Chef server provided by Chef DK—when converging the local workstation.

By providing the `-o` parameter, you override the run list of your local node so that the Chef client executes the default recipe from `my_cookbook`.

**There's more...**

Chef-zero saves all modifications made by your recipes to the local filesystem. It creates a JSON file containing all node attributes for your local workstation in the `nodes` directory. This way, the next time you run the Chef client in local mode, it will be aware of any changes your recipes made to the node.

**Running knife in local mode**

You can use knife in local mode, too. To set the run list of a node named `laptop` (instead of having to override it with `-o`), you can run the following command:

```
mma@laptop:~/chef-repo $ knife node run_list add -z laptop 'recipe[my_cookbook]'
```

**Moving to hosted Chef or your own Chef server**

When you're done editing and testing your cookbooks on your local workstation with `chef-zero`, you can seamlessly upload them to hosted Chef or your own Chef server:

Note: Make sure you bump the version number of modified cookbooks in their `metadata.rb` file and commit them to your version control system before uploading to the Chef Server.

```
mma@laptop:~/chef-repo $ berks upload  
Uploaded ...
```

### Using roles

Roles group nodes with similar configurations. Typical cases are using roles for web servers, database servers, and so on.

You can set custom run lists for all the nodes in your roles and override attribute values from within your roles.

Let's see how to create a simple role.

### Getting ready

For the following examples, I assume that you have a node named server and that you have at least one cookbook (I'll use the ntp cookbook) registered with your Chef server.

### How to do it...

Let's create a role and see what we can do with it:

1. Create a role:

```
mma@laptop:~/chef-repo $ subl roles/web_servers.rb
name "web_servers"
description "This role contains nodes, which act as web servers"
run_list "recipe[ntp]"
default_attributes 'ntp' => {
  'ntpdate' => {
    'disable' => true
  }
}
```

2. Upload the role on the Chef server:

```
mma@laptop:~/chef-repo $ knife role from file web_servers.rb
Updated Role web_servers
```

3. Assign the role to a node called server:

```
mma@laptop:~/chef-repo $ knife node run_list add server 'role[web_servers]'  
server:  
run_list: role[web_servers]
```

4. Log in to your node and run the Chef client:

```
user@server:~$ sudo chef-client  
...TRUNCATED OUTPUT...  
[2016-10-03T18:52:10+00:00] INFO: Run List is [role[web_servers]]  
[2016-10-03T18:52:10+00:00] INFO: Run List expands to [ntp]  
[2016-10-03T18:52:10+00:00] INFO: Starting Chef Run for server  
...TRUNCATED OUTPUT...
```

### How it works...

You define a role in a Ruby (or a JSON) file inside the `roles` folder of your Chef repository. A role consists of a `name` attribute and a `description` attribute. Additionally, a role usually contains a role-specific run list and role-specific attribute settings.

Every node with a role in its run list will have the role's run list expanded into its own. This means that all the recipes (and roles) that are in the role's run list will be executed on your nodes.

You need to upload your role to your Chef server by using the `knife role from file` command.

Only then should you add the role to your node's run list.

Running the Chef client on a node having your role in its run list will execute all the recipes listed in the role.

The attributes you define in your role will be merged with attributes from environments and cookbooks, according to the precedence rules described at <https://docs.chef.io/roles.html#attribute-precedence>.

### Using environments

Having separate environments for development, testing, and production is a good way to be able to develop and test cookbook updates, and other configuration changes in isolation. Chef enables you to group your nodes into separate environments so as to support an ordered development flow.

### Getting ready

For the following examples, let's assume that you have a node named server in the `_default` environment and that you have at least one cookbook (I'll use the `ntp` cookbook) registered with your Chef server.

### How to do it...

Let's see how to manipulate environments using knife:

**Note:**

This is only a good idea if you want to play around. For serious work, please create files describing your environments and put them under version control as described in the [There's more...](#) section of this recipe.

Make sure you've set your `EDITOR` environment variable to your preferred one.

1. Create your environment on-the-fly using knife. The following command will open your shell's default editor so that you can modify the environment definition:

```
mma@laptop:~/chef-repo $ knife environment create dev
{
  "name": "dev",
  "description": "",
```

```
"cookbook_versions": {  
},  
"json_class": "Chef::Environment",  
"chef_type": "environment",  
"default_attributes": {  
},  
"override_attributes": {  
}  
}
```

**Created dev**

2. List the available environments:

```
mma@laptop:~/chef-repo $ knife environment list  
 default  
 dev
```

3. List the nodes for all the environments:

```
mma@laptop:~/chef-repo $ knife node list  
server
```

4. Verify that the node server is not in the dev environment yet by listing nodes in the dev environment only:

```
mma@laptop:~/chef-repo $ knife node list -E dev  
mma@laptop:~/chef-repo $
```

5. Change the environment of the server to dev using knife:

```
mma@laptop:~/chef-repo $ knife node environment set server book  
server:  
chef_environment: dev
```

6. List the nodes in the dev environment again:

```
mma@laptop:~/chef-repo $ knife node list -E dev  
server
```

7. Use specific cookbook versions and override certain attributes for the environment:

```
mma@laptop:~/chef-repo $ knife environment edit dev
{
  "name": "dev",
  "description": "",
  "cookbook_versions": {
    "ntp": "1.6.8"
  },
  "json_class": "Chef::Environment",
  "chef_type": "environment",
  "default_attributes": {
  },
  "override_attributes": {
    "ntp": {
      "servers": ["0.europe.pool.ntp.org", "1.europe.pool.ntp.org",
      "2.europe.pool.ntp.org", "3.europe.pool.ntp.org"]
    }
  }
}
Saved dev
```

### How it works...

A common use of environments is to promote cookbook updates from development to staging and then into production. Additionally, they enable you to use different cookbook versions on separate sets of nodes and environment-specific attributes. You might have nodes with less memory in your staging environment as in your production environment. By using environment-specific default attributes, you can, for example, configure your MySQL service to consume less memory on staging than on production.

**Note:**

The Chef server always has an environment called `default`, which cannot be edited or deleted. All the nodes go in there if you don't specify any other environment. Make sure you've set

Be aware that roles are not environment-specific. You may use environment-specific run lists, though.

The node's environment can be queried using the `node.chef_environment` method inside your cookbooks.

**There's more...**

If you want your environments to be under version control (and you should), a better way to create a new environment is to create a new Ruby file in the `environments` directory inside your Chef repository:

```
mma@laptop:~/chef-repo $ cd environments  
mma@laptop:~/chef-repo $ subl dev.rb  
name "dev"
```

You should add, commit, and push your new environment file to GitHub:

```
mma@laptop:~/chef-repo $ git add environments/dev.rb  
mma@laptop:~/chef-repo $ git commit -a -m "the dev environment"  
mma@laptop:~/chef-repo $ git push
```

Now, you can create the environment on the Chef server from the newly created file using knife:

```
mma@laptop:~/chef-repo $ knife environment from file dev.rb  
Created environment dev
```

There is a way to migrate all the nodes from one environment to another by using knife exec:

```
mma@laptop:~/chef-repo $ knife exec -E  
'nodes.transform("chef_environment:_default") { |n| n.chef_environment("dev")}'
```

You can limit your search for nodes in a specific environment:

```
mma@laptop:~/chef-repo $ knife search node "chef_environment:dev"  
1 item found
```

### Freezing cookbooks

Uploading broken cookbooks that override your working ones is a major pain and can result in widespread outages throughout your infrastructure. If you have a cookbook version, you tested successfully with Test Kitchen, it's a good idea to freeze this version so that no one can overwrite the same version with broken code. When used together with version constraints that are specified in your environment manifests, freezing cookbooks can keep your production servers safe from accidental changes.

**Note**

Berkshelf takes care of freezing cookbooks automatically.

### Getting ready

Make sure you have at least one cookbook (I'll use the ntp cookbook) registered with your Chef server.

### How to do it...

Let's see what happens if we freeze a cookbook.

1. Upload a cookbook and freeze it:

```
mma@laptop:~/chef-repo $ knife cookbook upload ntp --freeze  
Uploading ntp [3.2.0]  
Uploaded 1 cookbook.
```

2. Try to upload the same cookbook version again:

```
mma@laptop:~/chef-repo $ knife cookbook upload ntp  
Uploading ntp [3.2.0]  
ERROR: Version 3.2.0 of cookbook ntp is frozen. Use --force to override.  
WARNING: Not updating version constraints for ntp in the environment as the  
cookbook is frozen.  
ERROR: Failed to upload 1 cookbook.
```

3. Change the cookbook version:

```
mma@laptop:~/chef-repo $ subl cookbooks/ntp/metadata.rb
```

...  
version "3.2.1"

4. Upload the cookbook again:

```
mma@laptop:~/chef-repo $ knife cookbook upload ntp
Uploading ntp [3.2.1]
Uploaded 1 cookbook.
```

### **How it works...**

By using the `--freeze` option when uploading a cookbook, you tell the Chef server that it should not accept any changes to the same version of the cookbook anymore. This is important if you're using environments and want to make sure that your production environment cannot be broken by uploading a corrupted cookbook.

By changing the version number of your cookbook, you can upload the new version. Then you can make, for example, your staging environment use that new cookbook version.

### **There's more...**

To support a more elaborate workflow, you can use the knife-spork knife plugin, which comes pre-installed with the Chef DK. It helps multiple developers work on the same Chef server and repository without treading on each other's toes. You can find more information about it at [https://docs.chef.io/plugin\\_knife\\_spork.html](https://docs.chef.io/plugin_knife_spork.html).

### **Running the Chef client as a daemon**

While you can run the Chef client on your nodes manually whenever you change something in your Chef repository, it's sometimes preferable to have the Chef client run automatically every so often. Letting the Chef client run automatically makes sure that no node misses out any updates.

### Getting ready

You need to have a node registered with your Chef server. It needs to be able to run chef-client without any errors.

### How to do it...

Let's see how to start the Chef client in daemon mode so that it runs automatically:

1. Start the Chef client in daemon mode, running every 30 minutes:

```
user@server:~$ sudo chef-client -i 1800
```

2. Validate that the Chef client runs as a daemon:

```
user@server:~$ ps auxw | grep chef-client
```

### How it works...

The -i parameter will start the Chef client as a daemon. The given number is the seconds between each Chef client run. In the previous example, we specified 1,800 seconds, which results in the Chef client running every 30 minutes.

You can use the same command in a service startup script.

```
#!/bin/bash
# Start the Chef Client as a service
# You can use the chef-client cookbook to install the Chef Client as a service
# See https://supermarket.chef.io/cookbooks/chef-client for details
```

### There's more...

Instead of running the Chef client as a daemon, you can use a Cronjob to run it every so often:

```
user@server:~$ subl /etc/cron.d/chef_client
PATH=/usr/local/bin:/usr/bin:/bin
```

```
# m h dommondow user command  
*/15 * * * * root chef-client -l warn | grep -v 'retrying [1234] / 5 in'
```

This cronjob will run the Chef client every 15 minutes and swallow the first four retrying warning messages. This is important to avoid Cron sending out e-mails if the connection to the Chef server is a little slow and the Chef client needs a few retries.

**Note**

It is possible to initiate a Chef client run at any time by sending the SIGUSR1 signal to the Chef client daemon:

```
user@server:~$ sudo killall -USR1 chef-client
```

## 2. Evaluating and Troubleshooting Cookbooks and Chef Runs

### Introduction

Developing cookbooks and making sure your nodes converge to the desired state is a complex endeavor. You need transparency about what is happening. This chapter will cover a lot of ways to see what's going on and make sure that everything is working as it should. From running basic checks on your cookbooks to a fully test-driven development approach, we'll see what the Chef ecosystem has to offer.

### Testing your Chef cookbooks with cookstyle and Rubocop

You know how annoying this is: you tweak a cookbook, run Test Kitchen, and, boom! it fails. What's even more annoying is that it fails only because you missed a mundane comma in the default recipe of the cookbook you just tweaked. Fortunately, there's a very quick and easy way to find such simple glitches before you go all in and try to run your cookbooks on Test Kitchen.

#### Getting

ready

Install the ntp cookbook by running the following command:

```
mma@laptop:~/chef-repo $ knife cookbook site install ntp  
Installing ntp to /Users/mma/work/chef-repo/cookbooks  
...TRUNCATED OUTPUT...
```

**Cookbook ntp version 3.2.0 successfully installed****How to do it...**

Carry out the following steps to test your cookbook; run cookstyle on the ntp cookbook:

```
mma@laptop:~/chef-repo $ cookstyle cookbooks/ntp
```

Inspecting 5 files

...C.

**Offenses:**

cookbooks/ntp/recipes/default.rb:25:1: C: Extra blank line detected.

5 files inspected, 1 offense detected

**How it works...**

Cookstyle is a wrapper around Rubocop and executes a Ruby syntax check on all Ruby files within the cookbook. Rubocop is a linting and style-checking tool built for Ruby. cookstyle defines some sane rules for Chef cookbooks.

**There's more...**

There exists a whole ecosystem of additional tools such as ChefSpec (behavior-driven testing for Chef), and Test Kitchen (an integration testing tool to run cookbooks on virtual servers), and then some.

**See also**

- Read more about Rubocop at <https://docs.chef.io/rubocop.html>
- Find the source code of Rubocop at GitHub: <https://github.com/bbatsov/rubocop>

- Read more about Cookstyle at: <https://github.com/chef/cookstyle>

### **Flagging problems in your Chef cookbooks with Foodcritic**

You might wonder what the proven ways to write cookbooks are. Foodcritic tries to identify possible issues with the logic and style of your cookbooks.

In this section, you'll learn how to use Foodcritic on some existing cookbooks.

#### **Getting ready**

Install version 6.0.0 of the mysql cookbook by running the following code:

```
mma@laptop:~/chef-repo $ knife cookbook site install mysql 6.0.0
Installing mysql to /Users/mma/work/chef-repo/cookbooks
...TRUNCATED OUTPUT...
Cookbook mysql version 6.0.0 successfully installed
```

#### **How to do it...**

Let's see how Foodcritic reports findings:

1. Run foodcritic on your cookbook:

```
mma@laptop:~/chef-repo $ foodcritic ./cookbooks/mysql
...TRUNCATED OUTPUT...
FC001: Use strings in preference to symbols to access node attributes:
./cookbooks/mysql/libraries/helpers.rb:273
FC005: Avoid repetition of resource declarations:
./cookbooks/mysql/libraries/provider_mysql_service.rb:77
...TRUNCATED OUTPUT...
```

2. Get a detailed list of the reported sections inside the mysql cookbook by using the -C flag:

```
mma@laptop:~/chef-repo $ foodcritic -C ./cookbooks/mysql
...TRUNCATED OUTPUT...
FC001: Use strings in preference to symbols to access node attributes
```

```

273| @pkginfo.set[:suse]['11.3']['5.5'][:server_package] = 'mysql'
274|
275| @pkginfo
276| end
cookbooks/mysql/libraries/provider_mysql_service.rb
FC005: Avoid repetition of resource declarations
74| end
75|
76| # Support directories
77| directory "#{$new_resource.name} :create #{etc_dir}" do
78|   path etc_dir
79|   owner new_resource.run_user
80|   group new_resource.run_group

```

### How it works...

Foodcritic defines a set of rules and checks your recipes against each of them. It comes with rules concerning various areas: style, correctness, attributes, strings, portability, search, services, files, metadata, and so on. Running Foodcritic against a cookbook tells you which of its rules matched a certain part of your cookbook. By default, it gives you a short explanation of what you should do along the concerned file and line number.

If you run foodcritic -C, it displays excerpts of the places where it found the rules to match.

In the preceding example, it didn't like it that the mysql cookbook version 6.0.0 uses symbols to access node attributes instead of strings:

```
@pkginfo.set[:suse]['11.3']['5.5'][:server_package] = 'mysql'
```

This could be rewritten as follows:

```
@pkginfo.set['suse']['11.3']['5.5']['server_package'] = 'mysql'
```

**There's more...**

Some of the rules, especially those from the styles section, are opinionated. You're able to exclude certain rules or complete sets of rules, such as `style`, when running Foodcritic:

```
mma@laptop:~/chef-repo $ foodcritic -t '~style' ./cookbooks/mysql  
mma@laptop:~/chef-repo $
```

In this case, the tilde negates the tag selection to exclude all rules with the style tag. Running without the tilde would run the style rules exclusively:

```
mma@laptop:~/chef-repo $ foodcritic -t style ./cookbooks/mysql
```

If you want to run foodcritic in a **continuous integration (CI)** environment, you can use the `-f` parameter to indicate which rules should fail the build:

```
mma@laptop:~/chef-repo $ foodcritic -f style ./cookbooks/mysql  
...TRUNCATED OUTPUT...  
FC001: Use strings in preference to symbols to access node attributes:  
./cookbooks/mysql/libraries/helpers.rb:273  
FC005: Avoid repetition of resource declarations:  
./cookbooks/mysql/libraries/provider_mysql_service.rb:77  
mma@laptop:~/chef-repo $ echo $?
```

In this example, we tell Foodcritic to fail if any rule in the style group fails. In our case, it returns a non-zero exit code instead of zero, as it would if either no rule matches or we omit the `-f` parameter. That non-zero exit code would fail your build on your continuous integration server. You can use `-f any` if you want Foodcritic to fail on all warnings.

**See also**

- ✓ Find out more about Foodcritic and its rules at <http://www.foodcritic.io>
- ✓ Learn how to make sure that your cookbooks compile in the **Testing your Chef cookbooks with cookstyle and Rubocop** recipe in this chapter
- ✓ Check out strainer, a tool to simultaneously test multiple things, such as Foodcritic, knife test, and other stuff, at <http://github.com/customink/strainer>



# AWS Compute

## Introduction

**Elastic Cloud Compute (EC2)** is by far the most utilized and complex service in the AWS catalogue. More than just **virtual machines**, EC2 provides a framework of sub-services to help you secure and manage your instances elastically.

## Creating a key pair

A key pair is used to access your instances via SSH. This is the quickest and easiest way to access your instances.

## Getting ready

To perform this, you must have your AWS CLI tool configured correctly.

## How to do it...

1. Create the key pair, and save it to disk:

```
aws ec2 create-key-pair \
--key-name MyEC2KeyPair \
--query 'KeyMaterial' \
--output text > ec2keypair.pem
```

2. Change the permissions on the created file:

```
chmod 600 ec2keypair.pem
```

## How it works...

This call requests a new private key from EC2. The response is then parsed using a JMESPath query, and the private key (in the KeyMaterial property) is saved to a new key file with the .pem extension.

Finally, we change the permissions on the key file so that it cannot be read by other users—this is required before SSH will allow you to use it.

### Launching an instance

There will be scenarios—usually when testing and developing your infrastructure code—when you need quick access to an instance. Creating it via the AWS CLI is the quickest and most consistent way to create one-off instances.

### Getting ready

We are launching an instance of AWS Linux using an AMI ID in the us-east-1 region. If you are working in a different region, you will need to update your image-id parameter.

You must have configured your AWS CLI tool with working credentials.

### How to do it...

Run the following AWS CLI command, using your own key-pair name:

```
aws ec2 run-instances \
--image-id ami-9be6f38c \
--instance-type t2.micro \
--key-name <your-key-pair-name>
```

### How it works...

While you can create an instance via the AWS web console, it involves many distracting options. When developing and testing, the CLI tool is the best way to provision instances.

While the key-name argument is optional, you will not be able to connect to your instance unless you have pre-configured some other way of logging in.

### Note

The t2.micro instance type used in this recipe is included in the AWS free tier. You can run one micro instance per month for free during the first 12 months of your usage. See <https://aws.amazon.com/free> for more information.

As no VPC or security groups are specified, the instance will be launched in your account's default VPC and security group. The default security group allows access from anywhere, on all ports, and so is not suitable for long-lived instances. You can modify an instance's security groups after it is launched, without stopping it.

### **There's more...**

If you have created your own AMI, then you can change the image-id argument to quickly launch your specific AMI.

You may also want to take note of the InstanceId value in the response from the API, as you may need it for future commands.

### **Attaching storage**

Ideally, you will have defined all your storage requirements up-front as code using a service such as CloudFormation. However, sometimes that is not possible due to application restrictions or changing requirements.

You can easily add additional storage to your instances while they are running by attaching a new volume.

### **Getting ready**

- A running instance's ID. It will start with i- followed by alphanumeric characters.
- The AZ the instance is running in. This looks like the region name with a letter after it; for example, us-east-1a.

In this section, we are using an AWS Linux instance. If you are using a different operating system, the steps to mount the volume will be different. We will be running an instance in the AZ us-east-1a.

You must have configured your AWS CLI tool with working credentials.

**How to do it...**

1. Create a volume:

```
aws ec2 create-volume --availability-zone us-east-1a
```

**Note**

Take note of the returned **VolumeId** in the response. It will start with **vol-** followed by alphanumeric characters.

2. Attach the volume to the instance, using the volume ID noted in the last step and the instance ID you started with:

```
aws ec2 attach-volume \
--volume-id <your-volume-id> \
--instance-id <your-instance-id> \
--device /dev/sdf
```

3. On the instance itself, mount the volume device:

```
mount /dev/xvdf /mnt/volume
```

**How it works...**

we start by creating a volume. Volumes are created from snapshots. If you do not specify a snapshot ID it uses a blank snapshot, and you get a blank volume.

While volumes are hosted redundantly, they are only hosted in a single AZ, so must be provisioned in the same AZ the instance is running in.

The create-volume command returns a response that includes the newly created volume's **VolumeId**. We then use this ID in the next step.

**Note**

It can sometimes take a few seconds for a volume to become available. If you are scripting these commands, use the `aws ec2 wait` command to wait for the volume to become available.

In step 3, we attach a volume to the instance. When attaching to an instance, you must specify the name of the device that it will be presented to the operating system as. Unfortunately, this does not guarantee what the device will appear as. In the case of AWS Linux, `/dev/sdf` becomes `/dev/xvdf`.

**Note**

Device naming is kernel-specific, so if you are using something other than AWS Linux, the device name may be different.

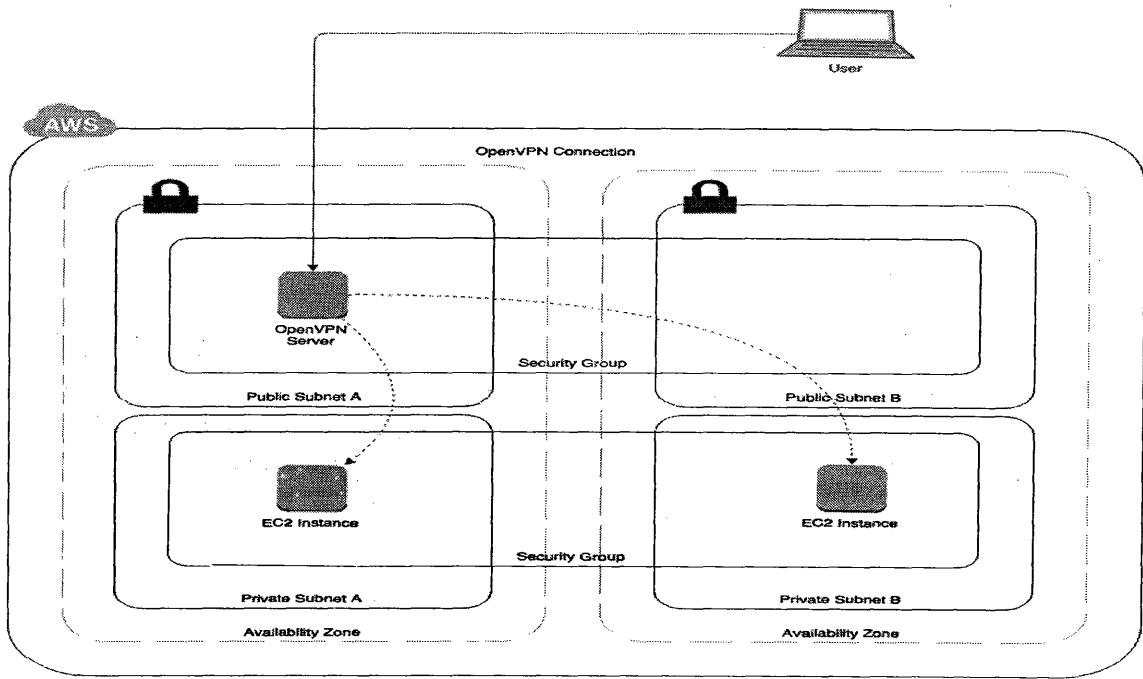
See [http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device\\_naming.html](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device_naming.html) for full details.

  
**Securely accessing private instances**

Any instance or resource living in a private subnet in your VPC will be inaccessible from the Internet. This makes good sense from a security perspective because it gives your instances a higher level of protection.

Of course, if they can't be accessed from the Internet, then they're not going to be easy to administer.

One common pattern is to use a VPN server as a single, highly controlled, entry point to your private network. This is what we're going to show you in this, as pictured in the following diagram:



Accessing private instances securely

### Getting ready

We're going to use OpenVPN for this example. They provide a free (for up to two users) AMI in the AWS marketplace, which has OpenVPN already installed and configured. You'll need to accept the terms and conditions for using this AMI. You can do so by visiting the AMI's marketplace page at <https://aws.amazon.com/marketplace/pp/B00MI40CAE/>.

You need to decide on a password, which will be your **temporary** admin password. We'll feed this password into a CloudFormation template and then change it after we create our stack.

#### Note

You can use the default VPC for this example.

#### How to do it...

1. Create a new CloudFormation template and add the following Mappings. This is a list of all the latest OpenVPN AMIs in each region. We're adding these to maximize region portability for our template—you can omit the regions you have no intention of using:

**Mappings:**

AWSRegion2AMI: # Latest OpenVPN AMI at time of publishing: 2.1.4  
us-east-1:  
    AMI: ami-bc3566ab  
us-east-2:  
    AMI: ami-10306a75  
us-west-2:  
    AMI: ami-d3e743b3  
us-west-1:  
    AMI: ami-4a02492a  
eu-west-1:  
    AMI: ami-f53d7386  
eu-central-1:  
    AMI: ami-ad1fe6c2  
ap-southeast-1:  
    AMI: ami-a859ffcb  
ap-northeast-1:  
    AMI: ami-e9da7c88  
ap-southeast-2:  
    AMI: ami-89477aea  
sa-east-1:  
    AMI: ami-0c069b60

2. We now need to define some Parameters. Firstly we'll need to know which VPC and subnet to deploy our VPN instance to. Note that you need to specify a **public** subnet here, otherwise you won't be able to access your OpenVPN server:

**VpcId:**

Type: AWS::EC2::VPC::Id  
Description: VPC where load balancer and instance will launch

**SubnetId:**

Type: List<AWS::EC2::Subnet::Id>  
Description: Subnet where OpenVPN server will launch  
(pick at least 1)

3. We also need to define InstanceType and KeyName. These are the EC2 instance class and SSH key pair to use to launch our OpenVPN server:

**InstanceType:**

Type: String  
Description: OpenVPN server instance type  
Default: m3.medium

**KeyName:**

Type: AWS::EC2::KeyPair::KeyName  
Description: EC2 KeyPair for SSH access

4. We need a parameter for AdminPassword. This is the temporary password which will be given to the openvpn user (administrator) when the server starts up:

AdminPassword:

Type: String  
Description: Password for 'openvpn' user  
Default: openvpn  
NoEcho: true

5. The last parameter is the CIDR block, which we wish to allow to connect to our VPN server. You may wish to lock this down to the public IP range of your corporate network, for example:

AllowAccessFromCIDR:

Type: String  
Description: IP range/address to allow VPN connections from  
Default: "0.0.0.0/0"

6. The first Resource we need to define is the security group our OpenVPN server will live in. You'll also use this security group to allow access to other resources in your network. Add it to your template as follows:

VPNSecurityGroup:

Type: AWS::EC2::SecurityGroup  
Properties:  
GroupDescription: Inbound access to OpenVPN server  
VpcId: !Ref VpcId  
SecurityGroupIngress:  
- CidrIp: !Ref AllowAccessFromCIDR  
FromPort: 443  
IpProtocol: tcp  
ToPort: 443  
- CidrIp: !Ref AllowAccessFromCIDR  
FromPort: 22  
IpProtocol: tcp  
ToPort: 22  
- CidrIp: !Ref AllowAccessFromCIDR  
FromPort: 1194  
IpProtocol: udp

ToPort: 1194

7. We can now define the actual OpenVPN instance itself. You'll notice that we are explicitly configuring the network interface. This is required, because we want to declare that this instance must get a public IP address (otherwise you won't be able to access it). In the UserData, we declare some variables that the OpenVPN software will pick up when it starts so that it can configure itself:

**OpenVPNInstance:**

Type: AWS::EC2::Instance

**Properties:**

ImageId: !FindInMap [ AWSRegion2AMI, !Ref "AWS::Region", AMI ]

InstanceType: !Ref InstanceType

KeyName: !Ref KeyName

**NetworkInterfaces:**

- AssociatePublicIpAddress: true

DeviceIndex: "0"

**GroupSet:**

- !Ref VPCSecurityGroup

SubnetId: !Select [ 0, Ref: SubnetId ]

**Tags:**

- Key: Name

Value: example-openvpn-server

**UserData:**

Fn::Base64: !Sub

- |

public\_hostname=openvpn

admin\_user=openvpn

admin\_pw=\${admin\_pw}

reroute\_gw=1

reroute\_dns=1

- admin\_pw: !Ref AdminPassword

8. Finally, we add some helpful Outputs:

**Outputs:****OpenVPNAdministration:****Value:**

Fn::Join:

- ""

- - https://

- !GetAtt OpenVPNInstance.PublicIp

- /admin/

Description: Admin URL for OpenVPN server  
OpenVPNClientLogin:

Value:

```
Fn::Join:  
- ""  
- - https://  
- !GetAtt OpenVPNInstance.PublicIp  
- /
```

Description: Client login URL for OpenVPN server

OpenVPNServerIPAddress:

Value: !GetAtt OpenVPNInstance.PublicIp

Description: IP address for OpenVPN server

9. Go ahead and launch this stack in the CloudFormation web console, or via the CLI, with the following command:

```
aws cloudformation create-stack \  
--template-body file://04-securely-access-private-instances.yaml \  
--stack-name example-vpn \  
--parameters \  
ParameterKey=KeyName,ParameterValue=<key-pair-name> \  
ParameterKey=VpcId,ParameterValue=<your-vpc-id> \  
ParameterKey=SubnetId,ParameterValue=<your-public-subnet-id>
```

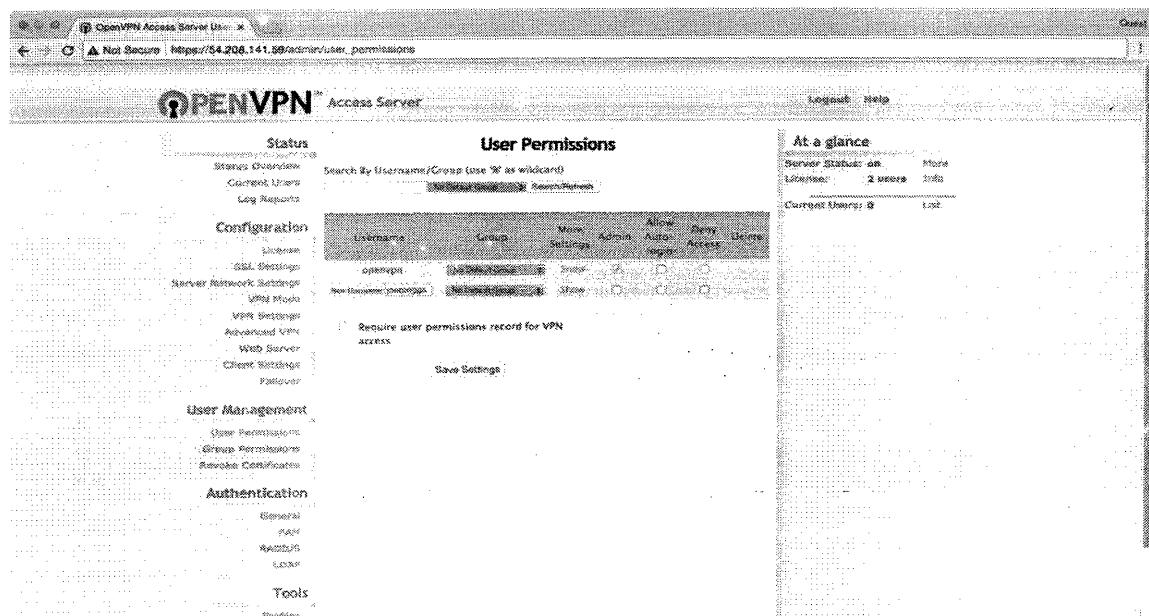
## Configuration

1. Once your stack is created, you'll want to change the password for the openvpn user (administrator). Go to the admin control panel and do this now: <https://<ip-or->

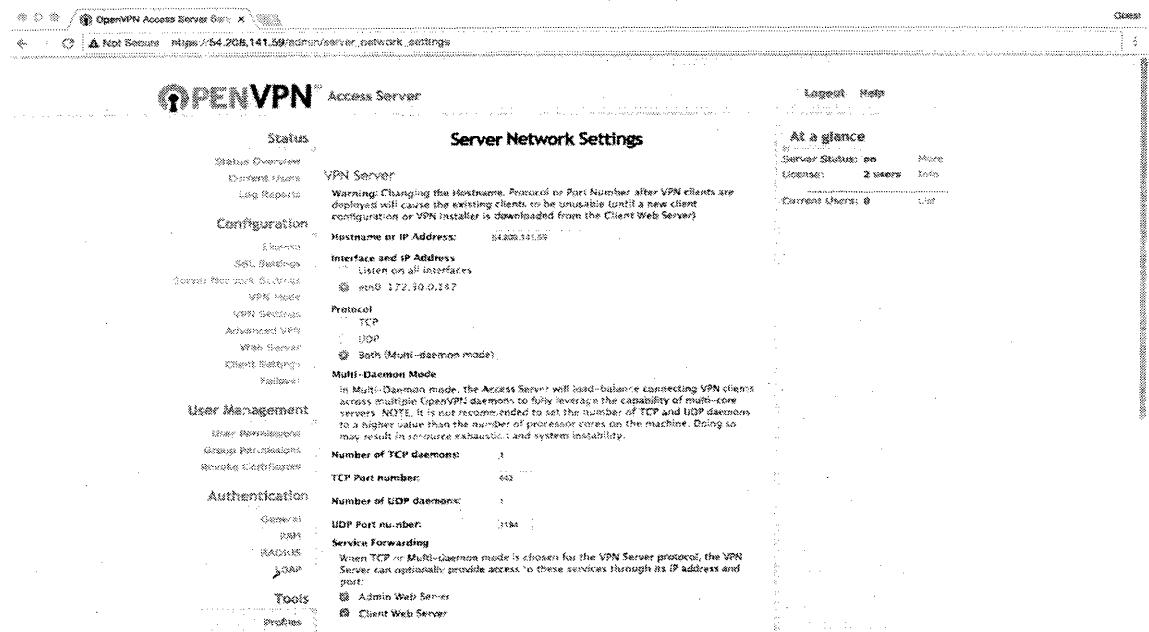
`hostname-of-vpn-server>/admin`. If your VPN server is operating as expected you'll be greeted with a status page after logging in, as pictured in the following screenshot:

The screenshot shows the 'Status Overview' page of the OpenVPN Access Server. The top navigation bar includes 'Logout' and 'Help'. On the left, there's a sidebar with links for 'Status', 'Configuration', 'User Management', 'Authentication', and 'Tools'. The main content area has three columns: 'Status Overview' (server is currently ON), 'Active Configuration' (listing various server settings like IP address, port, and authentication method), and 'At a glance' (showing 2 users online). A 'Documentation' section at the bottom provides links to command-line tools and advanced topics.

While you're there, you should create a non-administrator user account. This will be the account you'll use to connect to the VPN. Add this account on the **User Permissions** page as pictured in the following screenshot:



- Under **Server Network Settings**, in the **Hostname or IP address** field, enter the hostname or IP address of the server. This step is important, and when you download your OpenVPN config file from the server (next step), it will make your life much easier if it has the correct hostname or IP address in it. The next screenshot shows what you can expect to see on the **Server Network Settings** page:



**How it works...**

You should now be able to connect to your VPN server. Go to the user login page and log in with the credentials you gave to the previously mentioned non-administrator user:

<https://<ip-or-hostname-of-vpn-server>/>

After logging in, you will have the option to download the OpenVPN client with configuration which is specific to your account. Alternatively, if you already have a VPN client installed, you can just download the configuration on its own.

**There's more...**

There are a couple of important points you'll need to keep in mind now that you are up and running with an OpenVPN server:

- If you need to SSH to the instance, you must connect with the username `openvpnas`
- To access your other instances, you'll need to allow connections from the VPN security group

**Auto scaling an application server**

**Auto scaling** is a fundamental component of compute in the cloud. It provides not only the ability to scale up and down in response to application load, but also redundancy, by ensuring that capacity is always available. Even in the unlikely event of an AZ outage, the auto scaling group will ensure that instances are available to run your application.

Auto scaling also allows you to pay for only the EC2 capacity you need, because underutilized servers can be automatically de-provisioned.

**Getting ready**

You must supply two or more subnet IDs for this recipe to work.

The following example uses an AWS Linux AMI in the `us-east-1` region. Update the parameters as required if you are working in a different region.

**How to do it...**

1. Start by defining the template version and description:

```
AWSTemplateFormatVersion: "2010-09-09"  
Description: Create an Auto Scaling Group
```

2. Add a Parameters section with the required parameters that will be used later in the template:

```
Parameters:  
SubnetIds:  
Description: Subnet IDs where instances can be launched  
Type: List<AWS::EC2::Subnet::Id>
```

3. Still under the Parameters section, add the optional instance configuration parameters:

```
AmiId:  
Description: The application server's AMI ID  
Type: AWS::EC2::Image::Id  
Default: ami-9be6f38c # AWS Linux in us-east-1  
InstanceType:  
Description: The type of instance to launch  
Type: String  
Default: t2.micro
```

4. Still under the Parameters section, add the optional auto scaling group-configuration parameters:

```
MinSize:  
Description: Minimum number of instances in the group  
Type: Number  
Default: 1  
MaxSize:  
Description: Maximum number of instances in the group  
Type: Number  
Default: 4
```

```
ThresholdCPUHigh:
```

Description: Launch new instances when CPU utilization is over this threshold

Type: Number

Default: 60

ThresholdCPULow:

Description: Remove instances when CPU utilization is under this threshold

Type: Number

Default: 40

ThresholdMinutes:

Description: Launch new instances when over the CPU threshold for this many minutes

Type: Number

Default: 5

## 5. Add a Resources section, and define the auto scaling group resource:

Resources:

AutoScalingGroup:

Type: AWS::AutoScaling::AutoScalingGroup

Properties:

MinSize: !Ref MinSize

MaxSize: !Ref MaxSize

LaunchConfigurationName: !Ref LaunchConfiguration

Tags:

- Key: Name

Value: !Sub "\${AWS::StackName} server"

PropagateAtLaunch: true

VPCZoneIdentifier: !Ref SubnetIds

## 6. Still under the Resources section, define the launch configuration used by the auto scaling group:

LaunchConfiguration:

Type: AWS::AutoScaling::LaunchConfiguration

Properties:

ImageId: !Ref AmiId

InstanceType: !Ref InstanceType

UserData:

Fn::Base64: !Sub |

#!/bin/bash -xe

# This will be run on startup, launch your application here

7. Next, define two scaling policy resources—one to scale up and the other to scale down:

ScaleUpPolicy:

Type: AWS::AutoScaling::ScalingPolicy

Properties:

AdjustmentType: ChangeInCapacity

AutoScalingGroupName: !Ref AutoScalingGroup

Cooldown: 60

ScalingAdjustment: 1

ScaleDownPolicy:

Type: AWS::AutoScaling::ScalingPolicy

Properties:

AdjustmentType: ChangeInCapacity

AutoScalingGroupName: !Ref AutoScalingGroup

Cooldown: 60

ScalingAdjustment: -1

8. Define an alarm that will alert when the CPU goes over the ThresholdCPUHigh parameter:

CPULowAlarm:

Type: AWS::CloudWatch::Alarm

Properties:

ActionsEnabled: true

AlarmActions:

- !Ref ScaleDownPolicy

AlarmDescription: Scale down on CPU load

ComparisonOperator: LessThanThreshold

Dimensions:

- Name: AutoScalingGroupName

Value: !Ref AutoScalingGroup

EvaluationPeriods: !Ref ThresholdMinutes

Namespace: AWS/EC2

Period: 60

Statistic: Average

Threshold: !Ref ThresholdCPULow

MetricName: CPUUtilization

10. Save the template with the filename 04-auto-scaling-an-application-server.yaml.

11. Launch the template with the following AWS CLI command, supplying your subnet IDs:

```
aws cloudformation create-stack \
--stack-name asg \
--template-body file://04-auto-scaling-an-application-server.yaml \
--parameters \
ParameterKey=SubnetIds,ParameterValue='<subnet-id-1>', \
<subnet-id-2>'
```

### How it works...

This example defines an auto scaling group and the dependent resources. These include the following:

- A launch configuration to use when launching new instances
- Two scaling policies, one to scale the number of instances up, and an inverse policy to scale back down
- An alarm to alert when the CPU crosses a certain threshold, for a certain number of minutes

The auto scaling group and launch-configuration resource objects in this example use mostly default values. You will need to specify your own SecurityGroups and a KeyName parameter in the LaunchConfiguration resource configuration if you want to be able to connect to the instances (for example, via SSH).

AWS will automatically take care of spreading your instances evenly over the subnets you have configured, so make sure they are in different AZs! When scaling down, the oldest instances will be removed before the newer ones.

### Scaling policies

The scaling policies detail how many instances to create or delete when they are triggered. It also defines a Cooldownvalue, which helps prevent **flapping** servers—when servers are created and deleted before they have finished starting and are useful.

**Note**

While the scaling policies in this example use equal values, you might want to change that so your application can scale **up** quickly, and scale **down** slowly for the best user experience.

The minimum charge for an instance is **one hour**, so creating and destroying them multiple times in one hour may result in higher than expected charges.

**Creating machine images**

Creating or **baking** your own **Amazon Machine Images (AMIs)** is a key part of systems administration in AWS. Having a pre-baked image helps you provision your servers faster, easier, and more consistently than configuring it by hand.

Packer is the de facto standard tool that helps you make your own AMIs. By automating the launch, configuration, and clean-up of your instances, it makes sure you get a repeatable image every time.

In this recipe, we will create an image with the Apache web server pre-installed and configured. While this is a simple example, it is also a very common use-case.

By baking-in your web server, you can scale up your web serving layer to dynamically match the demands on your websites. Having the software already installed and configured means you get the fastest and most reliable start-up possible.

**Getting ready**

For this recipe, you must have the Packer tool available on your system. Download and install Packer from the project's website <https://www.packer.io/downloads.html>.

**How to do it...**

1. Create a new Packer template file, and start by defining an amazon-ebs builder in the builders section:

```
"builders": [
```

```
{  
  "type": "amazon-ebs",  
  "instance_type": "t2.micro",  
  "region": "us-east-1",  
  "source_ami": "ami-9be6f38c",  
  "ssh_username": "ec2-user",  
  "ami_name": "aws-linux-apache {{timestamp}}"  
}  
],
```

#### Note

The entire template file must be a valid JSON object. Remember to enclose the sections in curly braces: { ... }.

2. Create a provisioners section, and include the following snippet to install and activate Apache:

```
"provisioners": [  
  {  
    "type": "shell",  
    "inline": [  
      "sudo yum install -y httpd",  
      "sudo chkconfig httpd on"  
    ]  
  }  
]
```

3. Save the file with a specific name, such as 04-creating-machine-images.json.
4. Validate the configuration file you've created with the following packer validate command:

```
packer validate 04-creating-machine-images.json
```

6. When valid, build the AMI with the following command:  
**packer build 04-creating-machine-images.json**

Wait until the process is complete. While it is running, you will see an output similar to the following:

```
awsac - packer - packer build src/04-creating-machine-images.json - 314x81
$ packer build src/04-creating-machine-images.json
amazon-ebs output will be in this color.

==> amazon-ebs: Prevalidating AMI Name...
    amazon-ebs: Found Image ID: ami-9be6f38c
==> amazon-ebs: Creating temporary keypair: packer_587a9f63-1bcc-7e85-b9c9-eeec160cc172
==> amazon-ebs: Creating temporary security group for this instance...
==> amazon-ebs: Authorizing access to port 22 the temporary security group...
==> amazon-ebs: Launching a source AWS instance...
    amazon-ebs: Instance ID: i-0c249b526d0cabeb9b
==> amazon-ebs: Waiting for instance (i-0c249b526d0cabeb9b) to become ready...
==> amazon-ebs: Waiting for SSH to become available...
==> amazon-ebs: Connected to SSH!
==> amazon-ebs: Provisioning with shell script: /var/folders/vt/lkw7w5ns6hi6vt8j_tk08prm8000gn/T/packer-shell21831
1435
    amazon-ebs: Loaded plugins: priorities, update-motd, upgrade-helper
    amazon-ebs: Resolving Dependencies
    amazon-ebs: --> Running transaction check
    amazon-ebs: ---> Package httpd.x86_64 0:2.2.31-1.8.amzn1 will be installed
```

7. Take note of the AMI ID returned by Packer so that you can use it when launching instances in the future:

### How it works...

While this is very simple, there is a lot going on behind the scenes. This is why we recommend you use Packer to create your machine images.

### Template

In the builders section of the template, we define our build details.

We are using the most common type of AMI builder: amazon-ebs. There are other types of AWS builders, for instance, storage-backed instance types.

Next, we define the type of instance to use when baking.

**Note**

Make sure that you can often decrease the time it takes to bake your instance by using a larger instance size. Remember that the minimum price paid for an instance is one hour of billable time.

The `source_ami` property in this recipe is an AWS Linux AMI ID in the region we have specified. The `ssh_username` allows you to set the username used to connect and run provisioners on the instance. This will be determined by your operating system, which in our case is `ec2-user`.

Finally, the `ami_name` field includes the built-in Packer variable `{{timestamp}}`. This ensures the AMI you create will always have a unique name.

**Validate the template**

The `packer validate` command is a quick way to ensure your template is free of syntax errors before you launch any instances.

**Build the AMI**

Once you have created and validated your template, the `packer build` command does the following for you:

- Creates a one-time key pair for SSH access to the instance
- Creates a dedicated security group to control access to the instance
- Launches an instance
- Waits until SSH is ready to receive connections
- Runs the provisioner steps on the instance
- Stops the instance
- Generates an AMI from the stopped instance
- Terminates the instance

**There's more...**

While Packer makes the administration of images much easier on AWS, there are still a few things to watch out for.

**Debugging**

Obviously, with so many steps being automated for you, there are many things that can potentially go wrong. Packer gives you a few different ways to debug issues with your builds.

One of the most useful arguments to use with Packer is the `-debug` flag. This will force you to manually confirm each step **before** it takes place. Doing this makes it easy to work out exactly which step in the command is failing, which in turn usually makes it obvious what needs to be changed.

Another useful thing to do is to raise the level of logging output during a Packer command. You can do this by setting the `PACKER_LOG` variable to true. The easiest way to do this is with `PACKER_LOG=1` at the beginning of your Packer command line. This will mean you get a lot more information printed to the console (for example, SSH logs, AWS API calls, and so on) during the command. You may even want to run with this level of logging normally in your builds, for auditing purposes.

**Orphaned resources**

Packer does a great job of managing and cleaning up the resource it uses, but it can only do that while it is running.

If your Packer job aborts for any reason (most likely network issues) then there may be some resources left **orphaned**, or **unmanaged**. It is good practice to check for any Packer instances (they will have **Packer** in their name), and stop them if there are no active Packer jobs running.

You may also need to clean up any leftover key pairs and security groups, but this is less of an issue as there is no cost associated with them (unlike instances).

### Deregistering AMIs

As it becomes easier to create AMIs, you may find you end up with more than you need!

AMIs are made up of EC2 snapshots, which are stored in S3. There is a cost associated with storing snapshots, so you will want to clean them up periodically. Given the size of most AMIs (usually a few GBs), it is unlikely to be one of your major costs.

An even greater cost is the administrative overhead of managing too many AMIs. As your images improve and fixes are applied (especially security fixes), you may want to prevent people from using them.

To remove an AMI, you must first **deregister** it, and then remove the underlying snapshots.

#### Note

Make sure you do not deregister AMIs that are currently in use. For example, an auto scaling group that references a deregistered AMI will fail to launch new instances!

You can easily deregister snapshots through the web console or using the AWS CLI tool.

Once an AMI is no longer registered, you can remove the associated snapshots. Packer automatically adds the AMI ID to the snapshots' description. By searching your snapshots for the deregistered AMI ID, you can find which ones need to be deleted.

You will not be able to delete snapshots if the AMI has not been deregistered, or if the deregistration is still taking place (it can take a few minutes).

### Creating security groups

AWS describes security groups as **virtual firewalls**. While this analogy helps newcomers to the EC2 platform understand their purpose and function, it's probably more accurate to describe them as a **firewall-like** method of authorizing traffic. They don't offer all the functionality you'd find in a traditional firewall, but this simplification also makes them

extremely powerful, particularly when combined with Infrastructure as Code and modern SDLC practices.

We're going to go through a basic scenario involving a web server and load balancer. We want the load balancer to respond to HTTP requests from everywhere, and we want to isolate the web server from everything except the load balancer.

### Getting ready

Before we get started there's a small list of things you'll need to have ready:

- **AmiId:** This is the ID of an AMI in your region. For this recipe, we'd recommend using an AWS Linux AMI because our instance will attempt to run some yum commands on startup.
- **VPCID:** This is the ID of the VPC you wish to launch the EC2 server into.
- **SubnetIDs:** These are the subnets which our EC2 instance can launch in.

### How to do it...

1. Open up your text editor and create a new CloudFormation template. We're going to start by adding a few Parameters as follows:

```
AWSTemplateFormatVersion: '2010-09-09'  
Parameters:  
  AmiId:  
    Type: AWS::EC2::AMI::Id  
    Description: AMI ID to launch instances from  
  VPCID:  
    Type: AWS::EC2::VPC::Id  
    Description: VPC where load balancer and instance will launch  
  SubnetIDs:  
    Type: List<AWS::EC2::Subnet::Id>  
    Description: Subnets where load balancer and instance will launch  
    (pick at least 2)
```

2. Let's take a look at a security group we'll apply to a public load balancer:

```
ExampleELBSecurityGroup:  
  Type: AWS::EC2::SecurityGroup  
  Properties:
```

GroupDescription: Security Group for example ELB  
SecurityGroupIngress:

- IpProtocol: tcp
- CidrIp: 0.0.0.0/0
- FromPort: 80
- ToPort: 80

Anything which resides in this security group will allow inbound TCP connections on port 80 from anywhere (0.0.0.0/0). Note that a security group can contain more than one rule; we'd almost certainly want to also allow HTTPS (443), but we've left it out to simplify this recipe.

### 3. Now let's look at a security group for a web server sitting behind our load balancer:

ExampleEC2InstanceSecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: Security Group for example Instance

SecurityGroupIngress:

- IpProtocol: tcp
- SourceSecurityGroupName:
- Ref: ExampleELBSecurityGroup
- FromPort: 80
- ToPort: 80

Here you can see we are not specifying a source IP range. Instead, we're specifying a source security group, which we will accept connections from. In this case, we're saying that we want to allow anything from our ELB security group to connect to anything in our EC2 instance security group on port 80. Since this is the only rule we're specifying, our web server will not accept connections from anywhere except our load balancer, to port 80 or otherwise. Our web server isn't wide open to the Internet, and it is even isolated from other instances in our VPC

#### Note

Remember that multiple instances can reside in a security group. In a scenario where you have multiple web servers attached to this load balancer it would be unnecessary, inefficient, and somewhat of an anti-pattern to create a new security group for each web server. Given that all web servers attached to this load balancer would be serving the same role or function, it makes sense to apply the same security group to them.

This is where the power of security groups really comes in. If an EC2 instance is serving multiple roles—let's say you have an outbound HTTP proxy server in your VPC which you also want to act as an SMTP relay—then you can simply apply multiple security groups to it.

4. Next, we need to add our load balancer. This is probably the most basic load balancer configuration you'll come across. The following code will give you a load balancer, a listener and a target group containing our EC2 instance.

ExampleLoadBalancer:

Type: AWS::ElasticLoadBalancingV2::LoadBalancer

Properties:

Subnets:

- Fn::Select: [ 0, Ref: SubnetIDs ]
- Fn::Select: [ 1, Ref: SubnetIDs ]

SecurityGroups:

- Fn::GetAtt: ExampleELBSecurityGroup.GroupId

ExampleListener:

Type: AWS::ElasticLoadBalancingV2::Listener

Properties:

LoadBalancerArn:

Ref: ExampleLoadBalancer

DefaultActions:

- Type: forward

TargetGroupArn:

Ref: ExampleTargetGroup

Port: 80

Protocol: HTTP

ExampleTargetGroup:

Type: AWS::ElasticLoadBalancingV2::TargetGroup

Properties:

Port: 80

Protocol: HTTP

VpcId:

Ref: VPCID

Targets:

- Id:

Ref: ExampleEC2Instance

5. The last resource we'll add to our template is an EC2 server. This server will install and start nginx when it boots.

ExampleEC2Instance:

Type: AWS::EC2::Instance

Properties:

InstanceType: t2.nano

```
UserData:  
Fn::Base64:  
Fn::Sub: |  
#!/bin/bash -ex  
yum install -y nginx  
service nginx start  
exit 0  
ImageId:  
Ref: AmiId  
SecurityGroupIds:  
- Fn::GetAtt: ExampleEC2InstanceSecurityGroup.GroupId  
SubnetId:  
Fn::Select: [ 0, Ref: SubnetIDs ]
```

6. Lastly, we're going to add some **Outputs** to the template to make it a little more convenient to use our ELB and EC2 instance after the stack is created.

```
Outputs:  
ExampleEC2InstanceHostname:  
Value:  
Fn::GetAtt: [ ExampleEC2Instance, PublicDnsName ]  
ExampleELBURL:  
Value:  
Fn::Join:  
- "  
- [ 'http://', { 'Fn::GetAtt': [ ExampleLoadBalancer,  
DNSName ] }, '/' ]
```

7. Go ahead and launch this template using the CloudFormation web console or the AWS CLI.

### There's more...

You'll eventually run into circular dependency issues when configuring security groups using CloudFormation. Let's say you want all servers in our ExampleEC2InstanceSecurityGroup to be able to access each other on port 22 (SSH). In order to achieve this, you would need to add this rule as the separate resource type AWS::EC2::SecurityGroupIngress. This is because a security group can't refer to itself in CloudFormation when it is yet to be created. This is what the extra resource type looks like:

```
ExampleEC2InstanceIngress:  
Type: AWS::EC2::SecurityGroupIngress  
Properties:  
IpProtocol: tcp
```

```
SourceSecurityGroupName:  
  Ref: ExampleEC2InstanceSecurityGroup  
GroupName:  
  Ref: ExampleEC2InstanceSecurityGroup  
FromPort: 22  
ToPort: 22
```

### Differences from traditional firewalls

- Security groups can't be used to explicitly block traffic. Only rules of a permissive kind can be added; deny style rules are not supported. Essentially, all inbound traffic is denied unless you explicitly allow it.
- Your rules also may not refer to source ports; only destination ports are supported.
- When security groups are created, they will contain a rule which allows all outbound connections. If you remove this rule, new outbound connections will be dropped. It's a common pattern to leave this rule in place and filter all your traffic using inbound rules only.
- If you do replace the default outbound rule, it's important to note that only new outbound connections will be filtered. Any outbound traffic being sent in response to an inbound connection will still be allowed. This is because security groups are **stateful**.
- Unlike security groups, network ACLs are not stateful and do support DENY rules. You can use them as a complementary layer of security inside your VPC, especially if you need to control traffic flow between subnets.

### Creating a load balancer

AWS offers two kinds of load balancers:

- Classic load balancer
- Application load balancer

We're going to focus on the application load balancer. It's effectively an upgraded, second generation of the ELB service, and it offers a lot more functionality than the classic load balancer. HTTP/2 and WebSockets are supported natively, for example. The hourly rate also happens to be cheaper.

### Note

Application load balancers do not support layer-4 load balancing. For this kind of functionality, you'll need to use a classic load balancer.

**How to do it...**

1. Open up your text editor and create a new CloudFormation template. We're going to require a VPC ID and some subnet IDs as **Parameters**. Add them to your template like this:

```
AWSTemplateFormatVersion: '2010-09-09'
```

Parameters:

VPCID:

Type: AWS::EC2::VPC::Id

Description: VPC where load balancer and instance will launch

SubnetIDs:

Type: List<AWS::EC2::Subnet::Id>

Description: Subnets where load balancer and instance will launch  
(pick at least 2)

2. Next we need to add some **Mappings** of ELB account IDs. These will make it easier for us to give the load balancer permission to write logs to an S3 bucket. Your mappings should look like this:

**Note**

You can find the complete list of ELB account IDs here <http://docs.aws.amazon.com/elasticloadbalancing/latest/classic/enable-access-logs.html#attach-bucket-policy>.

Mappings:

ELBAccountMap:

us-east-1:

ELBAccountID: 127311923021

ap-southeast-2:

ELBAccountID: 783225319266

3. We can now start adding Resources to our template. First we're going to create an S3 bucket and bucket policy for storing our load balancer logs. In order to make this template portable, we'll omit a bucket name, but for convenience we'll include the bucket name in our outputs so that CloudFormation will echo the name back to us.

Resources:

ExampleLogBucket:

Type: AWS::S3::Bucket

ExampleBucketPolicy:

Type: AWS::S3::BucketPolicy  
Properties:  
    Bucket:  
        Ref: ExampleLogBucket  
    PolicyDocument:  
        Statement:  
            Action:  
                - "s3:PutObject"  
            Effect: "Allow"  
            Resource:  
                Fn::Join:  
                    - ""  
                    - "arn:aws:s3:::"  
                    - Ref: ExampleLogBucket  
                    - "/"\*"  
        Principal:  
            AWS:  
                Fn::FindInMap: [ ELBAccountMap, Ref: "AWS::Region",  
                                  ELBAccountID ]

4. Next, we need to create a security group for our load balancer to reside in. This security group will allow inbound connections to port 80 (HTTP). To simplify this recipe, we'll leave out port 443 (HTTPS), but we'll briefly cover how to add this functionality later in this section. Since we're adding a public load balancer, we want to allow connections to it from everywhere (0.0.0.0/0). This is what our security group looks like:

ExampleELBSecurityGroup:  
Type: AWS::EC2::SecurityGroup  
Properties:  
    GroupDescription: Security Group for example ELB  
    SecurityGroupIngress:  
        IpProtocol: tcp  
        CidrIp: 0.0.0.0/0  
        FromPort: 80  
        ToPort: 80

5. We now need to define a target group. Upon completion of this recipe, you can go ahead and register your instances in this group so that HTTP requests will be forwarded to it. Alternatively, you can attach the target group to an auto scaling group and AWS will take care of the instance registration and de-registration for you.

6. The target group is where we specify the health checks our load balancer should perform against the target instances. This health check is necessary to determine if a registered instance should receive traffic. The example provided with this recipe includes these health-check parameters with the values all set to their defaults. Go ahead and tweak these to suit your needs, or, optionally, remove them if the defaults work for you.

**ExampleTargetGroup:**

Type: AWS::ElasticLoadBalancingV2::TargetGroup

Properties:

Port: 80

Protocol: HTTP

HealthCheckIntervalSeconds: 30

HealthCheckProtocol: HTTP

HealthCheckPort: 80

HealthCheckPath: /

HealthCheckTimeoutSeconds: 5

HealthyThresholdCount: 5

UnhealthyThresholdCount: 2

Matcher:

HttpCode: '200'

VpcId:

Ref: VPCID

7. We need to define at least one listener to be added to our load balancer. A listener will **listen** for incoming requests to the load balancer on the port and protocol we configure for it. Requests matching the port and protocol will be forwarded through to our target group.

The configuration of our listener is going to be reasonably simple. We're listening for HTTP requests on port 80. We're also setting up a default action for this listener, which will forward our requests to the target group we've defined before. There is a limit of 10 listeners per load balancer.

**Note**

Currently, AWS only supports one action: forward.

**ExampleListener:**

Type: AWS::ElasticLoadBalancingV2::Listener

Properties:

LoadBalancerArn:  
Ref: ExampleLoadBalancer  
DefaultActions:  
- Type: forward  
TargetGroupArn:  
Ref: ExampleTargetGroup  
Port: 80  
Protocol: HTTP

8. Finally, now that we have all Resources we need, we can go ahead and set up our load balancer. We'll need to define at least two subnets for it to live in—these are included as Parameters in our example template:

ExampleLoadBalancer:  
Type: AWS::ElasticLoadBalancingV2::LoadBalancer  
Properties:  
LoadBalancerAttributes:  
- Key: access\_logs.s3.enabled  
Value: true  
- Key: access\_logs.s3.bucket  
Value:  
Ref: ExampleLogBucket  
- Key: idle\_timeout.timeout\_seconds  
Value: 60  
Scheme: internet-facing  
Subnets:  
- Fn::Select: [ 0, Ref: SubnetIDs ]  
- Fn::Select: [ 1, Ref: SubnetIDs ]  
SecurityGroups:  
- Fn::GetAtt: ExampleELBSecurityGroup.GroupId

9. Lastly, we're going to add some Outputs to our template for convenience. We're particularly interested in the name of the S3 bucket we created and the URL of the load balancer.

Outputs:  
ExampleELBURL:  
Value:  
Fn::Join:  
- "  
- [ 'http://', { 'Fn::GetAtt': [ ExampleLoadBalancer,  
DNSName ] }, '/' ]  
ExampleLogBucket:  
Value:

Ref: ExampleLogBucket

### How it works...

As you can see, we're applying a logging configuration which points to the S3 bucket we've created. We're configuring this load balancer to be Internet-facing, with an idle timeout of 60 seconds (the default).

All load balancers are Internet-facing by default, so it's not strictly necessary to define a Scheme in our example; however, it can be handy to include this anyway. This is especially the case if your CloudFormation template contains a mix of public and private load balancers.

### Note

If you specify a logging configuration but the load balancer can't access the S3 bucket, your CloudFormation stack will fail to complete.

Private ELBs are not Internet-facing and are available only to resources which live inside your VPC.

That's it! You now have a working application load balancer configured to ship logs to an S3 bucket.

### There's more...

Load balancers on AWS are highly configurable and there are many options available to you. Here are some of the more frequent ELB options you'll encounter:

### HTTPS/SSL

If you wish to accept HTTPS requests, you'll need to configure an additional listener. It will look something like the following:

ExampleHTTPSLListener:

Type: AWS::ElasticLoadBalancingV2::Listener

Properties:

Certificates:

- CertificateArn:

- arn:aws:acm:ap-southeast-2:123456789012:  
certificate/12345678-1234-1234-1234-123456789012

LoadBalancerArn:

- Ref: ExampleLoadBalancer

DefaultActions:

- Type: forward

- TargetGroupArn:

- Ref: ExampleTargetGroup

Port: 443

Protocol: HTTPS

The listener will need to reference a valid **Amazon Resource Name (ARN)** for the certificate you wish to use. It's really easy to have AWS Certificate Manager create a certificate for you, but it does require validation of the domain name you're generating the certificate for. You can, of course, bring your own certificate if you wish. You'll need to import it in to AWS Certificate Manager before you can use it with your ELB (or CloudFront distribution).

Unless you have specific requirements around ciphers, a good starting approach is to not define an SSL Policy and let AWS choose what is currently **best of breed**.

### Path-based routing

Once you are comfortable with ELB configuration, you can start to experiment with path-based routing. In a nutshell, it provides a way to inspect a request and proxy it to different targets based on the path requested.

One common scenario you might encounter is needing to route requests for /blog to a different set of servers running WordPress, instead of to your main server pool, which is running your Ruby on Rails application.

# AWS Networking with VPC



## Contents

|  |    |
|--|----|
| Introducing Amazon Web Services .....            | 3  |
| <b>AWS architecture and components</b> .....     | 3  |
| An overview of Amazon VPC.....                   | 8  |
| <b>VPC limits and costs</b> .....                | 16 |
| Working with VPCs.....                           | 17 |
| <b>VPC deployment scenarios</b> .....            | 17 |
| <b>Getting started with the VPC wizard</b> ..... | 18 |
| <b>Launching instances in your VPC</b> .....     | 31 |
| Planning next steps .....                        | 33 |
| Best practices and recommendations .....         | 35 |
| VPC Cloud Formation Recipes .....                | 36 |
| Building a secure network.....                   | 36 |
| <b>Getting ready</b> .....                       | 37 |
| <b>How to do it...</b> .....                     | 38 |
| <b>How it works...</b> .....                     | 45 |
| <b>There's more...</b> .....                     | 46 |
| Creating a NAT gateway .....                     | 47 |
| <b>Getting ready</b> .....                       | 47 |
| <b>How to do it...</b> .....                     | 48 |
| <b>How it works...</b> .....                     | 49 |
| Network logging and troubleshooting.....         | 49 |
| <b>Getting ready</b> .....                       | 50 |
| <b>How to do it...</b> .....                     | 50 |
| <b>How it works...</b> .....                     | 52 |
| <b>There's more...</b> .....                     | 52 |

## Introducing Amazon Web Services

Amazon Web Services or AWS is a comprehensive public cloud computing platform that offers a variety of web-based products and services on an on-demand and pay-per-use basis. AWS was earlier a part of the e-commerce giant Amazon.com, and it wasn't until 2006 that AWS became a separate entity of its own. Today, AWS operates globally with data centers located in USA, Europe, Brazil, Singapore, Japan, China, and Australia. AWS provides a variety of mechanisms, using which the end users can connect to and leverage its services, the most common form of interaction being the web-based dashboard also called as AWS Management Console.

## AWS architecture and components

Before we begin with the actual signup process, it is important to take a look at some of the key architecture and core components of services offered by AWS.

### Regions and availability zones

We do know that AWS is spread out globally and has its presence across USA, Europe, Asia, Australia, and so on. Each of these areas is termed as a region. AWS currently has about 10 regions, each containing multiple data centers within themselves. So what's with all these regions and why do they matter? In simple terms, the resources that are geographically close to your organization are served much faster! For example, an organization running predominantly from USA can leverage the USA's regions to host their resources and gain access to them much faster.

For most of the AWS services that you use, you will be prompted to select a region in which you want to deploy the service. Each region is completely isolated from the other and runs independently as well.

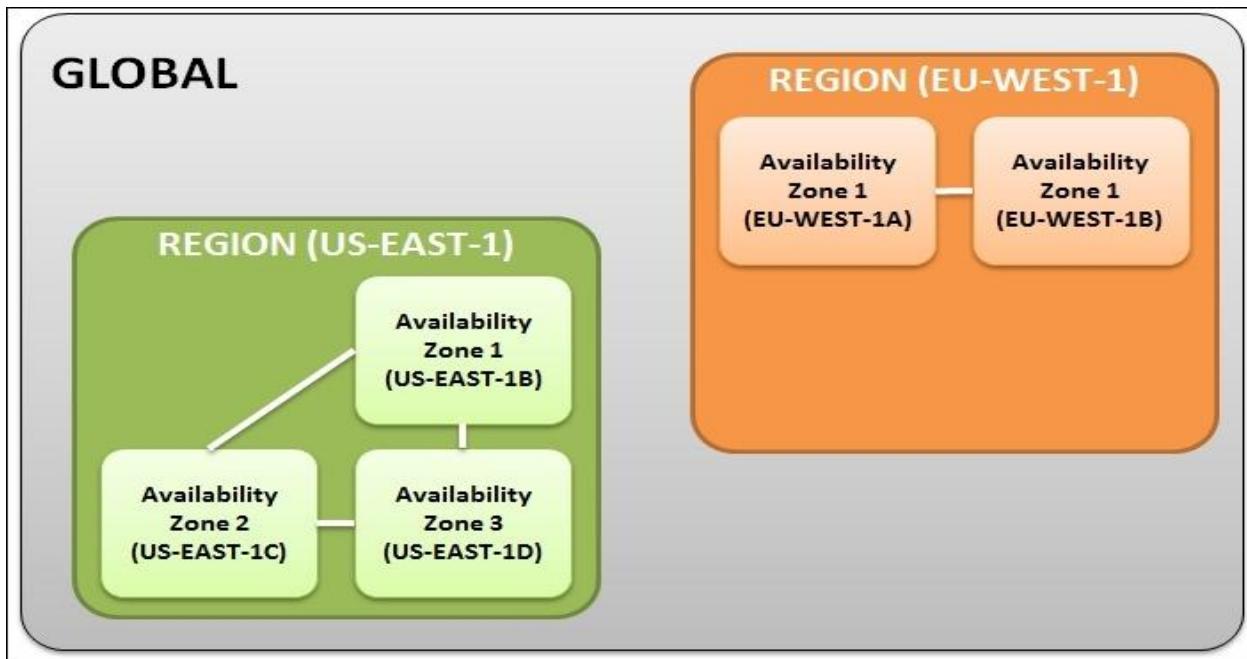
Note: AWS does not replicate resources across regions automatically. It is up to the end user to set up the replication process.

A list of regions and their corresponding codes is provided here for your reference. The code is basically how AWS refers to its multiple regions:

| Region        | Name                    | Code         |
|---------------|-------------------------|--------------|
| North America | US East (N. Virginia)   | us-east-1    |
|               | US West (N. California) | us-west-1    |
|               | US West (Oregon)        | us-west-2    |
| South America | Sao Paulo               | sa-east-1    |
| Europe        | EU (Frankfurt)          | eu-central-1 |
|               | EU (Ireland)            | eu-west-1    |

|      |                          |                |
|------|--------------------------|----------------|
| Asia | Asia Pacific (Tokyo)     | ap-northeast-1 |
|      | Asia Pacific (Singapore) | ap-southeast-1 |
|      | Asia Pacific (Sydney)    | ap-southeast-2 |
|      | Asia Pacific (Beijing)   | cn-north-1     |

Each region is split up into one or more **Availability Zones (AZs)** and pronounced as A-Zees. An A Z is an isolated location inside a region. AZs within a particular region connect to other AZs via low-latency links. What do these AZs contain? Well, ideally they are made up of one or more physical data centers that host AWS services on them. Just as with regions, even AZs have corresponding codes to identify them, generally they are regional names followed by a numerical value. For example, if you select and use us-east-1, which is the North Virginia region, then it would have AZs listed as us-east-1b, us-east-1c, us-east-1d, and so on:



AZs are very important from a design and deployment point of view. Being data centers, they are more than capable of failure and downtime, so it is always good practice to distribute your resources across multiple AZs and design your applications such that they can remain available even if one AZ goes completely offline.

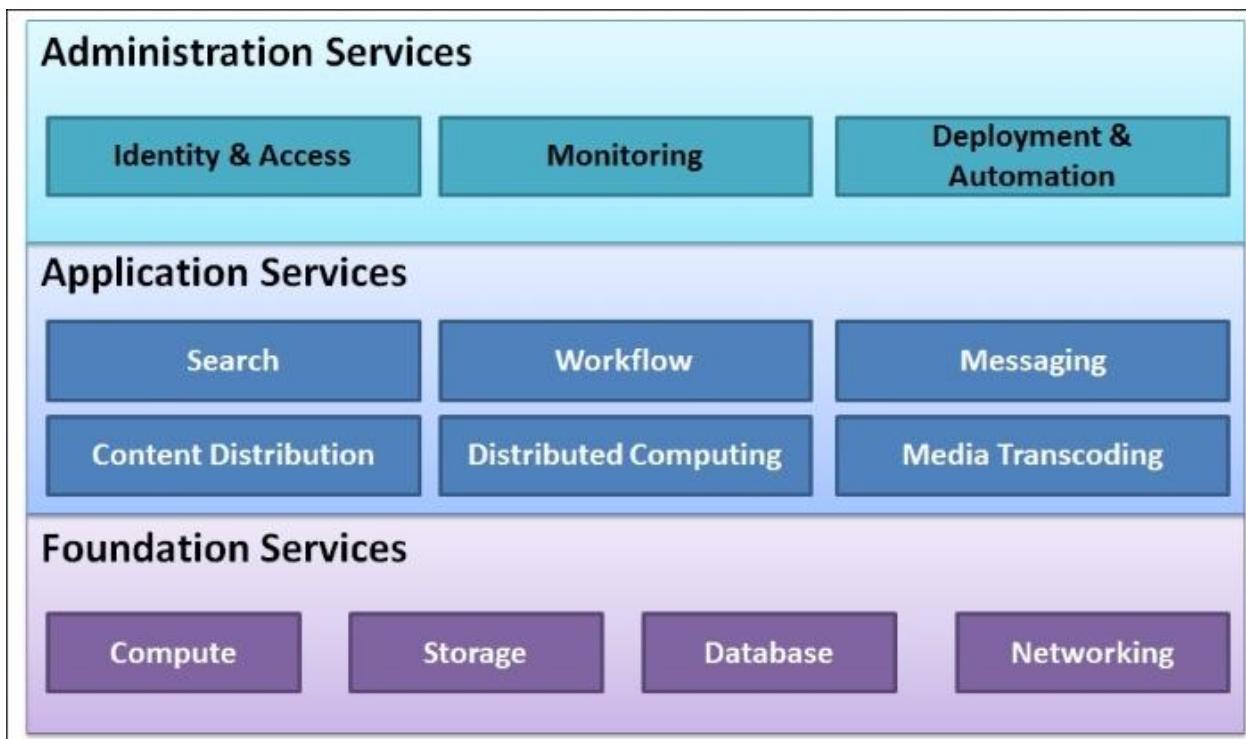
An important point to note here is that AWS will always provide the services and products to you as a customer; however, it is your duty to design and distribute your applications so that they do not suffer any potential outages or failures.

### AWS platform overview

The AWS platform consists of a variety of services that you can use either in isolation or in combination based on your organization's needs. This section will introduce you to some of the most commonly used services as well as some newly launched ones. To begin with, let's divide the services into three major classes:

- **Foundation services:** This is generally the pillars on which the entire AWS infrastructure commonly runs on, including the compute, storage, network, and databases.
- **Application services:** This class of services is usually more specific and generally used in conjunction with the foundation services to add functionality to your applications. For example, services such as distributed computing, messaging and Media Transcoding, and other services fall under this class.
- **Administration services:** This class deals with all aspects of your AWS environment, primarily with identity and access management tools, monitoring your AWS services and resources, application deployments, and automation.

Let's take a quick look at some of the key services provided by AWS. However, do note that this is not an exhaustive list:



We will discuss each of the foundation services.

## Compute

This includes the following services:

- **Elastic Compute Cloud (EC2):** When it comes to brute computation power and scalability, there must be very few cloud providers out there in the market that can match AWS's EC2 service. EC2 or Elastic Compute Cloud is a web service that provides flexible, resizable, and secure compute capacity on an on-demand basis. AWS started off with EC2 as one of its core services way back in 2006 and has not stopped

bringing changes and expanding the platform ever since. The compute infrastructure runs on a virtualized platform that predominantly consists of the open sourced Xen virtualization engine. We will be exploring EC2 and its subsequent services in detail in the coming chapters.

- **EC2 Container Service:** A recently launched service, the EC2 Container Service, allows you to easily run and manage docker containers across a cluster of specially created EC2 instances.
- **Amazon Virtual Private Cloud (VPC):** VPC enables you to create secure, fully customizable, and isolated private clouds within AWS's premises. They provide additional security and control than your standard EC2 along with connectivity options to on premise data centers.

## Storage

This includes the following services:

- **Simple Storage Service (S3):** S3 is a highly reliable, fault tolerant, and fully redundant data storage infrastructure provided by AWS. It was one of the first services offered by AWS way back in 2006, and it has not stopped growing since. As of April 2013, an approximate 2 trillion objects have been uploaded to S3, and these numbers are growing exponentially each year.
- **Elastic Block Storage (EBS):** EBS is a raw block device that can be attached to your compute EC2 instances to provide them with persistent storage capabilities.
- **Amazon Glacier:** It is a similar service offering to S3. Amazon Glacier offers long-term data storage, archival, and backup services to its customers.
- **Amazon Elastic File System:** Yet another very recent service offering introduced by AWS, **Elastic File System(EFS)** provides scalable and high-performance storage to EC2 compute instances in the form of an NFS filesystem.

## Databases

This includes the following services:

- **Amazon Relational Database Service (RDS):** RDS provides a scalable, high-performance relational database system such as MySQL, SQL Server, PostgreSQL, and Oracle in the cloud. RDS is a completely managed solution provided by AWS where all the database heavy lifting work is taken care of by AWS.
- **Amazon DynamoDB:** DynamoDB is a highly scalable NoSQL database as a service offering provided by AWS.
- **Amazon Redshift:** Amazon Redshift is a data warehouse service that is designed to handle and scale to petabytes of data. It is primarily used by organizations to perform real-time analytics and data mining.

## Networking

This includes the following services:

- **Elastic Load Balancer (ELB):** ELB is a dynamic load balancing service provided by AWS used to distribute traffic among EC2 instances. You will be learning about ELB a bit more in detail in subsequent chapters.
- **Amazon Route 53:** Route 53 is a highly scalable and available DNS web service provided by AWS. Rather than configuring DNS names and settings for your domain provider, you can leverage Route 53 to do the heavy lifting work for you.

## Distributed computing and analytics

This includes the following services:

- **Amazon Elastic MapReduce (EMR):** As the name suggests, this service provides users with a highly scalable and easy way to distribute and process large amounts of data using Apache's Hadoop. You can integrate the functionalities of EMR with Amazon S3 to store your large data or with Amazon DynamoDB as well.
- **Amazon Redshift:** This is a massive data warehouse that users can use to store, analyze, and query petabytes of data.

## Content distribution and delivery

**Amazon CloudFront** is basically a content delivery web service that can be used to distribute various types of content, such as media, files, and so on, with high data transfer speeds to end users globally. You can use CloudFront in conjunction with other AWS services such as EC2 and ELB as well.

## Workflow and messaging

This includes the following services:

- **Amazon Simple Notification Service (SNS):** SNS is a simple, fully managed push messaging service provided by AWS. You can use it to push your messages to mobile devices (SMS service) and even to other AWS services as API calls to trigger or notify certain activities.
- **Amazon Simple Email Service (SES):** As the name suggests, SES is used to send bulk e-mails to various recipients. These e-mails can be anything, from simple notifications to transactions messages, and so on. Think of it as a really large mail server that can scale as per your requirements and is completely managed by AWS! Awesome, isn't it!

## Monitoring

**Amazon CloudWatch** is a monitoring tool provided by AWS that you can use to monitor any and all aspects of your AWS environment, from EC2 instances to your RDS services to the

load on your ELBs, and so on. You can even create your own metrics, set thresholds, create alarms, and a whole lot of other activities as well.

## Identity and access management

AWS provides a rich set of tools and services to secure and control your infrastructure on the cloud. The most important and commonly used service for this is **identity and access management (IAM)**. Using IAM, you can, as an organizational administrator, create and manage users, assign them specific roles and permissions, and manage active directory federations as well. We will be using a lot of IAM in the next chapter, which covers this topic in greater depth.

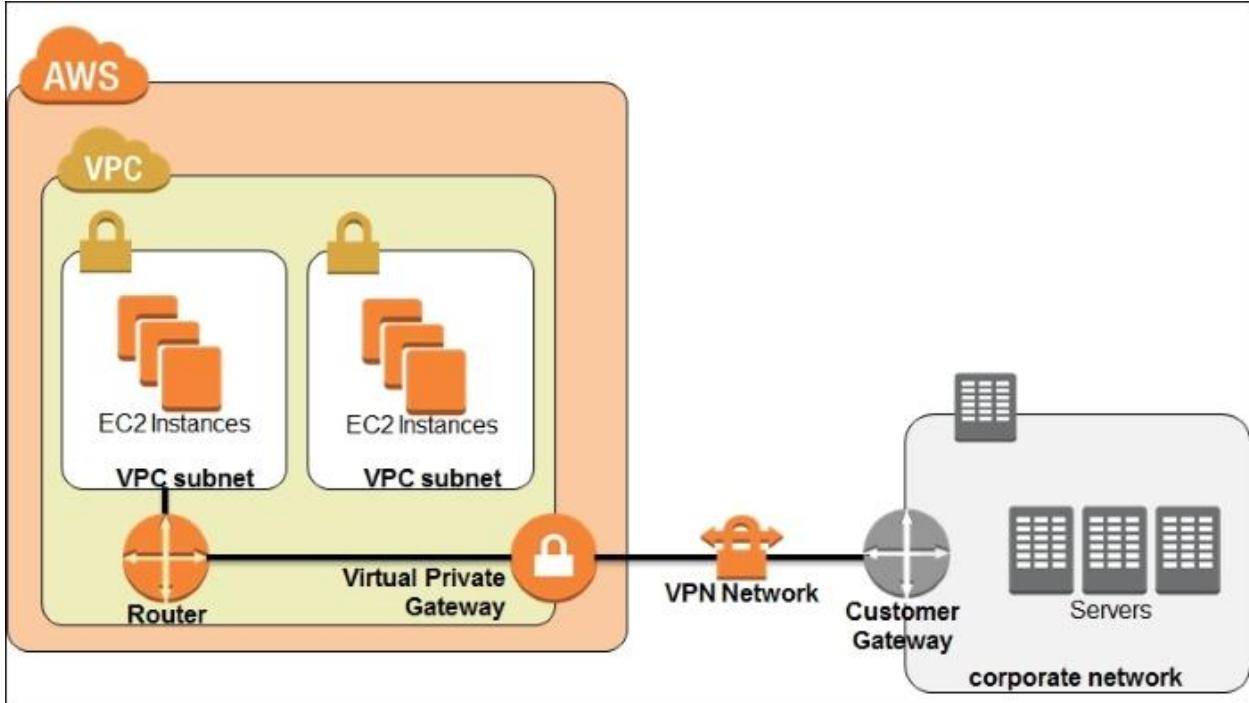
An overview of Amazon VPC

---

So far we have learnt a lot about EC2, its features, and uses, and how we can deploy scalable and fault tolerant applications using it, but EC2 does come with its own sets of minor drawbacks. For starters, you do not control the IP addressing of your instances, apart from adding an Elastic IP address to your instance. By design, each of your instances will get a single private and public IP address, which is routable on the Internet—again, something you cannot control. Also, EC2 security groups have the capability to add rules for inbound traffic only; there is no support for providing any outbound traffic rules. So, although EC2 is good for hosting your applications, it is still not that secure. The answer to all your problems is Amazon VPC!

Amazon VPC is a logically isolated part of the AWS cloud that enables you to build and use your own logical subnets and networks. In a simpler sense, you get to build your own network topology and spin up instances within it. But what actually separates VPC from your classic EC2 environment is the ability to isolate and secure your environment. Using VPCs, you can choose which instances are accessible over the Internet and which are not. You can create a bunch of different subnets within a single VPC, each with their own security policies and routing rules. VPCs also provided an added layer of protection by enforcing something called as **Network Access Control Lists (ACLs)** besides your standard use of security groups. This ensures that you have total control over what traffic is routed in and out of your subnets and the VPC as well.

VPCs also provide an added functionality using which you can connect and extend your on-premise datacenters to the AWS cloud. This is achieved using an IPsec VPN tunnel that connects from your on-premise datacenter's gateway device to the VPC's **Virtual Private Gateway**, as shown in the following image:



An important point to note here is that a VPC is still a part of the AWS Cloud. It is not physically separate hosting provided by AWS, it simply is a logically isolated part of the EC2 infrastructure. This isolation is done at the network layer and is very similar to a traditional datacenter's network isolation; it's just that we as end users are shielded from the complexities of it.

Note: To know more about VPN and virtual private gateways, refer to [http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC\\_VPN.html](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_VPN.html).

### Subnets

Perhaps the most important part of the VPC, the subnets are nothing more than a range of valid IP addresses that you specify. VPC provides you with two different subnet creation options: a publically or Internet routed subnet called as a **public subnet** and an isolated subnet called as a **private subnet**. You can launch your instances in either of these subnets depending on whether you wish your instances to be routed on the Internet or not.

How does it all work? Pretty simple! When you first create a VPC, you provide it with a set of IP addresses in the form of a CIDR, for example, 10.0.0.0/16. The /16 here indicates that this particular VPC can support up to 65,536 IP addresses ( $2^{(32-16)} = 65,536$ , IP address range 10.0.0.0 - 10.0.255.255)! Now that's a lot! Once the VPC's CIDR block is created, you can go ahead and carve out individual subnets from it. For example, a subnet with the CIDR block 10.0.1.0/24 for hosting your web servers and another CIDR block 10.0.5.0/24 for your database servers and so on and so forth.

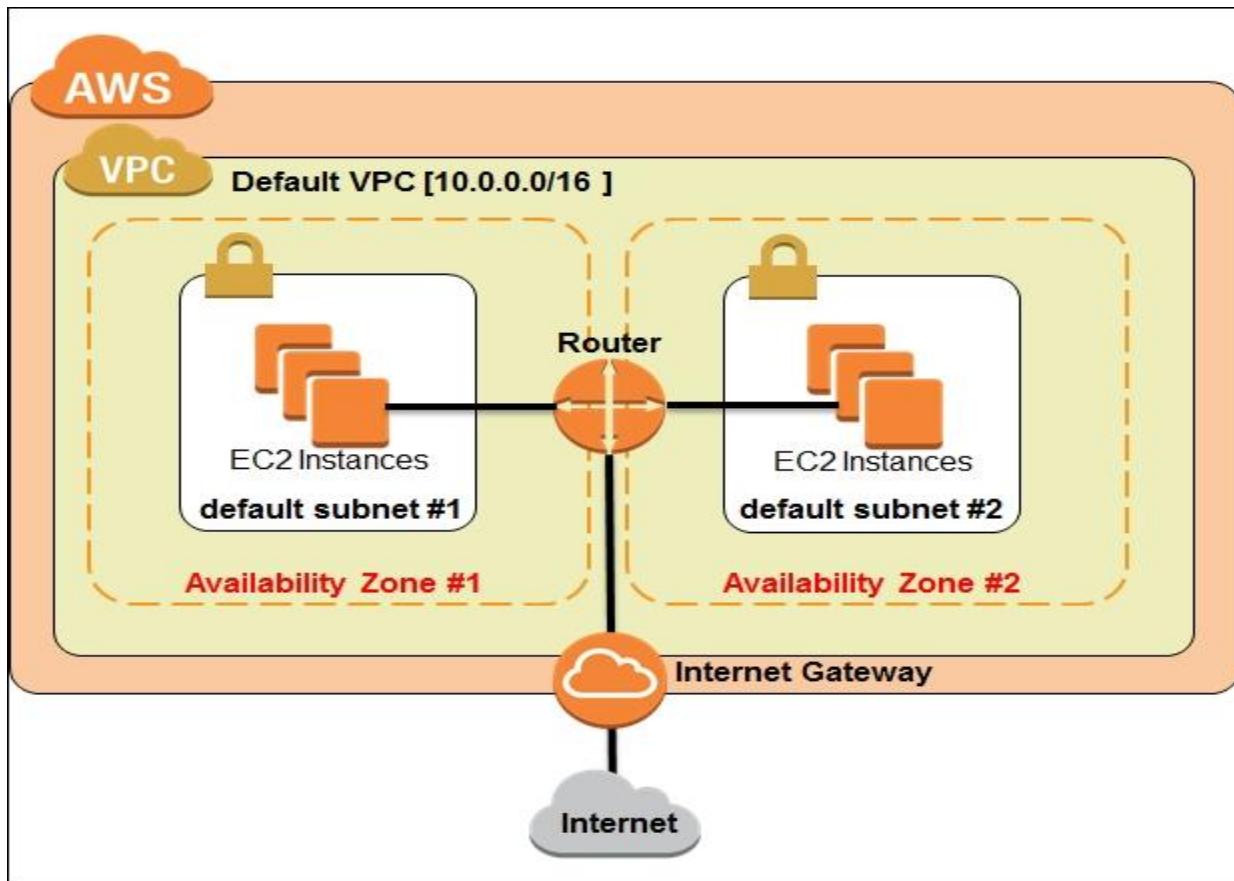
The idea here is that from the 65,536 IP address block, we carved out two subnets, each supporting 256 IPs (/24 CIDR includes 256 IP addresses in it). Now you can specify the subnet

for the web servers to be public, as they will need to be routed to the Internet and the subnet for the database servers to be private as they need to be isolated from the outside world.

To know more about CIDRs and how they work, refer to [https://en.wikipedia.org/wiki/Classless\\_Inter-Domain\\_Routing](https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing).

There is one additional thing worth mentioning here. By default, AWS will create a VPC for you in your particular region the first time you sign up for the service. This is called as the default VPC. The default VPC comes preconfigured with the following set of configurations:

- The default VPC is always created with a CIDR block of /16, which means it supports 65,536 IP addresses in it.
- A default subnet is created in each AZ of your selected region. Instances launched in these default subnets have both a public and a private IP address by default as well.
- An Internet Gateway is provided to the default VPC for instances to have Internet connectivity.
- A few necessary route tables, security groups, and ACLs are also created by default that enable the instance traffic to pass through to the Internet. Refer to the following figure:



You can use this default VPC just as any other VPC by creating additional subnets in it, provisioning route tables, security groups, and so on

## Security groups and network ACLs

We have talked a lot about security groups in the past two chapters already. We do know that security groups are nothing but simple firewall rules that you can configure to safeguard your instances. You can create a maximum of 100 security groups for a single VPC, with each Security Group containing up to 50 firewall rules in them. Also, it is very important to remember that a Security Group does not permit inbound traffic by default. You have to explicitly set inbound traffic rules to allow traffic to flow to your instance. However, all outbound traffic from the instance is allowed by default.

Network ACLs are something new. These provide an added security measure over security groups as they are instance specific, whereas Network ACLs are subnet specific. Unlike your security groups, however, you can both allow and restrict inbound and outbound traffic using ACL rules. Speaking of ACL rules, they are very much similar to your Security Group rules, however, with one small exception. Each ACL rule is evaluated by AWS based on a number. The number can be anything from 100 all the way up to 32,766. The ACL rules are evaluated in sequence starting from the smallest number and going all the way up to the maximum value. The following is a small example of how ACL rules look:

| <b>Inbound ACL rules</b>  |                  |                 |             |                   |
|---------------------------|------------------|-----------------|-------------|-------------------|
| <b>Rule No.</b>           | <b>Source IP</b> | <b>Protocol</b> | <b>Port</b> | <b>Allow/Deny</b> |
| 100                       | 0.0.0.0/0        | All             | All         | ALLOW             |
| *                         | 0.0.0.0/0        | All             | All         | DENY              |
| <b>Outbound ACL rules</b> |                  |                 |             |                   |
| <b>Rule No.</b>           | <b>Dest IP</b>   | <b>Protocol</b> | <b>Port</b> | <b>Allow/Deny</b> |
| 100                       | 0.0.0.0/0        | all             | all         | ALLOW             |
| *                         | 0.0.0.0/0        | all             | all         | DENY              |

These are the ACL rules created by AWS for your default VPC; as a result, this particular ACL is called as the default Network ACL as well. What do these rules mean? For starters, the rule number 100 for both the inbound and outbound ACL specifies the traffic to flow from any protocol running on any port in and out of the subnet. The \* is also considered as a rule number and is a must in all ACLs. It basically means that you drop any packets that do not match the ACL's rules. We will be checking out ACLs and security groups in action later on in this chapter when we create our very own VPC for the first time.

## Routing tables

Route tables are pretty straightforward and easy to implement in a VPC. They are nothing but simple rules or routes that are used to direct network traffic from a subnet. Each subnet in a VPC has to be associated with a single route table at any given time; however, you can attach multiple subnets to a single route table as well.

Remember the default VPC and the default subnets? Well, a similar default route table is also created when you first start using your VPC. This default route table is called as the **main route table** and it generally contains only one route information that enables traffic to flow within the VPC itself. Subnets that are not assigned to any route tables are automatically allocated to the main route table. You can edit and add multiple routes in the main route table as you see fit; however, you cannot modify the local route rule. The following is an example of a main route table viewed from the VPC Management dashboard:

| Summary                                  | Routes        | Subnet Associations | Route Propagation |
|--|---------------|---------------------|-------------------|
| Edit                                     |               |                     |                   |
| Destination                              | Target        | Status              | Propagated        |
| 10.0.0.0/16                              | local         | Active              | No                |
| pl-68a54001 (com.amazonaws.us-west-2.s3) | vpce-80cd2be9 | Active              | No                |

As you can see, there are a couple of entries made in this table. The first is the local route rule that allows traffic to flow within this particular subnet (**10.0.0.0/16**). The second route is something called as a route for VPC endpoints. This is a private connection made between your VPC and some other AWS service; in this case, the service is S3. Let's look at VPC endpoints a little closely.

## VPC endpoints

VPC endpoints basically allow you to securely connect your VPC with other AWS services. These are virtual devices that are highly available and fault tolerant by design. They are scaled and managed by AWS itself, so you don't have to worry about the intricacies of maintaining them. All you need to do is create a VPC endpoint connection between your VPC and an AWS service of your choice, and voila! Your instances can now communicate securely with other AWS services. The instances in the VPC communicate with other services using their private IP addresses itself, so there's no need to route the traffic over the Internet.

Note:

AWS currently only supports VPC endpoint connections for Amazon S3. More services are planned to be added shortly.

When an endpoint is created, you first need to select either of your VPC's route tables. The traffic between your VPC instances and the AWS service will be routed using this particular route table. Similar to any other route information, a VPC endpoint route also contains a **Destination** field and a **Target** field. The **Destination** field contains the AWS service's prefix list ID, which is generally represented by the following ID: pl-xxxxxxx. The **Target** field contains the endpoint ID, which is represented in the following format: vpce-xxxxxxx.

In the following route table example, the prefix list ID (**pl-68a54001**) represents the S3 service whereas the target **vpce-80cd2be9** represents the endpoint ID:

| VPC Endpoint Route |               |
|--------------------|---------------|
| Destination        | Target        |
| 10.0.0.0/16        | Local         |
| pl-68a54001        | vpce-80cd2be9 |

Endpoints also provided an additional feature using which you can control and govern access to the remote AWS service. This is achieved using something called as **endpoint policies**.

Endpoint policies are nothing more than simple IAM-based resource policies that are provided to you when an endpoint is first created. AWS creates a simple policy document that allows full access to the AWS service by default. The following is a sample endpoint policy that is created by AWS for the S3 service:

```
{  
  "Statement": [  
    {  
      "Action": "*",  
      "Effect": "Allow",  
      "Resource": "*",  
      "Principal": "*"  
    }  
  ]  
}
```

To know more about the endpoint policies and features, refer to <http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-endpoints.html>.

## Internet Gateways

Internet Gateways, as the name suggest, are primarily used to provide Internet connectivity to your VPC instances. All you have to do is create and attach an Internet Gateway device to your VPC and add a route entry in your public subnet's route table to point to the Internet Gateway! That's it! The default VPC comes with an Internet gateway already deployed in it. So, any instance that you launch from the default subnet obtains Internet connectivity automatically. This does not apply for non-default VPCs, however, as an instance launched in a non-default subnet does not receive a public IP address by default. You would have to either assign one to the instance during the launch phase or modify your non-default subnet's public IP address attributes.

Once you have created and attached an Internet Gateway to your VPC, you will also have to make sure that the public subnet's route table has an entry for this gateway.

Plus, you will also have to create the correct set of security groups and network ACL rules to allow your subnet's traffic to flow through the Internet. The following is an example of a VPC's route table showing the route for a subnet's traffic to the Internet Gateway (**igw-8c3066e9**):

| Destination                              | Target        | Status | Propagated |
|--|---------------|--------|------------|
| 10.0.0.0/16                              | local         | Active | No         |
| pl-68a54001 (com.amazonaws.us-west-2.s3) | vpce-80cd2be9 | Active | No         |
| 0.0.0.0/0                                | igw-8ce066e9  | Active | No         |

Besides the Internet connectivity, Internet Gateways also perform NAT on the instance's private IPs. The instances in a subnet are only aware of their private IP addresses that they use to communicate internally. The Internet Gateway maps the instance's private IP with an associated public or Elastic IP and then routes traffic outside the subnet to the Internet. Conversely, the Internet Gateway also maps inbound traffic from the Internet to a public or Elastic IP and then translates it to the instance's private IP address. This is how your instances receive Internet from within a VPC, which brings us to yet another interesting topic called as NAT instances.

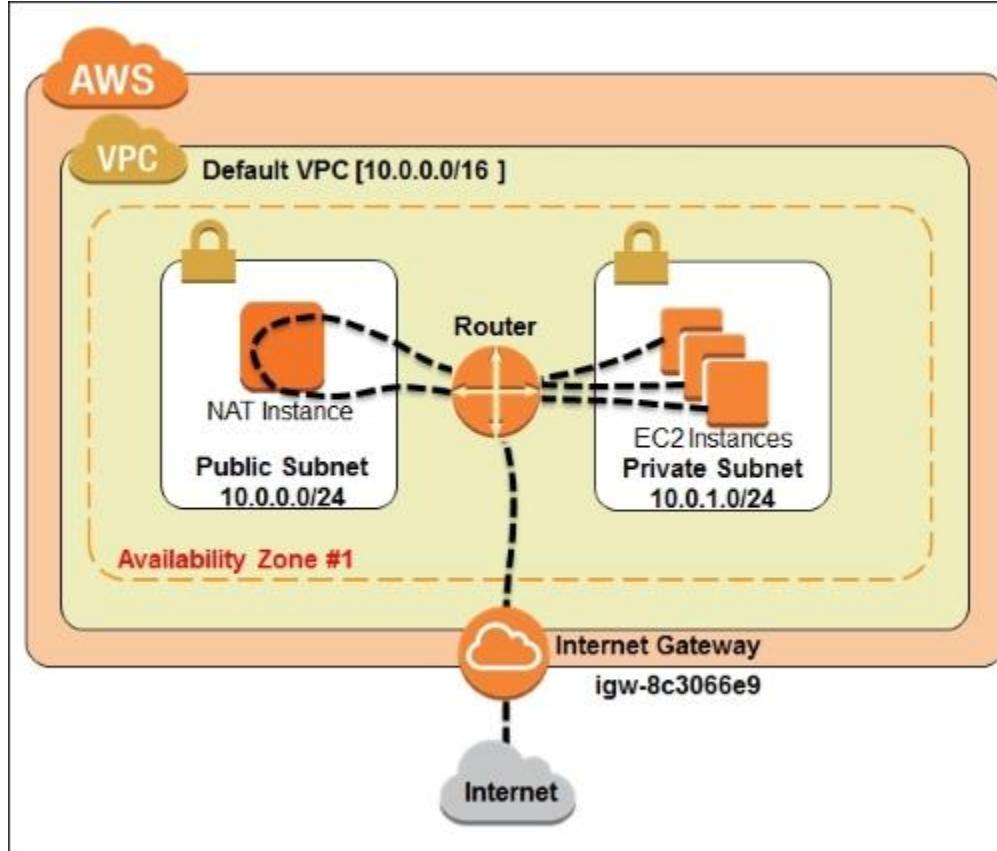
## NAT instances

So, we have just learnt that the Internet Gateway NATs the IP addresses of instances placed out in the public subnet so that they can communicate with the Internet, but what about instances in the private subnets? How do they communicate with the Internet without having direct Internet connectivity via the gateway?

That's where a NAT instance comes into play. A NAT Instance is a special instance created inside your public subnet that NATs outbound traffic from instances based in your private subnet to the Internet. It is important to note here that the NAT instance will only forward the outbound traffic and not allow any traffic from the Internet to reach the private subnets, similar to a one way street.

You can create a NAT Instance out of any AMI you wish; however, AWS provides few standard Linux-based AMIs that are well suited for such purposes. These special AMIs are listed out in the community AMIs page and all you need to do is filter out the amzn-ami-vpc-nat AMI from the list and spin up an instance from it.

The following example depicts the traffic flow from a private subnet (**10.0.1.0/24**) to the NAT instance inside a public subnet (**10.0.0.0/24**) via a route table:



In the preceding example, outbound traffic from the public subnet's route table is routed to the Internet Gateway (**igw-8c3066e9**) while the outbound traffic from the private subnet's route table is routed to the NAT instance. Along with the route tables, it is also essential that you correctly populate the Security Group for your NAT instance. The following is a simple NAT instance Security Group example for your reference:

| NAT instance - inbound security Rules |          |      |  |
|---------------------------------------|----------|------|--|
| Source                                | Protocol | Port | Remarks  |
| 10.0.1.0/24                           | TCP      | 80   | Permit inbound HTTP traffic from private subnet  |
| 10.0.1.0/24                           | TCP      | 443  | Permit inbound HTTPS traffic from private subnet |
| <HOSTIP>*                             | TCP      | 22   | Permit SSH login to NAT instance from remote N/W |

Note: The \* replace the <HOSTIP> field with the IP address of your local desktop machine.

The following are the outbound security rules:

| NAT Instance - outbound security rules |          |      |  |
|--|----------|------|--|
| Source                                 | Protocol | Port | Remarks  |
| 0.0.0.0/0                              | TCP      | 80   | Permit HTTP access to Internet for the NAT instance  |
| 0.0.0.0/0                              | TCP      | 443  | Permit HTTPS access to Internet for the NAT instance |

## DNS and DHCP Option Sets

VPCs provide an additional feature called as DHCP Option Sets using which you can set and customize the DNS and DHCP for your instances. The default VPC comes with a default DHCP Options Set that is used to provide the instances with a dynamic private IP address and a resolvable hostname. Using the DHCP Options Set, you can configure the following attributes for your VPC:

**Domain Name Servers (DNS):** You can list down up to four DNS servers here of your own choice or even provide the Amazon DNS server details. The Amazon DNS server is provided in your VPC and runs on a reserved IP address. For example, if your VPC has the subnet of 10.0.0.0/16, then the Amazon DNS Server will probably run on the IP 10.0.0.2. You can additionally provide the Amazon DNS Server's IP address, 169.254.169.253, or the value **AmazonProvidedDNS** as required. Values entered here are automatically added to your Linux instances /etc/resolv.conf file for name resolution.

**Domain name:** You can either provide your own domain name value or choose to use the default AWS domain name values using this option. The default AWS domain names can be provided only if you have selected **AmazonProvidedDNS** as your DNS server. For example, instances launched in the US West region with the Amazon DNS server value will get a resolvable private DNS hostname as us-west-2.compute.internal.

- **NTP servers:** You can list up to four NTP server IP addresses using the DHCP Options Set wizard. Note, however, that this will only accept IP address values and not FQDNs such as pool.ntp.org.
- **NetBIOS name server:** You can list down up to four NetBIOS name servers as well; however, this field is optional.
- **NetBIOS node type:** You can specify the NetBIOS node value, which can either be 1, 2, 4, or 8. AWS recommends that you specify 2 as broadcast, and multicasts are not currently supported.

You can create and attach only one DHCP option set with a VPC at a time. AWS uses the default DHCP option if you do not specify one explicitly for your VPC. Instances either running or newly launched will automatically pick up these DNS and DHCP settings, so there is no need for you to restart or relaunch your existing instances.

## VPC limits and costs

Okay, so far we have understood a lot about how the VPC works and what its components are, but what is the cost of all this? Very simple, it's nothing! VPC is a completely free of cost service provided by AWS; however, you do have to pay for the EC2 resources that you use, for example, the instances, the Elastic IP addresses, EBS volumes, and so on. Also, if you are using your VPC to connect to your on premise datacenter using the VPN option, then you need

to pay for the data transfers over the VPN connection as well as for the VPN connection itself. AWS charges \$0.05 per VPN connection hour.

Besides this, VPCs also have a few limits set on them by default. For example, you can have a maximum of five VPCs per region. Each VPC can have a max of one Internet gateway as well as one virtual private gateway. Also, each VPC has a limit of hosting a maximum of up to 200 subnets per VPC. You can increase these limit by simply requesting AWS to do so. To view the complete list of VPC limits, refer to [http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC\\_Appendix\\_Limits.html](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Appendix_Limits.html).

## Working with VPCs

---

### VPC deployment scenarios

VPC provides a simple, easy-to-use wizard that can spin up a fully functional VPC within a couple of minutes. All you need to do is select a particular deployment scenario out of the four scenarios provided and configure a few basic parameters such as subnet information, availability zones in which you want to launch your subnets, and so on, and the rest is all taken care of by AWS itself.

Let's have a quick look at the four VPC deployment scenarios:

- **VPC with a single public subnet:** This is by far the simplest of the four deployment scenarios. Using this scenario, VPC will provision a single public subnet with a default Internet Gateway attached to it. The subnet will also have a few simple and basic route tables, security groups, and network ACLs created. This type of deployment is ideal for small-scaled web applications or simple websites that don't require any separate application or subnet tiers.
- **VPC with public and private subnets (NAT):** Perhaps the most commonly used deployment scenario, this option will provide you with a public subnet and a private subnet as well. The public subnet will be connected to an Internet gateway and allow instances launched within it to have Internet connectivity, whereas the private subnet will not have any access to the outside world. This scenario will also provision a single NAT instance inside the public subnet using which your private subnet instances can connect with the outside world but not vice versa. Besides this, the wizard will also create and assign a route table to both the public and private subnets, each with the necessary routing information prefilled in them. This type of deployment is ideal for large-scale web applications and websites that leverage a mix of public facing (web servers) and non-public facing (database servers).
- **VPC with public and private subnets and hardware VPN access:** This deployment scenario is very much similar to the VPC with public and private subnets, however, with one component added additionally, which is the Virtual Private Gateway. This Virtual Private Gateway connects to your on premise network's gateway using a standard VPN connection. This type of deployment is well suited for organizations that

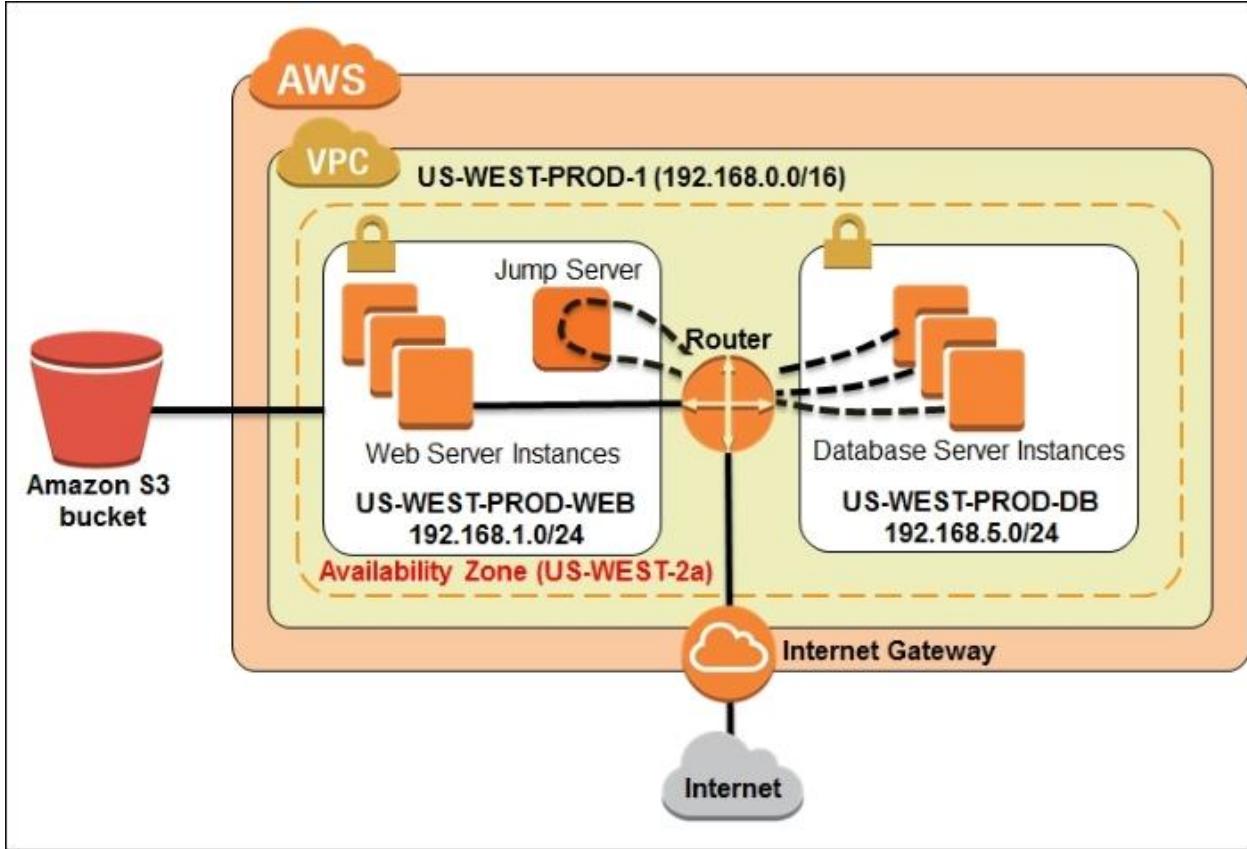
wish to extend their on premise datacenters and networks in to the public clouds while allowing their instances to communicate with the Internet.

- **VPC with a private subnet only and hardware VPN access:** Unlike the previous deployment scenario, this scenario only provides you with a private subnet that can connect to your on premise datacenters using standard VPN connections. There is no Internet Gateway provided and thus your instances remain isolated from the Internet. This deployment scenario is ideal for cases where you wish to extend your on premise datacenters into the public cloud but do not wish your instances to have any communication with the outside world.

## Getting started with the VPC wizard

Before we go ahead and deploy our VPC, let's first have a quick look at our use case. We need to create a secure website hosting environment for our friends at All-About-Dogs.com, complete with the following requirements:

- Create a VPC (US-WEST-PROD-1 - 192.168.0.0/16) with separate secure environments for hosting the web servers and database servers.
- Only the web servers environment (US-WEST-PROD-WEB - 192.168.1.0/24) should have direct Internet access.
- The database servers environment (US-WEST-PROD-DB - 192.168.5.0/24) should be isolated from any direct access from the outside world.
- The database servers can have restricted Internet access only through a jump server (**NAT Instance**). The jump server needs to be a part of the web server environment.
- The web servers environment should full have access to Amazon S3.



To get started with VPC, we first have to log in to the AWS Account using your IAM credentials. Next, from the **AWS Management Console**, select the **VPC** option from under the **Networking** group

This will bring up the **VPC Dashboard** using which you can create, manage, and delete VPCs as per your requirements. The VPC dashboard lists the currently deployed VPCs, Subnets, Network ACLs, and much more under the **Resources** section.

You can additionally view and monitor the health of your VPC service by viewing the status provided by the **Service Health** dashboard, as shown in the following screenshot. In my case, I'm operating my VPC out of the US West (Oregon) region.

| Resources  |                                      | Service Health  |                         |                |         |  |                               |  |                               |
|--|--------------------------------------|---|-------------------------|----------------|---------|--|-------------------------------|--|-------------------------------|
| <a href="#">Start VPC Wizard</a>   | <a href="#">Launch EC2 Instances</a> | <a href="#">Current Status</a>  | <a href="#">Details</a> |                |         |  |                               |  |                               |
| Note: Your Instances will launch in the US West (Oregon) region.                 |                                      | <table border="1"> <thead> <tr> <th>Current Status</th> <th>Details</th> </tr> </thead> <tbody> <tr> <td><span style="color: green;">✓</span> Amazon VPC - US West (Oregon)</td> <td>Service is operating normally</td> </tr> <tr> <td><span style="color: green;">✓</span> Amazon EC2 - US West (Oregon)</td> <td>Service is operating normally</td> </tr> </tbody> </table> |                         | Current Status | Details | <span style="color: green;">✓</span> Amazon VPC - US West (Oregon) | Service is operating normally | <span style="color: green;">✓</span> Amazon EC2 - US West (Oregon) | Service is operating normally |
| Current Status   | Details                              |   |                         |                |         |  |                               |  |                               |
| <span style="color: green;">✓</span> Amazon VPC - US West (Oregon)               | Service is operating normally        |   |                         |                |         |  |                               |  |                               |
| <span style="color: green;">✓</span> Amazon EC2 - US West (Oregon)               | Service is operating normally        |   |                         |                |         |  |                               |  |                               |
| You are using the following Amazon VPC resources in the US West (Oregon) region: |                                      | <a href="#">View complete service health details</a>  |                         |                |         |  |                               |  |                               |
| 1 VPC  | 1 Internet Gateway                   |   |                         |                |         |  |                               |  |                               |
| 3 Subnets  | 1 Route Table                        |   |                         |                |         |  |                               |  |                               |
| 1 Network ACL  | 0 Elastic IPs                        |   |                         |                |         |  |                               |  |                               |
| 2 Security Groups  | 0 Running Instances                  |   |                         |                |         |  |                               |  |                               |
| 0 VPC Peering Connections  | 0 Virtual Private Gateways           |   |                         |                |         |  |                               |  |                               |
| 0 VPN Connections  | 0 Customer Gateways                  |   |                         |                |         |  |                               |  |                               |
| 0 Endpoints  |                                      |   |                         |                |         |  |                               |  |                               |

The **VPC Dashboard** also lists any existing VPN connections that you might have set up earlier. You can view the **VPNConnections**, **Customer Gateways** information as well as the **Current Status** of the VPN connection by using this dashboard. Remember that a VPN connection has a cost associated with it when it is provisioned and in the available state. You can additionally use this dashboard and even launch your instances directly into a VPC using the **Launch EC2 Instances** option. These instances will most probably be launched in your default VPC in case you haven't already created another one.

With all this said and done, let's go ahead and create our VPC using the VPC Wizard. Select the **Start VPC Wizard** option. The wizard is a simple two-step process that will guide you with the required configuration settings for your VPC. You will be prompted to select any one out of the four VPC scenarios, so with our use case in mind, go ahead and select the **VPC with Public and Private Subnets** option. Do note that this will create a /16 network with two /24 subnets by default, one public subnet and the other a private subnet. You can always create more subnets as required once the VPC is created.

Also worth mentioning here is that this VPC scenario will create and launch a NAT instance as well in the public subnet. This instance will be powered on automatically once your VPC is created, so be aware about its existence and power it down unless you want to get charged for it.

The second step of the wizard is where you get to configure your VPC network and subnets. Fill in the following details as required:

- **IP CIDR block:** Provide the IP CIDR block address for your VPC's network. Ideally, provide a /16 subnet that will provide you with a good 65,531 IP addresses to use.
- **VPC name:** Provide a suitable name for your VPC. In this case, I have standardized and used the following naming convention: <REGION>-<DEV/PROD Environment>-<UNIQUE\_ID>; so in our case, this translates to US-WEST-PROD-1.
- **Public subnet:** Now, since we are going with a public and private subnet combination scenario, we have to fill in our public subnet details here. Provide a suitable subnet block for your instances to use. In this case, I have provided a /24 subnet which provides a good 251 usable IP addresses:

## Step 2: VPC with Public and Private Subnets

IP CIDR block\*: 192.168.0.0/16 (65531 IP addresses available)

VPC name: US-WEST-PROD-1

Public subnet\*: 192.168.1.0/24 (251 IP addresses available)

Availability Zone\*: us-west-2a

Public subnet name: US-WEST-PROD-WEB

Private subnet\*: 192.168.5.0/24 (251 IP addresses available)

Availability Zone\*: us-west-2a

Private subnet name: US-WEST-PROD-DB

You can add more subnets after AWS creates the VPC.

- **Availability Zone:** Here's the fun part! You can deploy your subnets in any availability zone available in that particular region. Now, US-WEST (Oregon) has three AZs and you can use any of those there. In my case, I have gone ahead and selected **us-west-2a** as the default option.
- **Public subnet name:** Provide a suitable public subnet reference name. Here, too, I have gone ahead and used the standard naming convention, so this particular subnet gets called as **US-WEST-PROD-WEB**, signifying the web server instances that will get deployed here.
- **Private subnet, Availability zone, Private subnet name:** Go ahead and fill out the private subnet's details using the similar IP addressing and naming conventions. Remember that although you can set up your private subnet in a different AZ, as compared to the public subnet, ideally doing that is not recommended. If you really want to set up a failover-like scenario, then create a separate public and private subnet environment in a different AZ altogether, for example, us-west-2c. So, even in case us-west-2a suffers an outage, which by the way can happen, your failover subnets will still be functioning out of the **us-west-2c** AZ. Refer to the following screenshot:

Specify the details of your NAT instance.

Instance type:<sup>\*</sup> m1.small

Key pair name: my-sample-keypair

Note: Instance rates apply. [View Rates](#).

Add endpoints for S3 to your subnets

Subnet: Public subnet

Policy:  Full Access - Allow access by any user or service within the VPC using credentials from any AWS accounts to any S3 resources  
 Custom

Enable DNS hostnames:<sup>\*</sup>  Yes  No

Hardware tenancy:<sup>\*</sup> Default

[Cancel and Exit](#) [Back](#) [Create VPC](#)

Next up, we specify the details of our NAT instance:

- **Instance type:** Select your NAT instance type from the available dropdown menu. In my case, I have gone ahead and selected the **t2.micro** instance type as that is the smallest type available. Do remember that selecting any other option will incur additional costs as only the t2.micro instance type is covered under the free tier eligibility.
- **Key pair name:** Select an already existing key pair from the dropdown list. Make sure you have this particular key pair stored safely on your local computer, as without it you will not be able to SSH into the NAT instance. You can alternatively create a new key pair here as well using the same EC2 Management Console.

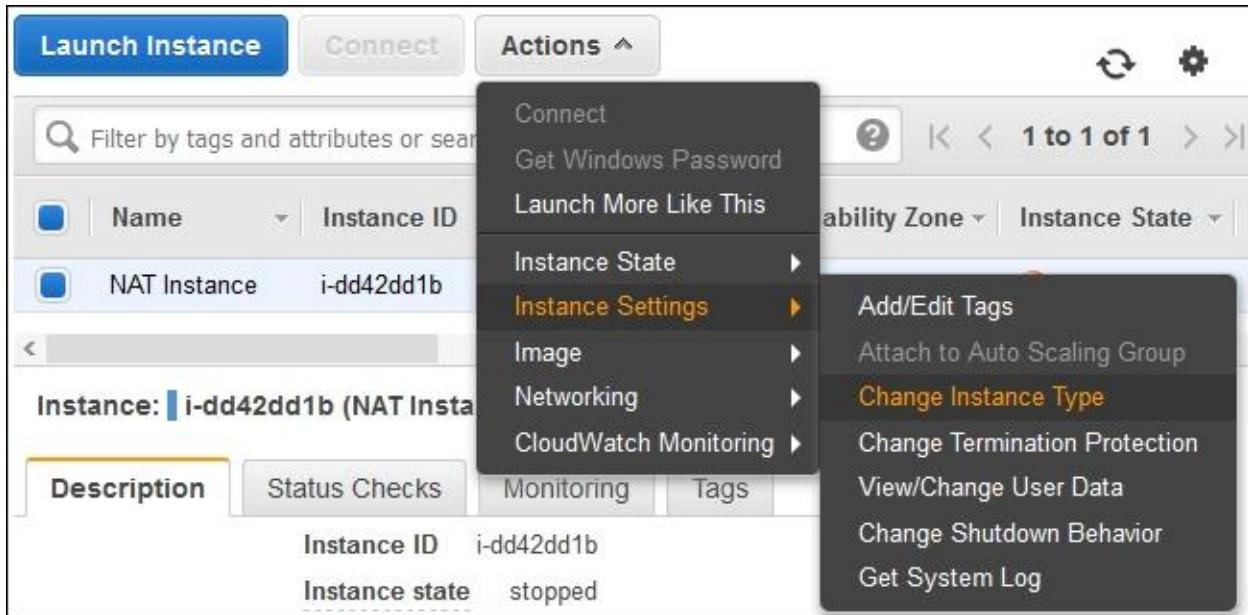
Moving on, we now add the S3 endpoints to our particular subnet. You can add the endpoint to either your private or public subnets, or both of them, depending on your requirements.

- **Subnet:** As per our VPC's requirements, the S3 endpoint is only made available to the public subnet, so go ahead and select the **Public subnet** option from the dropdown list.
- **Policy:** You can either choose to allow any user or service within the newly created VPC to access your S3 or specify a custom IAM access policy as you see fit. In our case, let's go ahead and select **Full Access** for now.
- **Enable DNS hostnames:** Enabling this option will provide your instances with the ability to resolve their DNS hostnames on the Internet. Select the **Yes** option and continue.
- **Hardware tenancy:** Although a VPC runs off a completely isolated network environment, the underlying server hardware is still shared by default. You can change this tenancy option by selecting either the **default** or **dedicated** option from the dropdown list provided.

Once all the required information is filled out, go ahead and click on the **Create VPC** option. The VPC creation takes a few seconds to complete. You can even view your new VPC's default routes, security groups, and Network ACLs being created. Toward the end you will notice your NAT instance powering on, and after a few seconds of deployment your VPC is now ready for use!

Here's a handy tip for all first timers! As soon as your NAT instance is created, you can go ahead and change its default instance type to t1.micro from the **EC2 Management Dashboard**.

To do so, first open up the **EC2 Management Dashboard** in a new tab on your browser. You should see an instance in the running state, as shown in the following screenshot. First up, stop your instance using the **Actions** tab. Select the **Instance State** option and click on **Stop**. Wait for the instance state to change to **Stop** before you proceed any further. Next, select the instance, and from the **Actions** tab, select **Instance Settings** and then **Change Instance Type**.



From the **Change Instance Type** dialog box, select the **t1.micro** option and click on **Apply**. Voila! Your NAT instance is now officially a part of your free tier as well! Simple, isn't it!

Let's have a quick look at what actually happens behind the scenes when you create a VPC using the VPC Wizard. First up, let's look at the VPC itself.

### Viewing VPCs

Once the VPC is created and ready, you can view it from the **VPC Management Dashboard** as well. Simply select the **Your VPCs** option from the **Navigation pane** provided on the left-hand side of the dashboard.

You should see your newly created VPC, as shown in the following screenshot. You can use the search bar to filter out results as well. Select the particular VPC to view its details.

The screenshot shows the AWS VPC Management Dashboard. At the top, there is a search bar with the text "QUS". Below it, a table lists one VPC entry:

| Name           | VPC ID       | State     | VPC CIDR       | DHCP options set |
|----------------|--------------|-----------|----------------|------------------|
| US-WEST-PROD-1 | vpc-d55a22b0 | available | 192.168.0.0/16 | dopt-703fcc15    |

Below the table, the VPC details are displayed:

vpc-d55a22b0 (192.168.0.0/16) | US-WEST-PROD-1

**Summary** **Flow Logs** **Tags**

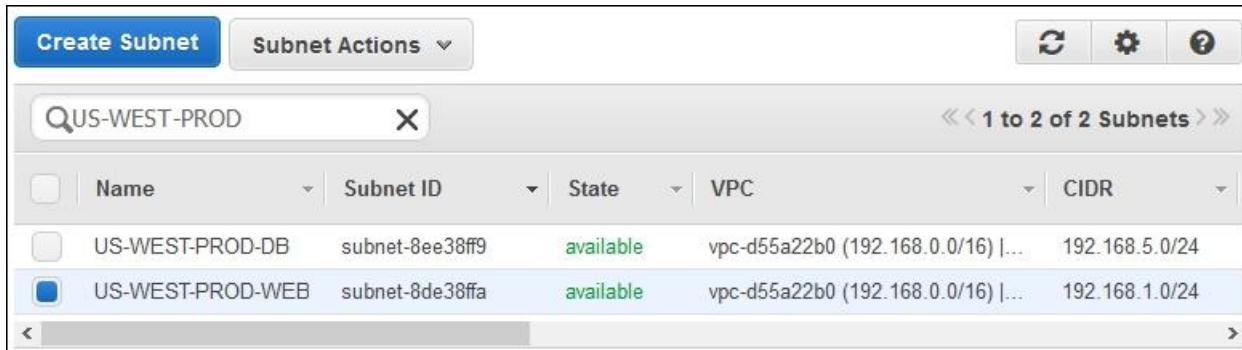
|                                 |                           |
|---------------------------------|---------------------------|
| vpc-d55a22b0                    | Network ACL: acl-c0354aa5 |
| VPC ID: US-WEST-                |                           |
| PROD-1                          |                           |
| State: available                | Tenancy: Default          |
| VPC CIDR: 192.168.0.0/16        | DNS resolution: yes       |
| DHCP options set: dopt-703fcc15 | DNS hostnames: yes        |
| Route table: rtb-853472e0       |                           |

Use the **Summary** tab to view the description of your selected VPC. Here, you can view the VPC's default **DHCP options** set as well as the **Tenancy** and **DNS hostnames** and **DNS resolution** options. You can optionally change these values by selecting the particular VPC and from the **Actions** tab, selecting either **Edit DHCP Options Set** or **Edit DNS Hostnames**.

You can create additional VPCs as well using the **Create VPC** option; however, as a good practice, always keep things simple and minimal. Don't go over creating VPCs. Rather use and create as many subnets as you require. Speaking of subnets, let's have a look at newly created VPC's two subnets!

## Listing out subnets

You can view, add, and modify existing subnets using the **VPC Management Dashboard** as well. Simply select the **Subnets** option from the **Navigation Pane**. This will list out all the subnets present in your account, so use the search bar to filter out the new ones that we just created. Type in the name of the subnet in the Search bar until the particular subnet gets listed out, as shown in the following screenshot:



The screenshot shows the AWS Subnet Actions dropdown menu open over a list of subnets. The menu includes options such as 'Delete Subnet', 'Create Flow Log', and 'Modify Auto-Assign Public IP'. The 'Modify Auto-Assign Public IP' option is highlighted in orange.

| Name             | Subnet ID       | State     | VPC                                | CIDR           |
|------------------|-----------------|-----------|------------------------------------|----------------|
| US-WEST-PROD-DB  | subnet-8ee38ff9 | available | vpc-d55a22b0 (192.168.0.0/16)  ... | 192.168.5.0/24 |
| US-WEST-PROD-WEB | subnet-8de38ffa | available | vpc-d55a22b0 (192.168.0.0/16)  ... | 192.168.1.0/24 |

You can view additional information associated with your subnet by simply selecting it and viewing the **Summary** tab. The **Summary** tab will list out the particular subnet's associated **Route table**, **Network ACL**, **CIDR**, and **State** as well. Besides these values, you can also configure your subnet's ability to auto assign public IPs to its instances. By default, this feature is disabled in your subnet, but you can always enable it as per your requirements. To do so, simply select your **Public Subnet**, and from the **Subnet Actions** tab, select the option, as shown in the following screenshot:



In the **Modify Auto-Assign Public IP** dialog box, simply select the **Enable auto-assign public IP** option and click on **Save** to complete the change setting. Do note that you can always override this behavior for each individual instance at its launch time.

Besides the **Summary** tab, the **Subnet Dashboard** option also provides additional tabs such as **Route Table**, **Network ACL**, and so on. You can use these tabs to view the subnet's associated route table as well the network ACL; however, you cannot add or modify the individual rules from here. To add or modify rules, you need to go to the **Network ACL** option or the **Route Tables** option from the **navigation pane**. Let's have a quick look at the route tables created for our VPC.

### Working with route tables

As discussed earlier in this chapter, VPCs come with a default route table (**Main Route Table**) associated with a subnet. So, since we have two subnets created in this VPC, we get two route tables as well, out of which one is the main route table. How do you tell whether a route table is the main one? Quite simple, actually! Just look for the **Main Column** in the **Route Table Dashboard**, as shown in the following screenshot. If the value in that column is **Yes**, then that

particular route table is your VPC's main route table. Now, here's a catch! If you do not explicitly associate a route table with a subnet, then the subnet ends up using the main route table. In our case, both the route tables created do not have any subnets associated with them by default, so let's first get that done.

| Name                | Route Table ID | Explicitly Associated Subnet | Main | VPC                                |
|---------------------|----------------|------------------------------|------|------------------------------------|
| US-WEST-PROD-WEB-RT | rtb-843472e1   | 1 Subnet                     | No   | vpc-d55a22b0 (192.168.0.0/16) [..] |
| US-WEST-PROD-DB-RT  | rtb-853472e0   | 1 Subnet                     | Yes  | vpc-d55a22b0 (192.168.0.0/16) [..] |

To associate a route table with a subnet explicitly, first you need to select the particular route table from the dashboard. In this case, I have selected the public subnet's router (**US-WEST-PROD-WEB-RT**). Next, from the **Subnet Associations** tab, click on the **Edit** button, as shown in the following screenshot. From here, you can select either of the two subnets that are listed down; however, since this is a public subnet's route table, let's go ahead and select the listed public subnet (**US-WEST-PROD-WEB**) as shown. Click on **Save** to save the configuration changes.

rtb-843472e1

Summary    Routes    **Subnet Associations**    Route Propagation

**Edit**

| Subnet  | CIDR           |
|---|----------------|
| subnet-8de38ffa (192.168.1.0/24)   US-WEST-PROD-WEB | 192.168.1.0/24 |

With this step completed, your subnet is now explicitly attached with a particular route table. But what about the individual route rules? How do we list and modify them? That's simple as well. Simply select the **Routes** tab to list your route table's existing rule set. Here, you should see at least three route rules, as shown in the following screenshot. The first rule is created by VPC for each and every route table, and it basically allows communication within the VPC itself. You cannot delete this rule, so don't even try it!

The next rule is basically a VPC endpoint route rule. Remember the S3 endpoint that we configured earlier with the public subnet? Well this rule will basically allow communication to occur between the instances belonging to this subnet and Amazon S3, and the best part is that this rule is auto-populated when you create a VPC endpoint!

| rtb-843472e1                             |               |                     |                   |
|--|---------------|---------------------|-------------------|
| Summary                                  | Routes        | Subnet Associations | Route Propagation |
| <b>Edit</b>                              |               |                     |                   |
| Destination                              | Target        | Status              | Propagated        |
| 192.168.0.0/16                           | local         | Active              | No                |
| pl-68a54001 (com.amazonaws.us-west-2.s3) | vpce-7b02e412 | Active              | No                |
| 0.0.0.0/0                                | igw-909c1ff5  | Active              | No                |

The final rule in the list basically allows for the instances to communicate over the Internet using the Internet Gateway as the target. You can optionally choose to edit these rules by selecting the **Edit** option. Once you have made your required changes, be sure to **Save** the configuration changes before you proceed with the next steps. Don't forget to associate the private subnet (**US-PROD-WEST-DB**) with the remaining route table (**US-WEST-PROD-DB-RT**) as well.

### Listing Internet Gateways

As discussed earlier in this chapter, Internet Gateways are scalable and redundant virtual devices that provide Internet connectivity for your instances present in the VPC. You can list currently available Internet Gateways within your VPC by selecting the **Internet Gateways** option from the VPC's **navigation pane**.

The VPC wizard will create and attach one Internet Gateway to your VPC automatically; however, you can create and attach an Internet Gateway to a VPC at any time using the Internet Gateway. Simply click on the **Create Internet Gateway** option and provide the **VPC** to which this Internet Gateway has to be attached, that's it! You can list down available Internet Gateways and filter the results using the search bar provided as well. To view your Internet Gateway's details, simply select the particular Internet Gateway and click on the **Summary** tab. You should see your **Internet Gateway's ID**, **State (attached or detached)**, as well as the **Attachment state (available or not available)**, as shown in the following screenshot:

The screenshot shows the AWS VPC console interface for managing Internet Gateways. At the top, there are buttons for 'Create Internet Gateway', 'Delete', 'Attach to VPC', and 'Detach from VPC'. Below this is a search bar and a navigation bar indicating '1 to 1 of 1 Internet Gateway'. The main list area shows a single Internet Gateway entry:

|                                     |                 |              |          |   |
|-------------------------------------|-----------------|--------------|----------|---|
| <input checked="" type="checkbox"/> | Name            | ID           | State    | VPC                                     |
| <input checked="" type="checkbox"/> | US-WEST-PROD-GW | igw-909c1ff5 | attached | vpc-d55a22b0 (192.168.0.0/16)   US-W... |

Below the list is a detailed view for the selected gateway ('igw-909c1ff5 | US-WEST-PROD-GW'). It includes tabs for 'Summary' (selected) and 'Tags'. The summary details are:

|   |  |
|---|--|
| igw-909c1ff5  <br>ID: US-WEST-<br>PROD-GW | Attached VPC ID:<br>(192.168.0.0/16)  <br>US-WEST-<br>PROD-1 |
| State: attached                           | Attachment state: available                                  |

Also remember that to really use the Internet Gateway, your public subnet's route table must contain a route rule that directs all Internet-bound traffic from the subnet to the Internet Gateway.

### Working with security groups and Network ACLs

The VPC is all about providing your applications a much more secure environment than what your traditional EC2 service can offer. This security is provided in two layers in the form of security groups and Network ACLs. The security groups can be used to set rules that can control both inbound and outbound traffic flow from the instance and hence work more at the instance level. The Network ACLs on the other hand operate at the subnet level, either allowing or disallowing certain type of traffic to flow in and out of your subnet. Important thing to remember here is that the Network ACLs are actually optional and can be avoided altogether if your security requirements are at a minimal. However, I would strongly recommend that you use both the security groups and Network ACLs for your VPC environments. As the saying goes - **better safe, than sorry!**

Coming back to your newly created VPC, the VPC wizard creates and populates a **default Security Group** and a **default Network ACL** option for you to use in an as-is condition. The default Security Group has a single inbound and outbound rule, as explained in the following:

| Default inbound security rule |          |            |   |
|-------------------------------|----------|------------|---|
| Source                        | Protocol | Port Range | Remarks   |
| Security_Group_ID             | All      | All        | Permits inbound traffic from instances belonging to the same security group |

| Default Outbound Security Rule |          |            |                                       |
|--------------------------------|----------|------------|---------------------------------------|
| Destination                    | Protocol | Port Range | Remarks                               |
| 0.0.0.0/0                      | All      | All        | Permits all outbound traffic from the |

|  |  |           |
|--|--|-----------|
|  |  | instances |
|--|--|-----------|

You can add, edit, and modify the rules in the default Security Group; however, you cannot delete it. As a good practice, it is always recommended that you do not use this default Security Group but rather create your own. So, let's go ahead and create three security groups: one for the web servers in the public subnet, one for the database servers in the private subnet, and one for the specially created NAT Instance.

To create a new Security Group using the VPC Dashboard, select the **Security Groups** option from the **navigation pane**. Next, from the **Security Groups dashboard**, select **Create Security Group**, as shown in the following screenshot:

The screenshot shows the AWS VPC Security Groups dashboard. At the top, there are two buttons: 'Create Security Group' (highlighted in blue) and 'Delete Security Group'. Below these is a search bar with the placeholder 'VPC-722' and a clear icon. Underneath the search bar is a table header with columns: Name tag, Group ID, Group Name, VPC, and Description. A single row is visible below the header, showing 'sg-16d76072' under 'Group ID', 'default' under 'Group Name', 'vpc-722e5d17 (10.0.0.0/16)' under 'VPC', and 'default VPC security group' under 'Description'.

Using the **Create Security Group** wizard, fill in the required information as described in the following:

- **Name tag:** A unique tag name for your Security Group.
- **Group name:** A suitable name for your Security Group. In this case, I have provided it as **US-WEST-PROD-WEB-SG**.
- **Description:** An optional description for your security group.
- **VPC:** Select the newly created VPC from the dropdown list, as shown in the following screenshot. Click on **Yes, Create** once done.

The screenshot shows the 'Create Security Group' wizard. It has four input fields: 'Name tag' (value: US-WEST-PROD-WEB), 'Group name' (value: US-WEST-PROD-WEB-SG), 'Description' (value: US-WEST-PROD-WEB Security Group), and 'VPC' (dropdown menu showing 'vpc-f7aad692 (10.0.0.0/16) | US-WEST-PROD-1'). At the bottom right are two buttons: 'Cancel' and a blue 'Yes, Create' button.

Once your Security Group has been created, select it from the **Security Groups dashboard** and click on the **Inbound Rules** tab. Click on the **Edit** option to add the following rule sets:

| Web server inbound security rule |          |            |  |
|----------------------------------|----------|------------|--|
| Source                           | Protocol | Port Range | Remarks  |
| 0.0.0.0/0                        | TCP      | 22         | Permit inbound SSH access to web server instance   |
| 0.0.0.0/0                        | TCP      | 80         | Permit inbound HTTP access to web server instance  |
| 0.0.0.0/0                        | TCP      | 443        | Permit inbound HTTPS access to web server instance |

Similarly, click on the **Outbound Rules** tab and fill out the Security Group's outbound rules as described in the following:

| Web server outbound security rule |          |            |   |
|-----------------------------------|----------|------------|---|
| Destination                       | Protocol | Port Range | Remarks   |
| DB_SECURITY_GROUP                 | TCP      | 1433       | Permits outbound Microsoft SQL Server traffic to the database servers |
| DB_SECURITY_GROUP                 | TCP      | 3306       | Permits outbound MySQL traffic to the database servers                |

Replace DB\_SECURITY\_GROUP with the Security Group ID of your database server's Security Group. Remember to save the rules by selecting the **Save** option, as shown in the following screenshot:

| Type                | Protocol | Port Range | Destination | Remove |
|---------------------|----------|------------|-------------|--------|
| MS SQL (1433)       | TCP (6)  | 1433       | sg-2ce95048 |        |
| MySQL/Aurora (3306) | TCP (6)  | 3306       | sg-2ce95048 |        |

Similarly, let's go ahead and create a Security Group for our database servers as well. Populate the inbound rules as described in the following:

| Database server inbound security rule |          |            |   |
|---------------------------------------|----------|------------|---|
| Source                                | Protocol | Port Range | Remarks   |
| WEB_SECURITY_GROUP                    | TCP      | 1433       | Permits Web Server instances to access the Microsoft SQL Server |
| WEB_SECURITY_GROUP                    | TCP      | 3306       | Permits Web Server instances to access the MySQL Server         |

Replace WEB\_SECURITY\_GROUP with the Security Group ID of your web server's Security Group ID and save the rules before you continue with the outbound rules additions:

| Database server outbound security rule |          |            |  |
|--|----------|------------|--|
| Source                                 | Protocol | Port Range | Remarks  |
| 0.0.0.0/0                              | TCP      | 80         | Permit outbound HTTP access to database server instance  |
| 0.0.0.0/0                              | TCP      | 443        | Permit outbound HTTPS access to database server instance |

Note that here we are permitting only the outbound Internet access to the database servers so that they can receive important patches and updates from the net. In reality, the Internet bound traffic from these servers will be routed through the NAT instance, which will forward the traffic to the Internet via your Internet Gateway.

Finally, go ahead and create the NAT instance's Security Group. Populate the inbound security rules as mentioned in the following:

| NAT instance inbound security rule  |          |            |  |
|-------------------------------------|----------|------------|--|
| Source                              | Protocol | Port Range | Remarks  |
| 0.0.0.0/0                           | TCP      | 22         | Permits inbound SSH access to the NAT Instance |
| 192.168.1.0/24                      | TCP      | 80         | Permit inbound HTTP access to the NAT instance |
| 192.168.1.0/24                      | TCP      | 443        | Permit inbound HTTPS access to NAT instance    |
| NAT instance outbound security rule |          |            |  |
| Source                              | Protocol | Port Range | Remarks  |
| 0.0.0.0/0                           | TCP      | 80         | Permit outbound HTTP access to NAT instance    |
| 0.0.0.0/0                           | TCP      | 443        | Permit outbound HTTPS access to NAT instance   |

With the security groups created, you are now ready to launch your instances into the VPC. Let's have a quick look at the steps required to do so!

### Launching instances in your VPC

Once your VPC is ready and the security groups and Network ACLs have been modified as per requirement, you are now ready to launch instances within your VPC. You can either launch instances directly from the **VPC Management Dashboard** or from the **EC2 Management Console** as well. In this case, let's go ahead and use the EC2 Management Console.

## Creating the web servers

From the **EC2 Management Console**, select the **Launch Instance** option. This will bring up the **Instance Wizard**, using which you can create and launch your web server instances

From the next page, select any instance type for the new web server instances. In my case, I went ahead and used the default **t1.micro** instance type.

Next, from the **Configure Instance Details** page, select the newly created VPC (**US-WEST-PROD-1**) from the **Network** dropdown list and provide the web server's public subnet (**US-WEST-PROD-WEB**), as shown in the following screenshot. You can optionally choose to change the **Auto-assign Public IP** setting; however, in this case, make sure that this setting is set to **Enable** otherwise your web server instances will not receive their public IPs.

**Step 3: Configure Instance Details**

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot Instances to take a discount, and more.

|                               |  |
|-------------------------------|--|
| Number of instances           | <input type="text" value="1"/>   |
| Purchasing option             | <input type="checkbox"/> Request Spot Instances  |
| Network                       | vpc-d55a22b0 (192.168.0.0/16)   US-WEST-PROD-1 <a href="#">Create new VPC</a>                                    |
| Subnet                        | subnet-8de38ffa(192.168.1.0/24)   US-WEST-PROD-1 <a href="#">Create new subnet</a><br>250 IP Addresses available |
| Auto-assign Public IP         | <input type="button" value="Use subnet setting (Enable)"/>   |
| IAM role                      | <input type="text" value="None"/> <a href="#">Create new IAM role</a>  |
| Shutdown behavior             | <input type="text" value="Stop"/>  |
| Enable termination protection | <input type="checkbox"/> Protect against accidental termination  |

Add the required **Storage**, **Tag** the instance, and provide it with the web server **Security Group** that we created a while back. Once you have completed the formalities, review your instance's settings and finally launch it in your VPC!

Once the instance starts, verify whether it received both the private and the public IP or not. Log in to your web server instance and check whether it can reach the Internet or not by simply pinging to one of Google's DNS servers like 8.8.8.8. If all goes well, then your web server instances are all ready for production use!

## Creating the database servers

The same process applies for the database servers as well. Simply remember to select the correct subnet (**US-WEST-PROD-DB**) for the database servers, as shown in the following screenshot:

The screenshot shows the AWS VPC configuration interface. It displays the following settings:

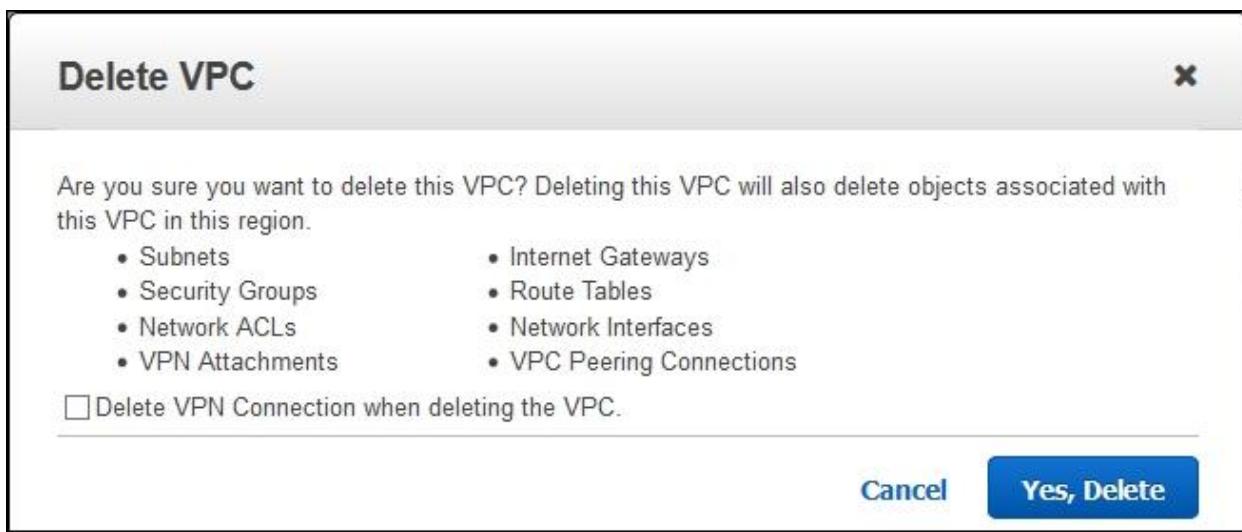
- Network:** vpc-d55a22b0 (192.168.0.0/16) | US-WEST-PROD-1
- Subnet:** subnet-8ee38ff9(192.168.5.0/24) | US-WEST-PROD-1  
251 IP Addresses available
- Auto-assign Public IP:** Use subnet setting (Disable)

On the right side, there are two buttons: "Create new VPC" and "Create new subnet".

Also note the **Auto-assign Public IP** setting for the database server's private subnet. By default, this should be disabled for the private subnet as we don't want our database instances to communicate with the Internet directly. All Internet-bound traffic from the database servers will pass via the NAT instance only. But how do you test whether your database servers are working correctly? By design, you cannot SSH into the database servers directly from your local desktops as the private subnet is isolated from the Internet. So, an alternative would be to set up something called as a **Bastion Host**. A Bastion Host is a special instance that acts as a proxy using which you can SSH into your database instances. This Bastion Host will be deployed in your public subnet and will basically only route SSH traffic from your local network over to the database server instances. But remember, this feature comes with its own set of security risks! Running a weak or poorly configured Bastion Host can prove to be harmful in production environments, so use them with care!

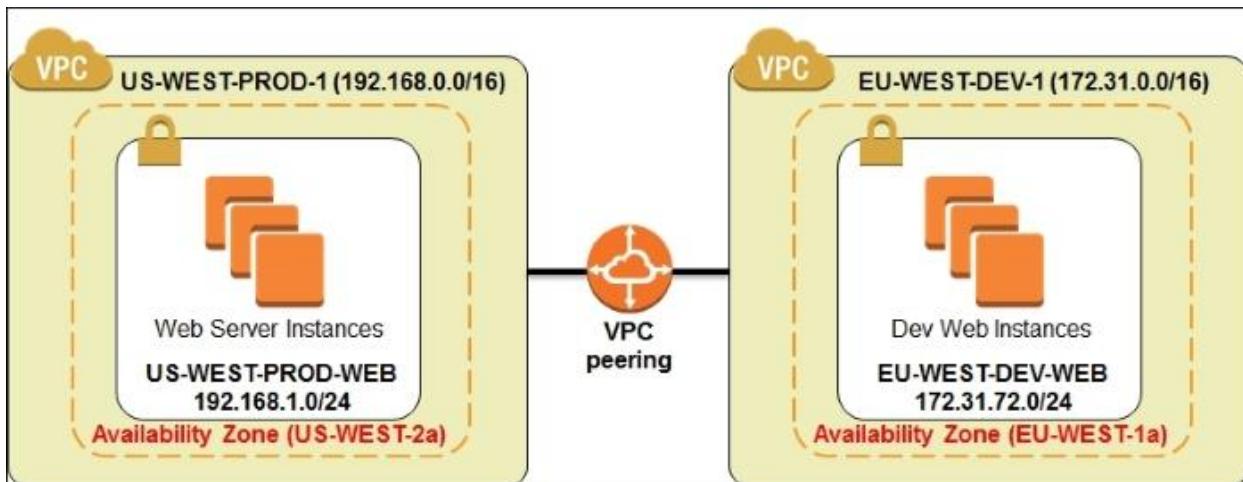
### Planning next steps

Well we have covered a lot in this chapter, but there are a few things still that you can try out on your own with regards to VPCs. First up, is cleaning up a VPC! Creating a VPC is easy enough and so is its deletion. You can delete an unused VPC from the **VPC Management dashboard** by simply selecting the **VPC**, clicking on the **Actions** tab, and selecting the **Delete VPC** option. This will bring up the **Delete VPC** dialog as shown in the following screenshot:



As you can see, the delete VPC option will delete all aspects of your VPC, including subnets, Network ACLs, Internet Gateways, and so on. You can optionally even delete any VPN connections as well by selecting the **Delete VPN Connections when deleting the VPC** checkbox. Remember that once you delete a VPC, you can't recover it back, so make sure that you don't have any active instances running on it before you go ahead and delete it. Also remember to clean up on the instances as well, especially the NAT Instance and the Bastion Host if you have created them.

The second thing that I would recommend trying out is called as **VPC peering**. VPC peering is nothing more than network connections between two different VPCs. Instances in one VPC communicate with instances present in another VPC using their private IP addresses alone, so there is no need to route the traffic over the Internet as well. You can connect your VPC with a different VPC that is either owned by you or by someone else's **Bastion Host**. All it needs is a request to be generated from the source VPC and sent to the destination VPC, along with a few route rules that will allow the traffic to flow from one point to the other. The following is the image describing the VPC peering:



You can read more about VPC peering at <http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-peering.html>.

The third thing that really is worth testing out is the hardware VPN connectivity with your VPC. I know you are probably thinking that since it's a hardware VPN connectivity, it means that I need some special hardware equipment like a router and so on. Well that's not quite true! You can set up an easy VPN connection using software as well, for example, OpenVPN. OpenVPN basically allows you to create a secure network connection from your local network to Amazon VPC using a VPN connection.

All you need to do is deploy an OpenVPN server in your VPC and configure that to accept incoming traffic from your private network. Then, install an OpenVPN client on your remote desktop and try connecting to the OpenVPN server placed in the VPC. If all goes well, you should have access to your VPC instances from your local desktop! Do note that you will have to open up additional security rules and network ACLs to allow this type of traffic to flow through your VPC subnet.

Last but not least, I would also recommend for you to have a look at VPC's Flow Logs. This is a simple logging feature provided in VPC to capture traffic information and store it using Amazon CloudWatch Logs. Flow Logs can help you analyze your network traffic flow for bottlenecks, observe certain traffic trends, as well as monitor traffic that reaches your instances. You can read more about Flow Logs at <http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/flow-logs.html>.

## Best practices and recommendations

The following are some key best practices and recommendations to keep in mind when using VPCs:

- Plan and design your VPC before actually implementing one. Determine the right choice of subnet that your application will need and build your VPC around it.
- Choose your VPC's network block allocation wisely. A /16 subnet can provide you with a potential 65,534 IP addresses that rarely will get utilized. So ideally, go for a /18 (16,382 IP addresses) or a /20 (4094 IP addresses) as your VPC network choice.
- Always plan and have a set of spare IP address capacity for your VPC. For example, consider the network block for my VPC as 192.168.0.0/18.
- In this case, we design the subnet IP addressing as follows:
  - 192.168.32.0/19 Public Subnet
  - 192.168.64.0/19 Public Subnet spares
  - 192.168.128.0/20 Private Subnet
  - 192.168.192.0/20 Private Subnet spares

Remember that you cannot edit a network block's size once it is created for a VPC. The only way to change the network block is by deleting this VPC and creating a new one in its place.

Use different security groups to secure and manage traffic flows from your instances. For example, a separate Security Group for web servers and a different one for your database servers. Avoid using the default security groups at all times.

Leverage multiple AZs to distribute your subnets across geographies. For example, the US-WEST region has three AZs, namely us-west-2a, us-west-2b, and us-west-2c. So an ideal situation would have you divide your VPC's network block and create subnets in each of these AZs evenly. The more AZs, the better the fault tolerance for your VPC.

Leverage IAM to secure your VPC at the user level as well. Create dedicated users with restricted access to your VPC and its resources.

Create and stick with a standard naming convention so that your VPC's resources can be easily identified and tagged. For example, in our scenarios, we named the VPC as US-

WEST-PROD-1, which clearly identifies this particular VPC to be hosted in the US-WEST region and to be a PRODUCTION environment.

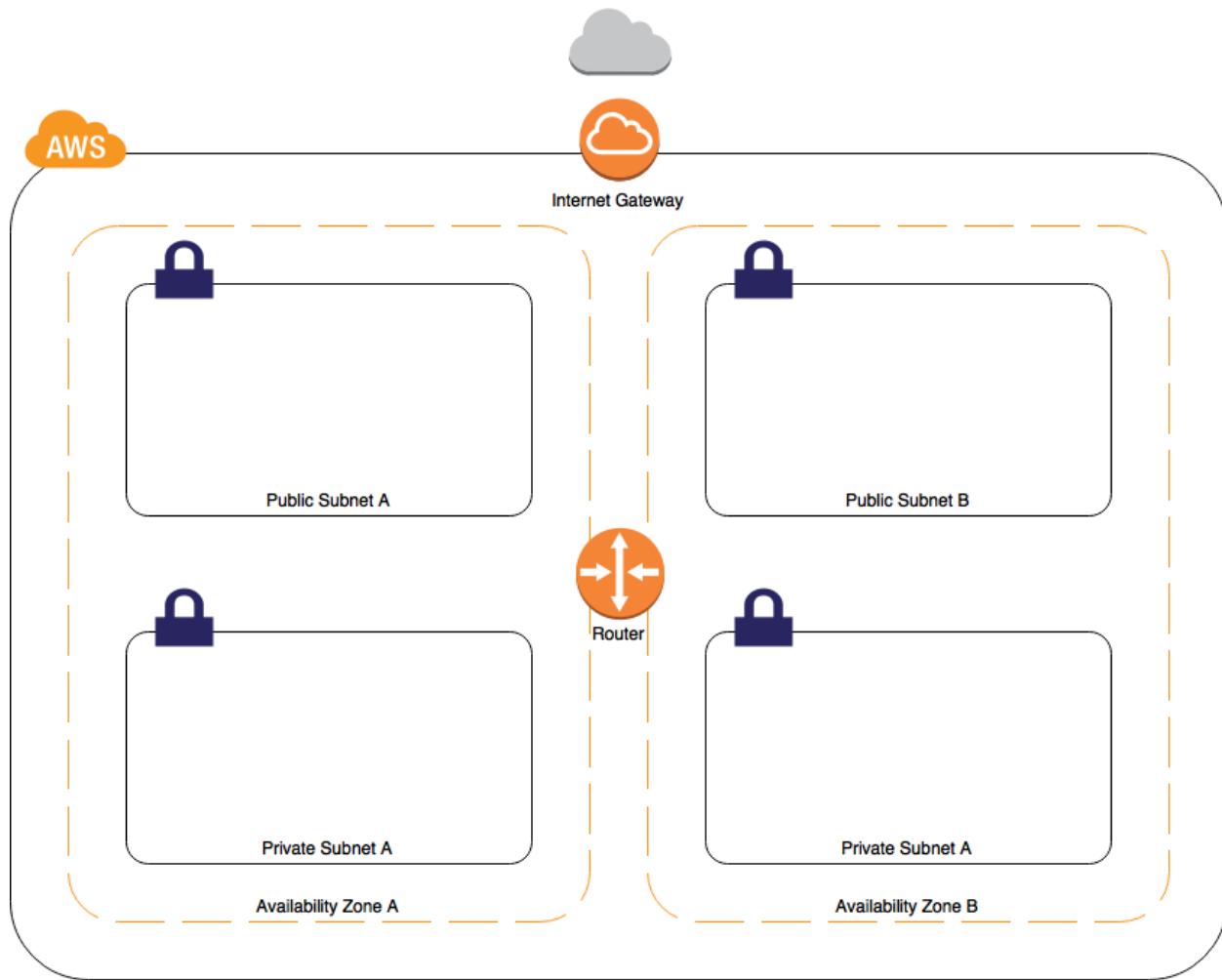
## VPCCloud Formation Recipes

Networking is a foundational component of using other AWS services such as EC2, RDS, and others. Using constructs such as VPCs and NAT gateways gives you the capability and confidence to secure your resources at a networking level. At a DNS level, Route 53 provides connectivity to your users in a responsive and fault-tolerant way that ensures the best performance in a variety of scenarios.

## Building a secure network

In this recipe, we're going to build a secure network (VPC) in AWS. This network will consist of two public and private subnets split across two Availability Zones. It will also allow inbound connections to the public subnets for the following:

- SSH (port 22)
- HTTP (port 80)
- HTTPS (port 443)



Building a secure network

## Getting ready

Before we proceed, you're going to need to know the names of at least two Availability Zones in the region we're deploying to. The recipes in this book will typically deploy to us-east-, so to get things moving you can just use the following:

- us-east-1a
- us-east-1b

Note:

When you create an AWS account, your zones are randomly allocated. This means that us-east-1a in your account isn't necessarily the same data center as us-east-1a in my account.

## How to do it...

Go ahead and create a new CloudFormation template for our VPC. Just a heads-up: this will be one of the larger templates that we'll create in this book:

1. The first two Parameters correspond to the Availability Zones we discussed previously. We don't provide any default values for these parameters, to maintain region portability:

Parameters:

AvailabilityZone1:

Description: Availability zone 1 name (e.g. us-east-1a)

Type: AWS::EC2::AvailabilityZone::Name

AvailabilityZone2:

Description: Availability zone 2 name (e.g. us-east-1b)

Type: AWS::EC2::AvailabilityZone::Name

2. The shell of our VPC has now been created. At this point, it's not connected to the Internet, so it's not entirely useful to us. We need to add an Internet gateway and attach it to our VPC. Go ahead and do that, as follows:

Resources:

# VPC & subnets

ExampleVPC:

Type: AWS::EC2::VPC

Properties:

CidrBlock: !Ref VPCCIDR

EnableDnsSupport: true

EnableDnsHostnames: true

Tags:

- { Key: Name, Value: Example VPC }

PublicSubnetA:

Type: AWS::EC2::Subnet

Properties:

AvailabilityZone: !Ref AvailabilityZone1

CidrBlock: !Ref PublicSubnetACIDR

MapPublicIpOnLaunch: true

VpcId: !Ref ExampleVPC

Tags:

- { Key: Name, Value: Public Subnet A }

PublicSubnetB:

Type: AWS::EC2::Subnet

Properties:

AvailabilityZone: !Ref AvailabilityZone2

CidrBlock: !Ref PublicSubnetBCIDR

MapPublicIpOnLaunch: true

```
VpcId: !Ref ExampleVPC
  Tags:
    - { Key: Name, Value: Public Subnet B }
PrivateSubnetA:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: !Ref AvailabilityZone1
    CidrBlock: !Ref PrivateSubnetACIDR
    VpcId: !Ref ExampleVPC
      Tags:
        - { Key: Name, Value: Private Subnet A }
PrivateSubnetB:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: !Ref AvailabilityZone2
    CidrBlock: !Ref PrivateSubnetBCIDR
    VpcId: !Ref ExampleVPC
      Tags:
        - { Key: Name, Value: Private Subnet B }
```

Note: AWS reserves a small number of IP addresses in your IP space for AWS-specific services. The VPC DNS server is one such example of this. It's usually located at the second (\*.2) IP address in the block allocated to your VPC.

3. The default values we provide for the subnets will allocate 512 IP addresses to each subnet:

```
VPCCIDR:
  Description: CIDR block for VPC
  Type: String
  Default: "172.31.0.0/21" # 2048 IP addresses
PublicSubnetACIDR:
  Description: CIDR block for public subnet A
  Type: String
  Default: "172.31.0.0/23" # 512 IP address
PublicSubnetBCIDR:
  Description: CIDR block for public subnet B
  Type: String
  Default: "172.31.2.0/23" # 512 IP address
PrivateSubnetACIDR:
  Description: CIDR block for private subnet A
  Type: String
  Default: "172.31.4.0/23" # 512 IP address
PrivateSubnetBCIDR:
  Description: CIDR block for private subnet B
  Type: String
```

Default: "172.31.6.0/23" # 512 IP address

- Now we can start to define Resources. We'll start by defining the VPC itself, as well as the two public and two private subnets inside it:

# Internet Gateway

ExampleIGW:

Type: AWS::EC2::InternetGateway

Properties:

Tags:

- { Key: Name, Value: Example Internet Gateway }

IGWAttachment:

Type: AWS::EC2::VPCGatewayAttachment

DependsOn: ExampleIGW

Properties:

VpcId: !Ref ExampleVPC

InternetGatewayId: !Ref ExampleIGW

- We need to create a couple of route tables. The first one we'll focus on is the public route table. We'll assign this route table to the two public subnets we've created. This route table will have just one route in it, which will direct all Internet-bound traffic to the Internet gateway we created in the previous step:

# Public Route Table

# Add a route for Internet bound traffic pointing to our IGW

# A route for VPC bound traffic will automatically be added

PublicRouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref ExampleVPC

Tags:

- { Key: Name, Value: Public Route Table }

PublicInternetRoute:

Type: AWS::EC2::Route

DependsOn: IGWAttachment

Properties:

RouteTableId: !Ref PublicRouteTable

GatewayId: !Ref ExampleIGW

DestinationCidrBlock: "0.0.0.0/0"

RouteAssociationPublicA:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PublicRouteTable

SubnetId: !Ref PublicSubnetA

RouteAssociationPublicB:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PublicRouteTable

SubnetId: !Ref PublicSubnetB

6. We'll create the private route table in a similar fashion. Since the private subnet is isolated from the Internet

```
# Private Route Table
```

```
# We don't add any entries to this route table because there is  
no NAT gateway
```

```
# However a route for VPC bound traffic will automatically be added
```

PrivateRouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref ExampleVPC

Tags:

```
- { Key: Name, Value: Private Route Table }
```

PrivateSubnetAssociationA:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PrivateRouteTable

SubnetId: !Ref PrivateSubnetA

PrivateSubnetAssociationB:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

RouteTableId: !Ref PrivateRouteTable

SubnetId: !Ref PrivateSubnetB

7. We can now focus on the security aspects of our network. Let's focus on the public subnets. These are the subnets you'll add your load balancers to; you'll also add things such as bastion boxes and NAT gateways. So we need to add a **Network ACL (NACL)** with several entries:

- Allow outbound traffic to all ports. Outbound access is unrestricted from hosts in our public subnets.
- Allow inbound traffic to ephemeral ports (above 1024). This ensures that packets returned to us from our outbound connections are not dropped.
- Allow inbound access to low port numbers for SSH, HTTP, and HTTPS (22, 80, and 443):

```
# Public NACL
```

PublicNACL:

Type: AWS::EC2::NetworkAcl

Properties:

```
VpcId: !Ref ExampleVPC
  Tags:
    - { Key: Name, Value: Example Public NACL }
    # Allow outbound to everywhere
  NACLRulePublicEgressAllowAll:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
      CidrBlock: "0.0.0.0/0"
        Egress: true
        Protocol: 6
      PortRange: { From: 1, To: 65535 }
      RuleAction: allow
      RuleNumber: 100
      NetworkAclId: !Ref PublicNACL
        # Allow outbound to VPC on all protocols
  NACLRulePublicEgressAllowAllToVPC:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
      CidrBlock: !Ref VPCCIDR
        Egress: true
        Protocol: -1
      RuleAction: allow
      RuleNumber: 200
      NetworkAclId: !Ref PublicNACL
        # Allow inbound from everywhere to ephemeral ports
        # (above 1024)
  NACLRulePublicIngressAllowEphemeral:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
      CidrBlock: "0.0.0.0/0"
        Protocol: 6
      PortRange: { From: 1024, To: 65535 }
      RuleAction: allow
      RuleNumber: 100
      NetworkAclId: !Ref PublicNACL
        # Allow inbound from everywhere on port 22 for SSH
  NACLRulePublicIngressAllowSSH:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
      CidrBlock: "0.0.0.0/0"
        Protocol: 6
      PortRange: { From: 22, To: 22 }
      RuleAction: allow
      RuleNumber: 200
      NetworkAclId: !Ref PublicNACL
        # Allow inbound from everywhere on port 443 for HTTPS
```

```
NACLRulePublicIngressAllowHTTPS:  
    Type: AWS::EC2::NetworkAclEntry  
    Properties:  
        CidrBlock: "0.0.0.0/0"  
        Protocol: 6  
        PortRange: { From: 443, To: 443 }  
        RuleAction: allow  
        RuleNumber: 300  
        NetworkAclId: !Ref PublicNACL  
            # Allow inbound from everywhere on port 80 for HTTP  
NACLRulePublicIngressAllowHTTP:  
    Type: AWS::EC2::NetworkAclEntry  
    Properties:  
        CidrBlock: "0.0.0.0/0"  
        Protocol: 6  
        PortRange: { From: 80, To: 80 }  
        RuleAction: allow  
        RuleNumber: 400  
        NetworkAclId: !Ref PublicNACL  
            # Allow inbound from VPC on all protocols  
NACLRulePublicIngressAllowFromVPC:  
    Type: AWS::EC2::NetworkAclEntry  
    Properties:  
        CidrBlock: !Ref VPCCIDR  
        Protocol: -1  
        RuleAction: allow  
        RuleNumber: 500  
        NetworkAclId: !Ref PublicNACL  
NACLAffiliationPublicSubnetA:  
    Type: AWS::EC2::SubnetNetworkAclAssociation  
    Properties:  
        NetworkAclId: !Ref PublicNACL  
        SubnetId: !Ref PublicSubnetA  
NACLAffiliationPublicSubnetB:  
    Type: AWS::EC2::SubnetNetworkAclAssociation  
    Properties:  
        NetworkAclId: !Ref PublicNACL  
        SubnetId: !Ref PublicSubnetB
```

8. We need to do the same for our private subnets. These subnets are somewhat easier to deal with. They should **only** be allowed to talk to hosts within our VPC, so we just need to add some NACLs allowing inbound and outbound traffic to our VPCs IP range:

```
# Private NACL
PrivateNACL:
    Type: AWS::EC2::NetworkAcl
    Properties:
        VpcId: !Ref ExampleVPC
        Tags:
            - { Key: Name, Value: Example Private NACL }
        # Allow all protocols from VPC range
NACLRulePrivateIngressAllowVPC:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
        CidrBlock: !Ref VPCCIDR
        Protocol: -1
        RuleAction: allow
        RuleNumber: 100
        NetworkAclId: !Ref PrivateNACL
        # Allow TCP responses from everywhere
NACLRulePrivateIngressAllowEphemeral:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
        CidrBlock: "0.0.0.0/0"
        Protocol: 6
        PortRange: { From: 1024, To: 65535 }
        RuleAction: allow
        RuleNumber: 200
        NetworkAclId: !Ref PrivateNACL
        # Allow outbound traffic to everywhere, all protocols
NACLRulePrivateEgressAllowVPC:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
        CidrBlock: "0.0.0.0/0"
        Egress: true
        Protocol: -1
        RuleAction: allow
        RuleNumber: 100
        NetworkAclId: !Ref PrivateNACL
NACLAutomationPrivateSubnetA:
    Type: AWS::EC2::SubnetNetworkAclAssociation
    Properties:
        NetworkAclId: !Ref PrivateNACL
        SubnetId: !Ref PrivateSubnetA
NACLAutomationPrivateSubnetB:
    Type: AWS::EC2::SubnetNetworkAclAssociation
    Properties:
        NetworkAclId: !Ref PrivateNACL
```

SubnetId: !Ref PrivateSubnetB

9. Finally, we'll add some Outputs to our template. These outputs are usually candidates for feeding into other templates or components of automation:

Outputs:

ExampleVPC:

    Value: !Ref ExampleVPC

PublicSubnetA:

    Value: !Ref PublicSubnetA

PublicSubnetB:

    Value: !Ref PublicSubnetB

PrivateRouteTable:

    Value: !Ref PrivateRouteTable

PublicRouteTable:

    Value: !Ref PublicRouteTable

PrivateSubnetA:

    Value: !Ref PrivateSubnetA

PrivateSubnetB:

    Value: !Ref PrivateSubnetB

10. You can go ahead and create your VPC in the web console or via the CLI using the following command:

```
awscloudformation create-stack \
--stack-name secure-vpc \
--template-body file://07-building-a-secure-network.yaml \
--parameters \
ParameterKey=AvailabilityZone1,ParameterValue=<az-1> \
ParameterKey=AvailabilityZone2,ParameterValue=<az-2>
```

## How it works...

When you run this template, AWS will go ahead and create an isolated, secure network just for you. While it contains a number of resources and concepts which will be familiar to network administrators, it's essentially an empty shell, which you can now go ahead and populate.

For example, each VPC contains a virtual router. You can't see it and you can't log into it to perform any special configuration, but you can customize its behavior by modifying the route tables in this template.

The NACLs we've deployed are not stateful and should **not** be considered a substitution for security groups. NACLs are **complementary** to security groups, which are stateful and frankly much easier to change and manage than NACLs. While the NACLs in our recipe allow everywhere (0.0.0.0/0) to make inbound connections to port 22, for example, you'll want to use security groups to lock this down to a specific IP range (your corporate data center, for example).

## There's more...

Actually, there's a **lot** more. Despite the amount of code in this recipe, we've really only covered the basics of what's possible with VPCs and networking in AWS. Here are some of the main VPC topics you'll encounter as you progress with your VPC usage:

- **Direct Connect:** This is a method of connecting your DC to your VPC using a private, dedicated pipe. Doing this often provides better network performance, and may also be cheaper than a VPN connection over the Internet.
- **Virtual Private Gateway (VPN):** You can configure your VPC to connect to your corporate DC over the Internet via VPN. This requires that you run supported VPN hardware in your DC.
- IPv6 support was added recently. We've left it out to keep things simple.
- **VPC endpoints:** This feature exposes AWS endpoints inside your VPC so that you don't have to route traffic over public Internet to consume them. Only S3 is supported at the time of writing.
- **VPC peering:** You can peer a VPC to one or more VPCs so that (unencrypted) traffic can flow between them. The IP ranges must not clash and, while the peering is free, you will still need to pay for traffic between VPCs. Transitive peering isn't supported, so if you need traffic to traverse VPCs you'll require a VPN/routing appliance of some kind. Cross-account VPC peering is supported (we use this feature quite often), but cross-region peering isn't yet available.
- **VPC sizing:**
  - IPv4: You can deploy networks between sizes /28 and /16.
  - IPv6: Your VPCs will be fixed in size at /56.
  - Once your VPC has been deployed you can't change its size. If you run out of IP space, your only option is to deploy a larger VPC and migrate everything (ouch!), or you can perhaps mitigate your problem with VPC peering.
- **VPC flow-logs:** You will want to enable VPC flow-logs in order to monitor traffic and do any kind of network debugging.
- Multicast traffic isn't supported.
- Subnets must reside in a single availability zone; they can't span Availability Zones.

- **Elastic Load Balancers (ELBs)** can scale out to use a lot of private IP addresses if you are sending a large amount of traffic through them. Keep this in mind when you're sizing your subnets.
- The number of VPCs you can deploy is limited to five per region, per account. You can request to increase this limit if necessary. Internet gateways have the same limit, and increasing one limit increases the other.
- The **default VPC**:
  - First and foremost, the default VPC is created automatically for you when you create your account. It has some different properties and behaviors to the VPCs you create for yourself.
  - If you try to launch an EC2 instance without specifying a subnet ID, AWS will attempt to launch it in your default VPC.
  - It consists of only public subnets. These subnets are configured to provide a public IP address to all instances by default.
  - It's possible to delete the default VPC in a region. If you do this by mistake, or have simply decided that you'd like to undo this action, you'll need to log a support ticket with AWS to have them create a new one for you.

## Creating a NAT gateway

Unless required, your instances should not be publicly exposed to the Internet. When your instances are on the Internet, you have to assume they will be attacked at some stage.

This means most of your workloads should run on instances in private subnets. Private subnets are those that are not connected directly to the Internet.

In order to give your private instances access to the Internet, you use **network address translation (NAT)**. A NAT gateway allows your instances to initiate a connection **to** the Internet, without allowing connections **from** the Internet.

## Getting ready

For this recipe, you must have the following existing resources:

- A VPC with an **Internet gateway (IGW)**
- A public subnet
- A private subnet route table

You will need the IDs for the public subnet and private subnet route table. Both of these resources should be in the same AZ.

## How to do it...

1. Start with the usual CloudFormation template version and description:

```
AWSTemplateFormatVersion: "2010-09-09"  
Description: Create NAT Gateway and associated route.
```

2. The template must take the following required parameters:

Parameters:

```
PublicSubnetId:  
  Description: Public Subnet ID to add the NAT Gateway to  
  Type: AWS::EC2::Subnet::Id  
RouteTableId:  
  Description: The private subnet route table to add the NAT  
  Gateway route to  
  Type: String
```

3. In the Resources section, define an **Elastic IP (EIP)** that will be assigned to the NAT gateway:

Resources:

```
EIP:  
  Type: AWS::EC2::EIP  
Properties:  
  Domain: vpc
```

4. Create the NAT gateway resource, assigning it the EIP you just defined in the public subnet:

```
NatGateway:  
  Type: AWS::EC2::NatGateway  
  Properties:  
    AllocationId: !GetAttEIP.AllocationId  
    SubnetId: !Ref PublicSubnetId
```

5. Finally, define the route to the NAT gateway and associate it with the private subnet's route table:

```
Route:  
  Type: AWS::EC2::Route  
  Properties:  
    RouteTableId: !Ref RouteTableId  
    DestinationCidrBlock: 0.0.0.0/0
```

NatGatewayId: !Ref NatGateway

6. Save the template with a known filename; for example, 07-nat-gateway.yaml.
7. Launch the template with the following CLI command:

```
awscloudformation create-stack \
--stack-name nat-gateway \
--template-body file://07-nat-gateway.yaml \
--parameters \
ParameterKey=RouteTableId,ParameterValue=<route-table-id> \
ParameterKey=PublicSubnetId,ParameterValue=<public-subnet-id>
```

## How it works...

The parameters required for this recipe are as follows:

- A public subnet ID
- A private subnet route table ID

The public subnet ID is needed to host the NAT gateway, which must have Internet access. The private subnet route table will be updated with a route to the NAT gateway.

Using the AWS NAT gateway service means that AWS takes care of hosting and securing the service for you. The service will be hosted redundantly in a single AZ.

In the unlikely (but possible) event of an AZ outage, you should deploy a NAT gateway per subnet. This means that if one NAT gateway goes offline, instances in the other AZ can continue to access the Internet as normal. You are deploying your application in multiple subnets, aren't you?

This recipe will only work if you have created your own private subnets, as the default subnets in a new AWS account are all **public**. Instances in a public subnet have direct access to the Internet (via an IGW), so they do not need a NAT gateway.

## Network logging and troubleshooting

One of the benefits of using virtualized infrastructure is that you can get a level of introspection that is difficult or costly with physical hardware. Being able to quickly switch on logging at a network-device level is an extremely useful feature, especially when getting used to the interactions between VPCs, subnets, NACLs, routing, and security groups.

In this recipe, we will turn on logging for our network resources. You could do this all the time, to give yourself another layer for monitoring and auditing, or you could selectively enable it during troubleshooting, saving yourself any additional datastorage charges.

## Getting ready

For this recipe, you must have a VPC to log activity on.

## How to do it...

1. Start by defining the template version and description:

```
AWSTemplateFormatVersion: "2010-09-09"  
Description: Flow logs for networking resources
```

2. Define the Parameters for the template. In this case, it is just the VpcId to turn logging on for:

```
Parameters:  
VpcId:  
    Type: String  
    Description: The VPC to create flow logs for
```

3. Create the Resources section of the template and define the log group to use to send our flow-logs to:

```
Resources:  
LogGroup:  
    Type: AWS::Logs::LogGroup  
DeletionPolicy: Delete  
Properties:  
LogGroupName: LogGroup
```

4. Next we define the IAM role that will give the flow-logs service permission to write the logs:

```
IamRole:
```

```
Type: AWS::IAM::Role
Properties:
AssumeRolePolicyDocument:
    Version: "2012-10-17"
    Statement:
        -
            Effect: Allow
            Principal:
                Service: vpc-flow-logs.amazonaws.com
                Action: sts:AssumeRole
Policies:
    -
        PolicyName: CloudWatchLogsAccess
        PolicyDocument:
            Version: "2012-10-17"
            Statement:
                -
                    Action:
                        - logs:CreateLogGroup
                        - logs:CreateLogStream
                        - logs:PutLogEvents
                        - logs:DescribeLogGroups
                        - logs:DescribeLogStreams
                    Effect: Allow
                    Resource: "*"
```

## 5. Finally, we define the flow-log itself:

```
FlowLog:
    Type: AWS::EC2::FlowLog
    DependsOn: LogGroup
    Properties:
        DeliverLogsPermissionArn: !GetAttIamRole.Arn
        LogGroupName: LogGroup
        ResourceId: !Ref VpcId
        ResourceType: VPC
        TrafficType: ALL
```

6. Save the template, and give it a known filename such as 07-flow-logs.yaml.
7. Create the flow-logs and associated resources by creating the template with the following command:

```
awscloudformation create-stack \
--stack-name VpcFlowLogs \
--template-body file://07-flow-logs.yaml \
```

```
--capabilities CAPABILITY_IAM \
--parameters ParameterKey=VpcId,ParameterValue=<your-vpc-id>
```

8. Once launched (and assuming you have network activity), you will be able to see your flow-log in the CloudWatch logs console.

### How it works...

The only parameter required for this template is the VPC ID to target. We specifically target a VPC to turn on flow-logging for, because it gives us the most **bang for buck**. While you can enable flow-logs for subnets and **Elastic Network Interfaces (ENIs)** individually, if you enable them on a VPC you get flow-logs for all the networking resources contained in that VPC—which includes subnets and ENIs.

In the resources section, we start by explicitly defining the log group to **hold** the flow-logs. If you don't create the log group yourself (and specify it in your flow-log resource configuration), a log group will be created for you. This means that you will still be able to use flow-logs, but the log group won't be managed by CloudFormation and will have to be maintained (for example, deleted) manually. We have also set a **deletion policy** of **delete** for our log group. This means it will be deleted if the CloudFormation stack is deleted, which is fine for a demonstration such as this. If using in a **real** environment (such as production), remove the `DeletionPolicy` property and its value.

Next we define the IAM role to use. Via the `AssumeRolePropertyDocument` value, we give the AWS flow-logs service permission to assume this role. Without this access, the flow-logs service cannot access the account. In the `Policies` property, we give the role permission to create and update log groups and streams.

Finally, now that we have created the dependent resources, we define the flow-log resource itself. You don't need to define the resources in order of dependencies, but it is usually easier to read if you do. In the resource, we also define a `DependsOn` relationship to the log group we defined earlier, so that the log group is ready to receive the flow-logs when it is created.

The final step is to launch the template you have created, passing the VPC ID as parameter. As this template creates an IAM role to allow the VPC service to send logs to CloudWatch logs, the command to create the stack must be given the `CAPABILITY_IAM` flag to signify that you are aware of the potential impact of launching this template.

### There's more...

Turning on logging is just the start of the troubleshooting process. There are a few other things you should be aware of when using flow-logs.

## Log format

Once logging is enabled, you can view the logs in the CloudWatch logs console. Here is a summary of the type of information you will see in the flow-log (in order):

- The VPC flow-logs version
- The AWS account ID
- The ID of the network interface
- The source IPv4 or IPv6 address
- The destination IPv4 or IPv6 address
- The source port of the traffic
- The destination port of the traffic
- The IANA protocol number of the traffic
- The number of packets transferred
- The number of bytes transferred
- The start time of the capture window (in Unix seconds)
- The end time of the capture window (in Unix seconds)
- The action associated with the traffic; for example, ACCEPT or REJECT
- The logging status of the flow-log; for example, OK, NODATA, or SKIPDATA

## Updates

You cannot update the configuration of an existing flow-log; you must delete it and recreate it if you want to change any settings associated. This is another reason why it is good to explicitly create and manage the associated log group.

## Omissions

Some traffic is not captured by the flow-logs service, as follows:

- Traffic to the Amazon DNS server (x.x.x.2 in your allocated range)
- Traffic for Amazon Windows license activation (obviously only applicable to Windows instances)
- Traffic to and from the instance metadata service (that is, IP address 169.254.169.254)
- DHCP traffic
- Traffic to the reserved VPC IP address for the default VPC router (x.x.x.1 in your allocated range)



**EC2 Administration****Contents**

|   |    |
|---|----|
| Introducing EC2! .....                      | 2  |
| EC2 use cases.....                          | 2  |
| Introducing images and instances.....       | 3  |
| Understanding images.....                   | 4  |
| Understanding instances.....                | 7  |
| EC2 instance pricing options .....          | 8  |
| Note .....                                  | 9  |
| Working with instances .....                | 11 |
| Configuring your instances.....             | 11 |
| Launching instances using the AWS CLI ..... | 12 |
| Note .....                                  | 15 |
| Cleaning up!.....                           | 16 |
| Recommendations and best practices .....    | 16 |
| Understanding EBS volumes .....             | 17 |
| Tip .....                                   | 18 |
| EBS volume types .....                      | 18 |



## Introducing EC2!

EC2 is a service that basically provides scalable compute capacity on an on-demand, pay-per-use basis to its end users. Let's break it up a bit to understand the terms a bit better. To start with, EC2 is all about server virtualization! And with server virtualization, we get a virtually unlimited capacity of virtual machines or, as AWS calls it, **instances**. Users can dynamically spin up these instances, as and when required, perform their activity on them, and then shut down the same while getting billed only for the resources they consume.

EC2 is also a highly scalable service, which means that you can scale up from just a couple of virtual servers to thousands in a matter of minutes, and vice versa—all achieved using a few simple clicks of a mouse button! This scalability accompanied by dynamicity creates an **elastic** platform that can be used for performing virtually any task you can think of! Hence, the term Elastic Compute Cloud! Now that's awesome!

With virtually unlimited compute capacity, you also get added functionality that helps you to configure your virtual server's network, storage, as well as security. You can also integrate your EC2 environment with other AWS services such as IAM, S3, SNS, RDS, and so on. To provide your applications with add-on services and tools such as security, scalable storage and databases, notification services, and so on and so forth.

### EC2 use cases

Let's have a quick look at some interesting and commonly employed use cases for AWS EC2:

- **Hosting environments:** EC2 can be used for hosting a variety of applications and software, websites, and even games on the cloud. The dynamic and scalable environment allows the compute capacity to grow along with the application's needs, thus ensuring better quality of service to end users at all

times. Companies such as Netflix, Reddit, Ubisoft, and many more leverage EC2 as their application hosting environments.

- **Dev/Test environments:** With the help of EC2, organizations can now create and deploy large scale development and testing environments with utmost ease. The best part of this is that they can easily turn on and off the service as per their requirements as there is no need for any heavy upfront investments for hardware.
- **Backup and disaster recovery:** EC2 can be also leveraged as a medium for performing disaster recovery by providing active or passive environments that can be turned up quickly in case of an emergency, thus resulting in faster failover with minimal downtime to applications.
- **Marketing and advertisements:** EC2 can be used to host marketing and advertising environments on the fly due to its low costs and rapid provisioning capabilities.
- **High Performance Computing (HPC):** EC2 provides specialized virtualized servers that provide high performance networking and compute power that can be used to perform CPU-intensive tasks such as Big Data analytics and processing. NASA's JPL and Pfizer are some of the companies that employ the use of HPC using EC2 instances.

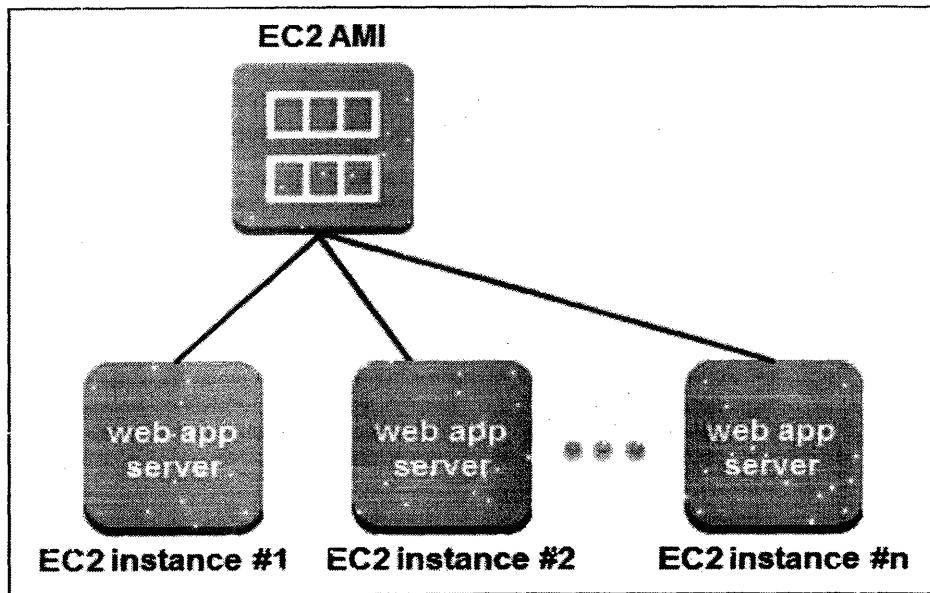
### **Introducing images and instances**

Almost all IT companies today run their workloads off virtualized platforms such as VMware vSphere or Citrix XenServer to even Microsoft's Hyper-V. AWS, too, got into the act but decided to use and modify a more off the shelf, open sourced Xen as its virtualization engine. And like any other virtualization technology, this platform was also used to spin up virtual machines using either some type of configuration files or some predefined templates. In AWS's vocabulary, these virtual machines came to be known as **instances** and their master templates came to be known as **images**.

### **Understanding images**

As discussed earlier, images are nothing more than preconfigured templates that you can use to launch one or more instances from. In AWS, we call these images **Amazon Machine Images (AMIs)**. Each AMI contains an operating system which can range from any modern Linux distro to even Windows Servers, plus some optional software application, such as a web server, or some application server installed on it.

It is important, however, to understand a couple of important things about AMIs. Just like any other template, AMIs are static in nature, which basically means that once they are created, their state remains unchanged. You can spin up or launch multiple instances using a single AMI and then perform any sort of modifications and alterations within the instance itself. There is also no restriction on the size of instances that you can launch based on your AMI. You can select anything from the smallest instance (also called as a **micro** instance) to the largest ones that are generally meant for high performance computing. Take a look at the following image of EC2 AMI:



Secondly, an AMI can contain certain launch permissions as well. These permissions dictate whether the AMI can be launched by anyone (**public**) or by someone or some

account which I specify (**explicit**) or I can even keep the AMI all to myself and not allow anyone to launch instances from it but me (**implicit**). Why have launch permissions? Well, there are cases where some AMIs can contain some form of propriety software or licensed application, which you do not want to share freely among the general public.

In that case, these permissions come in really handy! You can alternatively even create something called as a paid AMI. This feature allows you to share your AMI to the general public, however, with some support costs associated with it.

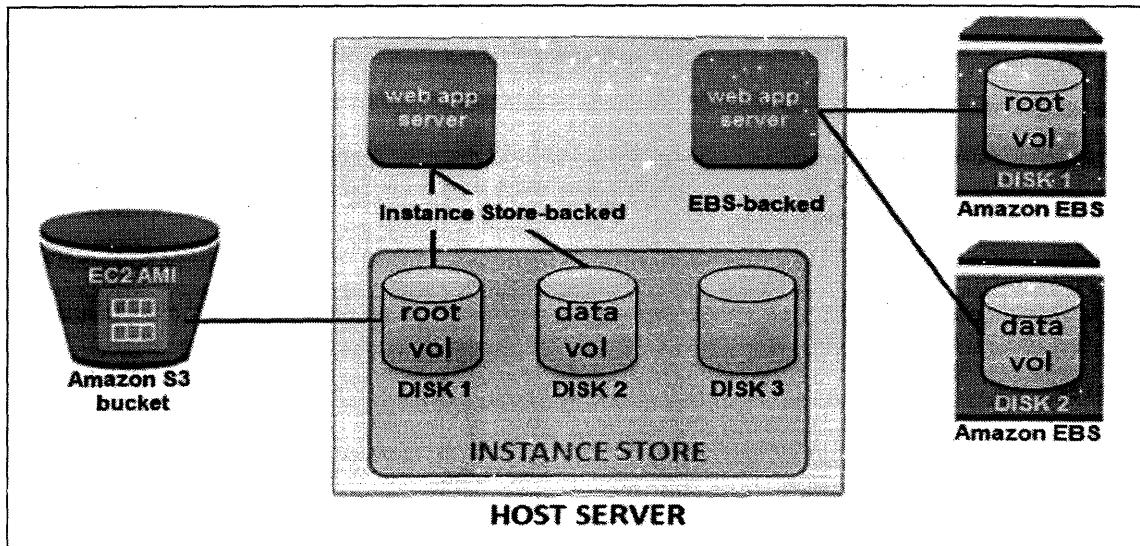
AMIs can be bought and sold using something called as the AWS Marketplace as well—a one stop shop for all your AMI needs! Here, AMIs are categorized according to their contents and you as an end user can choose and launch instances off any one of them. Categories include software infrastructure, development tools, business and collaboration tools, and much more! These AMIs are mostly created by third parties or commercial companies who wish to either sell or provide their products on the AWS platform.

AMIs can be broadly classified into two main categories depending on the way they store their root volume or hard drive:

- **EBS-backed AMI:** An EBS-backed AMI simply stores its entire root device on an **Elastic Block Store (EBS)** volume. EBS functions like a network shared drive and provides some really cool add on functionalities like snapshotting capabilities, data persistence, and so on. Even more, EBS volumes are not tied to any particular hardware as well. This enables them to be moved anywhere within a particular availability zone, kind of like a **Network Attached Storage (NAS)** drive. We shall be learning more about EBS-backed AMIs and instances in the coming chapter.
- **Instance store-backed AMI:** An instance store-backed AMI, on the other hand, stores its images on the AWS S3 service. Unlike its counterpart, instance store AMIs are not portable and do not provide data persistence capabilities as the root device data is directly stored on the instance's hard drive itself. During

deployment, the entire AMI has to be loaded from an S3 bucket into the instance store, thus making this type of deployment a slightly slow process.

The following image depicts the deployments of both the instance store-backed and EBS-backed AMIs. As you can see, the **root** and **data** volumes of the instance store-backed AMI are stored locally on the **HOST SERVER** itself, whereas the second instance uses EBS volumes to store its root device and data.



The following is a quick differentiator to help you understand some of the key differences between EB-backed and Instance store-backed AMIs:

|                  | EBS backed  | Instance store backed  |
|------------------|---|--|
| Root device      | Present on an EBS volume.   | Present on the instance itself.  |
| Disk size limit  | Up to 16 TB supported.  | Up to 10 GB supported.   |
| Data persistence | Data is persistent even after the instance is terminated.   | Data only persists during the lifecycle of the instance.                                       |
| Boot time        | Less than a minute. Only the parts of the AMI that are required for the boot process are retrieved for the instance to be made ready. | Up to 5 minutes. The entire AMI has to be retrieved from S3 before the instance is made ready. |
| Costs            | You are charged for the running instance plus the EBS volume's usage.   | You are charged for the running instance plus the storage costs incurred by S3.                |

### **Understanding instances**

So far we have only been talking about images; so now let's shift the attention over to instances! As discussed briefly earlier, instances are nothing but virtual machines or virtual servers that are spawned off from a single image or AMI. Each instance comes with its own set of resources, namely CPU, memory, storage, and network, which are differentiated by something called as instance families or instance types. When you first launch an instance, you need to specify its instance type. This will determine the amount of resources that your instance will obtain throughout its lifecycle.

AWS currently supports five instance types or families, which are briefly explained as follows:



- **General purpose:** This group of instances is your average, day-to-day, balanced instances. Why balanced? Well, because they provide a good mix of CPU, memory, and disk space that most applications can suffice with while not compromising on performance. The general purpose group comprises the commonly used instance types such as t2.micro, t2.small, t2.medium, and the m3 and m4 series which comprises m4.large, m4.xlarge, and so on and so forth. On average, this family contains instance types that range from 1 VCPU and 1 GB RAM (t2.micro) all the way to 40 VCpus and 160 GB RAM (m4.10xlarge).
- **Compute optimized:** As the name suggests, these are specialized group of instances that are commonly used for CPU-intensive applications. The group comprises two main instances types, that is, C3 and C4. On an average, this family contains instances that can range from 2 VCpus and 2.75 GB RAM (c4.large) to 36 VCpus and 60 GB RAM (c4.8xlarge).
- **Memory optimized:** Similar to the compute optimized, this family comprises instances that require or consume more RAM than CPU. Ideally, databases and analytical applications fall into this category. This group consists of

a single instance type called R3 instances, and they can range anywhere from 2 VCPUs and 15.25 GB RAM (r3.large) to 32 VCPUs and 244 GB RAM (r3.8xlarge).

- **Storage optimized:** This family of instances comprises specialized instances that provide fast storage access and writes using SSD drives. These instances are also used for high I/O performance and high disk throughput applications. The group also comprises two main instance types, namely the I2 and D2 (no, this doesn't have anything to do with R2D2!). These instances can provide SSD enabled storage ranging from 800 GB (i2.large) all the way up to 48 TB (d2.8xlarge)—now that's impressive!
- **GPU instances:** Similar to the compute optimized family, the GPU instances are specially designed for handling high CPU-intensive tasks but by using specialized NVIDIA GPU cards. This instance family is generally used for applications that require video encoding, machine learning or 3D rendering, and so on. This group consists of a single instance type called G2, and it can range between 1 GPU (g2.2xlarge) and 4 GPU (g2.8xlarge).

To know more about the various instance types and their use cases, refer to <http://aws.amazon.com/ec2/instance-types/>

### **EC2 instance pricing options**

Apart from the various instance types, EC2 also provides three convenient instance pricing options to choose from, namely on-demand, reserved, and spot instances. You can use either or all of these pricing options at the same time to suit your application's needs. Let's have a quick look at all three options to get a better understanding of them.

#### **On-demand instances**

Pretty much the most commonly used instance deployment method, the on-demand instances are created only when you require them, hence the term on-demand. On-

demand instances are priced by the hour with no upfront payments or commitments required. This, in essence, is the true pay-as-you-go payment method that we always end up mentioning when talking about clouds. These are standard computational resources that are ready whenever you request them and can be shut down anytime during its tenure.

By default, you can have a max of 20 such on-demand instances launched within a single AWS account at a time. If you wish to have more such instances, then you simply have to raise a support request with AWS using the **AWS Management Console's Support** tab. A good use case for such instances can be an application running unpredictable workloads, such as a gaming website or social website. In this case, you can leverage the flexibility of on-demand instances accompanied with their low costs to only pay for the compute capacity you need and use and not a dime more!

#### Note

On-demand instance costs vary based on whether the underlying OS is a Linux or Windows, as well as in the regions that they are deployed in.

#### Reserved instances

Deploying instances using the on-demand model has but one slight drawback, which is that AWS does not guarantee the deployment of your instance. Why, you ask? Well to put it simply, using on-demand model, you can create and terminate instances on the go without having to make any commitments whatsoever. It is up to AWS to match this dynamic requirement and make sure that adequate capacity is present in its datacenters at all times. However, in very few and rare cases, this does not happen, and that's when AWS will fail to power on your on-demand instance.

In such cases, you are better off by using something called as reserved instances, where AWS actually guarantees your instances with resource capacity reservations and significantly lower costs as compared to the on-demand model. You can choose between three payment options when you purchase reserved instances: all upfront, partial

upfront, and no upfront. As the name suggests, you can choose to pay some upfront costs or the full payment itself for reserving your instances for a minimum period of a year and maximum up to three years.

Consider our earlier example of the t2.micro instance costing \$0.0013 per hour. The following table summarizes the upfront costs you will need to pay for a period of one year for a single t2.micro instance using the reserved instance pricing model:

| Payment method  | Upfront cost | Monthly cost | Hourly cost | Savings over on-demand |
|-----------------|--------------|--------------|-------------|------------------------|
| No upfront      | \$0          | \$6.57       | \$0.009     | 31%                    |
| Partial upfront | \$51         | \$2.19       | \$0.0088    | 32%                    |
| All upfront     | \$75         | \$0          | \$0.0086    | 34%                    |

Reserved instances are the best option when the application loads are steady and consistent. In such cases, where you don't have to worry about unpredictable workloads and spikes, you can reserve a bunch of instances in EC2 and end up saving on additional costs.

### **Spot instances**

Spot instances allow you to bid for unused EC2 compute capacity. These instances were specially created to address a simple problem of excess EC2 capacity in AWS. How does it all work? Well, it's just like any other bidding system. AWS sets the hourly price for a particular spot instance that can change as the demand for the spot instances either grows or shrinks. You as an end user have to place a bid on these spot instances, and when your bid exceeds that of the current spot price, your instances are then made to run! It is important to also note that these instances will stop the moment someone else out bids you, so host your application accordingly. Ideally, applications that are non-critical in nature and do not require large processing times, such as image resizing operations, are ideally run on spot instances.

Let's look at our trusty t2.micro instance example here as well. The on-demand cost for a t2.micro instance is \$0.013 per hour; however, I place a bid of \$0.0003 per hour to run

my application. So, if the current bid cost for the t2.micro instance falls below my bid, then EC2 will spin up the requested t2.micro instances for me until either I choose to terminate them or someone else out bids me on the same—simple, isn't it?

Spot instances compliment the reserved and on-demand instances; hence, ideally, you should use a mixture of spot instances working on-demand or reserved instances just to be sure that your application has some compute capacity on standby in case it needs it.

### **Working with instances**

Okay, so we have seen the basics of images and instances along with various instance types and some interesting instance pricing strategies as well. Now comes the fun part! Actually deploying your very own instance on the cloud!

In this section, we will be using the AWS Management Console and launching our very first t2.micro instance on the AWS cloud. Along the way, we shall also look at some instance lifecycle operations such as start, stop, reboot, and terminate along with steps, using which you can configure your instances as well. So, what are we waiting for? Let's get busy!

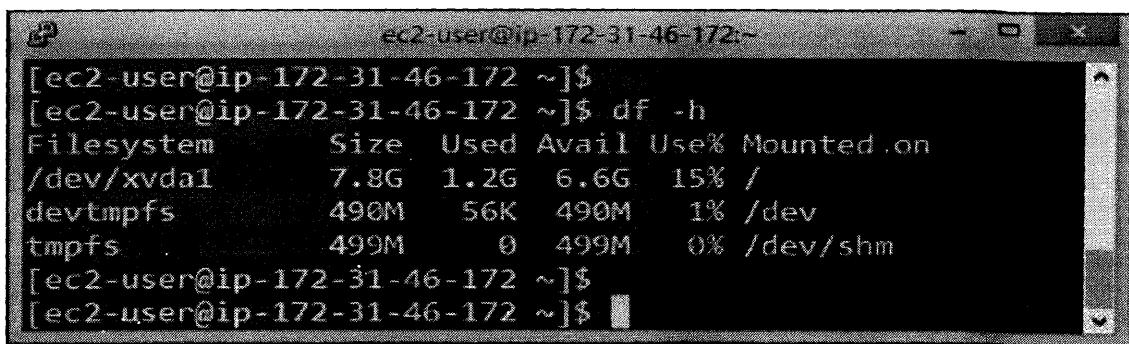
#### **Configuring your instances**

Once your instances are launched, you can configure virtually anything in it, from packages, to users, to some specialized software or application, anything and everything goes!

Let's begin by running some simple commands first. Go ahead and type the following command to check your instance's disk size:

**df -h**

Here is the output showing the configuration of the instance:



```
[ec2-user@ip-172-31-46-172 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1     7.8G  1.2G  6.6G  15% /
devtmpfs        490M   56K  490M   1% /dev
tmpfs          499M     0  499M   0% /dev/shm
[ec2-user@ip-172-31-46-172 ~]$
```

You should see an 8 GB disk mounted on the root (/) partition, as shown in the preceding screenshot. Not bad, eh! Let's try something else, like updating the operating system. AWS Linux AMIs are regularly patched and provided with necessary package updates, so it is a good idea to patch them from time to time. Run the following command to update the Amazon Linux OS:

**sudo yum update -y**

Why sudo? Well, as discussed earlier, you are not provided with root privileges when you log in to your instance. You can change that by simple changing the current user to root after you login; however, we are going to stick with the ec2-user itself for now.

What else can we do over here? Well, let's go ahead and install some specific software for our instance. Since this instance is going to act as a web server, we will need to install and configure a basic Apache HTTP web server package on it.

Type in the following set of commands that will help you install the Apache HTTP web server on your instance:

**sudo yum install httpd  
# sudo service httpd start  
# sudo chkconfig httpd on**

Launching instances using the AWS CLI

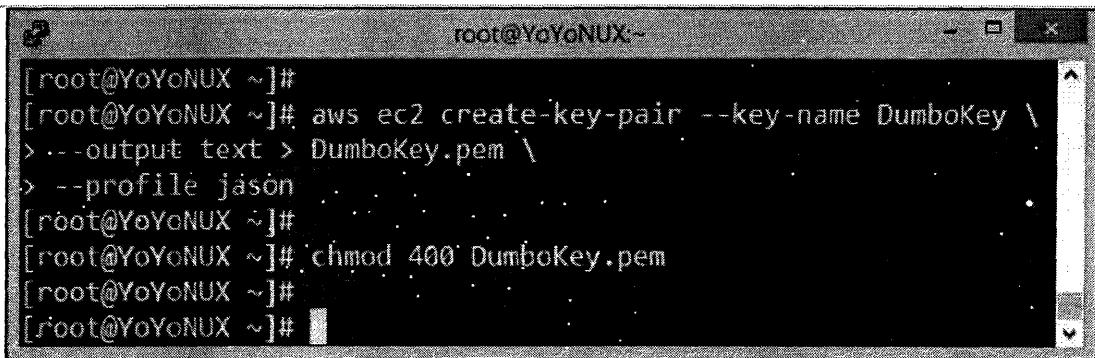
So far, we have seen how to launch and manage instances in EC2 using the EC2 dashboard. In this section, we are going to see how to leverage the AWS CLI to launch your instance in the cloud! For this exercise, I'll be using my trusty old CentOS 6.5 machine

Stage 1 – create a key pair

```
aws ec2 create-key-pair --key-name <Key_Pair_Name> \
> --output text ><Key_Pair_Name>.pem
```

Once the key pair has been created, remember to change its permissions using the following command:

```
chmod 400 <Key_Pair_Name>.pem
```



The screenshot shows a terminal window titled 'root@YoYoNUX'. The command entered is:

```
[root@YoYoNUX ~]# aws ec2 create-key-pair --key-name DumboKey \
> --output text > DumboKey.pem \
> --profile jason
[root@YoYoNUX ~]# chmod 400 DumboKey.pem
[root@YoYoNUX ~]#
[root@YoYoNUX ~]#
```

Stage 2 – create a security group

```
# aws ec2 create-security-group --group-name <SG_Name> \
> --description "<SG_Description>"
```

For creating security groups, you are only required to provide a security group name and an optional description field along with it. Make sure that you provide a simple yet meaningful name here:

```
[root@YoYoNUX ~]# aws ec2 create-security-group --group-name DumboSG \
> --description "This is yet another simple security group" \
> --profile jason
[CreateSecurityGroup]
+-----+
| GroupId | sg-6431ef00 |
+-----+
[root@YoYoNUX ~]#
[root@YoYoNUX ~]#
```

Once executed, you will be provided with the new security group's ID as the output. Make a note of this ID as it will be required in the next few steps.

#### Stage 3 – add rules to your security group

With your new security group created, the next thing to do is to add a few firewall rules to it. We will be discussing a lot more on this topic in the next chapter, so to keep things simple, let's add one rule to allow inbound SSH traffic to our instance. Type in the following command to add the new rule:

```
aws ec2 authorize-security-group-ingress --group-name <SG_Name> \
> --protocol tcp --port 22 --cidr 0.0.0.0/0
```

To add a firewall rule, you will be required to provide the security group's name to which the rule has to be applied. You will also need to provide the protocol, port number, and network CIDR values as per your requirements:

```
[root@YoYoNUX ~]#
[root@YoYoNUX ~]# aws ec2 authorize-security-group-ingress --group-name DumboSG \
> --protocol tcp --port 22 --cidr 0.0.0.0/0 \
> --profile jason
[root@YoYoNUX ~]#
[root@YoYoNUX ~]#
```

#### Stage 4 – launch the instance

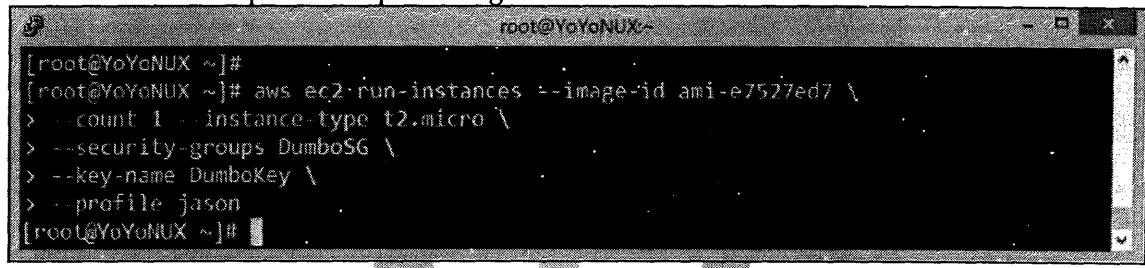
With the key pair and security group created and populated, the final thing to do is to launch your new instance. For this step, you will need a particular AMI ID along with a

few other key essentials such as your security group name, the key pair, and the instance launch type, along with the number of instances you actually wish to launch.

Type in the following command to launch your instance:

```
# aws ec2 run-instances --image-id ami-e7527ed7 \
> --count 1 --instance-type t2.micro \
> --security-groups <SG_Name> \
> --key-name <Key_Pair_Name>
```

And here is the output of the preceding commands:

A screenshot of a terminal window titled "root@YoYoNUX ~". The window contains the following text:

```
[root@YoYoNUX ~]# [root@YoYoNUX ~]# aws ec2 run-instances --image-id ami-e7527ed7 \
> --count 1 --instance-type t2.micro \
> --security-groups DumboSG \
> --key-name DumboKey \
> --profile jason
[root@YoYoNUX ~]#
```

The terminal window has a dark background with white text and a light gray border.

#### Note

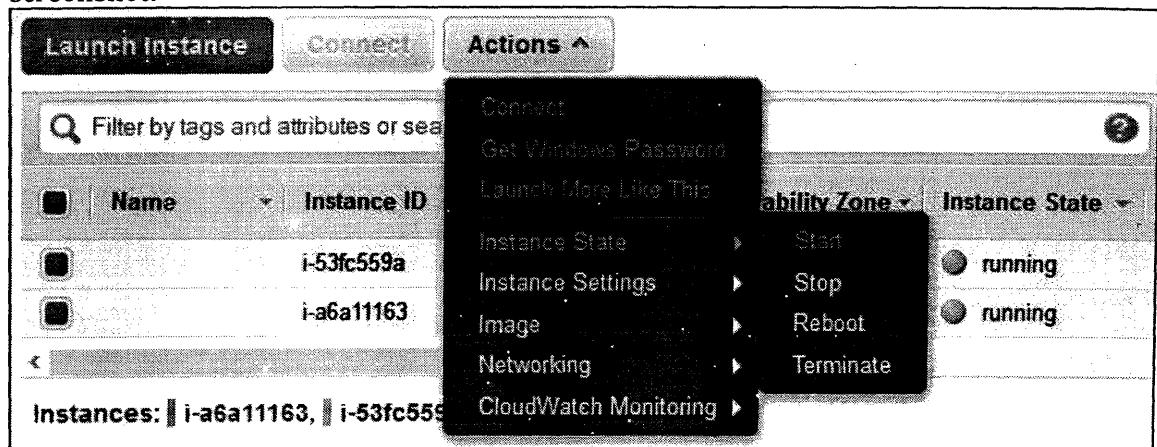
In this case, we are using the same Amazon Linux AMI (**ami-e7527ed7**) that we used during the launch of our first instance using the EC2 dashboard.

The instance will take a good two or three minutes to spin up, so be patient! Make a note of the instance's ID from the output of the ec2 run-instance command. We will be using this instance ID to find out the instance's public IP address using the EC2 describe-instance command as shown:

```
# aws ec2 describe-instances --instance-ids <Instance_ID>
```

**Cleaning up!**

Spinning up instances is one thing; you should also know how to stop and terminate them! To perform any power operations on your instance from the EC2 dashboard, all you need to do is select the particular instance and click on the **Actions** tab as shown. Next, from the **Instance State** submenu, select whether you want to **Stop**, **Reboot**, or **Terminate** your instance, as shown in the following screenshot:



It is important to remember that you only have instance stopping capabilities when working with EBS-backed instances. Each time an EBS-backed instance is stopped, the hourly instance billing stops too; however, you are still charged for the EBS volume that your instance is using. Similarly, if your EBS-backed instance is terminated or destroyed, then by default the EBS root volume attached to it is also destroyed, unless specified otherwise, during the instance launch phase.

**Recommendations and best practices**

- First and foremost, create and use separate IAM users for working with EC2. **DO NOT USE** your standard root account credentials!

- Use IAM roles if you need to delegate access to your EC2 account to other people for some temporary period of time. Do not share your user passwords and keys with anyone.
- Use a standard and frequently deployed set of AMIs as they are tried and tested by AWS thoroughly.
- Make sure that you understand the difference between instance store-backed and EBS-backed AMIs. Use the instance store with caution and remember that you are responsible for your data, so take adequate backups of it.
- Don't create too many firewall rules on a single security group. Make sure that you apply the least permissive rules for your security groups.
- Stop your instances when not in use. This will help you save up on costs as well.
- Use tags to identify your EC2 instances. Tagging your resources is a good practice and should be followed at all times.
- Save your key pairs in a safe and accessible location. Use passphrases as an added layer of security if you deem it necessary.
- Monitor your instances at all times. We will be looking at instance monitoring in depth in the coming chapters; however, you don't have to wait until then! Use the EC2 Status and Health Check tabs whenever required.

### **Understanding EBS volumes**

First up, let's understand EBS volumes a bit better. EBS volumes are nothing more than block-level storage devices that you can attach to your EC2 instances. They are highly durable and can provide a host of additional functionalities to your instances, such as data persistence, encryption, snapshotting capabilities, and so on. Majority of the time, these EBS volumes are used for storing data for a variety of applications and databases, however you can use it just as a normal hard drive as well. The best part of EBS volumes is that they can persist independently from your instances. So powering down an instance

or even terminating it will not affect the state of your EBS volumes. Your data will stay on it unless and until you explicitly delete it.

Let's look at some of the key features and benefits that EBS volumes have to offer:

- **High availability:** Unlike your instance store-backed drives, EBS volumes are automatically replicated by AWS within the availability zone in which they are created. You can create an EBS volume and attach it to any instance present in the same availability zone; however, one EBS volume cannot be attached to multiple instances at the same time. A single instance, however, can have multiple EBS volumes attached to it at any given time.
- **Encryption capabilities:** EBS volumes provide an add-on feature using which you can encrypt your volumes using standard encryption algorithms, such as AES-256, and keys as well. These keys are autogenerated the first time you employ encryption on a volume using the **AWS Key Management Service (KMS)**. You can additionally even use IAM to provide fine-grained access control and permissions to your EBS volumes.
- **Snapshot capabilities:** The state of an EBS volume can be saved using point-in-time snapshots. These snapshots are all stored incrementally on your Amazon S3 account and can be used for a variety of purposes, such as creating new volumes based on an existing one, resizing volumes, backup and data recovery, and so on.

### **Tip**

EBS volumes cannot be copied from one AWS region to another. In such cases, you can take a snapshot of the volume and copy the snapshot over to a different region using the steps mentioned at <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-copy-snapshot.html>

### **EBS volume types**

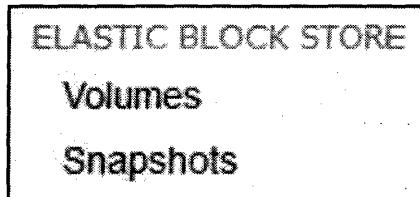
There are three different types of EBS volumes available today, each with their own sets of performance characteristics and associated costs. Let's briefly look into each one of them and their potential uses:

- **General purpose volumes (SSD):** These are by far the most commonly used EBS volume types as they provide a good balance between cost and overall performance. By default, this volume provides a standard 3 IOPS per GB of storage, so a 10 GB general purpose volume will get approximately 30 IOPS and so on so forth, with a max value of 10,000 IOPS. You can create general purpose volumes that can range in size from 1 GB to a maximum of 16 TB. Such volumes can be used for a variety of purposes, such as instance root volumes, data disks for dev and test environments, database storage, and so on.
- **Provisioned IOPS volumes (SSD):** These are a specialized set of SSDs that can consistently provide a minimum of 100 IOPS burstable up to 20,000 IOPS. You can create Provisioned IOPS Volumes that range in size from a minimum of 4 GB all the way up to 16 TB. Such volumes are ideally suited for applications that are IO intensive, such as databases, parallel computing workloads such as Hadoop, and so on.
- **Magnetic volumes:** Very similar to traditional tape drives and magnetic disks, these volumes are a good match for workloads where data is accessed infrequently, such as log storage, data backup and recovery, and so on. On an average, these volumes provide up to a 100 IOPS with an ability to burst up to 1,000 IOPS. You can create Magnetic volumes that range in size from a minimum of 1 GB all the way up to 1 TB.

### **Getting started with EBS Volumes**

Now that we have a fair idea of what an EBS Volume is, let's look at some simple ways that you can create, attach, and manage these volumes.

To view and access your account's EBS Volumes using **AWS Management Console**, simply select the **Volumes** option from the EC2 dashboard's navigation pane, as shown here:



Each EBS-backed instance's volume will appear here in the **Volume Management** dashboard. You can use this same dashboard to perform a host of activities on your volumes, such as create, attach, detach, and monitor performance, to name a few.

| Description  | Volume ID | Size | Type      | IOPS          | Snapshot                              | Created    | Availability Zone | State |
|--------------|-----------|------|-----------|---------------|---------------------------------------|------------|-------------------|-------|
| vol-a0f422b5 | 8 GiB     | gp2  | 24 / 3000 | snap-bfb086e1 | August 5, 2015 at 8:00:19 PM UTC+5:30 | us-west-2a | in-use            |       |

| Description  | Volume ID | Size                                  | Created | State                           | Attachment information | Alarm status  | Snapshot   | Availability Zone | Encrypted | KMS Key ID |
|--------------|-----------|---------------------------------------|---------|---------------------------------|------------------------|---------------|------------|-------------------|-----------|------------|
| vol-a0f422b5 | 8 GiB     | August 5, 2015 at 8:00:19 PM UTC+5:30 | in-use  | i-53fc559a:/dev/xvda (attached) | None                   | snap-bfb086e1 | us-west-2a | Not Encrypted     |           |            |

You can view any particular EBS Volume's details by simply selecting it and viewing its related information in the **Description** tab, as shown. Here, you can view the volume's ID, **Size**, **Created date**, the volume's current **State** as well as its **Attachment information**, which displays the volume's mount point on a particular instance. Additionally, you can also view the volume's health and status by selecting the **Monitoring** and **Status Checks** tab, respectively. For now, let's go ahead and create a new volume using the volume management dashboard.

### Creating EBS volumes

From the **Volume Management** dashboard, select the **Create Volume** option. This will pop up the **Create Volume** dialog box as shown here:

The screenshot shows the 'Create Volume' dialog box. The 'Type' field is set to 'General Purpose (SSD)'. The 'Size (GiB)' field contains '10' with a note '(Min: 1 GiB, Max: 16384 GiB)'. The 'IOPS' field shows '30 / 3000' with a note '(Baseline of 3 IOPS per GiB)'. The 'Availability Zone' dropdown is set to 'us-west-2a'. The 'Snapshot ID' field has 'Search (case-insensitive)' placeholder text. The 'Encryption' checkbox is checked. The 'Master Key' dropdown is set to '(default) aws/ebs'. At the bottom are 'Cancel' and 'Create' buttons.

Fill in the details as required in the **Create Volume** dialog box. For this tutorial, I went ahead and created a simple 10-GB general purpose volume:

- **Type:** From the **Type** drop-down list, select either **General Purpose (SSD)**, **Provisioned IOPS (SSD)**, or **Magnetic** as per your requirements.
- **Size (GiB):** Provide the size of your volume in GB. Here, I provided 10 GB.
- **IOPS:** This field will only be editable if you have selected Provisioned IOPS (SSD) as the volume's type. Enter the max **IOPS value** as per your requirements.
- **Availability Zone:** Select the appropriate **availability zone** in which you wish to create the volume. Remember, an EBS volume can span availability zones, but not regions.
- **Snapshot ID:** This is an optional field. You can choose to populate your EBS volume based on a third party's **snapshot ID**. In this case, we have left this field blank.

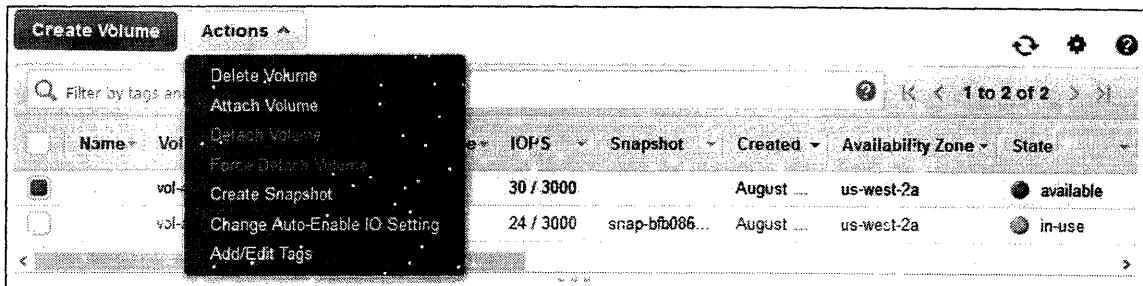
- Encryption:** As mentioned earlier, you can choose whether or not you wish to encrypt your EBS Volume. Select **Encrypt this volume checkbox** if you wish to do so.
- Master Key:** On selecting the **Encryption** option, AWS will automatically create a default key pair for the AWS's KMS. You can make a note of the KMS Key ID as well as the KMS Key ARN as these values will be required during the volume's decryption process as well.

Once your configuration settings are filled in, select **Create** to complete the volume's creation process. The new volume will take a few minutes to be available for use. Once the volume is created, you can now go ahead and attach this volume to your running instance.

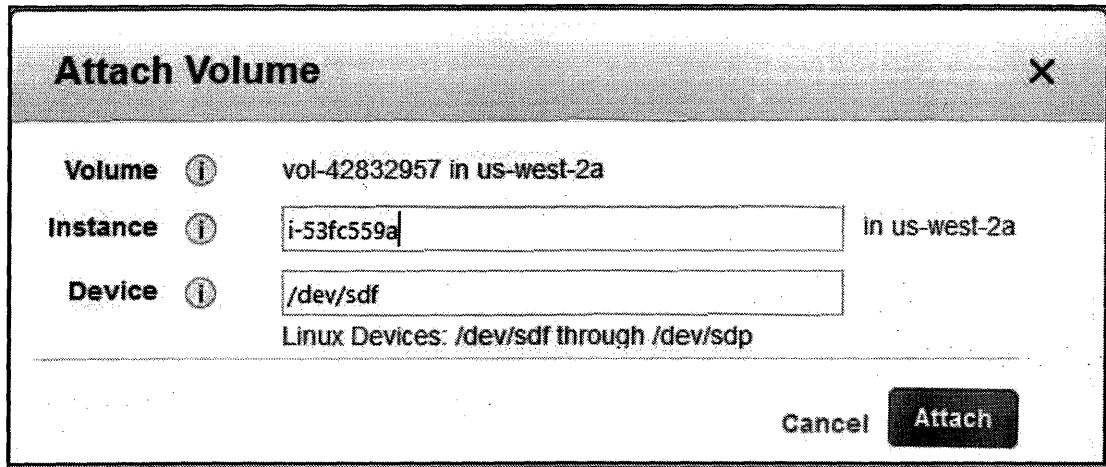
#### Attaching EBS volumes

Once the EBS volume is created, make sure it is in the **available** state before you go ahead and attach it to an instance. You can attach multiple volumes to a single instance at a time, with each volume having a unique device name. Some of these device names are reserved, for example, /dev/sda1 is reserved for the root device volume. You can find the complete list of potential and recommended device names at [http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device\\_naming.html](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device_naming.html).

The following screenshot shows the option of attaching a volume:



To attach a volume, select the **volume** from the **Volume Management** dashboard. Then select the **Actions** tab and click on the **Attach Volume** option. This will pop up the **Attach Volume** dialog box, as shown:



Type in your **instance's ID** in the **Instance** field and provide a suitable name in the **Device** field as shown. In this case, I provided the recommended device name of **/dev/sdf** to this volume. Click on **Attach** once done. The volume attachment process takes a few minutes to complete. Once done, you are now ready to make the volume accessible from your instance.

#### Accessing volumes from an instance

Once the volume is attached to an instance, you can basically format it and use it like any other block device.

To get started, first up connect to your running instance using putty or any other SSH client of your choice. Next, type in the following command to check the current disk partitioning of your instance:

**sudodf -h**

Next, run the following command to list out partitions on your current instance. You should see a default /dev/xvda partition along with its partition table and an unformatted disk partition with the name /dev/xvdf as shown in the following screenshot. The /dev/xvdf command is the newly added EBS volume that we will need to format in the upcoming steps:

**sudofdisk -l**

```
[ec2-user@ip-172-31-46-172 ~]$ sudo fdisk -l
Disk /dev/xvda: 8589 MB, 8589934592 bytes, 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: gpt
#      Start    End     Size   Type      Name
 1        4096  16777182      8G  Linux filesystem  Linux
128       2048        4095      1M  BIOS boot partition  BIOS Boot Partition.

Disk /dev/xvdf: 10.7 GB, 10737418240 bytes, 20971520 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

[ec2-user@ip-172-31-46-172 ~]$
```

Once you have verified the name of your newly added disk, you can go ahead and format with a filesystem of your choice. In this case, I have gone ahead and used the ext4 filesystem for my new volume:

**sudomkfs -t ext4 /dev/xvdf**

Now that your volume is formatted, you can create a new directory on your Linux instance and mount the volume to it using your standard Linux commands:

**sudomkdir /my-new-dir  
# sudo mount /dev/xvdf /my-new-dir**

Here is the screenshot of creating new directory using the preceding commands:

```
[ec2-user@ip-172-31-46-172 ~]$ sudo mkfs -t ext4 /dev/xvdf
[ec2-user@ip-172-31-46-172 ~]$
[ec2-user@ip-172-31-46-172 ~]$ sudo mkdir /my-new-dir
[ec2-user@ip-172-31-46-172 ~]$
[ec2-user@ip-172-31-46-172 ~]$ sudo mount /dev/xvdf /my-new-dir/
[ec2-user@ip-172-31-46-172 ~]$
[ec2-user@ip-172-31-46-172 ~]$ df -h .
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1     7.8G  1.2G  6.5G  16% /
/dev/xvdf     9.8G   23M  9.2G   1% /my-new-dir
[ec2-user@ip-172-31-46-172 ~]$
[ec2-user@ip-172-31-46-172 ~]$
```

Here's a useful tip! Once you have mounted your new volume, you can optionally edit the Linux instance's fstabfile and add the volume's mount information there. This will make sure that the volume is mounted and available even if the instance is rebooted. Make sure you take a backup copy of the fstab file before you edit it, just as a precautionary measure.

#### Detaching EBS volumes

Detaching EBS volumes is a fairly simple and straightforward process. You will first need to unmount the volume from your instance and then simply detach it using **Volume Management dashboard**.

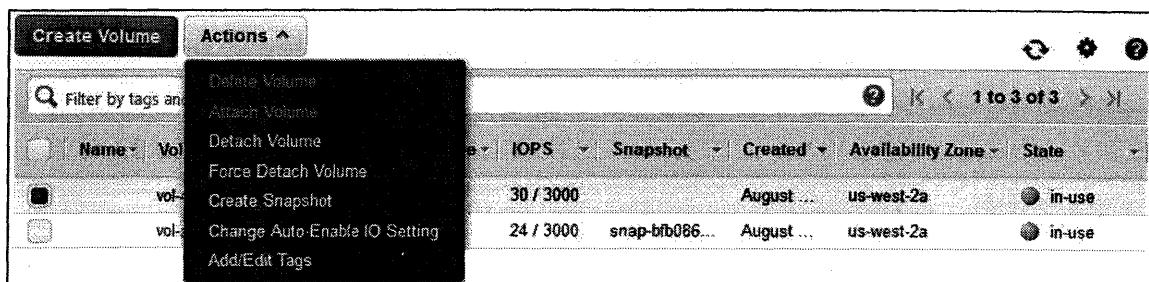
Run the following command to unmount the EBS volume from the instance:

```
sudoumount /dev/sdf
```

#### Note

Make sure you are unmounting the correct volume from the instance. **Do not** try and unmount the /dev/sda or any other root partitions.

Once the volume is successfully unmounted from the instance, detach the volume by selecting the **Detach Volume** option from the **Actions** tab, as shown here:



### Managing EBS volumes using the AWS CLI

You can create, attach, and manage EBS volumes using the AWS CLI as well. Let's go ahead and create a new EBS volume using the AWS CLI. Type in the following command:

```
# aws ec2 create-volume \
--size 5 --region us-west-2 --availability-zone us-west-2a \
--volume-type gp2
```

#### Note

The --volume-type attribute accepts any one of these three values:

**gp2:** General Purpose instances (SSD)

**io1:** Provisioned IOPS volumes (SSD)

**standard:** Magnetic volumes

The following code will create a 5 GB General Purpose volume in the us-west-2a availability zone.

The new volume will take a couple of minutes to be created. You should see a similar output as the following screenshot. Make a note of the new volume's Volume ID before proceeding to the next steps.

The screenshot shows a terminal window titled 'root@YoYoNUX ~' with the command 'aws ec2 create-volume' being run. The output shows the creation of a volume named 'vol-40993355' in the 'us-west-2a' availability zone. The volume has a size of 5 GiB, a volume type of gp2, and is currently in a 'creating' state.

```
[root@YoYoNUX ~]# aws ec2 create-volume \
> --size 5 --region us-west-2 --availability-zone us-west-2a \
> --volume-type gp2 \
> --profile jason

[...]
| CreateVolume
+-----+
| AvailabilityZone | us-west-2a
| CreateTime       | 2015-08-19T14:02:46.117Z
| Encrypted        | False
| Iops             | 15
| Size             | 5
| SnapshotId      |
| State            | creating
| VolumeId         | vol-40993355
| VolumeType       | gp2
+-----+
[root@YoYoNUX ~]#
```

Now that the new volume is created, we can go ahead and attach it to our instance. Type in the following command:

```
# aws ec2 attach-volume \
--volume-id vol-40993355 \
--instance-id i-53fc559a \
--device /dev/sdg
```

The following command will attach the volume with the volume ID vol-40993355 to our instance (i-53fc559a), and the device name will be /dev/sdg. Once again, you can supply any meaningful device name here, but make sure that it abides by AWS's naming conventions and best practices as mentioned in [http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device\\_naming.html](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device_naming.html).

Once the volume is attached to your instance, the next steps are pretty easy and straightforward. First format the new volume with a suitable filesystem of your choice. Next up, create a new directory inside your instance and mount the newly formatted volume on it. Voila! Your volume is now ready for use.

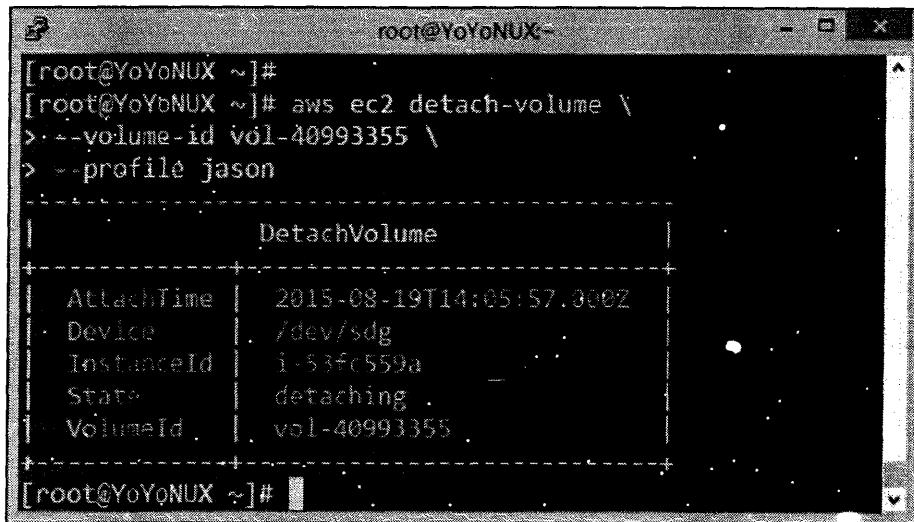
You can detach and delete the volume as well using the AWS CLI. First up, we will need to unmount the volume from the instance. To do so, type in the following command in your instance:

```
unmount /dev/sdg
```

Make sure you are unmounting the correct volume and not the root partition. Once the volume is unmounted, simply detach it from the instance using the following AWS CLI code:

```
aws ec2 detach-volume \  
--volume-id vol-40993355
```

The output of the preceding command is as follows:



A terminal window titled "root@YoYoNUX~" showing the command "aws ec2 detach-volume --volume-id vol-40993355 --profile jason". The output shows a table with the following data:

| DetachVolume |                          |
|--------------|--------------------------|
| AttachTime   | 2015-08-19T14:05:57.000Z |
| Device       | /dev/sdg                 |
| InstanceId   | i-53fc559a               |
| Status       | detaching                |
| VolumeId     | vol-40993355             |

Finally, go ahead and delete the volume using the following AWS CLI code:

```
# aws ec2 delete-volume \  
--volume-id vol-40993355
```

Remember that you cannot delete volumes if they are attached or in use by an instance, so make sure that you follow the detachment process before deleting it.

**Backing up volumes using EBS snapshots**

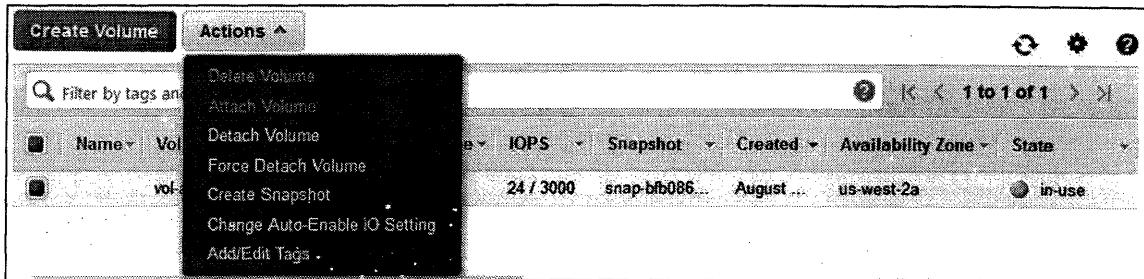
We do know for a fact that AWS automatically replicates EBS volumes so that your data is preserved even in case the complete drive fails. But this replication is limited only to the availability zone in which the drive or EBS volume was created, which means if that particular availability zone was to go down for some reason, then there is no way for you to back up your data. Fortunately for us, AWS provides a very simple yet highly efficient method of backing EBS volumes, called as EBS snapshots.

An EBS snapshot in simple terms is a state of your volume at a particular point in time. You can take a snapshot of a volume anytime you want. Each snapshot that you take is stored incrementally in Amazon S3, but, you will not be able to see these snapshots in your S3 buckets; they are kind of hidden away and stored separately.

You can achieve a wide variety of tasks using snapshots. A few are listed as follows:

- **Create new volumes based on existing ones:** Snapshots are a great and easy way to spin up new volumes. A new volume spawned from a snapshot is an exact replica of the original volume, down to the last detail.
- **Expand existing volumes:** Snapshots can also be used to expand an existing EBS Volume's size as well. It is a multistep process, which involves you taking a snapshot of your existing EBS volume and creating a larger new volume from the snapshot.
- **Share your volumes:** Snapshots can be shared within your own account (**private**) as well publicly.
- **Backup and disaster recovery:** Snapshots are a handy tool when it comes to backing up your volumes. You can create multiple replicates of an existing volume within an AZ, across AZs that belong to a particular region, as well as across regions, using something called an EBS Snapshot copy mechanism.

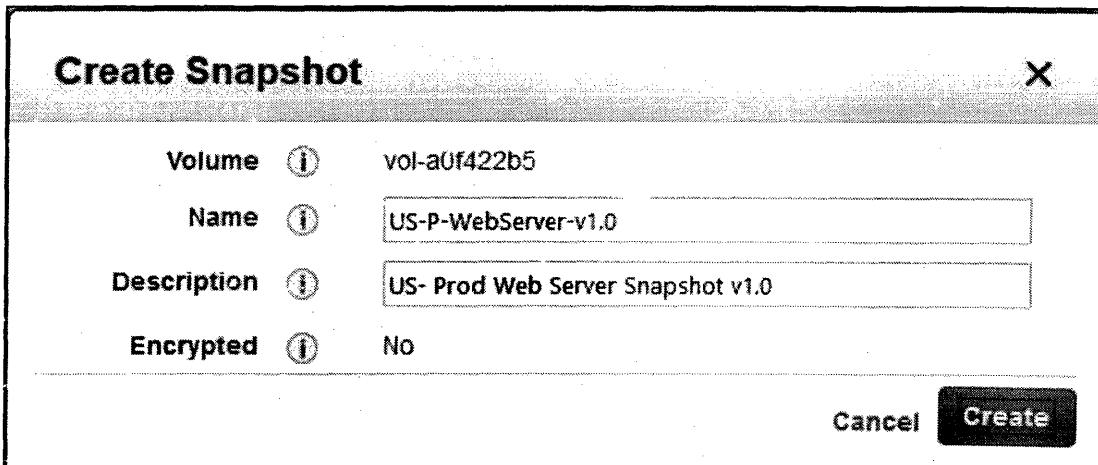
To create a snapshot of your volumes, all you need to do is select the particular **volume** from the **Volume Management dashboard**. Click on the **Actions** tab and select the **Create Snapshot** option, as shown here:



#### Tip

It is really a good practice to stop your instance before taking a snapshot if you are taking a snapshot of its root volume. This ensures a consistent and complete snapshot of your volume at all times.

You should see the **Create Snapshot** dialog box as shown in the following screenshot. Provide a suitable **Name** and **Description** for your new snapshot. An important thing to note here is that this particular snapshot is not supporting encryption, but why? Well, that's simple! Because the original volume was not encrypted, neither will the snapshot be encrypted. Snapshots of encrypted volumes are automatically encrypted. Even new volumes created from an encrypted snapshot are encrypted automatically. Once you have finished providing the details, click on **Create** to complete the snapshot process:



You will be shown a confirmation box, which will display this particular snapshot's ID. Make a note of this ID for future reference.

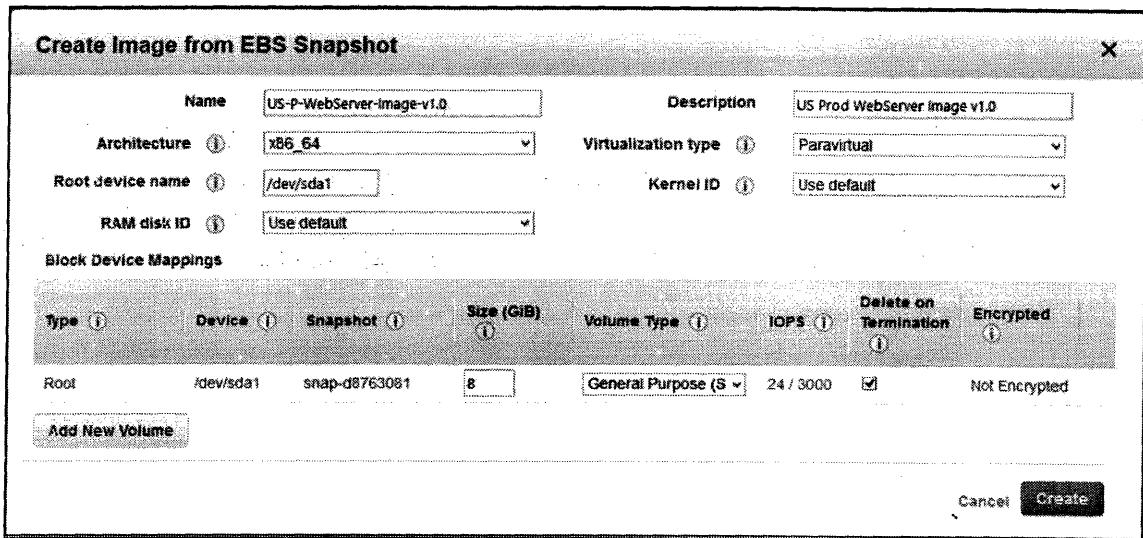
The new snapshot will take a good 3-4 minutes to go from **Pending** to **Completed**. You can check the status of your snapshot by viewing the **Status** as well as the **Progress** fields in the **Description** tab, as shown here:

The screenshot shows the AWS Snapshot Management interface. At the top, there are buttons for 'Create Snapshot' and 'Actions'. A dropdown menu shows 'Owned By Me'. A search bar says 'Filter by tags and attributes or search by keyword'. Below this is a table header with columns: Name, Snapshot ID, Size, Description, and Status. A single row is visible: 'US-P-WebS...' with Snapshot ID 'snap-d8763081', Size '8 GiB', Description 'US- Prod Web Server Snapshot v1.0', and Status 'completed'. Below the table, a section titled 'Snapshot: snap-d8763081 (US-P-WebServer-v1.0)' shows detailed information: Snapshot ID 'snap-d8763081', Progress '0%', Status 'pending', Capacity '8 GiB', Volume 'vol-a0f422b5', Encrypted 'Not Encrypted', Started 'August 19,' and KMS Key ID. There are also three small icons at the bottom right of this section.

Once the snapshot process is completed, you can use this particular snapshot and **Create Volume**, **Copy** this snapshot from one region to another, and **Modify Snapshot Permissions** to private or public as you see fit. These options are all present in the **Actions** tab of your **Snapshot Management** dashboard:

This screenshot is similar to the previous one, but the 'Actions' tab is currently selected. A dropdown menu is open under 'Actions' with several options: Delete, Create Volume, Create Image, Copy, Modify Snapshot Permissions, and Add/Edit Tags. The rest of the interface is identical to the first screenshot, showing the completed snapshot details.

But for now, let's go ahead and use this snapshot to create our very first AMI. Yes, you can use snapshots to create AMIs as well. From the **Actions** tab, select the **Create Image** option. You should see the **Create Image from EBS Snapshot** wizard as shown here. Fill in the required details and click on **Create** to create your very first AMI:

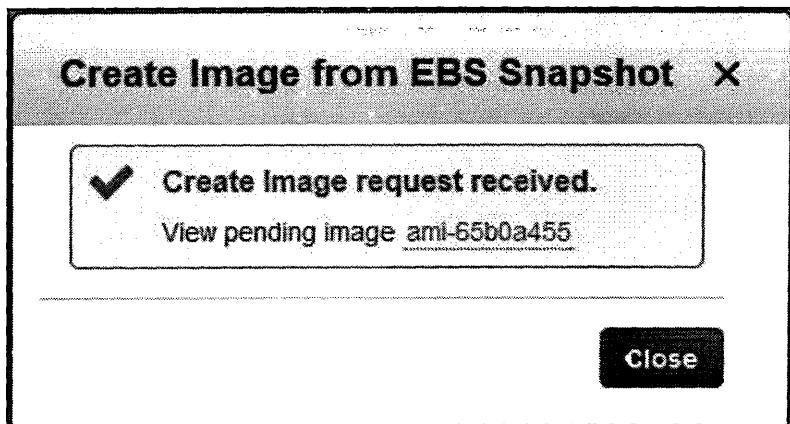


The details contain the following options:

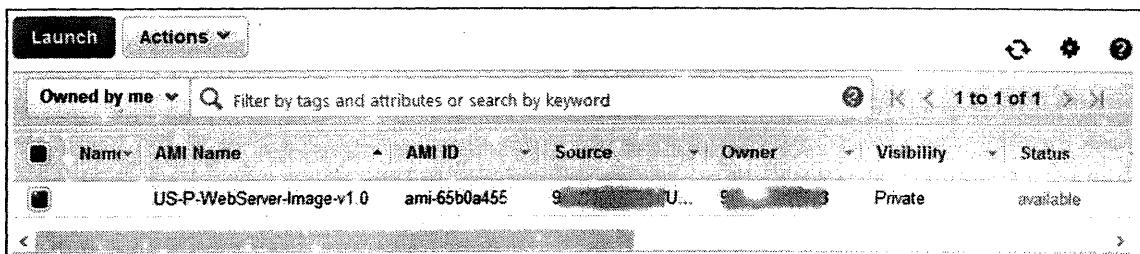
- **Name:** Provide a suitable and meaningful name for your AMI.
- **Description:** Provide a suitable description for your new AMI.
- **Architecture:** You can either choose between **i386** (32 bit) or **x86\_64** (64 bit).
- **Root device name:** Enter a suitable name for your root device volume. Ideally, a root device volume should be labelled as **/dev/sda1** as per EC2's device naming best practices.
- **Virtualization type:** You can choose whether the instances launched from this particular AMI will support **Paravirtualization (PV)** or **Hardware Virtual Machine (HVM)** virtualization.
- **RAM disk ID , Kernel ID:** You can select and provide your AMI with its own RAM disk ID (ARI) and Kernel ID (AKI); however, in this case I have opted to keep the default ones.

- **Block Device Mappings:** You can use this dialog to either expand your root volume's size or add additional volumes to it. You can change the **Volume Type** from **General Purpose (SSD)** to **Provisioned IOPS (SSD)** or **Magnetic** as per your AMI's requirements. For now, I have left these to their default values.

Click on **Create** to complete the AMI creation process. The new AMI will take a few minutes to spin up. In the meantime, you can make a note of the new AMI ID from the **Create Image from EBS Snapshot** confirmation box, as shown:



You can view your newly created AMI under the AMIs option from the EC2 dashboard's navigation pane:



So, here we have it! You very own AMI, created and ready to use.

