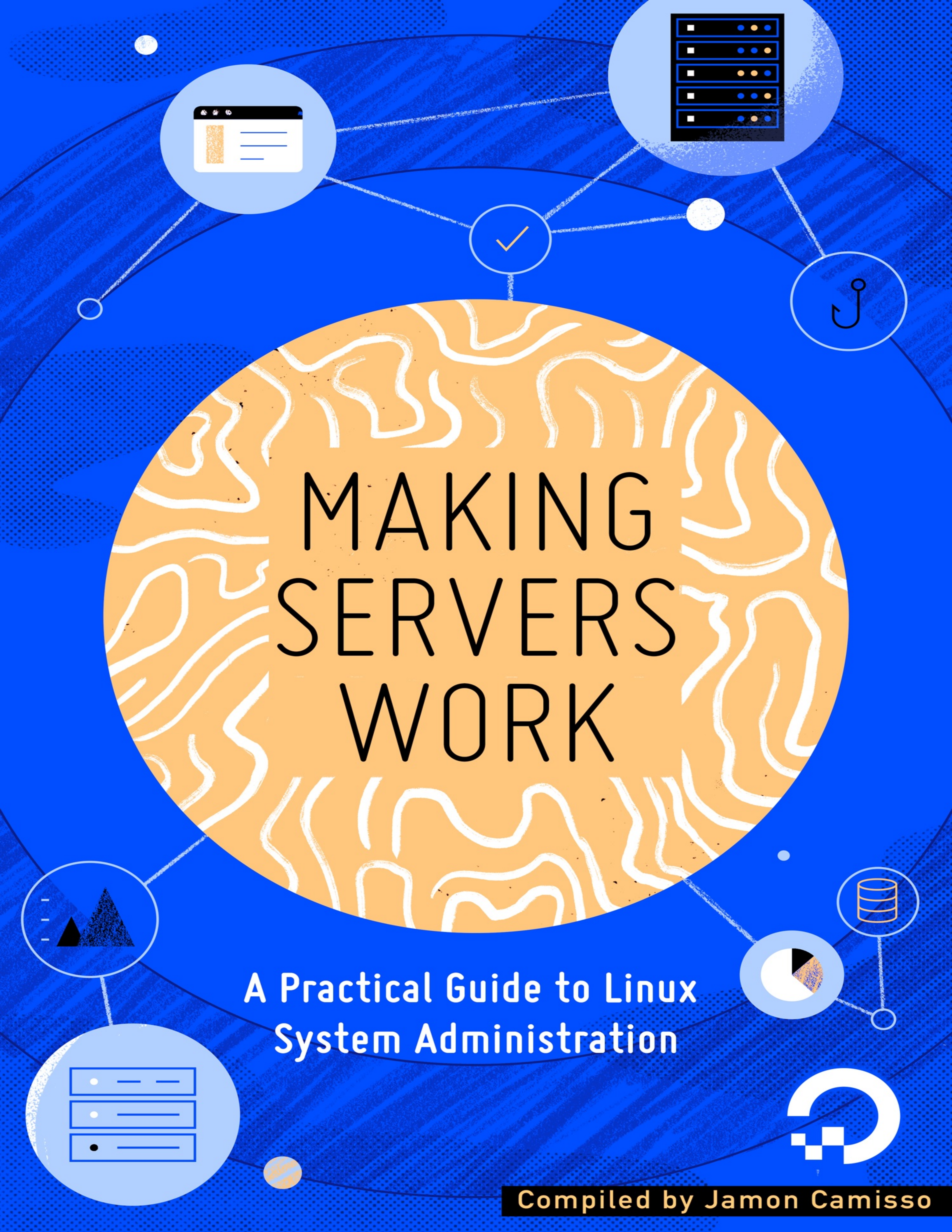


MAKING SERVERS WORK

A Practical Guide to Linux System Administration

Compiled by Jamon Camisso

ISBN 978-0-9997730-4-8

Making Servers Work: A Practical Guide to Linux System Administration

Compiled by Jamon Camisso

DigitalOcean, New York City, New York, USA

2020-03

Making Servers Work: A Practical Guide to Linux System Administration

1. About DigitalOcean
2. Preface - Getting Started with this Book
3. Introduction
4. An Introduction to the Linux Terminal
5. Basic Linux Navigation and File Management
6. An Introduction to Linux Permissions
7. An Introduction to Linux I/O Redirection
8. Initial Server Setup with Ubuntu 18.04
9. How to Add and Delete Users on Ubuntu 18.04
10. How To Install the Apache Web Server on Ubuntu 18.04
11. How To Install Nginx on Ubuntu 18.04
12. How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04
13. How To Install Linux, Nginx, MySQL, PHP (LEMP stack) on Ubuntu 18.04
14. How To Secure Apache with Let’s Encrypt on Ubuntu 18.04
15. How To Secure Nginx with Let’s Encrypt on Ubuntu 18.04
16. How To Set Up a Firewall with UFW on Ubuntu 18.04
17. How to Use Ansible to Automate Initial Server Setup on Ubuntu 18.04
18. How to Use Ansible to Install and Set Up LAMP on Ubuntu 18.04

19. How to Use Ansible to Install and Set Up LEMP on Ubuntu 18.04
20. How To Acquire a Let’s Encrypt Certificate Using Ansible on Ubuntu 18.04
21. How To Install Git on Ubuntu 18.04
22. How To Use Git Effectively
23. How To Install Jenkins on Ubuntu 18.04
24. How To Configure Jenkins with SSL Using an Nginx Reverse Proxy on Ubuntu 18.04

About DigitalOcean

DigitalOcean is a cloud services platform delivering the simplicity developers love and businesses trust to run production applications at scale. It provides highly available, secure and scalable compute, storage and networking solutions that help developers build great software faster.

Founded in 2012 with offices in New York and Cambridge, MA, DigitalOcean offers transparent and affordable pricing, an elegant user interface, and one of the largest libraries of open source resources available. For more information, please visit https://www.digitalocean.com or follow @digitalocean on Twitter.

Preface - Getting Started with this Book

We recommend that you begin with a clean, new server to start learning about system administration. However, the examples in this book will work with any up-to-date system running Ubuntu or Debian, from a laptop to a remote server running in a cloud provider's environment.

Chapter 1 of this book goes into detail about how to use a terminal to connect to and administer a Linux server, but it will be helpful to prepare in advance and ensure that you can connect to your new server. To connect to your new server with a terminal, use one of these guides based on your computer's operating system.

- Linux and macOS users: How to Connect to Droplets with SSH
- Windows users: If you have Bash on Windows or Windows Subsystem for Linux, you can use the guide above. Otherwise you can use PuTTY on Windows to connect to your Ubuntu server.

Once you have connected to your server, everything should be ready for you to start following along with the examples in this book.

Introduction

Why Learn About System Administration

Many system administrators do not set out to become system administrators. Instead, they learn system administration through experience in other areas like support or development and grow into the role over time. Often, the boundaries are blurred between system administration, network administration, engineering, DevOps, security, and support. In a small start up environment or research lab, a sysadmin may even occupy all of these roles at once, in addition to their actual role of founder or researcher. At home, sysadmin tasks can mean supporting family and friends with various devices: phones, tablets, smart devices, e-readers, networking equipment, and computers to name a few.

This experiential approach to learning about system administration is practical, but can also be limiting due to a lack of broad and diverse exposure to different tools, approaches to technical tasks and techniques. A sysadmin may become an expert at managing web servers and databases through experience, but may be unfamiliar with how to automate backups, or how to deploy automated configuration management tools to deploy applications at scale. Every computing environment and organization is different, and without experience across various tools, architectures, configurations, and automation methods, there can be gaps in a sysadmin's knowledge that more focused and deliberate learning can address.

This book is written with the belief that familiarity with some core areas of system administration will benefit anyone who uses computers, from individuals at home who would like to learn how to automate their systems and day to day tasks, to teams running thousands of servers in datacenters. Computers are complex systems, and complex systems need administering to ensure they are reliable, perform well over time, and behave as expected for users. Learning about sysadmin and understanding how computers and operating systems work behind the scenes is a great way to accomplish those goals.

Everyone from a curious beginner just starting out with some system administration tasks to the most seasoned sysadmin can always add to and refine their knowledge of system administration. Tools, methods, requirements, and networks are always changing, which is what makes system administration so interesting, challenging, and rewarding — there's always something new to learn about.

Motivation for this Book

This book is written to fill a gap in the existing literature about system administration. Many existing resources focus solely on theory, and others focus too specifically on the nuances of system utilities and services. Anyone who is interested in technology can benefit from learning about system administration. This book intends to highlight practical sysadmin skills, common architectures that you'll encounter, and best practices that apply to automating and running systems at any scale, from one laptop or server to 1,000 or more.

Learning Goals and Outcomes

The goal of this book is not to make you an expert sysadmin. That level of expertise can only come with time, practice, and familiarity with many varied systems. Instead, our goals are more modest: to familiarize you with the fundamentals of system administration; to highlight best practices that apply to one or many servers; and to provide a reference for future areas of focus as you develop your system administration skills.

In terms of concrete learning outcomes, this book is structured to help you progress from learning how to connect to a Linux server, to automating your servers with Ansible, all the way to using Git and Continuous Integration to manage deployed software on your servers.

In the first section of the book, you will start by learning how to access and modify users, data, and configuration on existing servers. Once you are familiar with how to access servers and manage users, you will learn how to install and configure the popular web servers Apache and Nginx that you are likely to encounter as a system administrator.

When you are confident with installing Apache or Nginx, the next section of the book will guide you through adding MySQL database and language support for PHP to each web server. These combinations of Linux, Apache, MySQL, and PHP (LAMP), or Linux, Nginx, MySQL, and PHP (LEMP) are very common, so familiarity with both will be very useful.

Once you have a working LAMP or LEMP server setup, it is important to learn how to secure it. In the third section of this book, you will learn about firewalls and how to configure the UFW firewall tool to restrict access to your Linux servers, ensuring that only traffic directed to Apache or Nginx is allowed. After creating firewall rules, you will learn how to add Transport Layer Security (TLS) certificates to your web server of choice. TLS certificates are important for every system administrator to understand and configure, since they are used to encrypt traffic to and from web, mail, database, VPN, and other types of servers.

The fourth section of this book will guide you through automating all the server set up steps from Sections 1 and 2. Instead of manually installing packages, editing configuration files, and adding firewall rules, the chapters in this section will demonstrate how to automate all these steps using Ansible. There are chapters that explain how to automate LAMP and LEMP stacks respectively. After you have automated either stack, there is a chapter on using Letsencrypt with Ansible to secure both kinds of servers.

The last section of the curriculum will familiarize you with how to use Git to version control server files and manage application code. Finally, when you are comfortable using Git, the last chapters explain how to set up Jenkins with Nginx and TLS for Continuous Integration (CI) so that you can automate building and deploying code to your servers.

Feel free to pick topics in this book that interest you and explore them using these chapters as guides. Working through this book will expose you to a wide variety of technologies, technical terms, and conceptual approaches to managing Linux servers. You can work through each chapter or section at your own pace, and in any order that you choose.

For example, if you are familiar with building a LAMP based server, but haven’t used Nginx before, then maybe try creating a new LEMP server to learn about it. If you have used Jenkins for continuous integration before, try automating the process of installing and configuring it with Ansible, using the chapters here as a guide. When you feel confident that you understand a concept or process to configure a server a certain way, you can move on to a new set of chapters, or continue learning and experimenting with automation.

After focusing on the fundamentals through this book, we hope that you will continue to explore more resources to support you in achieving your sysadmin goals. Once you finish this book, be sure to visit the DigitalOcean Community site site for more free tutorials written by sysadmins and developers, and an active community who can help answer questions as you continue to learn.

An Introduction to the Linux Terminal

Written by Mitchell Anicas

In this chapter you will learn about how to interact with a Linux system using commands and a terminal emulation program. This chapter explains terminal options for Windows, macOS, and Linux so you will be able to use any operating system to interact with a Linux server.

After explaining what terminals are, this chapter discusses the shell environment with a focus on the Bourne-Again shell (usually referred to as `bash`).

From there, you will learn about the command prompt on a remote server. Specifically, how it is structured to provide you information about the remote server, and how it lets you enter commands. You'll also learn about tools like `ls` and how arguments to command line programs work.

Finally, this chapter explores environment variables, and how you can set them, and use them in your command prompt to do things like add references to installed applications so that you can invoke them using the command line.

This tutorial, which is the first in a series that teaches Linux basics to get new users on their feet, covers getting started with the terminal, the Linux command line, and executing commands. If you are new to Linux, you will want to familiarize yourself with the terminal, as it is the standard way to interact with a Linux server. Using the command line may seem like a daunting task but it is actually very easy if you start with the basics, and build your skills from there.

If you would like to get the most out of this tutorial, you will need a Linux server to connect to and use. If you do not already have one, you can quickly spin one up by following this link: How To Create A DigitalOcean Droplet. This tutorial is based on an Ubuntu 14.04 server but the general principles apply to any other distribution of Linux.

Let's get started by going over what a terminal emulator is.

Terminal Emulator

A terminal emulator is a program that allows the use of the terminal in a graphical environment. As most people use an OS with a graphical user interface (GUI) for their day-to-day computer needs, the use of a terminal emulator is a necessity for most Linux server users.

Here are some free, commonly-used terminal emulators by operating system:

- Mac OS X: Terminal (default), iTerm 2
- Windows: PuTTY
- Linux: Terminal, KDE Konsole, XTerm

Each terminal emulator has its own set of features, but all of the listed ones work great and are easy to use.

The Shell

In a Linux system, the shell is a command-line interface that interprets a user's commands and script files, and tells the server's operating system what to do with them. There are several shells that are widely used, such as Bourne shell (`sh`) and C shell (`csh`). Each shell has its own feature set and intricacies, regarding how commands are interpreted, but they all feature input and output redirection, variables, and condition-testing, among other things.

This tutorial was written using the Bourne-Again shell, usually referred to as `bash`, which is the default shell for most Linux distributions, including Ubuntu, CentOS, and RedHat.

The Command Prompt

When you first login to a server, you will typically be greeted by the Message of the Day (MOTD), which is typically an informational message that includes miscellaneous information such as the version of the Linux distribution that the server is running. After the MOTD, you will be dropped into the command prompt, or shell prompt, which is where you can issue commands to the server.

The information that is presented at the command prompt can be customized by the user, but here is an example of the default Ubuntu 14.04 command prompt:

```
sammy@webapp:~$
```

Here is a breakdown of the composition of the command prompt:

- `sammy`: The username of the current user
- `webapp`: The hostname of the server
- `~`: The current directory. In `bash`, which is the default shell, the `~`, or tilde, is a special character that expands to the path of the current user's home directory; in this case, it represents `/home/sammy`
- `$`: The prompt symbol. This denotes the end of the command prompt, after which the user's keyboard input will appear

Here is an example of what the command prompt might look like, if logged in as `root` and in the `/var/log` directory:

```
root@webapp:/var/log#
```

Note that the symbol that ends the command prompt is a `#`, which is the standard prompt symbol for `root`. In Linux, the `root` user is the superuser account, which is a special user account that can perform system-wide administrative functions–it is an unrestricted user that has permission to perform any task on a server.

Executing Commands

Commands can be issued at the command prompt by specifying the name of an executable file, which can be a binary program or a script. There are many standard Linux commands and utilities that are installed with the OS, that allow you navigate the file system, install and software packages, and configure the system and applications.

An instance of a running command is known as a process. When a command is executed in the foreground, which is the default way that commands are executed, the user must wait for the process to finish before being returned to the command prompt, at which point they can continue issuing more commands.

It is important to note that almost everything in Linux is case-sensitive, including file and directory names, commands, arguments, and options. If something is not working as expected, double-check the spelling and case of your commands!

We will run through a few examples that will cover the basics of executing commands.

Note: If you're not already connected to a Linux server, now is a good time to log in. If you have a Linux server but are having trouble connecting, follow this link: How to Connect to Your Droplet with SSH.

Without Arguments or Options

To execute a command without any arguments or options, simply type in the name of the command and hit `RETURN`.

If you run a command like this, it will exhibit its default behavior, which varies from command to command. For example, if you run the `cd` command without any arguments, you will be returned to your current user's home directory. The `ls` command will print a listing of the current directory's files and directories. The `ip` command without any arguments will print a message that shows you how to use the `ip` command.

Try running the `ls` command with no arguments to list the files and directories in your current directory (there may be none):

```
ls
```

With Arguments

Many commands accept arguments, or parameters, which can affect the behavior of a command. For example, the most common way to use the `cd` command is to pass it a single argument that specifies which directory to change to. For example, to change to the `/usr/bin` directory, where many standard commands are installed, you would issue this command:

```
cd /usr/bin
```

The `cd` component is the command, and the first argument `/usr/bin` follows the command. Note how your command prompt's current path has updated.

If you would like, try running the `ls` command to see the files that are in your new current directory.

```
ls
```

With Options

Most commands accept options, also known as flags or switches, that modify the behavior of the command. As they are special arguments, options follow a command, and are indicated by a single `-` character followed by one or more options, which are represented by individual upper- or lower-case letters. Additionally, some options start with `--`, followed by a single, multi-character (usually a descriptive word) option.

For a basic example of how options work, let's look at the `ls` command. Here are a couple of common options that come in handy when using `ls`:

- `-l`: print a "long listing", which includes extra details such as permissions, ownership, file sizes, and timestamps
- `-a`: list all of a directory's files, including hidden ones (that start with `.`)

To use the `-l` flag with `ls`, use this command:

```
ls -l
```

Note that the listing includes the same files as before, but with additional information about each file.

As mentioned earlier, options can often be grouped together. If you want to use the `-l` and `-a` option together, you could run `ls -l -a`, or just combine them like in this command:

```
ls -la
```

Note that the listing includes the hidden . and .. directories in the listing, because of the -a option.

With Options and Arguments

Options and arguments can almost always be combined, when running commands.

For example, you could check the contents of /home, regardless of your current directory, by running this ls command:

```
ls -la /home
```

ls is the command, -la are the options, and /home is the argument that indicates which file or directory to list. This should print a detailed listing of the /home directory, which should contain the home directories of all of the normal users on the server.

Environment Variables

Environment variables are named values that are used to change how commands and processes are executed. When you first log in to a server, several environment variables will be set according to a few configuration files by default.

View All Environment Variables

To view all of the environment variables that are set for a particular terminal session, run the env command:

```
env
```

There will likely be a lot of output, but try and look for PATH entry:

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr
/bin:/sbin:/bin:/usr/games:/usr/local/games
```

The PATH environment variable is a colon-delimited list of directories where the shell will look for executable programs or scripts when a command is issued. For example, the env command is located in /usr/bin, and we are able to execute it without specifying its fully-qualified location because its path is in the PATH environment variable.

View the Value of a Variable

The value of an environment variable can be retrieved by prefixing the variable name with a $. Doing so will expand the referenced variable to its value.

For example, to print out the value of the PATH variable, you may use the echo command:

```
echo $PATH
```

Or you could use the HOME variable, which is set to your user's home directory by default, to change to your home directory like this:

```
cd $HOME
```

If you try to access an environment variable that hasn't been set, it will be expanded to nothing; an empty string.

Setting Environment Variables

Now that you know how to view your environment variables, you should learn how to set them.

To set an environment variable, all you need to do is start with a variable name, followed immediately by an = sign, followed immediately by its desired value:

```
VAR=value
```

Note that if you set an existing variable, the original value will be overwritten. If the variable did not exist in the first place, it will be created.

Bash includes a command called `export` which exports a variable so it will be inherited by child processes. In simple terms, this allows you to use scripts that reference an exported environment variable from your current session. If you're still unclear on what this means, don't worry about it for now.

You can also reference existing variables when setting a variable. For example, if you installed an application to `/opt/app/bin`, you could add that directory to the end of your `PATH` environment variable with this command:

```
export PATH=$PATH:/opt/app/bin
```

Now verify that `/opt/app/bin` has been added to the end of your `PATH` variable with `echo`:

```
echo $PATH
```

Keep in mind that setting environment variables in this way only sets them for your current session. This means if you log out or otherwise change to another session, the changes you made to the environment will not be preserved. There is a way to permanently change environment variables, but this will be covered in a later tutorial.

Conclusion

Now that you have learned about the basics of the Linux terminal (and a few commands), you should have a good foundation for expanding your knowledge of Linux commands. Read the next tutorial in this series to learn how to navigate, view, and edit files and their permissions.

Basic Linux Navigation and File Management

Written by Justin Ellingwood

This chapter will introduce you to the primary tools that you can use to navigate filesystems and manipulate files on a Linux server. You will learn about the shell prompt and how to interact with it by invoking commands and programs. The programs that are demonstrated in this chapter are some of the most important and commonly used command line tools on a Linux server.

To start off you will learn how to display where you are located in a filesystem using the `pwd` command. You will learn how to list the contents of a directory with the `ls` command, and then how to navigate between directories using the `cd` command.

After getting acquainted with how to navigate around a Linux system, you will learn how to create and view files using the `touch` and `less` commands respectively. You will also learn how to create and remove directories, and view and edit files.

If you do not have much experience working with Linux systems, you may be overwhelmed by the prospect of controlling an operating system from the command line. In this guide, we will attempt to get you up to speed with the basics.

This guide will not cover everything you need to know to effectively use a Linux system. However, it should give you a good jumping-off point for future exploration. This guide will give you the bare minimum you need to know before moving on to other guides.

Prerequisites and Goals

In order to follow along with this guide, you will need to have access to a Linux server. If you need information about connecting to your server for the first time, you can follow our guide on connecting to a Linux server using SSH.

You will also want to have a basic understanding of how the terminal works and what Linux commands look like. This guide covers terminal basics, so you should check it out if you are new to using terminals.

All of the material in this guide can be accomplished with a regular, non-root (non-administrative) user account. You can learn how to configure this type of user account by following your distribution's initial server setup guide (Ubuntu 14.04, CentOS 7).

When you are ready to begin, connect to your Linux server using SSH and continue below.

Navigation and Exploration

The most fundamental skills you need to master are moving around the filesystem and getting an idea of what is around you. We will discuss the tools that allow you to do this in this section.

Finding Where You Are with the "pwd" Command

When you log into your server, you are typically dropped into your user account's home directory. A home directory is a directory set aside for your user to store files and create directories. It is the location in the filesystem where you have full dominion.

To find out where your home directory is in relationship to the rest of the filesystem, you can use the `pwd` command. This command displays the directory that we are currently in:

```
pwd
```

You should get back some information that looks like this:

```
/home/demo
```

The home directory is named after the user account, so the above example is what the value would be if you were logged into the server with an account called **demo**. This directory is within a directory called `/home`, which is itself within the top-level directory, which is called "root" but represented by a single slash "/".

Looking at the Contents of Directories with "ls"

Now that you know how to display the directory that you are in, we can show you how to look at the contents of a directory.

Currently, your home directory that we saw above does not have much to see, so we will go to another, more populated directory to explore. Type the following in your terminal to move to this directory (we will explain the details of moving directories in the next section). Afterward, we'll use `pwd` to confirm that we successfully moved:

```
cd /usr/share
pwd
/usr/share
```

Now that we are in a new directory, let's look at what's inside. To do this, we can use the `ls` command:

```
ls
adduser            groff
pam-configs
applications       grub
perl
apport             grub-gfxpayload-lists
perl5
apps               hal
pixmaps
apt                i18n
pkgconfig
aptitude           icons
polkit-1
apt-xapian-index   info
popularity-contest
. . .
```

As you can see, there are many items in this directory. We can add some optional flags to the command to modify the default behavior. For instance, to list all of the contents in an extended form, we can use the `-l` flag (for “long” output):

```
ls -l
total 440
drwxr-xr-x   2 root root  4096 Apr 17  2014
adduser
drwxr-xr-x   2 root root  4096 Sep 24 19:11
applications
drwxr-xr-x   6 root root  4096 Oct  9 18:16 apport
```

```
drwxr-xr-x   3 root root  4096 Apr 17  2014 apps
drwxr-xr-x   2 root root  4096 Oct  9 18:15 apt
drwxr-xr-x   2 root root  4096 Apr 17  2014
aptitude
drwxr-xr-x   4 root root  4096 Apr 17  2014 apt-
xapian-index
drwxr-xr-x   2 root root  4096 Apr 17  2014 awk
. . .
```

This view gives us plenty of information, most of which looks rather unusual. The first block describes the file type (if the first column is a “d” the item is a directory, if it is a “-”, it is a normal file) and permissions. Each subsequent column, separated by white space, describes the number of hard links, the owner, group owner, item size, last modification time, and the name of the item. We will describe some of these at another time, but for now, just know that you can view this information with the `-l` flag of `ls`.

To get a listing of all files, including hidden files and directories, you can add the `-a` flag. Since there are no real hidden files in the `/usr/share` directory, let’s go back to our home directory and try that command. You can get back to the home directory by typing `cd` with no arguments:

```
cd
ls -a
.  ..  .bash_logout  .bashrc  .profile
```

As you can see, there are three hidden files in this demonstration, along with `.` and `..`, which are special indicators. You will find that often, configuration files are stored as hidden files, as is the case here.

For the dot and double dot entries, these aren't exactly directories as much as built-in methods of referring to related directories. The single dot indicates the current directory, and the double dot indicates this directory's parent directory. This will come in handy in the next section.

Moving Around the Filesystem with "cd"

We have already made two directory moves in order to demonstrate some properties of `ls` in the last section. Let's take a better look at the command here.

Begin by going back to the `/usr/share` directory by typing this:

```
cd /usr/share
```

This is an example of changing a directory by giving an absolute path. In Linux, every file and directory is under the top-most directory, which is called the "root" directory, but referred to by a single leading slash "/". An absolute path indicates the location of a directory in relation to this top-level directory. This lets us refer to directories in an unambiguous way from any place in the filesystem. Every absolute path must begin with a slash.

The alternative is to use relative paths. Relative paths refer to directories in relation to the current directory. For directories close to the current directory in the hierarchy, this is usually easier and shorter. Any directory within the current directory can be referenced by name without a leading slash. We can change to the `locale` directory within `/usr/share` from our current location by typing:

```
cd locale
```

We can likewise move multiple directory levels with relative paths by providing the portion of the path that comes after the current directory's path. From here, we can get to the `LC_MESSAGES` directory within the `en` directory by typing:

```
cd en/LC_MESSAGES
```

To go back up, travelling to the parent of the current directory, we use the special double dot indicator we talked about earlier. For instance, we are now in the `/usr/share/locale/en/LC_MESSAGES` directory. To move up one level, we can type:

```
cd ..
```

This takes us to the `/usr/share/locale/en` directory.

A shortcut that you saw earlier that will always take you back to your home directory is to use `cd` without providing a directory:

```
cd
pwd
/home/demo
```

To learn more about how to use these three commands, you can check out our guide on exploring the Linux filesystem.

Viewing Files

In the last section, we learned a bit about how to navigate the filesystem. You probably saw some files when using the `ls` command in various directories. In this section, we'll discuss different ways that you can use to view files. In contrast to some operating systems, Linux and other Unix-like operating systems rely on plain text files for vast portions of the system.

The main way that we will view files is with the `less` command. This is what we call a “pager”, because it allows us to scroll through pages of a file. While the previous commands immediately executed and returned you to the command line, `less` is an application that will continue to run and occupy the screen until you exit.

We will open the `/etc/services` file, which is a configuration file that contains service information that the system knows about:

```
less /etc/services
```

The file will be opened in `less`, allowing you to see the portion of the document that fits in the area of the terminal window:

```
# Network services, Internet style
#
# Note that it is presently the policy of IANA to
assign a single well-known
# port number for both TCP and UDP; hence,
officially ports have two entries
# even if the protocol doesn't support UDP
operations.
#
# Updated from
http://www.iana.org/assignments/port-numbers and
other
# sources like
http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/serv
ices .
# New ports will be added on request if they have
been officially assigned
# by IANA and used in the real-world or are needed
by a debian package.
```

```
# If you need a huge list of used numbers please
install the nmap package.

tcpmux          1/tcp                          #
TCP port service multiplexer
echo            7/tcp
. . .
```

To scroll, you can use the up and down arrow keys on your keyboard. To page down one whole screens-worth of information, you can use either the space bar, the “Page Down” button on your keyboard, or the `CTRL-f` shortcut.

To scroll back up, you can use either the “Page Up” button, or the `CTRL-b` keyboard shortcut.

To search for some text in the document, you can type a forward slash “/” followed by the search term. For instance, to search for “mail”, we would type:

```
/mail
```

This will search forward through the document and stop at the first result. To get to another result, you can type the lower-case `n` key:

```
n
```

To move backwards to the previous result, use a capital `N` instead:

```
N
```

When you wish to exit the `less` program, you can type `q` to quit:

```
q
```

While we focused on the `less` tool in this section, there are many other ways of viewing a file that come in handy in certain circumstances. The `cat` command displays a file’s contents and returns you to the prompt immediately. The `head` command, by default, shows the first 10 lines of a file. Likewise, the `tail` command shows the last 10 lines by default. These commands display file contents in a way that is useful for “piping” to other programs. We will discuss this concept in a future guide.

Feel free to see how these commands display the `/etc/services` file differently.

File and Directory Manipulation

We learned in the last section how to view a file. In this section, we’ll demonstrate how to create and manipulate files and directories.

Create a File with “touch”

Many commands and programs can create files. The most basic method of creating a file is with the `touch` command. This will create an empty file using the name and location specified.

First, we should make sure we are in our home directory, since this is a location where we have permission to save files. Then, we can create a file called `file1` by typing:

```
cd
touch file1
```

Now, if we view the files in our directory, we can see our newly created file:

```
ls
file1
```

If we use this command on an existing file, the command simply updates the data our filesystem stores on the time when the file was last accessed and modified. This won’t have much use for us at the moment.

We can also create multiple files at the same time. We can use absolute paths as well. For instance, if our user account is called demo, we could type:

```
touch /home/demo/file2 /home/demo/file3
ls
file1  file2  file3
```

Create a Directory with "mkdir"

Similar to the touch command, the mkdir command allows us to create empty directories.

For instance, to create a directory within our home directory called test, we could type:

```
cd
mkdir test
```

We can make a directory within the test directory called example by typing:

```
mkdir test/example
```

For the above command to work, the test directory must already exist. To tell mkdir that it should create any directories necessary to construct a given directory path, you can use the -p option. This allows you to create nested directories in one step. We can create a directory structure that looks like some/other/directories by typing:

```
mkdir -p some/other/directories
```

The command will make the some directory first, then it will create the other directory inside of that. Finally it will create the directories directory within those two directories.

Moving and Renaming Files and Directories with “mv”

We can move a file to a new location using the mv command. For instance, we can move file1 into the test directory by typing:

```
mv file1 test
```

For this command, we give all of the items that we wish to move, with the location to move them at the end. We can move that file back to our home directory by using the special dot reference to refer to our current directory. We should make sure we’re in our home directory, and then execute the command:

```
cd
mv test/file1 .
```

This may seem unintuitive at first, but the mv command is also used to rename files and directories. In essence, moving and renaming are both just adjusting the location and name for an existing item.

So to rename the test directory to testing, we could type:

```
mv test testing
```

Note: It is important to realize that your Linux system will not prevent you from certain destructive actions. If you are renaming a file and choose a name that already exists, the previous file will be overwritten by the file you are moving. There is no way to recover the previous file if you accidentally overwrite it.

Copying Files and Directories with “cp”

With the mv command, we could move or rename a file or directory, but we could not duplicate it. The cp command can make a new copy of an existing item.

For instance, we can copy `file3` to a new file called `file4`:

```
cp file3 file4
```

Unlike a `mv` operation, after which `file3` would no longer exist, we now have both `file3` and `file4`.

Note: As with the `mv` command, it is possible to overwrite a file if you are not careful about the filename you are using as the target of the operation. For instance, if `file4` already existed in the above example, its content would be completely replaced by the content of `file3`.

In order to copy directories, you must include the `-r` option to the command. This stands for "recursive", as it copies the directory, plus all of the directory's contents. This option is necessary with directories, regardless of whether the directory is empty.

For instance, to copy the `some` directory structure to a new structure called `again`, we could type:

```
cp -r some again
```

Unlike with files, with which an existing destination would lead to an overwrite, if the target is an existing directory, the file or directory is copied into the target:

```
cp file1 again
```

This will create a new copy of `file1` and place it inside of the `again` directory.

Removing Files and Directories with "rm" and "rmdir"

To delete a file, you can use the `rm` command.

Note: Be extremely careful when using any destructive command like `rm`. There is no "undo" command for these actions so it is possible to accidentally destroy important files permanently.

To remove a regular file, just pass it to the `rm` command:

```
cd
rm file4
```

Likewise, to remove empty directories, we can use the `rmdir` command. This will only succeed if there is nothing in the directory in question. For instance, to remove the `example` directory within the `testing` directory, we can type:

```
rmdir testing/example
```

If you wish to remove a non-empty directory, you will have to use the `rm` command again. This time, you will have to pass the `-r` option, which removes all of the directory's contents recursively, plus the directory itself.

For instance, to remove the `again` directory and everything within it, we can type:

```
rm -r again
```

Once again, it is worth reiterating that these are permanent actions. Be entirely sure that the command you typed is the one that you wish to execute.

Editing Files

Currently, we know how to manipulate files as objects, but we have not learned how to actually edit them and add content to them.

The `nano` command is one of the simplest command-line Linux text editors, and is a great starting point for beginners. It operates somewhat similarly to the `less` program discussed above, in that it occupies the entire terminal for the duration of its use.

The nano editor can open existing files, or create a file. If you decide to create a new file, you can give it a name when you call the nano editor, or later on, when you wish to save your content.

We can open the file1 file for editing by typing:

```
cd
nano file1
```

The nano application will open the file (which is currently blank). The interface looks something like this:

```
  GNU nano 2.2.6                   File: file1










                                     [ Read 0 lines ]
^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev
Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next
Page  ^U UnCut Text ^T To Spell
```

Along the top, we have the name of the application and the name of the file we are editing. In the middle, the content of the file, currently blank, is displayed. Along the bottom, we have a number of key combinations that indicate some basic controls for the editor. For each of these, the ^ character means the CTRL key.

To get help from within the editor, type:

```
CTRL-G
```

When you are finished browsing the help, type `CTRL-X` to get back to your document.

Type in or modify any text you would like. For this example, we'll just type these two sentences:

```
Hello there.


Here is some text.
```

To save our work, we can type:

```
CTRL-O
```

This is the letter "o", not a zero. It will ask you to confirm the name of the file you wish to save to:

```
File Name to Write: file1
^G Get Help          M-D DOS Format       M-A
Append           M-B Backup File
^C Cancel            M-M Mac Format       M-P
Prepend
```

As you can see, the options at the bottom have also changed. These are contextual, meaning they will change depending on what you are trying to do. If `file1` is still the file you wish to write to, hit "ENTER".

If we make some additional changes and wish to save the file and exit the program, we will see a similar prompt. Add a new line, and then try to exit the program by typing:

```
CTRL-X
```

If you have not saved after making your modification, you will be asked whether you wish to save the modifications you made:

```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
 Y Yes
 N No           ^C Cancel
```

You can type “Y” to save your changes, “N” to discard your changes and exit, or “CTRL-C” to cancel the exit operation. If you choose to save, you will be given the same file prompt that you received before, confirming that you want to save the changes to the same file. Press ENTER to save the file and exit the editor.

You can see the contents of the file you created using either the `cat` program to display the contents, or the `less` program to open the file for viewing. After viewing with `less`, remember that you should hit `q` to get back to the terminal.

```
less file1
Hello there.

Here is some text.

Another line.
```

Another editor that you may see referenced in certain guides is `vim` or `vi`. This is a more advanced editor that is very powerful, but comes with a very steep learning curve. If you are ever told to use `vim` or `vi`, feel free to use `nano` instead. If you wish to learn how to use `vim`, read our guide to getting started with vim.

Conclusion

By now, you should have a basic understanding of how to get around your Linux server and how to see the files and directories available. You should also know some basic file manipulation commands that will allow you to view, copy, move, or delete files. Finally, you should be comfortable with some basic editing using the `nano` text editor.

With these few skills, you should be able to continue on with other guides and learn how to get the most out of your server. In our next guide, we will discuss how to view and understand Linux permissions.

An Introduction to Linux Permissions

Written by Mitchell Anicas

In this chapter you will learn about users and groups, file ownership, and permissions. These three concepts act in tandem to ensure security on a Linux system. To begin you will learn about users and groups on a system, since ownership and permission apply to files based on a user or group role.

Once you are familiar with users and groups, the chapter explains how to view who owns a file, and determine what permissions it has (its mode). The way that permissions are displayed and configured in Linux may be unfamiliar to some, so the chapter explains how to understand file modes in some detail with examples of common modes that you are likely to encounter.

Linux is a multi-user OS that is based on the Unix concepts of file ownership and permissions to provide security at the file system level. If you are planning to improve your Linux skills, it is essential that you have a decent understanding of how ownership and permissions work. There are many intricacies when dealing with file ownership and permissions, but we will try our best to distill the concepts down to the details that are necessary for a foundational understanding of how they work.

In this tutorial, we will cover how to view and understand Linux ownership and permissions. If you are looking for a tutorial on how to modify permissions, check out this guide: Linux Permissions Basics and How to Use Umask on a VPS

Prerequisites

Make sure you understand the concepts covered in the prior tutorials in this series:

- An Introduction to the Linux Terminal
- Basic Linux Navigation and File Management

Access to a Linux server is not strictly necessary to follow this tutorial, but having one to use will let you get some first-hand experience. If you want to set one up, check out this link for help.

About Users

As mentioned in the introduction, Linux is a multi-user system. We must understand the basics of Linux users and groups before we can talk about ownership and permissions, because they are the entities that the ownership and permissions apply to. Let's get started with the basics of what users are.

In Linux, there are two types of users: system users and regular users. Traditionally, system users are used to run non-interactive or background processes on a system, while regular users used for logging in and running processes interactively. When you first log in to a Linux system, you may notice that it starts out with many system users that run the services that the OS depends on–this is completely normal.

An easy way to view all of the users on a system is to look at the contents of the `/etc/passwd` file. Each line in this file contains information about a single user, starting with its user name (the name before the first `:`). Print the `passwd` file with this command:

```
cat /etc/passwd
```

Superuser

In addition to the two user types, there is the superuser, or root user, that has the ability to override any file ownership and permission restrictions. In practice, this means that the superuser has the rights to access anything on its own server. This user is used to make system-wide changes, and must be kept secure.

It is also possible to configure other user accounts with the ability to assume “superuser rights”. In fact, creating a normal user that has `sudo` privileges for system administration tasks is considered to be best practice.

About Groups

Groups are collections of zero or more users. A user belongs to a default group, and can also be a member of any of the other groups on a server.

An easy way to view all the groups and their members is to look in the `/etc/group` file on a server. We won’t cover group management in this article, but you can run this command if you are curious about your groups:

```
cat /etc/group
```

Now that you know what users and groups are, let’s talk about file ownership and permissions!

Viewing Ownership and Permissions

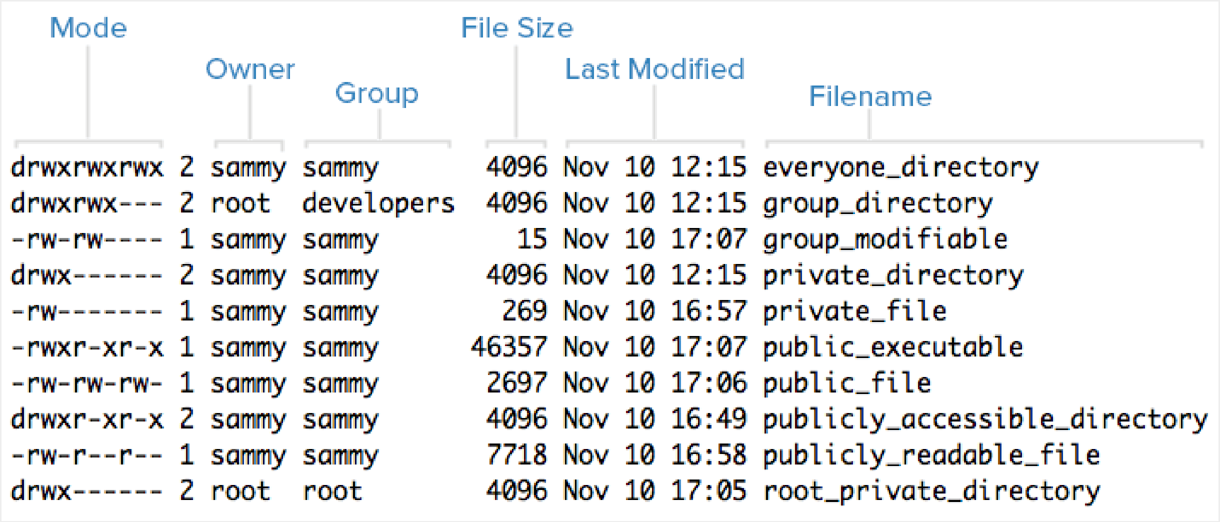
In Linux, each and every file is owned by a single user and a single group, and has its own access permissions. Let’s look at how to view the ownership and permissions of a file.

The most common way to view the permissions of a file is to use `ls` with the long listing option, e.g. `ls -l myfile`. If you want to view the permissions of all of the files in your current directory, run the command without an argument, like this:

```
ls -l
```

Hint: If you are in an empty home directory, and you haven’t created any files to view yet, you can follow along by listing the contents of the `/etc` directory by running this command: `ls -l /etc`

Here is an example screenshot of what the output might look like, with labels of each column of output:



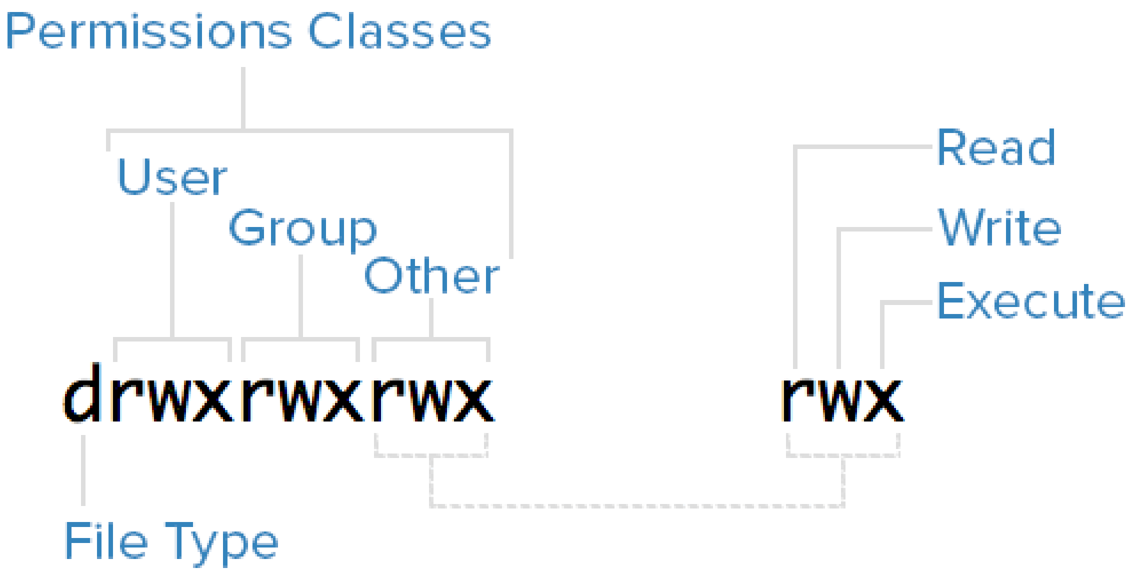
```
drwxrwxrwx 2 sammy sammy      4096 Nov 10 12:15 everyone_directory
drwxrwx--- 2 root  developers 4096 Nov 10 12:15 group_directory
-rw-rw---- 1 sammy sammy        15 Nov 10 17:07 group_modifiable
drwx------ 2 sammy sammy      4096 Nov 10 12:15 private_directory
-rw------- 1 sammy sammy       269 Nov 10 16:57 private_file
-rwxr-xr-x 1 sammy sammy     46357 Nov 10 17:07 public_executable
-rw-rw-rw- 1 sammy sammy      2697 Nov 10 17:06 public_file
drwxr-xr-x 2 sammy sammy      4096 Nov 10 16:49 publicly_accessible_directory
-rw-r--r-- 1 sammy sammy      7718 Nov 10 16:58 publicly_readable_file
drwx------ 2 root  root       4096 Nov 10 17:05 root_private_directory
```

ls -l

Note that each file’s mode (which contains permissions), owner, group, and name are listed. Aside from the Mode column, this listing is fairly easy to understand. To help explain what all of those letters and hyphens mean, let's break down the Mode column into its components.

Understanding Mode

To help explain what all the groupings and letters mean, take a look at this closeup of the mode of the first file in the example above:



Mode and permissions breakdown

File Type

In Linux, there are two basic types of files: normal and special. The file type is indicated by the first character of the mode of a file–in this guide, we refer to this as the file type field.

Normal files can be identified by files with a hyphen (−) in their file type fields. Normal files are just plain files that can contain data. They are called normal, or regular, files to distinguish them from special files.

Special files can be identified by files that have a non-hyphen character, such as a letter, in their file type fields, and are handled by the OS differently than normal files. The character that appears in the file type field indicates the kind of special file a particular file is. For example, a directory, which is the most common kind of special file, is identified by the `d` character that appears in its file type field (like in the previous screenshot). There are several other kinds of special files but they are not essential what we are learning here.

Permissions Classes

From the diagram, we know that Mode column indicates the file type, followed by three triads, or classes, of permissions: user (owner), group, and other. The order of the classes is consistent across all Linux distributions.

Let's look at which users belong to each permissions class:

- User: The owner of a file belongs to this class
- Group: The members of the file's group belong to this class
- Other: Any users that are not part of the user or group classes belong to this class.

Reading Symbolic Permissions

The next thing to pay attention to are the sets of three characters, or triads, as they denote the permissions, in symbolic form, that each class has for a given file.

In each triad, read, write, and execute permissions are represented in the following way:

- Read: Indicated by an `r` in the first position
- Write: Indicated by a `w` in the second position
- Execute: Indicated by an `x` in the third position. In some special cases, there may be a different character here

A hyphen (`-`) in the place of one of these characters indicates that the respective permission is not available for the respective class. For example, if the group triad for a file is `r--`, the file is “read-only” to the group that is associated with the file.

Understanding Read, Write, Execute

Now that you know how to read which permissions of a file, you probably want to know what each of the permissions actually allow users to do. We will explain each permission individually, but keep in mind that they are often used in combination with each other to allow for meaningful access to files and directories.

Here is a quick breakdown of the access that the three basic permission types grant a user.

Read

For a normal file, read permission allows a user to view the contents of the file.

For a directory, read permission allows a user to view the names of the file in the directory.

Write

For a normal file, write permission allows a user to modify and delete the file.

For a directory, write permission allows a user to delete the directory, modify its contents (create, delete, and rename files in it), and modify the contents of files that the user can read.

Execute

For a normal file, execute permission allows a user to execute a file (the user must also have read permission). As such, execute permissions must be set for executable programs and shell scripts before a user can run them.

For a directory, execute permission allows a user to access, or traverse, into (i.e. `cd`) and access metadata about files in the directory (the information that is listed in an `ls -l`).

Examples of Modes (and Permissions)

Now that know how to read the mode of a file, and understand the meaning of each permission, we will present a few examples of common modes, with brief explanations, to bring the concepts together.

- `-rw-------`: A file that is only accessible by its owner
- `-rwxr-xr-x`: A file that is executable by every user on the system. A “world-executable” file
- `-rw-rw-rw-`: A file that is open to modification by every user on the system. A “world-writable” file
- `drwxr-xr-x`: A directory that every user on the system can read and access

- `drwxrwx---`: A directory that is modifiable (including its contents) by its owner and group
- `drwxr-x---`: A directory that is accessible by its group

As you may have noticed, the owner of a file usually enjoys the most permissions, when compared to the other two classes. Typically, you will see that the group and other classes only have a subset of the owner's permissions (equivalent or less). This makes sense because files should only be accessible to users who need access to them for a particular reason.

Another thing to note is that even though many permissions combinations are possible, only certain ones make sense in most situations. For example, write or execute access is almost always accompanied by read access, since it's hard to modify, and impossible to execute, something you can't read.

Modifying Ownership and Permissions

To keep this tutorial simple, we will not cover how to modify file ownership and permissions here. To learn how to use `chown`, `chgrp`, and `chmod` to accomplish these tasks, refer to this guide: Linux Permissions Basics and How to Use Umask on a VPS.

Conclusion

You should now have a good understanding of how ownership and permissions work in Linux. If you would like to learn more about Linux basics, it is highly recommended that you read the next tutorial in this series:

- An Introduction to Linux I/O Redirection

An Introduction to Linux I/O Redirection

Written by David Collazo

This chapter will introduce you to input and output streams on a Linux system. You will learn about the three standard stream systems: Standard Input, Standard Output, and Standard Error. You will also learn how to interact and redirect the contents of a stream to or from another stream or file. Once you are familiar with how streams work, you will learn how to use pipes and filters with stream data in order to process input or output using various command line tools.

The redirection capabilities built into Linux provide you with a robust set of tools used to make all sorts of tasks easier to accomplish. Whether you're writing complex software or performing file management through the command line, knowing how to manipulate the different I/O streams in your environment will greatly increase your productivity.

Streams

Input and output in the Linux environment is distributed across three streams. These streams are:

- standard input (stdin)
- standard output (stdout)
- standard error (stderr)

The streams are also numbered:

- stdin (0)
- stdout (1)
- stderr (2)

During standard interactions between the user and the terminal, standard input is transmitted through the user's keyboard. Standard output and standard error are displayed on the user's terminal as text. Collectively, the three streams are referred to as the standard streams.

Standard Input

The standard input stream typically carries data from a user to a program. Programs that expect standard input usually receive input from a device, such as a keyboard. Standard input is terminated by reaching EOF (end-of-file). As described by its name, EOF indicates that there is no more data to be read.

To see standard input in action, run the cat program. Cat stands for concatenate, which means to link or combine something. It is commonly used to combine the contents of two files. When run on its own, cat opens a looping prompt.

```
cat
```

After opening cat, type a series of numbers as it is running.

```
1
2
3
ctrl-d
```

When you type a number and press enter, you are sending standard input to the running cat program, which is expecting said input. In turn, the cat program is sending your input back to the terminal display as standard output.

EOF can be input by the user by pressing ctrl-d. After the cat program receives EOF, it stops.

Standard Output

Standard output writes the data that is generated by a program. When the standard output stream is not redirected, it will output text to the terminal. Try the following example:

```
echo Sent to the terminal through standard output
```

When used without any additional options, the echo command displays any argument that is passed to it on the command line. An argument is something that is received by a program.

Run echo without any arguments:

```
echo
```

It will return an empty line, since there are no arguments.

Standard Error

Standard error writes the errors generated by a program that has failed at some point in its execution. Like standard output, the default destination for this stream is the terminal display.

When a program’s standard error stream is piped to a second program, the piped data (consisting of program errors) is simultaneously sent to the terminal as well.

Let’s see a basic example of standard error using the ls command. ls lists a directory’s contents.

When run without an argument, ls lists the contents within the current directory. If ls is run with a directory as an argument, it will list the contents of the provided directory.

```
ls %
```

Since % is not an existing directory, this will send the following text to standard error:

```
ls: cannot access %: No such file or directory
```

Stream Redirection

Linux includes redirection commands for each stream. These commands write standard output to a file. If a non-existent file is targetted (either by a single-bracket or double-bracket command), a new file with that name will be created prior to writing.

Commands with a single bracket overwrite the destination’s existing contents.

Overwrite

- > - standard output
- < - standard input
- 2> - standard error

Commands with a double bracket do not overwrite the destination’s existing contents.

Append

- >> - standard output
- << - standard input
- 2>> - standard error

Let’s see an example:

```
cat > write_to_me.txt
a
b
c
ctrl-d
```

Here, cat is being used to write to a file, which is created as a result of the loop.

View the contents of write_to_me.txt using cat:

```
cat write_to_me.txt
```

It should have the following contents:

```
a
b
c
```

Redirect cat to write_to_me.txt again, and enter three numbers.

```
cat > write_to_me.txt
1
2
3
ctrl-d
```

When you use cat to view write_to_me.txt, you will see the following:

```
1
2
3
```

The prior contents are no longer there, as the file was overwritten by the single-bracket command.

Do one more cat redirection, this time using double brackets:

```
cat >> write_to_me.txt
a
b
c
ctrl-d
```

Open write_to_me.txt again, and you will see this:

```
1
2
3
a
b
c
```

The file now contains text from both uses of cat, as the second one did not override the first one.

Pipes

Pipes are used to redirect a stream from one program to another. When a program’s standard output is sent to another through a pipe, the first program’s data, which is received by the second program, will not be displayed on the terminal. Only the filtered data returned by the second program will be displayed.

The Linux pipe is represented by a vertical bar.

```
*|*
```

An example of a command using a pipe:

```
ls | less
```

This takes the output of ls, which displays the contents of your current directory, and pipes it to the less program. less displays the data sent to it one line at a time.

ls normally displays directory contents across multiple rows. When you run it through less, each entry is placed on a new line.

Though the functionality of the pipe may appear to be similar to that of > and >> (standard output redirect), the distinction is that pipes redirect data from one command to another, while > and >> are used to redirect exclusively to files.

Filters

Filters are commands that alter piped redirection and output. Note that filter commands are also standard Linux commands that can be used without pipes.

- find - Find returns files with filenames that match the argument passed to find.
- grep - Grep returns text that matches the string pattern passed to grep.
- tee - Tee redirects standard input to both standard output and one or more files.
- tr - tr finds-and-replaces one string with another.
- wc - wc counts characters, lines, and words.

Examples

Now that you have been introduced to redirection, piping, and basic filters, let’s look at some basic redirection patterns and examples.

command > file

This pattern redirects the standard output of a command to a file.

```
ls ~ > root_dir_contents.txt
```

The command above passes the contents of your system’s root directory as standard output, and writes the output to a file named root_dir_contents.txt. It will delete any prior contents in the file, as it is a single-bracket command.

command > /dev/null

/dev/null is a special file that is used to trash any data that is redirected to it. It is used to discard standard output that is not needed, and that might otherwise interfere with the functionality of a command or a script. Any output that is sent to /dev/null is discarded. In the future, you may find the practice of redirecting standard output and standard error to /dev/null when writing shell scripts.

```
ls > /dev/null
```

This command discards the standard output stream returned from the command ls by passing it to /dev/null.

command 2> file

This pattern redirects the standard error stream of a command to a file, overwriting existing contents.

```
mkdir '' 2> mkdir_log.txt
```

This redirects the error raised by the invalid directory name ’’, and writes it to log.txt. Note that the error is still sent to the terminal and displayed as text.

command >> file

This pattern redirects the standard output of a command to a file without overwriting the file’s existing contents.

```
echo Written to a new file > data.txt
echo Appended to an existing file's contents >>
data.txt
```

This pair of commands first redirects the text inputted by the user through echo to a new file. It then appends the text received by the second echo command to the existing file, without overwriting its contents.

command 2>> file

The pattern above redirects the standard error stream of a command to a file without overwriting the file’s existing contents. This pattern is useful for creating error logs for a program or service, as the log file will not have its previous content wiped each time the file is written to.

```
find '' 2> stderr_log.txt
wc '' 2>> stderr_log.txt
```

The above command redirects the error message caused by an invalid find argument to a file named stderr_log.txt. It then appends the error message caused by an invalid wc argument to the same file.

command | command

Redirects the standard output from the first command to the standard input of the second command.

```
find /var lib | grep deb
```

This command searches through /var and its subfolders for filenames and extensions that match the string deb, and returns the file paths for the files, with the matching portion in each path highlighted in red.

command | tee file

This pattern (which includes the tee command) redirects the standard output of the command to a file and overwrites its contents. Then, it displays the redirected output in the terminal. It creates a new file if the file does not already exist.

In the context of this pattern, tee is typically used to view a program's output while simultaneously saving it to a file.

```
wc /etc/magic | tee magic_count.txt
```

This pipes the counts for characters, lines, and words in the magic file (used by the Linux shell to determine file types) to the tee command, which then splits wc's output in two directions, and sends it to the terminal display and the magic_count.txt file. For the tee command, imagine the letter T. The bottom part of the letter is the initial data, and the top part is the data being split in two different directions (standard output and the terminal).

Multiple pipes can be used to redirect output across multiple commands and/or filters.

command | command | command >> file

This pattern predirects the standard output of the first command and filters it through the next two commands. It then appends the final result to a file.

```
ls ~ | grep *tar | tr e E >> ls_log.txt
```

This begins by running ls in your root directory (~) and piping the result to the grep command. In this case, grep returns a list of files containing tar in their filename or extension.

The results from grep are then piped to tr, which replaces occurrences of the letter e with E, since e is being passed as the first argument (the string to search for), and E is passed as the second argument (the string that replaces any matches for the first argument). This final result is then appended to the file ls_log.txt, which is created if it does not already exist).

Conclusion

Learning how to use the redirection capabilities built into the Linux command line can be a bit daunting, but you are well on your way to mastering this skillset after completing this tutorial. Now that you have seen the basics of how redirections and pipes work, you'll be able to begin your foray into the world of shell scripting, which makes frequent use of the programs and patterns highlighted in this guide.

If you would like to dig deeper into the commands that were introduced in this tutorial, you can do so with man command | less. For example:

```
man tee | less
```

This will show you the full list of commands available for the tee program. You can use this pattern to display information and usage options for any Linux command or program.

Googling for specific commands, or for something that you would like to do in the command line (e.g. "delete all files in a directory that begin with an uppercase letter") can also prove helpful when you need to accomplish a specific task using the command line.

By David Collazo

Initial Server Setup with Ubuntu 18.04

Written by Justin Ellingwood

In this chapter you will learn about some of the common set up tasks that you should complete when you create a new Linux server. Specifically, you will learn how to login as the `root` user using SSH, and then create an unprivileged user that you can use for most tasks. After you create a new user, you will learn how to grant administrative privileges so that the user can run commands with elevated system permissions.

Once you have a new user with administrative access in place, you will learn how to set up a firewall to restrict traffic to specific services from the public Internet. Finally, you will learn how to ensure that any new unprivileged users that you create can use SSH to connect to a a server.

When you first create a new Ubuntu 18.04 server, there are a few configuration steps that you should take early on as part of the basic setup. This will increase the security and usability of your server and will give you a solid foundation for subsequent actions.

Note: The guide below demonstrates how to manually complete the steps we recommend for new Ubuntu 18.04 servers. Following this procedure manually can be useful to learn some basic system administration skills and as an exercise to fully understand the actions being taken on your server. As an alternative, if you wish to get up and running more quickly, you can run our initial server setup script which automates these steps.

Step 1 — Logging in as Root

To log into your server, you will need to know your server's public IP address. You will also need the password or, if you installed an SSH key for authentication, the private key for the root user's account. If you have not already logged into your server, you may want to follow our guide on how to connect to your Droplet with SSH, which covers this process in detail.

If you are not already connected to your server, go ahead and log in as the root user using the following command (substitute the highlighted portion of the command with your server's public IP address):

```
ssh root@your_server_ip
```

Accept the warning about host authenticity if it appears. If you are using password authentication, provide your root password to log in. If you are using an SSH key that is passphrase protected, you may be prompted to enter the passphrase the first time you use the key each session. If this is your first time logging into the server with a password, you may also be prompted to change the root password.

About Root

The root user is the administrative user in a Linux environment that has very broad privileges. Because of the heightened privileges of the root account, you are discouraged from using it on a regular basis. This is because part of the power inherent with the root account is the ability to make very destructive changes, even by accident.

The next step is to set up an alternative user account with a reduced scope of influence for day-to-day work. We'll teach you how to gain increased privileges during the times when you need them.

Step 2 — Creating a New User

Once you are logged in as root, we're prepared to add the new user account that we will use to log in from now on.

This example creates a new user called sammy, but you should replace it with a username that you like:

```
adduser sammy
```

You will be asked a few questions, starting with the account password.

Enter a strong password and, optionally, fill in any of the additional information if you would like. This is not required and you can just hit ENTER in any field you wish to skip.

Step 3 — Granting Administrative Privileges

Now, we have a new user account with regular account privileges. However, we may sometimes need to do administrative tasks.

To avoid having to log out of our normal user and log back in as the root account, we can set up what is known as "superuser" or root privileges for our normal account. This will allow our normal user to run commands with administrative privileges by putting the word sudo before each command.

To add these privileges to our new user, we need to add the new user to the sudo group. By default, on Ubuntu 18.04, users who belong to the sudo group are allowed to use the sudo command.

As root, run this command to add your new user to the sudo group (substitute the highlighted word with your new user):

```
usermod -aG sudo sammy
```

Now, when logged in as your regular user, you can type sudo before commands to perform actions with superuser privileges.

Step 4 — Setting Up a Basic Firewall

Ubuntu 18.04 servers can use the UFW firewall to make sure only connections to certain services are allowed. We can set up a basic firewall very easily using this application.

Note: If your servers are running on DigitalOcean, you can optionally use DigitalOcean Cloud Firewalls instead of the UFW firewall. We recommend using only one firewall at a time to avoid conflicting rules that may be difficult to debug.

Different applications can register their profiles with UFW upon installation. These profiles allow UFW to manage these applications by name. OpenSSH, the service allowing us to connect to our server now, has a profile registered with UFW.

You can see this by typing:

```
ufw app list
```

Output

```
Available applications:
  OpenSSH
```

We need to make sure that the firewall allows SSH connections so that we can log back in next time. We can allow these connections by typing:

```
ufw allow OpenSSH
```

Afterwards, we can enable the firewall by typing:

```
ufw enable
```

Type “y” and press `ENTER` to proceed. You can see that SSH connections are still allowed by typing:

```
ufw status
```

Output

```
Status: active

To                          Action      From
--                          ------      ----
OpenSSH                     ALLOW       Anywhere
OpenSSH (v6)                ALLOW       Anywhere (v6)
```

As the firewall is currently blocking all connections except for SSH, if you install and configure additional services, you will need to adjust the firewall settings to allow acceptable traffic in. You can learn some common UFW operations in this guide.

Step 5 — Enabling External Access for Your Regular User

Now that we have a regular user for daily use, we need to make sure we can SSH into the account directly.

Note: Until verifying that you can log in and use `sudo` with your new user, we recommend staying logged in as root. This way, if you have problems, you can troubleshoot and make any necessary changes as root. If you are using a DigitalOcean Droplet and experience problems with your root SSH connection, you can log into the Droplet using the DigitalOcean Console.

The process for configuring SSH access for your new user depends on whether your server’s root account uses a password or SSH keys for authentication.

If the Root Account Uses Password Authentication

If you logged in to your root account using a password, then password authentication is enabled for SSH. You can SSH to your new user account by opening up a new terminal session and using SSH with your new username:

```
ssh sammy@your_server_ip
```

After entering your regular user’s password, you will be logged in. Remember, if you need to run a command with administrative privileges, type sudo before it like this:

```
sudo command_to_run
```

You will be prompted for your regular user password when using sudo for the first time each session (and periodically afterwards).

To enhance your server’s security, we strongly recommend setting up SSH keys instead of using password authentication. Follow our guide on setting up SSH keys on Ubuntu 18.04 to learn how to configure key-based authentication.

If the Root Account Uses SSH Key Authentication

If you logged in to your root account using SSH keys, then password authentication is disabled for SSH. You will need to add a copy of your local public key to the new user's `~/.ssh/authorized_keys` file to log in successfully.

Since your public key is already in the root account's `~/.ssh/authorized_keys` file on the server, we can copy that file and directory structure to our new user account in our existing session.

The simplest way to copy the files with the correct ownership and permissions is with the `rsync` command. This will copy the root user's `.ssh` directory, preserve the permissions, and modify the file owners, all in a single command. Make sure to change the highlighted portions of the command below to match your regular user's name:

Note: The `rsync` command treats sources and destinations that end with a trailing slash differently than those without a trailing slash. When using `rsync` below, be sure that the source directory (`~/.ssh`) does not include a trailing slash (check to make sure you are not using `~/.ssh/`).

If you accidentally add a trailing slash to the command, `rsync` will copy the contents of the root account's `~/.ssh` directory to the `sudo` user's home directory instead of copying the entire `~/.ssh` directory structure. The files will be in the wrong location and SSH will not be able to find and use them.

```
rsync --archive --chown=sammy:sammy ~/.ssh
/home/sammy
```

Now, open up a new terminal session and using SSH with your new username:

```
ssh sammy@your_server_ip
```

You should be logged in to the new user account without using a password. Remember, if you need to run a command with administrative privileges, type sudo before it like this:

```
sudo command_to_run
```

You will be prompted for your regular user password when using sudo for the first time each session (and periodically afterwards).

Where To Go From Here?

At this point, you have a solid foundation for your server. You can install any of the software you need on your server now.

How to Add and Delete Users on Ubuntu 18.04

Written by Jamon Camisso

In this chapter you will learn how to manage users on a Linux system. You will add a user to a server, grant it access to run privileged commands using `sudo`, and then delete the users. This chapter will also explain how to verify the privileges that are granted to a user with the `sudo` command.

Adding and removing users on a Linux system is one of the most important system administration tasks to familiarize yourself with. When you create a new system, you are often only given access to the root account by default.

While running as the root user gives you complete control over a system and its users, it is also dangerous and can be destructive. For common system administration tasks, it is a better idea to add an unprivileged user and carry out those tasks without root privileges. You can also create additional unprivileged accounts for any other users you may have on your system. Each user on a system should have their own separate account.

For tasks that require administrator privileges, there is a tool installed on Ubuntu systems called `sudo`. Briefly, `sudo` allows you to run a command as another user, including users with administrative privileges. In this guide we will cover how to create user accounts, assign `sudo` privileges, and delete users.

Prerequisites

To follow along with this guide, you will need:

- Access to a server running Ubuntu 18.04. Ensure that you have root access to the server. To set this up, follow our Initial Server Setup Guide for Ubuntu 18.04.

Adding a User

If you are signed in as the root user, you can create a new user at any time by typing:

```
adduser newuser
```

If you are signed in as a non-root user who has been given `sudo` privileges, you can add a new user by typing:

```
sudo adduser newuser
```

Either way, you will be asked a series of questions. The procedure will be:

- Assign and confirm a password for the new user
- Enter any additional information about the new user. This is entirely optional and can be skipped by hitting `ENTER` if you don't wish to utilize these fields.
- Finally, you'll be asked to confirm that the information you provided was correct. Enter `Y` to continue.

Your new user is now ready for use. You can now log in using the password that you entered.

If you need your new user to have access to administrative functionality, continue on to the next section.

Granting a User Sudo Privileges

If your new user should have the ability to execute commands with root (administrative) privileges, you will need to give the new user access to sudo. Let's examine two approaches to this problem: adding the user to a pre-defined sudo user group, and specifying privileges on a per-user basis in sudo's configuration.

Adding the New User to the Sudo Group

By default, sudo on Ubuntu 18.04 systems is configured to extend full privileges to any user in the sudo group.

You can see what groups your new user is in with the groups command:

```
groups newuser
```

Output

```
newuser : newuser
```

By default, a new user is only in their own group which adduser creates along with the user profile. A user and its own group share the same name. In order to add the user to a new group, we can use the usermod command:

```
usermod -aG sudo newuser
```

The -aG option here tells usermod to add the user to the listed groups.

Specifying Explicit User Privileges in /etc/sudoers

As an alternative to putting your user in the sudo group, you can use the `visudo` command, which opens a configuration file called `/etc/sudoers` in the system's default editor, and explicitly specify privileges on a per-user basis.

Using `visudo` is the only recommended way to make changes to `/etc/sudoers`, because it locks the file against multiple simultaneous edits and performs a sanity check on its contents before overwriting the file. This helps to prevent a situation where you misconfigure `sudo` and are prevented from fixing the problem because you have lost `sudo` privileges.

If you are currently signed in as root, type:

```
visudo
```

If you are signed in as a non-root user with `sudo` privileges, type:

```
sudo visudo
```

Traditionally, `visudo` opened `/etc/sudoers` in the `vi` editor, which can be confusing for inexperienced users. By default on new Ubuntu installations, `visudo` will instead use `nano`, which provides a more convenient and accessible text editing experience. Use the arrow keys to move the cursor, and search for the line that looks like this:

/etc/sudoers

```
root    ALL=(ALL:ALL) ALL
```

Below this line, add the following highlighted line. Be sure to change **`newuser`** to the name of the user profile that you would like to grant `sudo` privileges:

/etc/sudoers

```
root    ALL=(ALL:ALL) ALL
newuser ALL=(ALL:ALL) ALL
```

Add a new line like this for each user that should be given full `sudo` privileges. When you are finished, you can save and close the file by hitting `CTRL+X`, followed by `Y`, and then `ENTER` to confirm. ## Testing Your User's Sudo Privileges

Now, your new user is able to execute commands with administrative privileges.

When signed in as the new user, you can execute commands as your regular user by typing commands as normal:

```
some_command
```

You can execute the same command with administrative privileges by typing `sudo` ahead of the command:

```
sudo some_command
```

You will be prompted to enter the password of the regular user account you are signed in as.

Deleting a User

In the event that you no longer need a user, it is best to delete the old account.

You can delete the user itself, without deleting any of their files, by typing the following command as root:

```
deluser newuser
```

If you are signed in as another non-root user with sudo privileges, you could instead type:

```
sudo deluser newuser
```

If, instead, you want to delete the user's home directory when the user is deleted, you can issue the following command as root:

```
deluser --remove-home newuser
```

If you're running this as a non-root user with sudo privileges, you would instead type:

```
sudo deluser --remove-home newuser
```

If you had previously configured sudo privileges for the user you deleted, you may want to remove the relevant line again by typing:

```
visudo
```

Or use this if you are a non-root user with sudo privileges:

```
sudo visudo
root    ALL=(ALL:ALL) ALL
newuser ALL=(ALL:ALL) ALL   # DELETE THIS LINE
```

This will prevent a new user created with the same name from being accidentally given sudo privileges.

Conclusion

You should now have a fairly good handle on how to add and remove users from your Ubuntu 18.04 system. Effective user management will allow you to separate users and give them only the access that they are required to do their job.

For more information about how to configure sudo, check out our guide on how to edit the sudoers file here.

How To Install the Apache Web Server on Ubuntu 18.04

Written by Justin Ellingwood and Kathleen Juell

This chapter explains how to install and manage the Apache webserver. Apache is the most widely used web server on the Internet, so it is worth learning to install and configure it.

In this chapter you will learn how to install Apache, configure firewall rules to allow it to send and receive HTTP data, and verify the server is configured correctly.

You will also learn hwo to manage the Apache process, build more advanced `VirtualHost` configurations, and where to look and how to modify other important Apache configuration files and directories.

The Apache HTTP server is the most widely-used web server in the world. It provides many powerful features including dynamically loadable modules, robust media support, and extensive integration with other popular software.

In this guide, we'll explain how to install an Apache web server on your Ubuntu 18.04 server.

Prerequisites

Before you begin this guide, you should have a regular, non-root user with sudo privileges configured on your server. Additionally, you will need to enable a basic firewall to block non-essential ports. You can learn how to configure a regular user account and set up a firewall for your server by following our initial server setup guide for Ubuntu 18.04.

When you have an account available, log in as your non-root user to begin.

Step 1 — Installing Apache

Apache is available within Ubuntu's default software repositories, making it possible to install it using conventional package management tools.

Let's begin by updating the local package index to reflect the latest upstream changes:

```
sudo apt update
```

Then, install the `apache2` package:

```
sudo apt install apache2
```

After confirming the installation, `apt` will install Apache and all required dependencies.

Step 2 — Adjusting the Firewall

Before testing Apache, it's necessary to modify the firewall settings to allow outside access to the default web ports. Assuming that you followed the instructions in the prerequisites, you should have a UFW firewall configured to restrict access to your server.

During installation, Apache registers itself with UFW to provide a few application profiles that can be used to enable or disable access to Apache through the firewall.

List the `ufw` application profiles by typing:

```
sudo ufw app list
```

You will see a list of the application profiles:

Output

```
Available applications:
  Apache
  Apache Full
  Apache Secure
  OpenSSH
```

As you can see, there are three profiles available for Apache:

- Apache: This profile opens only port 80 (normal, unencrypted web traffic)
- Apache Full: This profile opens both port 80 (normal, unencrypted web traffic) and port 443 (TLS/SSL encrypted traffic)
- Apache Secure: This profile opens only port 443 (TLS/SSL encrypted traffic)

It is recommended that you enable the most restrictive profile that will still allow the traffic you've configured. Since we haven't configured SSL for our server yet in this guide, we will only need to allow traffic on port 80:

```
sudo ufw allow 'Apache'
```

You can verify the change by typing:

```
sudo ufw status
```

You should see HTTP traffic allowed in the displayed output:

Output

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Apache                     ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache (v6)                ALLOW       Anywhere (v6)
```

As you can see, the profile has been activated to allow access to the web server.

Step 3 — Checking your Web Server

At the end of the installation process, Ubuntu 18.04 starts Apache. The web server should already be up and running.

Check with the `systemd` init system to make sure the service is running by typing:

```
sudo systemctl status apache2
```

Output

```
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled;
vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Tue 2018-04-24 20:14:39 UTC; 9min
ago
 Main PID: 2583 (apache2)
    Tasks: 55 (limit: 1153)
   CGroup: /system.slice/apache2.service
           ├─2583 /usr/sbin/apache2 -k start
           ├─2585 /usr/sbin/apache2 -k start
           └─2586 /usr/sbin/apache2 -k start
```

As you can see from this output, the service appears to have started successfully. However, the best way to test this is to request a page from Apache.

You can access the default Apache landing page to confirm that the software is running properly through your IP address. If you do not know your server’s IP address, you can get it a few different ways from the command line.

Try typing this at your server’s command prompt:

```
hostname -I
```

You will get back a few addresses separated by spaces. You can try each in your web browser to see if they work.

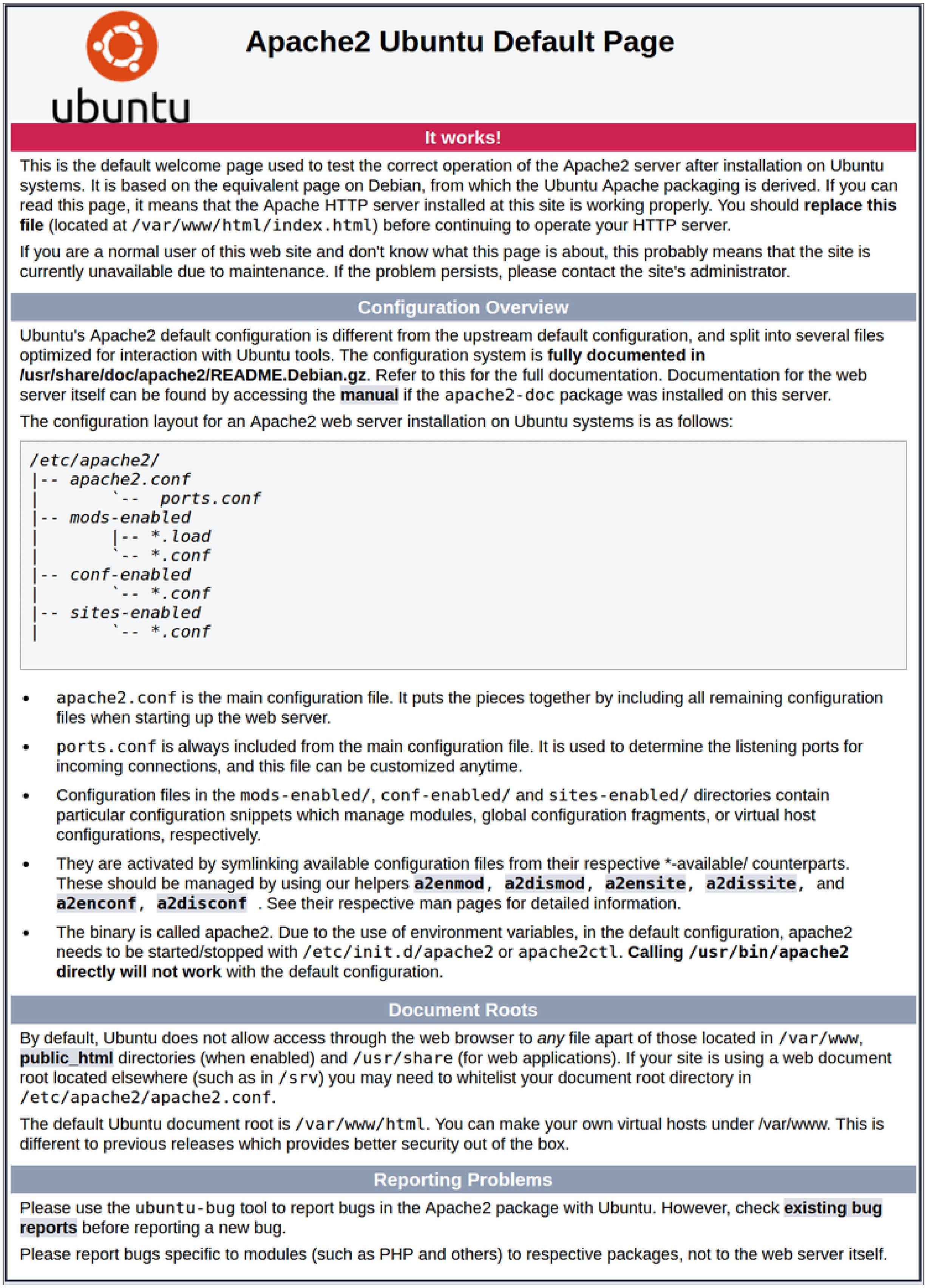
An alternative is typing this, which should give you your public IP address as seen from another location on the internet:

```
curl -4 icanhazip.com
```

When you have your server’s IP address, enter it into your browser’s address bar:

```
http://your_server_ip
```

You should see the default Ubuntu 18.04 Apache web page:



Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers **`a2enmod`**, **`a2dismod`**, **`a2ensite`**, **`a2dissite`**, and **`a2enconf`**, **`a2disconf`** . See their respective man pages for detailed information.
- The binary is called apache2. Due to the use of environment variables, in the default configuration, apache2 needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

Document Roots

By default, Ubuntu does not allow access through the web browser to *any* file apart of those located in `/var/www`, **public_html** directories (when enabled) and `/usr/share` (for web applications). If your site is using a web document root located elsewhere (such as in `/srv`) you may need to whitelist your document root directory in `/etc/apache2/apache2.conf`.

The default Ubuntu document root is `/var/www/html`. You can make your own virtual hosts under /var/www. This is different to previous releases which provides better security out of the box.

Reporting Problems

Please use the `ubuntu-bug` tool to report bugs in the Apache2 package with Ubuntu. However, check **existing bug reports** before reporting a new bug.

Please report bugs specific to modules (such as PHP and others) to respective packages, not to the web server itself.

Apache default page

This page indicates that Apache is working correctly. It also includes some basic information about important Apache files and directory locations.

Step 4 — Managing the Apache Process

Now that you have your web server up and running, let’s go over some basic management commands.

To stop your web server, type:

```
sudo systemctl stop apache2
```

To start the web server when it is stopped, type:

```
sudo systemctl start apache2
```

To stop and then start the service again, type:

```
sudo systemctl restart apache2
```

If you are simply making configuration changes, Apache can often reload without dropping connections. To do this, use this command:

```
sudo systemctl reload apache2
```

By default, Apache is configured to start automatically when the server boots. If this is not what you want, disable this behavior by typing:

```
sudo systemctl disable apache2
```

To re-enable the service to start up at boot, type:

```
sudo systemctl enable apache2
```

Apache should now start automatically when the server boots again.

Step 5 — Setting Up Virtual Hosts (Recommended)

When using the Apache web server, you can use virtual hosts (similar to server blocks in Nginx) to encapsulate configuration details and host more than one domain from a single server. We will set up a domain called your_domain, but you should replace this with your own domain name. To learn more about setting up a domain name with DigitalOcean, see our Introduction to DigitalOcean DNS.

Apache on Ubuntu 18.04 has one server block enabled by default that is configured to serve documents from the `/var/www/html` directory. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying `/var/www/html`, let's create a directory structure within `/var/www` for a your_domain site, leaving `/var/www/html` in place as the default directory to be served if a client request doesn't match any other sites.

Create the directory for your_domain as follows:

```
sudo mkdir /var/www/your_domain
```

Next, assign ownership of the directory with the `$USER` environment variable:

```
sudo chown -R $USER:$USER /var/www/your_domain
```

The permissions of your web roots should be correct if you haven't modified your `unmask` value, but you can make sure by typing:

```
sudo chmod -R 755 /var/www/your_domain
```

Next, create a sample `index.html` page using `nano` or your favorite editor:

```
nano /var/www/your_domain/index.html
```

Inside, add the following sample HTML:

/var/www/your_domain/index.html

```
<html>
    <head>
        <title>Welcome to Your_domain!</title>
    </head>
    <body>
        <h1>Success!  The your_domain virtual host is working!</h1>
    </body>
</html>
```

Save and close the file when you are finished.

In order for Apache to serve this content, it's necessary to create a virtual host file with the correct directives. Instead of modifying the default configuration file located at `/etc/apache2/sites-available/000-default.conf` directly, let's make a new one at `/etc/apache2/sites-available/`**`your_domain`**`.conf`:

```
sudo nano /etc/apache2/sites-available/your_domain.conf
```

Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:

/etc/apache2/sites-available/your_domain.conf

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName your_domain
    ServerAlias www.your_domain
    DocumentRoot /var/www/your_domain
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Notice that we've updated the `DocumentRoot` to our new directory and `ServerAdmin` to an email that the your_domain site administrator can access. We've also added two directives: `ServerName`, which establishes the base domain that should match for this virtual host definition, and `ServerAlias`, which defines further names that should match as if they were the base name.

Save and close the file when you are finished.

Let's enable the file with the `a2ensite` tool:

```
sudo a2ensite your_domain.conf
```

Disable the default site defined in `000-default.conf`:

```
sudo a2dissite 000-default.conf
```

Next, let's test for configuration errors:

```
sudo apache2ctl configtest
```

You should see the following output:

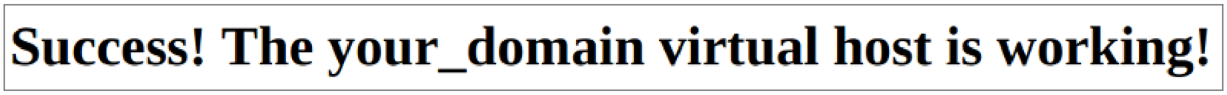
Output

```
Syntax OK
```

Restart Apache to implement your changes:

```
sudo systemctl restart apache2
```

Apache should now be serving your domain name. You can test this by navigating to `http://`**`your_domain`**, where you should see something like this:



Success! The your_domain virtual host is working!

Apache virtual host example

Step 6 – Getting Familiar with Important Apache Files and Directories

Now that you know how to manage the Apache service itself, you should take a few minutes to familiarize yourself with a few important directories and files.

Content

- `/var/www/html`: The actual web content, which by default only consists of the default Apache page you saw earlier, is served out of the `/var/www/html` directory. This can be changed by altering Apache configuration files.

Server Configuration

- `/etc/apache2`: The Apache configuration directory. All of the Apache configuration files reside here.
- `/etc/apache2/apache2.conf`: The main Apache configuration file. This can be modified to make changes to the Apache global configuration. This file is responsible for loading many of the other files in the configuration directory.
- `/etc/apache2/ports.conf`: This file specifies the ports that Apache will listen on. By default, Apache listens on port 80 and additionally listens on port 443 when a module providing SSL capabilities is enabled.
- `/etc/apache2/sites-available/`: The directory where per-site virtual hosts can be stored. Apache will not use the configuration files found in this directory unless they are linked to the `sites-enabled` directory. Typically, all server block configuration is done in this directory, and then enabled by linking to the other directory with the `a2ensite` command.
- `/etc/apache2/sites-enabled/`: The directory where enabled per-site virtual hosts are stored. Typically, these are created by linking to configuration files found in the `sites-available` directory with the `a2ensite`. Apache reads the configuration files and links found in this directory when it starts or reloads to compile a complete configuration.
- `/etc/apache2/conf-available/`, `/etc/apache2/conf-enabled/`: These directories have the same relationship as the `sites-available` and `sites-enabled` directories, but are used to store configuration fragments that do not belong in a virtual host. Files in the `conf-available` directory can be enabled with

the a2enconf command and disabled with the a2disconf command.

- /etc/apache2/mods-available/, /etc/apache2/mods-enabled/: These directories contain the available and enabled modules, respectively. Files in ending in .load contain fragments to load specific modules, while files ending in .conf contain the configuration for those modules. Modules can be enabled and disabled using the a2enmod and a2dismod command.

Server Logs

- /var/log/apache2/access.log: By default, every request to your web server is recorded in this log file unless Apache is configured to do otherwise.
- /var/log/apache2/error.log: By default, all errors are recorded in this file. The LogLevel directive in the Apache configuration specifies how much detail the error logs will contain.

Conclusion

Now that you have your web server installed, you have many options for the type of content you can serve and the technologies you can use to create a richer experience.

If you'd like to build out a more complete application stack, you can look at this article on how to configure a LAMP stack on Ubuntu 18.04.

How To Install Nginx on Ubuntu 18.04

Written by Justin Ellingwood and Kathleen Juell

In the previous chapter you learned how to install the Apache webserver. This chapter explains how to install and manage the Nginx webserver, a fully featured alternative to Apache that powers some of the busiest websites in the world.

You will learn how to install Nginx, configure firewall rules to allow it to send and receive HTTP data, and verify the server is configured correctly.

You will also learn hwo to manage the Nginx process, build more advanced server configurations, and where to look and how to modify other important Nginx configuration files and directories.

Nginx is one of the most popular web servers in the world and is responsible for hosting some of the largest and highest-traffic sites on the internet. It is more resource-friendly than Apache in most cases and can be used as a web server or reverse proxy.

In this guide, we'll discuss how to install Nginx on your Ubuntu 18.04 server.

Prerequisites

Before you begin this guide, you should have a regular, non-root user with sudo privileges configured on your server. You can learn how to configure a regular user account by following our initial server setup guide for Ubuntu 18.04.

When you have an account available, log in as your non-root user to begin.

Step 1 – Installing Nginx

Because Nginx is available in Ubuntu's default repositories, it is possible to install it from these repositories using the `apt` packaging system.

Since this is our first interaction with the `apt` packaging system in this session, we will update our local package index so that we have access to the most recent package listings. Afterwards, we can install `nginx`:

```
sudo apt update
sudo apt install nginx
```

After accepting the procedure, `apt` will install Nginx and any required dependencies to your server.

Step 2 – Adjusting the Firewall

Before testing Nginx, the firewall software needs to be adjusted to allow access to the service. Nginx registers itself as a service with `ufw` upon installation, making it straightforward to allow Nginx access.

List the application configurations that `ufw` knows how to work with by typing:

```
sudo ufw app list
```

You should get a listing of the application profiles:

Output

```
Available applications:
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenSSH
```

As you can see, there are three profiles available for Nginx:

- Nginx Full: This profile opens both port 80 (normal, unencrypted web traffic) and port 443 (TLS/SSL encrypted traffic)
- Nginx HTTP: This profile opens only port 80 (normal, unencrypted web traffic)
- Nginx HTTPS: This profile opens only port 443 (TLS/SSL encrypted traffic)

It is recommended that you enable the most restrictive profile that will still allow the traffic you've configured. Since we haven't configured SSL for our server yet in this guide, we will only need to allow traffic on port 80.

You can enable this by typing:

```
sudo ufw allow 'Nginx HTTP'
```

You can verify the change by typing:

```
sudo ufw status
```

You should see HTTP traffic allowed in the displayed output:

Output

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Nginx HTTP                 ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Nginx HTTP (v6)            ALLOW       Anywhere (v6)
```

Step 3 – Checking your Web Server

At the end of the installation process, Ubuntu 18.04 starts Nginx. The web server should already be up and running.

We can check with the `systemd` init system to make sure the service is running by typing:

```
systemctl status nginx
```

Output

```
● nginx.service - A high performance web server and a reverse proxy
server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled;
vendor preset: enabled)
   Active: active (running) since Fri 2018-04-20 16:08:19 UTC; 3
days ago
     Docs: man:nginx(8)
 Main PID: 2369 (nginx)
    Tasks: 2 (limit: 1153)
   CGroup: /system.slice/nginx.service
           ├─2369 nginx: master process /usr/sbin/nginx -g daemon
on; master_process on;
           └─2380 nginx: worker process
```

As you can see above, the service appears to have started successfully. However, the best way to test this is to actually request a page from Nginx.

You can access the default Nginx landing page to confirm that the software is running properly by navigating to your server’s IP address. If you do not know your server’s IP address, you can get it a few different ways.

Try typing this at your server’s command prompt:

```
ip addr show eth0 | grep inet | awk '{ print $2;
}' | sed 's/\/.*$//'
```

You will get back a few lines. You can try each in your web browser to see if they work.

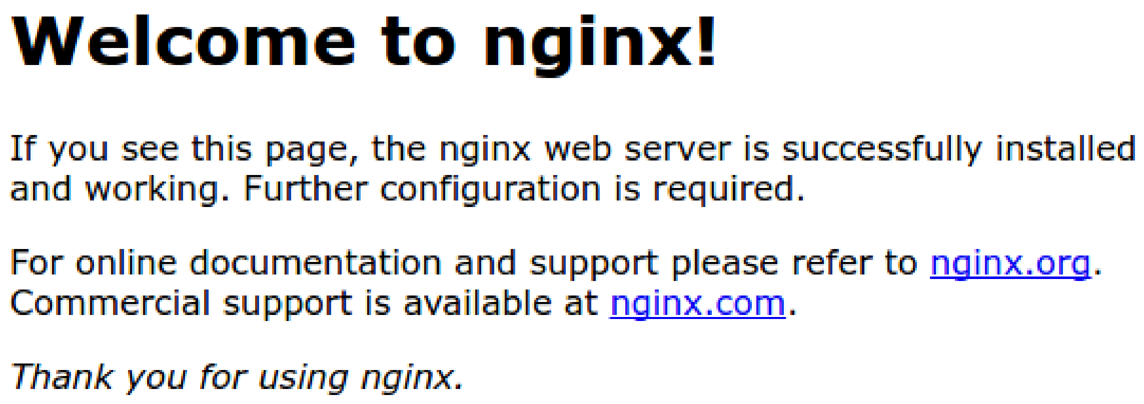
An alternative is typing this, which should give you your public IP address as seen from another location on the internet:

```
curl -4 icanhazip.com
```

When you have your server’s IP address, enter it into your browser’s address bar:

```
http://your_server_ip
```

You should see the default Nginx landing page:



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Nginx default page

This page is included with Nginx to show you that the server is running correctly.

Step 4 – Managing the Nginx Process

Now that you have your web server up and running, let’s review some basic management commands.

To stop your web server, type:

```
sudo systemctl stop nginx
```

To start the web server when it is stopped, type:

```
sudo systemctl start nginx
```

To stop and then start the service again, type:

```
sudo systemctl restart nginx
```

If you are simply making configuration changes, Nginx can often reload without dropping connections. To do this, type:

```
sudo systemctl reload nginx
```

By default, Nginx is configured to start automatically when the server boots. If this is not what you want, you can disable this behavior by typing:

```
sudo systemctl disable nginx
```

To re-enable the service to start up at boot, you can type:

```
sudo systemctl enable nginx
```

Step 5 – Setting Up Server Blocks (Recommended)

When using the Nginx web server, server blocks (similar to virtual hosts in Apache) can be used to encapsulate configuration details and host more than one domain from a single server. We will set up a domain called example.com, but you should replace this with your own domain name. To learn more about setting up a domain name with DigitalOcean, see our Introduction to DigitalOcean DNS.

Nginx on Ubuntu 18.04 has one server block enabled by default that is configured to serve documents out of a directory at `/var/www/html`. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying `/var/www/html`, let’s create a directory structure within `/var/www` for our example.com site, leaving `/var/www/html` in place as the default directory to be served if a client request doesn't match any other sites.

Create the directory for example.com as follows, using the `-p` flag to create any necessary parent directories:

```
sudo mkdir -p /var/www/example.com/html
```

Next, assign ownership of the directory with the `$USER` environment variable:

```
sudo chown -R $USER:$USER
/var/www/example.com/html
```

The permissions of your web roots should be correct if you haven't modified your `umask` value, but you can make sure by typing:

```
sudo chmod -R 755 /var/www/example.com
```

Next, create a sample `index.html` page using `nano` or your favorite editor:

```
nano /var/www/example.com/html/index.html
```

Inside, add the following sample HTML:

/var/www/example.com/html/index.html

```
<html>
    <head>
        <title>Welcome to Example.com!</title>
    </head>
    <body>
        <h1>Success!  The example.com server block is working!</h1>
    </body>
</html>
```

Save and close the file when you are finished.

In order for Nginx to serve this content, it's necessary to create a server block with the correct directives. Instead of modifying the default configuration file directly, let's make a new one at `/etc/nginx/sites-available/`**`example.com`**:

```
sudo nano /etc/nginx/sites-available/example.com
```

Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:

/etc/nginx/sites-available/example.com

```
server {
        listen 80;
        listen [::]:80;

        root /var/www/example.com/html;
        index index.html index.htm index.nginx-debian.html;

        server_name example.com www.example.com;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

Notice that we've updated the `root` configuration to our new directory, and the `server_name` to our domain name.

Next, let's enable the file by creating a link from it to the `sites-enabled` directory, which Nginx reads from during startup:

```
sudo ln -s /etc/nginx/sites-available/example.com
/etc/nginx/sites-enabled/
```

Two server blocks are now enabled and configured to respond to requests based on their `listen` and `server_name` directives (you can read more about how Nginx processes these directives here):

- `example.com`: Will respond to requests for `example.com` and `www.example.com`.
- `default`: Will respond to any requests on port 80 that do not match the other two blocks.

To avoid a possible hash bucket memory problem that can arise from adding additional server names, it is necessary to adjust a single value in the `/etc/nginx/nginx.conf` file. Open the file:

```
sudo nano /etc/nginx/nginx.conf
```

Find the `server_names_hash_bucket_size` directive and remove the # symbol to uncomment the line:

/etc/nginx/nginx.conf

```
...
http {
    ...
    server_names_hash_bucket_size 64;
    ...
}
...
```

Next, test to make sure that there are no syntax errors in any of your Nginx files:

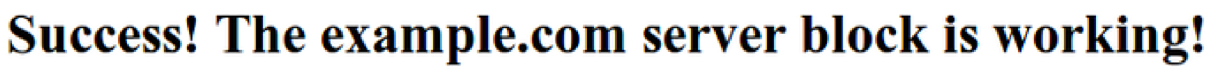
```
sudo nginx -t
```

Save and close the file when you are finished.

If there aren’t any problems, restart Nginx to enable your changes:

```
sudo systemctl restart nginx
```

Nginx should now be serving your domain name. You can test this by navigating to `http://`**`example.com`**, where you should see something like this:



Success! The example.com server block is working!

Nginx first server block

Step 6 – Getting Familiar with Important Nginx Files and Directories

Now that you know how to manage the Nginx service itself, you should take a few minutes to familiarize yourself with a few important directories and files.

Content

- `/var/www/html`: The actual web content, which by default only consists of the default Nginx page you saw earlier, is served out of the `/var/www/html` directory. This can be changed by altering Nginx configuration files.

Server Configuration

- `/etc/nginx`: The Nginx configuration directory. All of the Nginx configuration files reside here.
- `/etc/nginx/nginx.conf`: The main Nginx configuration file. This can be modified to make changes to the Nginx global configuration.
- `/etc/nginx/sites-available/`: The directory where per-site server blocks can be stored. Nginx will not use the configuration files found in this directory unless they are linked to the `sites-enabled` directory. Typically, all server block configuration is done in this directory, and then enabled by linking to the other directory.
- `/etc/nginx/sites-enabled/`: The directory where enabled per-site server blocks are stored. Typically, these are created by linking to configuration files found in the `sites-available` directory.

- `/etc/nginx/snippets`: This directory contains configuration fragments that can be included elsewhere in the Nginx configuration. Potentially repeatable configuration segments are good candidates for refactoring into snippets.

Server Logs

- `/var/log/nginx/access.log`: Every request to your web server is recorded in this log file unless Nginx is configured to do otherwise.
- `/var/log/nginx/error.log`: Any Nginx errors will be recorded in this log.

Conclusion

Now that you have your web server installed, you have many options for the type of content to serve and the technologies you want to use to create a richer experience.

If you'd like to build out a more complete application stack, check out this article on how to configure a LEMP stack on Ubuntu 18.04.

How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04

Written by Mark Drake

This chapter explains how to set up a Linux, Apache, MySQL, and PHP server, commonly referred to as the LAMP stack. In this chapter you will learn how to install all the required LAMP packages, create an Apache configuration, secure MySQL, set up the UFW firewall to allow traffic to the server, and finally add a test PHP script to demonstrate that all the components of the LAMP stack are working correctly.

A previous version of this tutorial was written by Brennan Bearnes.

A "LAMP" stack is a group of open-source software that is typically installed together to enable a server to host dynamic websites and web apps. This term is actually an acronym which represents the Linux operating system, with the Apache web server. The site data is stored in a MySQL database, and dynamic content is processed by PHP.

In this guide, we will install a LAMP stack on an Ubuntu 18.04 server.

Prerequisites

In order to complete this tutorial, you will need to have an Ubuntu 18.04 server with a non-root `sudo`-enabled user account and a basic firewall. This can be configured using our initial server setup guide for Ubuntu 18.04.

Step 1 — Installing Apache and Updating the Firewall

The Apache web server is among the most popular web servers in the world. It’s well-documented and has been in wide use for much of the history of the web, which makes it a great default choice for hosting a website.

Install Apache using Ubuntu’s package manager, `apt`:

```
sudo apt update
sudo apt install apache2
```

Since this is a `sudo` command, these operations are executed with root privileges. It will ask you for your regular user’s password to verify your intentions.

Once you’ve entered your password, `apt` will tell you which packages it plans to install and how much extra disk space they’ll take up. Press `Y` and hit `ENTER` to continue, and the installation will proceed.

Adjust the Firewall to Allow Web Traffic

Next, assuming that you have followed the initial server setup instructions and enabled the UFW firewall, make sure that your firewall allows HTTP and HTTPS traffic. You can check that UFW has an application profile for Apache like so:

```
sudo ufw app list
```

Output

```
Available applications:
  Apache
  Apache Full
  Apache Secure
  OpenSSH
```

If you look at the `Apache Full` profile, it should show that it enables traffic to ports `80` and `443`:

```
sudo ufw app info "Apache Full"
```

Output

```
Profile: Apache Full
Title: Web Server (HTTP,HTTPS)
Description: Apache v2 is the next generation of the omnipresent
Apache web
server.

Ports:
  80,443/tcp
```

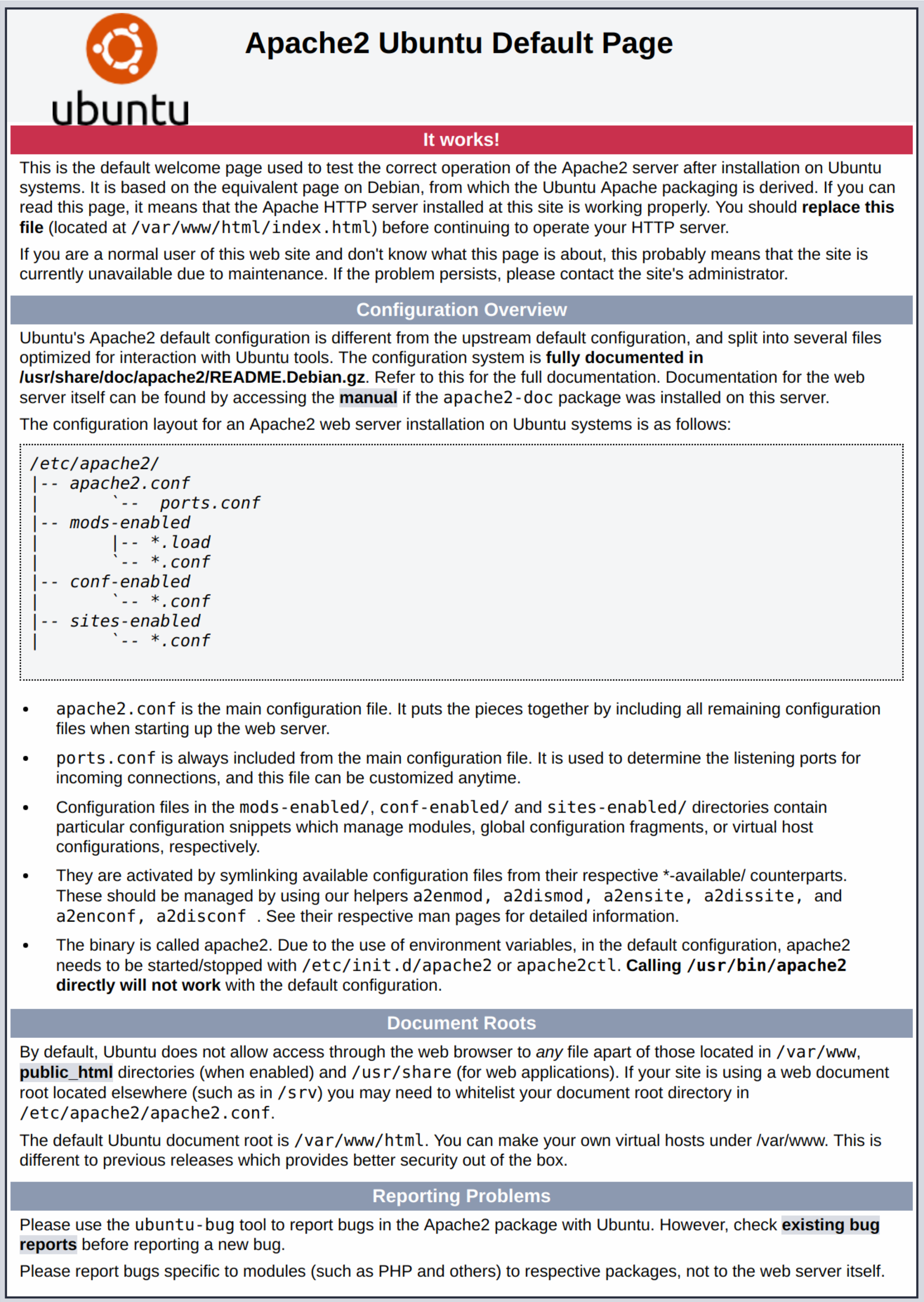
Allow incoming HTTP and HTTPS traffic for this profile:

```
sudo ufw allow in "Apache Full"
```

You can do a spot check right away to verify that everything went as planned by visiting your server's public IP address in your web browser (see the note under the next heading to find out what your public IP address is if you do not have this information already):

```
http://your_server_ip
```

You will see the default Ubuntu 18.04 Apache web page, which is there for informational and testing purposes. It should look something like this:



Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf` . See their respective man pages for detailed information.
- The binary is called apache2. Due to the use of environment variables, in the default configuration, apache2 needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

Document Roots

By default, Ubuntu does not allow access through the web browser to *any* file apart of those located in `/var/www`, **public_html** directories (when enabled) and `/usr/share` (for web applications). If your site is using a web document root located elsewhere (such as in `/srv`) you may need to whitelist your document root directory in `/etc/apache2/apache2.conf`.

The default Ubuntu document root is `/var/www/html`. You can make your own virtual hosts under /var/www. This is different to previous releases which provides better security out of the box.

Reporting Problems

Please use the `ubuntu-bug` tool to report bugs in the Apache2 package with Ubuntu. However, check **existing bug reports** before reporting a new bug.

Please report bugs specific to modules (such as PHP and others) to respective packages, not to the web server itself.

Ubuntu 18.04 Apache default

If you see this page, then your web server is now correctly installed and accessible through your firewall.

How To Find your Server’s Public IP Address

If you do not know what your server’s public IP address is, there are a number of ways you can find it. Usually, this is the address you use to connect to your server through SSH.

There are a few different ways to do this from the command line. First, you could use the `iproute2` tools to get your IP address by typing this:

```
ip addr show eth0 | grep inet | awk '{ print $2;
}' | sed 's/\/.*$//'
```

This will give you two or three lines back. They are all correct addresses, but your computer may only be able to use one of them, so feel free to try each one.

An alternative method is to use the `curl` utility to contact an outside party to tell you how it sees your server. This is done by asking a specific server what your IP address is:

```
sudo apt install curl
curl http://icanhazip.com
```

Regardless of the method you use to get your IP address, type it into your web browser’s address bar to view the default Apache page.

Step 2 — Installing MySQL

Now that you have your web server up and running, it is time to install MySQL. MySQL is a database management system. Basically, it will organize and provide access to databases where your site can store information.

Again, use `apt` to acquire and install this software:

```
sudo apt install mysql-server
```

Note: In this case, you do not have to run `sudo apt update` prior to the command. This is because you recently ran it in the commands above to install Apache. The package index on your computer should already be up-to-date.

This command, too, will show you a list of the packages that will be installed, along with the amount of disk space they'll take up. Enter `Y` to continue.

When the installation is complete, run a simple security script that comes pre-installed with MySQL which will remove some dangerous defaults and lock down access to your database system. Start the interactive script by running:

```
sudo mysql_secure_installation
```

This will ask if you want to configure the `VALIDATE PASSWORD PLUGIN`.

Note: Enabling this feature is something of a judgment call. If enabled, passwords which don't match the specified criteria will be rejected by MySQL with an error. This will cause issues if you use a weak password in conjunction with software which automatically configures MySQL user credentials, such as the Ubuntu packages for phpMyAdmin. It is safe to leave validation disabled, but you should always use strong, unique passwords for database credentials.

Answer `Y` for yes, or anything else to continue without enabling.

```
VALIDATE PASSWORD PLUGIN can be used to test
passwords
and improve security. It checks the strength of
password
and allows the users to set only those passwords
which are
secure enough. Would you like to setup VALIDATE
PASSWORD plugin?

Press y|Y for Yes, any other key for No:
```

If you answer “yes”, you’ll be asked to select a level of password validation. Keep in mind that if you enter `2` for the strongest level, you will receive errors when attempting to set any password which does not contain numbers, upper and lowercase letters, and special characters, or which is based on common dictionary words.

```
There are three levels of password validation
policy:

LOW    Length >= 8
MEDIUM Length >= 8, numeric, mixed case, and
special characters
STRONG Length >= 8, numeric, mixed case, special
characters and dictionary                  file

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 1
```

Regardless of whether you chose to set up the `VALIDATE PASSWORD PLUGIN`, your server will next ask you to select and confirm a password for the MySQL root user. This is an administrative account in MySQL that has increased privileges. Think of it as being similar to the root account for the server itself (although the one you are configuring now is a MySQL-specific account). Make sure this is a strong, unique password, and do not leave it blank.

If you enabled password validation, you'll be shown the password strength for the root password you just entered and your server will ask if you want to change that password. If you are happy with your current password, enter `N` for "no" at the prompt:

```
Using existing password for root.

Estimated strength of the password: 100
Change the password for root ? ((Press y|Y for
Yes, any other key for No) : n
```

For the rest of the questions, press `Y` and hit the `ENTER` key at each prompt. This will remove some anonymous users and the test database, disable remote root logins, and load these new rules so that MySQL immediately respects the changes you have made.

Note that in Ubuntu systems running MySQL 5.7 (and later versions), the root MySQL user is set to authenticate using the `auth_socket` plugin by default rather than with a password. This allows for some greater security and usability in many cases, but it can also complicate things when you need to allow an external program (e.g., phpMyAdmin) to access the user.

If you prefer to use a password when connecting to MySQL as root, you will need to switch its authentication method from `auth_socket` to `mysql_native_password`. To do this, open up the MySQL prompt from your terminal:

```
sudo mysql
```

Next, check which authentication method each of your MySQL user accounts use with the following command:

```
SELECT user,authentication_string,plugin,host FROM
mysql.user;
```

Output

```
+------------------+-------------------------------------------+---
--------------------+-----------+
| user             | authentication_string                     |
plugin                | host      |
+------------------+-------------------------------------------+---
--------------------+-----------+
| root             |                                           |
auth_socket           | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
mysql_native_password | localhost |
| debian-sys-maint | *CC744277A401A7D25BE1CA89AFF17BF607F876FF |
mysql_native_password | localhost |
+------------------+-------------------------------------------+---
--------------------+-----------+
4 rows in set (0.00 sec)
```

In this example, you can see that the root user does in fact authenticate using the `auth_socket` plugin. To configure the root account to authenticate with a password, run the following `ALTER USER` command. Be sure to change **password** to a strong password of your choosing:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH
mysql_native_password BY 'password';
```

Then, run FLUSH PRIVILEGES which tells the server to reload the grant tables and put your new changes into effect:

```
FLUSH PRIVILEGES;
```

Check the authentication methods employed by each of your users again to confirm that root no longer authenticates using the auth_socket plugin:

```
SELECT user,authentication_string,plugin,host FROM
mysql.user;
```

Output

```
+------------------+-------------------------------------------+---
--------------------+-----------+
| user             | authentication_string                     |
plugin                | host      |
+------------------+-------------------------------------------+---
--------------------+-----------+
| root             | *3636DACC8616D997782ADD0839F92C1571D6D78F |
mysql_native_password | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
mysql_native_password | localhost |
| debian-sys-maint | *CC744277A401A7D25BE1CA89AFF17BF607F876FF |
mysql_native_password | localhost |
+------------------+-------------------------------------------+---
--------------------+-----------+
4 rows in set (0.00 sec)
```

You can see in this example output that the root MySQL user now authenticates using a password. Once you confirm this on your own server, you can exit the MySQL shell:

```
exit
```

At this point, your database system is now set up and you can move on to installing PHP, the final component of the LAMP stack.

Step 3 — Installing PHP

PHP is the component of your setup that will process code to display dynamic content. It can run scripts, connect to your MySQL databases to get information, and hand the processed content over to your web server to display.

Once again, leverage the `apt` system to install PHP. In addition, include some helper packages this time so that PHP code can run under the Apache server and talk to your MySQL database:

```
sudo apt install php libapache2-mod-php php-mysql
```

This should install PHP without any problems. We'll test this in a moment.

In most cases, you will want to modify the way that Apache serves files when a directory is requested. Currently, if a user requests a directory from the server, Apache will first look for a file called `index.html`. We want to tell the web server to prefer PHP files over others, so make Apache look for an `index.php` file first.

To do this, type this command to open the `dir.conf` file in a text editor with root privileges:

```
sudo nano /etc/apache2/mods-enabled/dir.conf
```

It will look like this:

/etc/apache2/mods-enabled/dir.conf

```
<IfModule mod_dir.c>
    DirectoryIndex index.html index.cgi index.pl index.php
index.xhtml index.htm
</IfModule>
```

Move the PHP index file (highlighted above) to the first position after the `DirectoryIndex` specification, like this:

/etc/apache2/mods-enabled/dir.conf

```
<IfModule mod_dir.c>
    DirectoryIndex index.php index.html index.cgi index.pl
index.xhtml index.htm
</IfModule>
```

When you are finished, save and close the file by pressing `CTRL+X`. Confirm the save by typing `Y` and then hit `ENTER` to verify the file save location.

After this, restart the Apache web server in order for your changes to be recognized. Do this by typing this:

```
sudo systemctl restart apache2
```

You can also check on the status of the `apache2` service using `systemctl`:

```
sudo systemctl status apache2
```

Sample Output

```
● apache2.service - LSB: Apache2 web server
   Loaded: loaded (/etc/init.d/apache2; bad; vendor preset:
enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Tue 2018-04-23 14:28:43 EDT; 45s
ago
     Docs: man:systemd-sysv-generator(8)
  Process: 13581 ExecStop=/etc/init.d/apache2 stop (code=exited,
status=0/SUCCESS)
  Process: 13605 ExecStart=/etc/init.d/apache2 start (code=exited,
status=0/SUCCESS)
    Tasks: 6 (limit: 512)
   CGroup: /system.slice/apache2.service
           ├─13623 /usr/sbin/apache2 -k start
           ├─13626 /usr/sbin/apache2 -k start
           ├─13627 /usr/sbin/apache2 -k start
           ├─13628 /usr/sbin/apache2 -k start
           ├─13629 /usr/sbin/apache2 -k start
           └─13630 /usr/sbin/apache2 -k start
```

Press `Q` to exit this status output.

To enhance the functionality of PHP, you have the option to install some additional modules. To see the available options for PHP modules and libraries, pipe the results of `apt search` into `less`, a pager which lets you scroll through the output of other commands:

```
apt search php- | less
```

Use the arrow keys to scroll up and down, and press `Q` to quit.

The results are all optional components that you can install. It will give you a short description for each:

```
bandwidthd-pgsql/bionic 2.0.1+cvs20090917-
10ubuntu1 amd64
  Tracks usage of TCP/IP and builds html files
with graphs

bluefish/bionic 2.2.10-1 amd64
  advanced Gtk+ text editor for web and software
development

cacti/bionic 1.1.38+ds1-1 all
  web interface for graphing of monitoring systems

ganglia-webfrontend/bionic 3.6.1-3 all
  cluster monitoring toolkit - web front-end

golang-github-unknwon-cae-dev/bionic
0.0~git20160715.0.c6aac99-4 all
  PHP-like Compression and Archive Extensions in
Go

haserl/bionic 0.9.35-2 amd64
  CGI scripting program for embedded environments
```

```
kdevelop-php-docs/bionic 5.2.1-1ubuntu2 all
  transitional package for kdevelop-php

kdevelop-php-docs-l10n/bionic 5.2.1-1ubuntu2 all
  transitional package for kdevelop-php-l10n
…
:
```

To learn more about what each module does, you could search the internet for more information about them. Alternatively, look at the long description of the package by typing:

```
apt show package_name
```

There will be a lot of output, with one field called `Description` which will have a longer explanation of the functionality that the module provides.

For example, to find out what the `php-cli` module does, you could type this:

```
apt show php-cli
```

Along with a large amount of other information, you'll find something that looks like this:

Output

```
…
Description: command-line interpreter for the PHP scripting
language (default)
 This package provides the /usr/bin/php command interpreter, useful
for
 testing PHP scripts from a shell or performing general shell
scripting tasks.
 .
 PHP (recursive acronym for PHP: Hypertext Preprocessor) is a
widely-used
 open source general-purpose scripting language that is especially
suited
 for web development and can be embedded into HTML.
 .
 This package is a dependency package, which depends on Ubuntu's
default
 PHP version (currently 7.2).
…
```

If, after researching, you decide you would like to install a package, you can do so by using the `apt install` command like you have been doing for the other software.

If you decided that `php-cli` is something that you need, you could type:

```
sudo apt install php-cli
```

If you want to install more than one module, you can do that by listing each one, separated by a space, following the `apt install` command, like this:

```
sudo apt install package1 package2 ...
```

At this point, your LAMP stack is installed and configured. Before you do anything else, we recommend that you set up an Apache virtual host where you can store your server's configuration details.

Step 4 — Setting Up Virtual Hosts (Recommended)

When using the Apache web server, you can use virtual hosts (similar to server blocks in Nginx) to encapsulate configuration details and host more than one domain from a single server. We will set up a domain called your_domain, but you should replace this with your own domain name. To learn more about setting up a domain name with DigitalOcean, see our Introduction to DigitalOcean DNS.

Apache on Ubuntu 18.04 has one server block enabled by default that is configured to serve documents from the `/var/www/html` directory. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying `/var/www/html`, let's create a directory structure within `/var/www` for our your_domain site, leaving `/var/www/html` in place as the default directory to be served if a client request doesn't match any other sites.

Create the directory for your_domain as follows:

```
sudo mkdir /var/www/your_domain
```

Next, assign ownership of the directory with the `$USER` environment variable:

```
sudo chown -R $USER:$USER /var/www/your_domain
```

The permissions of your web roots should be correct if you haven't modified your `unmask` value, but you can make sure by typing:

```
sudo chmod -R 755 /var/www/your_domain
```

Next, create a sample `index.html` page using `nano` or your favorite editor:

```
nano /var/www/your_domain/index.html
```

Inside, add the following sample HTML:

/var/www/your_domain/index.html

```
<html>
    <head>
        <title>Welcome to Your_domain!</title>
    </head>
    <body>
        <h1>Success!  The your_domain server block is working!</h1>
    </body>
</html>
```

Save and close the file when you are finished.

In order for Apache to serve this content, it's necessary to create a virtual host file with the correct directives. Instead of modifying the default configuration file located at `/etc/apache2/sites-available/000-default.conf` directly, let's make a new one at `/etc/apache2/sites-available/your_domain.conf`:

```
sudo nano /etc/apache2/sites-available/your_domain.conf
```

Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:

/etc/apache2/sites-available/your_domain.conf

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName your_domain
    ServerAlias www.your_domain
    DocumentRoot /var/www/your_domain
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Notice that we've updated the `DocumentRoot` to our new directory and `ServerAdmin` to an email that the your_domain site administrator can access. We've also added two directives: `ServerName`, which establishes the base domain that should match for this virtual host definition, and `ServerAlias`, which defines further names that should match as if they were the base name.

Save and close the file when you are finished.

Let's enable the file with the `a2ensite` tool:

```
sudo a2ensite your_domain.conf
```

Disable the default site defined in `000-default.conf`:

```
sudo a2dissite 000-default.conf
```

Next, let’s test for configuration errors:

```
sudo apache2ctl configtest
```

You should see the following output:

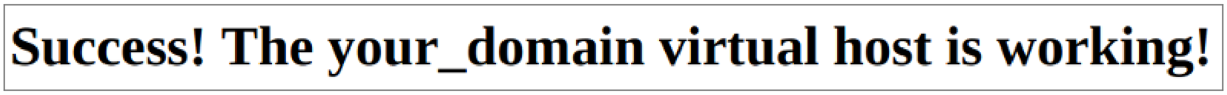
Output

```
Syntax OK
```

Restart Apache to implement your changes:

```
sudo systemctl restart apache2
```

Apache should now be serving your domain name. You can test this by navigating to `http://your_domain`, where you should see something like this:



Success! The your_domain virtual host is working!

Apache virtual host example

With that, you virtual host is fully set up. Before making any more changes or deploying an application, though, it would be helpful to proactively test out your PHP configuration in case there are any issues that should be addressed.

Step 5 — Testing PHP Processing on your Web Server

In order to test that your system is configured properly for PHP, create a very basic PHP script called `info.php`. In order for Apache to find this file and serve it correctly, it must be saved to your web root directory.

Create the file at the web root you created in the previous step by running:

```
sudo nano /var/www/your_domain/info.php
```

This will open a blank file. Add the following text, which is valid PHP code, inside the file:

info.php

```
<?php
phpinfo();
?>
```

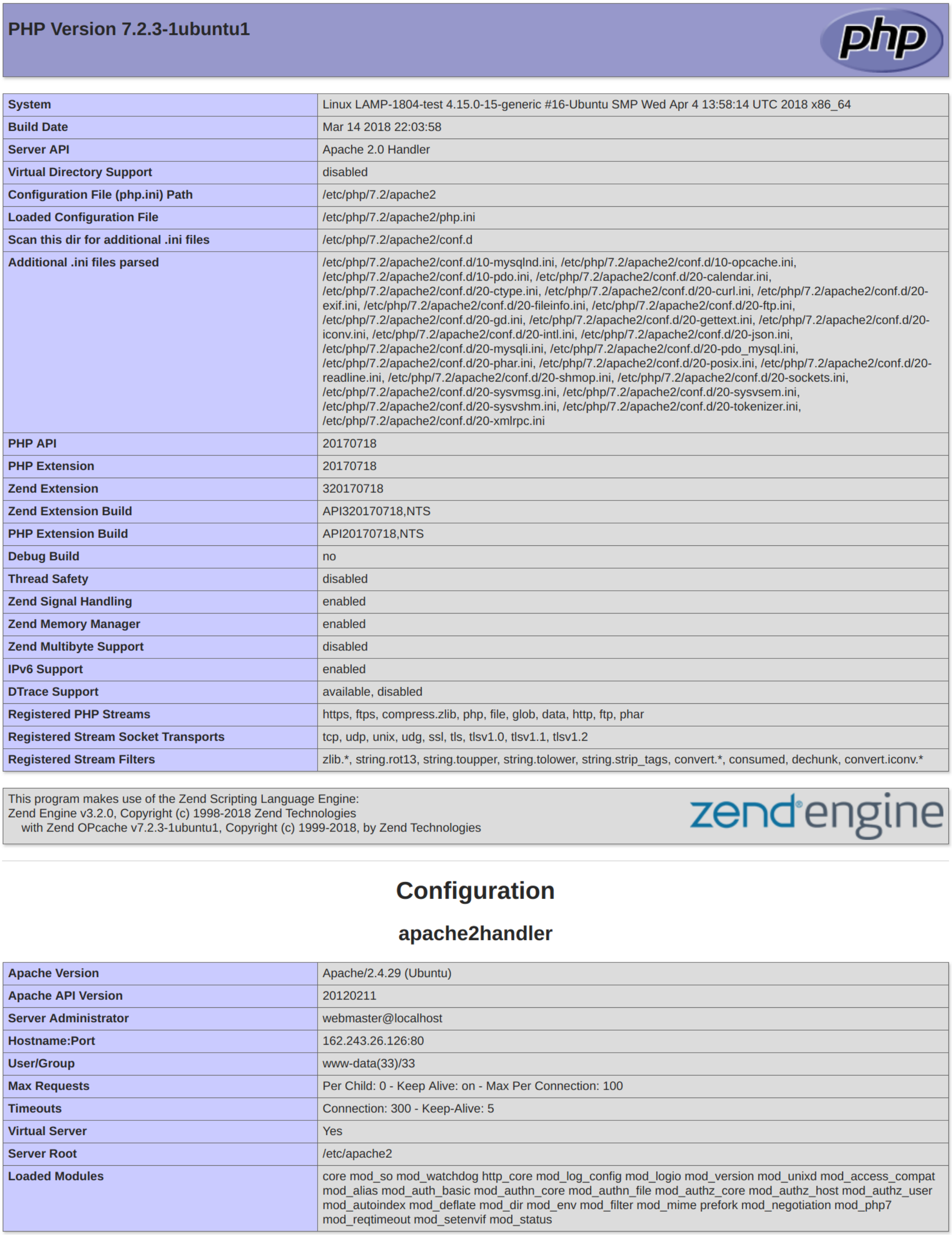
When you are finished, save and close the file.

Now you can test whether your web server is able to correctly display content generated by this PHP script. To try this out, visit this page in your web browser. You’ll need your server’s public IP address again.

The address you will want to visit is:

```
http://your_domain/info.php
```

The page that you come to should look something like this:



PHP Version 7.2.3-1ubuntu1

System	Linux LAMP-1804-test 4.15.0-15-generic #16-Ubuntu SMP Wed Apr 4 13:58:14 UTC 2018 x86_64
Build Date	Mar 14 2018 22:03:58
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/apache2
Loaded Configuration File	/etc/php/7.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/apache2/conf.d
Additional .ini files parsed	/etc/php/7.2/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-curl.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-fileinfo.ini, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gd.ini, /etc/php/7.2/apache2/conf.d/20-gettext.ini, /etc/php/7.2/apache2/conf.d/20-iconv.ini, /etc/php/7.2/apache2/conf.d/20-intl.ini, /etc/php/7.2/apache2/conf.d/20-json.ini, /etc/php/7.2/apache2/conf.d/20-mysqli.ini, /etc/php/7.2/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.2/apache2/conf.d/20-phar.ini, /etc/php/7.2/apache2/conf.d/20-posix.ini, /etc/php/7.2/apache2/conf.d/20-readline.ini, /etc/php/7.2/apache2/conf.d/20-shmop.ini, /etc/php/7.2/apache2/conf.d/20-sockets.ini, /etc/php/7.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.2/apache2/conf.d/20-sysvsem.ini, /etc/php/7.2/apache2/conf.d/20-sysvshm.ini, /etc/php/7.2/apache2/conf.d/20-tokenizer.ini, /etc/php/7.2/apache2/conf.d/20-xmlrpc.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
    with Zend OPcache v7.2.3-1ubuntu1, Copyright (c) 1999-2018, by Zend Technologies

Configuration

apache2handler

Apache Version	Apache/2.4.29 (Ubuntu)
Apache API Version	20120211
Server Administrator	webmaster@localhost
Hostname:Port	162.243.26.126:80
User/Group	www-data(33)/33
Max Requests	Per Child: 0 - Keep Alive: on - Max Per Connection: 100
Timeouts	Connection: 300 - Keep-Alive: 5
Virtual Server	Yes
Server Root	/etc/apache2
Loaded Modules	core mod_so mod_watchdog http_core mod_log_config mod_logio mod_version mod_unixd mod_access_compat mod_alias mod_auth_basic mod_authn_core mod_authn_file mod_authz_core mod_authz_host mod_authz_user mod_autoindex mod_deflate mod_dir mod_env mod_filter mod_mime prefork mod_negotiation mod_php7 mod_reqtimeout mod_setenvif mod_status

Ubuntu 18.04 default PHP info

This page provides some basic information about your server from the perspective of PHP. It is useful for debugging and to ensure that your settings are being applied correctly.

If you can see this page in your browser, then your PHP is working as expected.

You probably want to remove this file after this test because it could actually give information about your server to unauthorized users. To do this, run the following command:

```
sudo rm /var/www/your_domain/info.php
```

You can always recreate this page if you need to access the information again later.

Conclusion

Now that you have a LAMP stack installed, you have many choices for what to do next. Basically, you've installed a platform that will allow you to install most kinds of websites and web software on your server.

As an immediate next step, you should ensure that connections to your web server are secured, by serving them via HTTPS. The easiest option here is to use Let's Encrypt to secure your site with a free TLS/SSL certificate.

Some other popular options are:

- Install Wordpress the most popular content management system on the internet.
- Set Up PHPMyAdmin to help manage your MySQL databases from web browser.
- Learn how to use SFTP to transfer files to and from your server.

How To Install Linux, Nginx, MySQL, PHP (LEMP stack) on Ubuntu 18.04

Written by Justin Ellingwood and Mark Drake

This chapter explains how to set up a Linux, Nginx, MySQL, and PHP server, commonly referred to as the LEMP stack. In this guide you will learn how to install all the required LEMP packages, create an Nginx configuration, secure MySQL, set up the UFW firewall to allow traffic to the server, and finally add a test PHP script to demonstrate that all the components of the LEMP stack are working correctly.

The LEMP software stack is a group of software that can be used to serve dynamic web pages and web applications. This is an acronym that describes a Linux operating system, with an Nginx (pronounced like "Engine-X") web server. The backend data is stored in the MySQL database and the dynamic processing is handled by PHP.

This guide demonstrates how to install a LEMP stack on an Ubuntu 18.04 server. The Ubuntu operating system takes care of the first requirement. We will describe how to get the rest of the components up and running.

Prerequisites

Before you complete this tutorial, you should have a regular, non-root user account on your server with `sudo` privileges. Set up this account by completing our initial server setup guide for Ubuntu 18.04.

Once you have your user available, you are ready to begin the steps outlined in this guide.

Step 1 – Installing the Nginx Web Server

In order to display web pages to our site visitors, we are going to employ Nginx, a modern, efficient web server.

All of the software used in this procedure will come from Ubuntu’s default package repositories. This means we can use the `apt` package management suite to complete the necessary installations.

Since this is our first time using `apt` for this session, start off by updating your server’s package index. Following that, install the server:

```
sudo apt update
sudo apt install nginx
```

On Ubuntu 18.04, Nginx is configured to start running upon installation.

If you have the `ufw` firewall running, as outlined in the initial setup guide, you will need to allow connections to Nginx. Nginx registers itself with `ufw` upon installation, so the procedure is rather straightforward.

It is recommended that you enable the most restrictive profile that will still allow the traffic you want. Since you haven’t configured SSL for your server in this guide, you will only need to allow traffic on port `80`.

Enable this by typing:

```
sudo ufw allow 'Nginx HTTP'
```

You can verify the change by running:

```
sudo ufw status
```

This command’s output will show that HTTP traffic is allowed:

Output

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Nginx HTTP                 ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Nginx HTTP (v6)            ALLOW       Anywhere (v6)
```

With the new firewall rule added, you can test if the server is up and running by accessing your server’s domain name or public IP address in your web browser.

If you do not have a domain name pointed at your server and you do not know your server’s public IP address, you can find it by running the following command:

```
ip addr show eth0 | grep inet | awk '{ print $2;
}' | sed 's/\/.*$//'
```

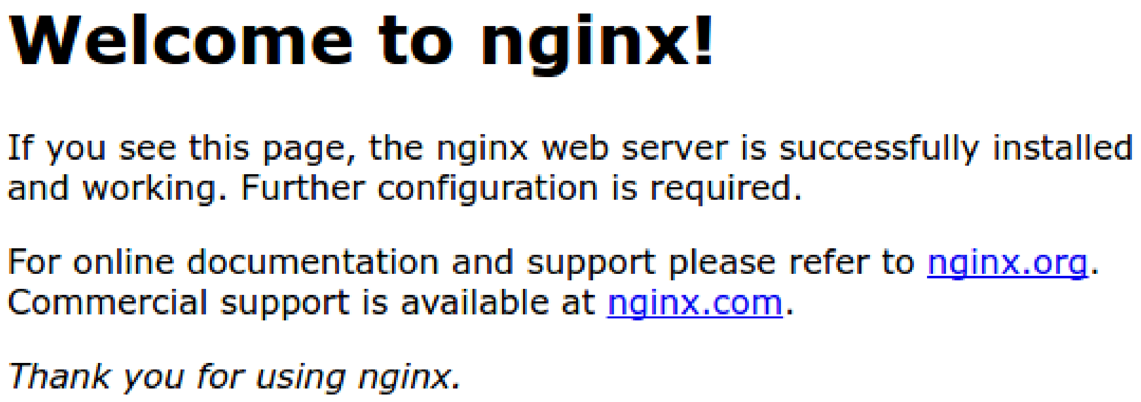
This will print out a few IP addresses. You can try each of them in turn in your web browser.

As an alternative, you can check which IP address is accessible, as viewed from other locations on the internet:

```
curl -4 icanhazip.com
```

Type the address that you receive in your web browser and it will take you to Nginx’s default landing page:

```
http://server_domain_or_IP
```



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Nginx default page

If you see the above page, you have successfully installed Nginx.

Step 2 – Installing MySQL to Manage Site Data

Now that you have a web server, you need to install MySQL (a database management system) to store and manage the data for your site.

Install MySQL by typing:

```
sudo apt install mysql-server
```

The MySQL database software is now installed, but its configuration is not yet complete.

To secure the installation, MySQL comes with a script that will ask whether we want to modify some insecure defaults. Initiate the script by typing:

```
sudo mysql_secure_installation
```

This script will ask if you want to configure the `VALIDATE PASSWORD PLUGIN`.

Warning: Enabling this feature is something of a judgment call. If enabled, passwords which don’t match the specified criteria will be rejected by MySQL with an error. This will cause issues if you use a weak password in conjunction with software which automatically configures MySQL user credentials, such as the Ubuntu packages for phpMyAdmin. It is safe to leave validation disabled, but you should always use strong, unique passwords for database credentials.

Answer `Y` for yes, or anything else to continue without enabling.

```
VALIDATE PASSWORD PLUGIN can be used to test
passwords
and improve security. It checks the strength of
password
and allows the users to set only those passwords
which are
secure enough. Would you like to setup VALIDATE
PASSWORD plugin?

Press y|Y for Yes, any other key for No:
```

If you've enabled validation, the script will also ask you to select a level of password validation. Keep in mind that if you enter 2 – for the strongest level – you will receive errors when attempting to set any password which does not contain numbers, upper and lowercase letters, and special characters, or which is based on common dictionary words.

```
There are three levels of password validation
policy:

LOW    Length >= 8
MEDIUM Length >= 8, numeric, mixed case, and
special characters
```

```
STRONG Length >= 8, numeric, mixed case, special
characters and dictionary                  file

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 1
```

Next, you’ll be asked to submit and confirm a root password:

```
Please set the password for root here.

New password:

Re-enter new password:
```

For the rest of the questions, you should press `Y` and hit the `ENTER` key at each prompt. This will remove some anonymous users and the test database, disable remote root logins, and load these new rules so that MySQL immediately respects the changes we have made.

Note that in Ubuntu systems running MySQL 5.7 (and later versions), the root MySQL user is set to authenticate using the `auth_socket` plugin by default rather than with a password. This allows for some greater security and usability in many cases, but it can also complicate things when you need to allow an external program (e.g., phpMyAdmin) to access the user.

If using the `auth_socket` plugin to access MySQL fits with your workflow, you can proceed to Step 3. If, however, you prefer to use a password when connecting to MySQL as root, you will need to switch its authentication method from `auth_socket` to `mysql_native_password`. To do this, open up the MySQL prompt from your terminal:

```
sudo mysql
```

Next, check which authentication method each of your MySQL user accounts use with the following command:

```
SELECT user,authentication_string,plugin,host FROM
mysql.user;
```

Output

```
+------------------+-------------------------------------------+---
--------------------+-----------+
| user             | authentication_string                     |
plugin                | host      |
+------------------+-------------------------------------------+---
--------------------+-----------+
| root             |                                           |
auth_socket           | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
mysql_native_password | localhost |
| debian-sys-maint | *CC744277A401A7D25BE1CA89AFF17BF607F876FF |
mysql_native_password | localhost |
+------------------+-------------------------------------------+---
--------------------+-----------+
4 rows in set (0.00 sec)
```

In this example, you can see that the root user does in fact authenticate using the `auth_socket` plugin. To configure the root account to authenticate with a password, run the following `ALTER USER` command. Be sure to change **password** to a strong password of your choosing:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH
mysql_native_password BY 'password';
```

Then, run `FLUSH PRIVILEGES` which tells the server to reload the grant tables and put your new changes into effect:

```
FLUSH PRIVILEGES;
```

Check the authentication methods employed by each of your users again to confirm that root no longer authenticates using the `auth_socket` plugin:

```
SELECT user,authentication_string,plugin,host FROM
mysql.user;
```

Output

```
+------------------+-------------------------------------------+---
--------------------+-----------+
| user             | authentication_string                     |
plugin                | host      |
+------------------+-------------------------------------------+---
--------------------+-----------+
| root             | *3636DACC8616D997782ADD0839F92C1571D6D78F |
mysql_native_password | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
mysql_native_password | localhost |
| debian-sys-maint | *CC744277A401A7D25BE1CA89AFF17BF607F876FF |
mysql_native_password | localhost |
+------------------+-------------------------------------------+---
--------------------+-----------+
4 rows in set (0.00 sec)
```

You can see in this example output that the root MySQL user now authenticates using a password. Once you confirm this on your own server, you can exit the MySQL shell:

```
exit
```

Note: After configuring your root MySQL user to authenticate with a password, you'll no longer be able to access MySQL with the `sudo mysql` command used previously. Instead, you must run the following:

```
mysql -u root -p
```

After entering the password you just set, you will see the MySQL prompt.

At this point, your database system is now set up and you can move on to installing PHP.

Step 3 – Installing PHP and Configuring Nginx to Use the PHP Processor

You now have Nginx installed to serve your pages and MySQL installed to store and manage your data. However, you still don't have anything that can generate dynamic content. This is where PHP comes into play.

Since Nginx does not contain native PHP processing like some other web servers, you will need to install `php-fpm`, which stands for "fastCGI process manager". We will tell Nginx to pass PHP requests to this software for processing.

Note: Depending on your cloud provider, you may need to add Ubuntu's `universe` repository, which includes free and open-source software maintained by the Ubuntu community, before installing the `php-fpm` package. You can do this by typing:

```
sudo add-apt-repository universe
```

Install the `php-fpm` module along with an additional helper package, `php-mysql`, which will allow PHP to communicate with your database backend. The installation will pull in the necessary PHP core files. Do this by typing:

```
sudo apt install php-fpm php-mysql
```

You now have all of the required LEMP stack components installed, but you still need to make a few configuration changes in order to tell Nginx to use the PHP processor for dynamic content.

This is done on the server block level (server blocks are similar to Apache's virtual hosts). To do this, open a new server block configuration file within the `/etc/nginx/sites-available/` directory. In this example, the new server block configuration file is named `example.com`, although you can name yours whatever you'd like:

```
sudo nano /etc/nginx/sites-available/example.com
```

By editing a new server block configuration file, rather than editing the default one, you'll be able to easily restore the default configuration if you ever need to.

Add the following content, which was taken and slightly modified from the default server block configuration file, to your new server block configuration file:

/etc/nginx/sites-available/example.com

```
server {
        listen 80;
        root /var/www/html;
        index index.php index.html index.htm index.nginx-debian.html;
        server_name example.com;

        location / {
                try_files $uri $uri/ =404;
        }

        location ~ \.php$ {
                include snippets/fastcgi-php.conf;
                fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;
        }

        location ~ /\.ht {
                deny all;
        }
}
```

Here’s what each of these directives and location blocks do:

- `listen` — Defines what port Nginx will listen on. In this case, it will listen on port `80`, the default port for HTTP.

- `root` — Defines the document root where the files served by the website are stored.
- `index` — Configures Nginx to prioritize serving files named `index.php` when an index file is requested, if they're available.
- `server_name` — Defines which server block should be used for a given request to your server. Point this directive to your server's domain name or public IP address.
- `location /` — The first location block includes a `try_files` directive, which checks for the existence of files matching a URI request. If Nginx cannot find the appropriate file, it will return a 404 error.
- `location ~ \.php$` — This location block handles the actual PHP processing by pointing Nginx to the `fastcgi-php.conf` configuration file and the `php7.2-fpm.sock` file, which declares what socket is associated with `php-fpm`.
- `location ~ /\.ht` — The last location block deals with `.htaccess` files, which Nginx does not process. By adding the `deny all` directive, if any `.htaccess` files happen to find their way into the document root they will not be served to visitors.

After adding this content, save and close the file. Enable your new server block by creating a symbolic link from your new server block configuration file (in the `/etc/nginx/sites-available/` directory) to the `/etc/nginx/sites-enabled/` directory:

```
sudo ln -s /etc/nginx/sites-available/example.com
/etc/nginx/sites-enabled/
```

Then, unlink the default configuration file from the `/sites-enabled/` directory:

```
sudo unlink /etc/nginx/sites-enabled/default
```

Note: If you ever need to restore the default configuration, you can do so by recreating the symbolic link, like this:

```
sudo ln -s /etc/nginx/sites-available/default
/etc/nginx/sites-enabled/
```

Test your new configuration file for syntax errors by typing:

```
sudo nginx -t
```

If any errors are reported, go back and recheck your file before continuing.

When you are ready, reload Nginx to make the necessary changes:

```
sudo systemctl reload nginx
```

This concludes the installation and configuration of your LEMP stack. However, it's prudent to confirm that all of the components can communicate with one another.

Step 4 – Creating a PHP File to Test Configuration

Your LEMP stack should now be completely set up. You can test it to validate that Nginx can correctly hand `.php` files off to the PHP processor.

To do this, use your text editor to create a test PHP file called `info.php` in your document root:

```
sudo nano /var/www/html/info.php
```

Enter the following lines into the new file. This is valid PHP code that will return information about your server:

/var/www/html/info.php

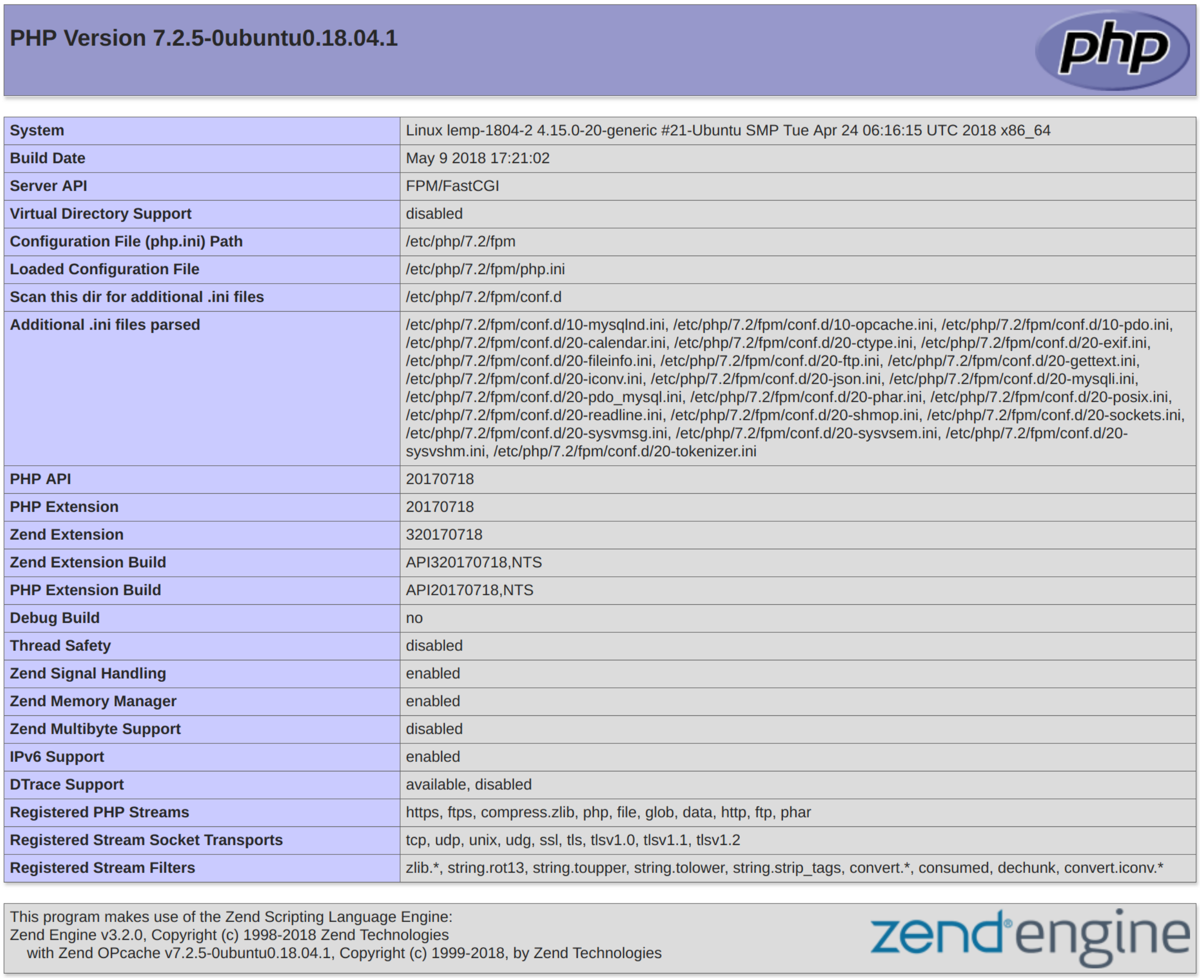
```
<?php

phpinfo();
```

When you are finished, save and close the file.

Now, you can visit this page in your web browser by visiting your server's domain name or public IP address followed by `/info.php`:

`http://`**`your_server_domain_or_IP`**`/info.php`

You should see a web page that has been generated by PHP with information about your server:



PHP Version 7.2.5-0ubuntu0.18.04.1

System	Linux lemp-1804-2 4.15.0-20-generic #21-Ubuntu SMP Tue Apr 24 06:16:15 UTC 2018 x86_64
Build Date	May 9 2018 17:21:02
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/fpm
Loaded Configuration File	/etc/php/7.2/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/fpm/conf.d
Additional .ini files parsed	/etc/php/7.2/fpm/conf.d/10-mysqlnd.ini, /etc/php/7.2/fpm/conf.d/10-opcache.ini, /etc/php/7.2/fpm/conf.d/10-pdo.ini, /etc/php/7.2/fpm/conf.d/20-calendar.ini, /etc/php/7.2/fpm/conf.d/20-ctype.ini, /etc/php/7.2/fpm/conf.d/20-exif.ini, /etc/php/7.2/fpm/conf.d/20-fileinfo.ini, /etc/php/7.2/fpm/conf.d/20-ftp.ini, /etc/php/7.2/fpm/conf.d/20-gettext.ini, /etc/php/7.2/fpm/conf.d/20-iconv.ini, /etc/php/7.2/fpm/conf.d/20-json.ini, /etc/php/7.2/fpm/conf.d/20-mysqli.ini, /etc/php/7.2/fpm/conf.d/20-pdo_mysql.ini, /etc/php/7.2/fpm/conf.d/20-phar.ini, /etc/php/7.2/fpm/conf.d/20-posix.ini, /etc/php/7.2/fpm/conf.d/20-readline.ini, /etc/php/7.2/fpm/conf.d/20-shmop.ini, /etc/php/7.2/fpm/conf.d/20-sockets.ini, /etc/php/7.2/fpm/conf.d/20-sysvmsg.ini, /etc/php/7.2/fpm/conf.d/20-sysvsem.ini, /etc/php/7.2/fpm/conf.d/20-sysvshm.ini, /etc/php/7.2/fpm/conf.d/20-tokenizer.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
    with Zend OPcache v7.2.5-0ubuntu0.18.04.1, Copyright (c) 1999-2018, by Zend Technologies

PHP page info

If you see a page that looks like this, you've set up PHP processing with Nginx successfully.

After verifying that Nginx renders the page correctly, it's best to remove the file you created as it can actually give unauthorized users some hints about your configuration that may help them try to break in. You can always regenerate this file if you need it later.

For now, remove the file by typing:

```
sudo rm /var/www/html/info.php
```

With that, you now have a fully-configured and functioning LEMP stack on your Ubuntu 18.04 server.

Conclusion

A LEMP stack is a powerful platform that will allow you to set up and serve nearly any website or application from your server.

There are a number of next steps you could take from here. For example, you should ensure that connections to your server are secured. To this end, you could secure your Nginx installation with Let’s Encrypt. By following this guide, you will acquire a free TLS/SSL certificate for your server, allowing it to serve content over HTTPS.

How To Secure Apache with Let’s Encrypt on Ubuntu 18.04

Written by Kathleen Juell and Erika Heidi

In this chapter you will learn how to secure Apache using TLS/SSL certificates that are issued by Let’s Encrypt and managed with Certbot. By the end of this chapter you will have an Apache web server set up with a `VirtualHost` for a custom domain that is secured using TLS/SSL encryption. You will also configure the certificate to renew automatically every ninety days using Certbot.

Let’s Encrypt is a Certificate Authority (CA) that provides an easy way to obtain and install free TLS/SSL certificates, thereby enabling encrypted HTTPS on web servers. It simplifies the process by providing a software client, Certbot, that attempts to automate most (if not all) of the required steps. Currently, the entire process of obtaining and installing a certificate is fully automated on both Apache and Nginx.

In this tutorial, you will use Certbot to obtain a free SSL certificate for Apache on Ubuntu 18.04 and set up your certificate to renew automatically.

This tutorial will use a separate Apache virtual host file instead of the default configuration file. We recommend creating new Apache virtual host files for each domain because it helps to avoid common mistakes and maintains the default files as a fallback configuration.

Prerequisites

To follow this tutorial, you will need:

- One Ubuntu 18.04 server set up by following this initial server setup for Ubuntu 18.04 tutorial, including a sudo non-root user and a firewall.
- A fully registered domain name. This tutorial will use your_domain as an example throughout. You can purchase a domain name on Namecheap, get one for free on Freenom, or use the domain registrar of your choice.
- Both of the following DNS records set up for your server. You can follow this introduction to DigitalOcean DNS for details on how to add them.
  - An A record with **`your_domain`** pointing to your server's public IP address.
  - An A record with `www.`**`your_domain`** pointing to your server's public IP address.
- Apache installed by following How To Install Apache on Ubuntu 18.04. Be sure that you have a virtual host file for your domain. This tutorial will use `/etc/apache2/sites-available/`**`your_domain`**`.conf` as an example.

Step 1 — Installing Certbot

The first step to using Let's Encrypt to obtain an SSL certificate is to install the Certbot software on your server.

Certbot is in very active development, so the Certbot packages provided by Ubuntu tend to be outdated. However, the Certbot developers maintain a Ubuntu software repository with up-to-date versions, so we'll use that repository instead.

First, add the repository:

```
sudo add-apt-repository ppa:certbot/certbot
```

You'll need to press `ENTER` to accept.

Install Certbot's Apache package with `apt`:

```
sudo apt install python-certbot-apache
```

Certbot is now ready to use, but in order for it to configure SSL for Apache, we need to verify some of Apache's configuration.

Step 2 — Set Up the SSL Certificate

Certbot needs to be able to find the correct virtual host in your Apache configuration for it to automatically configure SSL. Specifically, it does this by looking for a `ServerName` directive that matches the domain you request a certificate for.

If you followed the virtual host set up step in the Apache installation tutorial, you should have a VirtualHost block for your domain at `/etc/apache2/sites-available/your_domain.com.conf` with the `ServerName` directive already set appropriately.

To check, open the virtual host file for your domain using `nano` or your favorite text editor:

```
sudo nano /etc/apache2/sites-available/your_domain.conf
```

Find the existing `ServerName` line. It should look like this:

/etc/apache2/sites-available/your_domain.conf

```
...
ServerName your_domain;
...
```

If it does, exit your editor and move on to the next step.

If it doesn't, update it to match. Then save the file, quit your editor, and verify the syntax of your configuration edits:

```
sudo apache2ctl configtest
```

If you get an error, reopen the virtual host file and check for any typos or missing characters. Once your configuration file's syntax is correct, reload Apache to load the new configuration:

```
sudo systemctl reload apache2
```

Certbot can now find the correct VirtualHost block and update it.

Next, let's update the firewall to allow HTTPS traffic.

Step 3 — Allowing HTTPS Through the Firewall

If you have the `ufw` firewall enabled, as recommended by the prerequisite guides, you'll need to adjust the settings to allow for HTTPS traffic. Luckily, Apache registers a few profiles with `ufw` upon installation.

You can see the current setting by typing:

```
sudo ufw status
```

It will probably look like this, meaning that only HTTP traffic is allowed to the web server:

Output

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Apache                     ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache (v6)                ALLOW       Anywhere (v6)
```

To additionally let in HTTPS traffic, allow the Apache Full profile and delete the redundant Apache profile allowance:

```
sudo ufw allow 'Apache Full'
sudo ufw delete allow 'Apache'
```

Your status should now look like this:

```
sudo ufw status
```

Output

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Apache Full                ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache Full (v6)           ALLOW       Anywhere (v6)
```

Next, let’s run Certbot and fetch our certificates.

Step 4 — Obtaining an SSL Certificate

Certbot provides a variety of ways to obtain SSL certificates through plugins. The Apache plugin will take care of reconfiguring Apache and reloading the config whenever necessary. To use this plugin, type the following:

```
sudo certbot --apache -d your_domain -d
www.your_domain
```

This runs `certbot` with the `--apache` plugin, using `-d` to specify the names you’d like the certificate to be valid for.

If this is your first time running `certbot`, you will be prompted to enter an email address and agree to the terms of service. After doing so, `certbot` will communicate with the Let’s Encrypt server, then run a challenge to verify that you control the domain you’re requesting a certificate for.

If that’s successful, `certbot` will ask how you’d like to configure your HTTPS settings:

Output

```
Please choose whether or not to redirect HTTP traffic to HTTPS,
removing HTTP access.
--------------------------------------------------------------------
------------
1: No redirect - Make no further changes to the webserver
configuration.
2: Redirect - Make all requests redirect to secure HTTPS access.
Choose this for
new sites, or if you're confident your site works on HTTPS. You can
undo this
change by editing your web server's configuration.
--------------------------------------------------------------------
------------
Select the appropriate number [1-2] then [enter] (press 'c' to
cancel):
```

Select your choice then hit `ENTER`. The configuration will be updated, and Apache will reload to pick up the new settings. `certbot` will wrap up with a message telling you the process was successful and where your certificates are stored:

Output

```
IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/your_domain/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/your_domain/privkey.pem
   Your cert will expire on 2018-07-23. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot
again
   with the "certonly" option. To non-interactively renew *all* of
   your certificates, run "certbot renew"
 - Your account credentials have been saved in your Certbot
   configuration directory at /etc/letsencrypt. You should make a
   secure backup of this folder now. This configuration directory
will
   also contain certificates and private keys obtained by Certbot
so
   making regular backups of this folder is ideal.
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:
https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

Your certificates are downloaded, installed, and loaded. Try reloading your website using `https://` and notice your browser's security indicator. It should indicate that the site is properly secured, usually with a green lock icon. If you test your server using the SSL Labs Server Test, it will get an A grade.

Let’s finish by testing the renewal process.

Step 5 — Verifying Certbot Auto-Renewal

Let’s Encrypt’s certificates are only valid for ninety days. This is to encourage users to automate their certificate renewal process. The `certbot` package we installed takes care of this for us by adding a renew script to `/etc/cron.d`. This script runs twice a day and will automatically renew any certificate that’s within thirty days of expiration.

To test the renewal process, you can do a dry run with `certbot`:

```
sudo certbot renew --dry-run
```

If you see no errors, you’re all set. When necessary, Certbot will renew your certificates and reload Apache to pick up the changes. If the automated renewal process ever fails, Let’s Encrypt will send a message to the email you specified, warning you when your certificate is about to expire.

Conclusion

In this tutorial, you installed the Let’s Encrypt client `certbot`, downloaded SSL certificates for your domain, configured Apache to use these certificates, and set up automatic certificate renewal. If you have further questions about using Certbot, their documentation is a good place to start.

How To Secure Nginx with Let’s Encrypt on Ubuntu 18.04

Written by Hazel Virdó and Kathleen Juell

In this chapter you will learn how to secure Nginx using TLS/SSL certificates that are issued by Let’s Encrypt and managed with Certbot. By the end of this chapter you will have an Nginx web server set up with a server block for a custom domain that is secured using TLS/SSL encryption. You will also configure the certificate to renew automatically every ninety days using Certbot.

A previous version of this tutorial was written by Hazel Virdó

Let’s Encrypt is a Certificate Authority (CA) that provides an easy way to obtain and install free TLS/SSL certificates, thereby enabling encrypted HTTPS on web servers. It simplifies the process by providing a software client, Certbot, that attempts to automate most (if not all) of the required steps. Currently, the entire process of obtaining and installing a certificate is fully automated on both Apache and Nginx.

In this tutorial, you will use Certbot to obtain a free SSL certificate for Nginx on Ubuntu 18.04 and set up your certificate to renew automatically.

This tutorial will use a separate Nginx server block file instead of the default file. We recommend creating new Nginx server block files for each domain because it helps to avoid common mistakes and maintains the default files as a fallback configuration.

Prerequisites

To follow this tutorial, you will need:

- One Ubuntu 18.04 server set up by following this initial server setup for Ubuntu 18.04 tutorial, including a sudo non-root user and a firewall.
- A fully registered domain name. This tutorial will use example.com throughout. You can purchase a domain name on Namecheap, get one for free on Freenom, or use the domain registrar of your choice.
- Both of the following DNS records set up for your server. You can follow this introduction to DigitalOcean DNS for details on how to add them.
    - An A record with **`example.com`** pointing to your server's public IP address.
    - An A record with `www.`**`example.com`** pointing to your server's public IP address.
- Nginx installed by following How To Install Nginx on Ubuntu 18.04. Be sure that you have a server block for your domain. This tutorial will use `/etc/nginx/sites-available/`**`example.com`** as an example.

Step 1 — Installing Certbot

The first step to using Let's Encrypt to obtain an SSL certificate is to install the Certbot software on your server.

Certbot is in very active development, so the Certbot packages provided by Ubuntu tend to be outdated. However, the Certbot developers maintain a Ubuntu software repository with up-to-date versions, so we'll use that repository instead.

First, add the repository:

```
sudo add-apt-repository ppa:certbot/certbot
```

You'll need to press ENTER to accept.

Install Certbot's Nginx package with apt:

```
sudo apt install python-certbot-nginx
```

Certbot is now ready to use, but in order for it to configure SSL for Nginx, we need to verify some of Nginx's configuration.

Step 2 — Confirming Nginx's Configuration

Certbot needs to be able to find the correct server block in your Nginx configuration for it to be able to automatically configure SSL. Specifically, it does this by looking for a server_name directive that matches the domain you request a certificate for.

If you followed the server block set up step in the Nginx installation tutorial, you should have a server block for your domain at /etc/nginx/sites-available/**example.com** with the server_name directive already set appropriately.

To check, open the server block file for your domain using nano or your favorite text editor:

```
sudo nano /etc/nginx/sites-available/example.com
```

Find the existing server_name line. It should look like this:

/etc/nginx/sites-available/example.com

```
...
server_name example.com www.example.com;
...
```

If it does, exit your editor and move on to the next step.

If it doesn't, update it to match. Then save the file, quit your editor, and verify the syntax of your configuration edits:

```
sudo nginx -t
```

If you get an error, reopen the server block file and check for any typos or missing characters. Once your configuration file's syntax is correct, reload Nginx to load the new configuration:

```
sudo systemctl reload nginx
```

Certbot can now find the correct `server` block and update it.

Next, let's update the firewall to allow HTTPS traffic.

Step 3 — Allowing HTTPS Through the Firewall

If you have the `ufw` firewall enabled, as recommended by the prerequisite guides, you'll need to adjust the settings to allow for HTTPS traffic. Luckily, Nginx registers a few profiles with `ufw` upon installation.

You can see the current setting by typing:

```
sudo ufw status
```

It will probably look like this, meaning that only HTTP traffic is allowed to the web server:

Output

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Nginx HTTP                 ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Nginx HTTP (v6)            ALLOW       Anywhere (v6)
```

To additionally let in HTTPS traffic, allow the Nginx Full profile and delete the redundant Nginx HTTP profile allowance:

```
sudo ufw allow 'Nginx Full'
sudo ufw delete allow 'Nginx HTTP'
```

Your status should now look like this:

```
sudo ufw status
```

Output

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Nginx Full                 ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Nginx Full (v6)            ALLOW       Anywhere (v6)
```

Next, let’s run Certbot and fetch our certificates.

Step 4 — Obtaining an SSL Certificate

Certbot provides a variety of ways to obtain SSL certificates through plugins. The Nginx plugin will take care of reconfiguring Nginx and reloading the config whenever necessary. To use this plugin, type the following:

```
sudo certbot --nginx -d example.com -d www.example.com
```

This runs `certbot` with the `--nginx` plugin, using `-d` to specify the names we’d like the certificate to be valid for.

If this is your first time running `certbot`, you will be prompted to enter an email address and agree to the terms of service. After doing so, `certbot` will communicate with the Let’s Encrypt server, then run a challenge to verify that you control the domain you’re requesting a certificate for.

If that’s successful, `certbot` will ask how you’d like to configure your HTTPS settings.

Output

```
Please choose whether or not to redirect HTTP traffic to HTTPS,
removing HTTP access.
--------------------------------------------------------------------
------------
1: No redirect - Make no further changes to the webserver
configuration.
2: Redirect - Make all requests redirect to secure HTTPS access.
Choose this for
new sites, or if you're confident your site works on HTTPS. You can
undo this
change by editing your web server's configuration.
--------------------------------------------------------------------
------------
Select the appropriate number [1-2] then [enter] (press 'c' to
cancel):
```

Select your choice then hit `ENTER`. The configuration will be updated, and Nginx will reload to pick up the new settings. `certbot` will wrap up with a message telling you the process was successful and where your certificates are stored:

Output

```
IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/example.com/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/example.com/privkey.pem
   Your cert will expire on 2018-07-23. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot
again
   with the "certonly" option. To non-interactively renew *all* of
   your certificates, run "certbot renew"
 - Your account credentials have been saved in your Certbot
   configuration directory at /etc/letsencrypt. You should make a
   secure backup of this folder now. This configuration directory
will
   also contain certificates and private keys obtained by Certbot
so
   making regular backups of this folder is ideal.
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:
https://letsencrypt.org/donate
   Donating to EFF:                   https://eff.org/donate-le
```

Your certificates are downloaded, installed, and loaded. Try reloading your website using `https://` and notice your browser's security indicator. It should indicate that the site is properly secured, usually with a green lock icon. If you test your server using the SSL Labs Server Test, it will get an A grade.

Let's finish by testing the renewal process.

Step 5 — Verifying Certbot Auto-Renewal

Let's Encrypt's certificates are only valid for ninety days. This is to encourage users to automate their certificate renewal process. The `certbot` package we installed takes care of this for us by adding a renew script to `/etc/cron.d`. This script runs twice a day and will automatically renew any certificate that's within thirty days of expiration.

To test the renewal process, you can do a dry run with `certbot`:

```
sudo certbot renew --dry-run
```

If you see no errors, you're all set. When necessary, Certbot will renew your certificates and reload Nginx to pick up the changes. If the automated renewal process ever fails, Let's Encrypt will send a message to the email you specified, warning you when your certificate is about to expire.

Conclusion

In this tutorial, you installed the Let's Encrypt client `certbot`, downloaded SSL certificates for your domain, configured Nginx to use these certificates, and set up automatic certificate renewal. If you have further questions about using Certbot, their documentation is a good place to start.

How To Set Up a Firewall with UFW on Ubuntu 18.04

Written by Brian Boucheron

In this chapter you will learn how to set up a firewall with UFW on Ubuntu 18.04. UFW, or Uncomplicated Firewall, is an interface to `iptables` that removes much of the complexity of configuring firewall rules by hand.

This chapter explains how to enable UFW and how you can create, modify and delete firewall rules. It also demonstrates how to use UFW with IPv6 enabled network interfaces. By the end of this chapter you will have a set of firewall rules that allows SSH traffic. You will also be able to add or remove rules for other kinds of traffic to and from servers.

A previous version of this tutorial was written by Hazel Virdó

UFW, or Uncomplicated Firewall, is an interface to `iptables` that is geared towards simplifying the process of configuring a firewall. While `iptables` is a solid and flexible tool, it can be difficult for beginners to learn how to use it to properly configure a firewall. If you’re looking to get started securing your network, and you’re not sure which tool to use, UFW may be the right choice for you.

This tutorial will show you how to set up a firewall with UFW on Ubuntu 18.04.

Prerequisites

To follow this tutorial, you will need:

- One Ubuntu 18.04 server with a sudo non-root user, which you can set up by following Steps 1–3 in the Initial Server Setup with Ubuntu 18.04 tutorial.

UFW is installed by default on Ubuntu. If it has been uninstalled for some reason, you can install it with `sudo apt install ufw`.

Step 1 — Using IPv6 with UFW (Optional)

This tutorial is written with IPv4 in mind, but will work for IPv6 as well as long as you enable it. If your Ubuntu server has IPv6 enabled, ensure that UFW is configured to support IPv6 so that it will manage firewall rules for IPv6 in addition to IPv4. To do this, open the UFW configuration with `nano` or your favorite editor.

```
sudo nano /etc/default/ufw
```

Then make sure the value of `IPV6` is `yes`. It should look like this:

/etc/default/ufw excerpt

```
IPV6=yes
```

Save and close the file. Now, when UFW is enabled, it will be configured to write both IPv4 and IPv6 firewall rules. However, before enabling UFW, we will want to ensure that your firewall is configured to allow you to connect via SSH. Let’s start with setting the default policies.

Step 2 — Setting Up Default Policies

If you’re just getting started with your firewall, the first rules to define are your default policies. These rules control how to handle traffic that does not explicitly match any other rules. By default, UFW is set to deny all incoming connections and allow all outgoing connections. This means anyone trying to reach your server would not be able to connect, while any application within the server would be able to reach the outside world.

Let’s set your UFW rules back to the defaults so we can be sure that you’ll be able to follow along with this tutorial. To set the defaults used by UFW, use these commands:

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
```

These commands set the defaults to deny incoming and allow outgoing connections. These firewall defaults alone might suffice for a personal computer, but servers typically need to respond to incoming requests from outside users. We’ll look into that next.

Step 3 — Allowing SSH Connections

If we enabled our UFW firewall now, it would deny all incoming connections. This means that we will need to create rules that explicitly allow legitimate incoming connections — SSH or HTTP connections, for example — if we want our server to respond to those types of requests. If you’re using a cloud server, you will probably want to allow incoming SSH connections so you can connect to and manage your server.

To configure your server to allow incoming SSH connections, you can use this command:

```
sudo ufw allow ssh
```

This will create firewall rules that will allow all connections on port `22`, which is the port that the SSH daemon listens on by default. UFW knows what port `allow ssh` means because it's listed as a service in the `/etc/services` file.

However, we can actually write the equivalent rule by specifying the port instead of the service name. For example, this command works the same as the one above:

```
sudo ufw allow 22
```

If you configured your SSH daemon to use a different port, you will have to specify the appropriate port. For example, if your SSH server is listening on port `2222`, you can use this command to allow connections on that port:

```
sudo ufw allow 2222
```

Now that your firewall is configured to allow incoming SSH connections, we can enable it.

Step 4 — Enabling UFW

To enable UFW, use this command:

```
sudo ufw enable
```

You will receive a warning that says the command may disrupt existing SSH connections. We already set up a firewall rule that allows SSH connections, so it should be fine to continue. Respond to the prompt with `y` and hit `ENTER`.

The firewall is now active. Run the `sudo ufw status verbose` command to see the rules that are set. The rest of this tutorial covers how to use UFW in more detail, like allowing or denying different kinds of connections.

Step 5 — Allowing Other Connections

At this point, you should allow all of the other connections that your server needs to respond to. The connections that you should allow depends on your specific needs. Luckily, you already know how to write rules that allow connections based on a service name or port; we already did this for SSH on port `22`. You can also do this for:

- HTTP on port 80, which is what unencrypted web servers use, using `sudo ufw allow http` or `sudo ufw allow 80`
- HTTPS on port 443, which is what encrypted web servers use, using `sudo ufw allow https` or `sudo ufw allow 443`

There are several others ways to allow other connections, aside from specifying a port or known service.

Specific Port Ranges

You can specify port ranges with UFW. Some applications use multiple ports, instead of a single port.

For example, to allow X11 connections, which use ports `6000-6007`, use these commands:

```
sudo ufw allow 6000:6007/tcp
sudo ufw allow 6000:6007/udp
```

When specifying port ranges with UFW, you must specify the protocol (`tcp` or `udp`) that the rules should apply to. We haven’t mentioned this before because not specifying the protocol automatically allows both protocols, which is OK in most cases.

Specific IP Addresses

When working with UFW, you can also specify IP addresses. For example, if you want to allow connections from a specific IP address, such as a work or home IP address of `203.0.113.4`, you need to specify `from`, then the IP address:

```
sudo ufw allow from 203.0.113.4
```

You can also specify a specific port that the IP address is allowed to connect to by adding `to any port` followed by the port number. For example, If you want to allow `203.0.113.4` to connect to port `22` (SSH), use this command:

```
sudo ufw allow from 203.0.113.4 to any port 22
```

Subnets

If you want to allow a subnet of IP addresses, you can do so using CIDR notation to specify a netmask. For example, if you want to allow all of the IP addresses ranging from `203.0.113.1` to `203.0.113.254` you could use this command:

```
sudo ufw allow from 203.0.113.0/24
```

Likewise, you may also specify the destination port that the subnet `203.0.113.0/24` is allowed to connect to. Again, we'll use port `22` (SSH) as an example:

```
sudo ufw allow from 203.0.113.0/24 to any port 22
```

Connections to a Specific Network Interface

If you want to create a firewall rule that only applies to a specific network interface, you can do so by specifying "allow in on" followed by the name of the network interface.

You may want to look up your network interfaces before continuing. To do so, use this command:

```
ip addr
```

Output Excerpt

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state
. . .
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group
default
. . .
```

The highlighted output indicates the network interface names. They are typically named something like eth0 or enp3s2.

So, if your server has a public network interface called eth0, you could allow HTTP traffic (port 80) to it with this command:

```
sudo ufw allow in on eth0 to any port 80
```

Doing so would allow your server to receive HTTP requests from the public internet.

Or, if you want your MySQL database server (port 3306) to listen for connections on the private network interface eth1, for example, you could use this command:

```
sudo ufw allow in on eth1 to any port 3306
```

This would allow other servers on your private network to connect to your MySQL database.

Step 6 — Denying Connections

If you haven't changed the default policy for incoming connections, UFW is configured to deny all incoming connections. Generally, this simplifies the process of creating a secure firewall policy by requiring you to create rules that explicitly allow specific ports and IP addresses through.

However, sometimes you will want to deny specific connections based on the source IP address or subnet, perhaps because you know that your server is being attacked from there. Also, if you want to change your default incoming policy to allow (which is not recommended), you would need to create deny rules for any services or IP addresses that you don't want to allow connections for.

To write deny rules, you can use the commands described above, replacing allow with deny.

For example, to deny HTTP connections, you could use this command:

```
sudo ufw deny http
```

Or if you want to deny all connections from `203.0.113.4` you could use this command:

```
sudo ufw deny from 203.0.113.4
```

Now let's take a look at how to delete rules.

Step 7 — Deleting Rules

Knowing how to delete firewall rules is just as important as knowing how to create them. There are two different ways to specify which rules to delete: by rule number or by the actual rule (similar to how the rules were specified when they were created). We'll start with the delete by rule number method because it is easier.

By Rule Number

If you’re using the rule number to delete firewall rules, the first thing you’ll want to do is get a list of your firewall rules. The UFW status command has an option to display numbers next to each rule, as demonstrated here:

```
sudo ufw status numbered
```

Numbered Output:

```
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 22                         ALLOW IN    15.15.15.0/24
[ 2] 80                         ALLOW IN    Anywhere
```

If we decide that we want to delete rule 2, the one that allows port 80 (HTTP) connections, we can specify it in a UFW delete command like this:

```
sudo ufw delete 2
```

This would show a confirmation prompt then delete rule 2, which allows HTTP connections. Note that if you have IPv6 enabled, you would want to delete the corresponding IPv6 rule as well.

By Actual Rule

The alternative to rule numbers is to specify the actual rule to delete. For example, if you want to remove the `allow http` rule, you could write it like this:

```
sudo ufw delete allow http
```

You could also specify the rule by `allow 80`, instead of by service name:

```
sudo ufw delete allow 80
```

This method will delete both IPv4 and IPv6 rules, if they exist.

Step 8 — Checking UFW Status and Rules

At any time, you can check the status of UFW with this command:

```
sudo ufw status verbose
```

If UFW is disabled, which it is by default, you’ll see something like this:

Output

```
Status: inactive
```

If UFW is active, which it should be if you followed Step 3, the output will say that it’s active and it will list any rules that are set. For example, if the firewall is set to allow SSH (port `22`) connections from anywhere, the output might look something like this:

Output

```
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW IN    Anywhere
```

Use the `status` command if you want to check how UFW has configured the firewall.

Step 9 — Disabling or Resetting UFW (optional)

If you decide you don’t want to use UFW, you can disable it with this command:

```
sudo ufw disable
```

Any rules that you created with UFW will no longer be active. You can always run `sudo ufw enable` if you need to activate it later.

If you already have UFW rules configured but you decide that you want to start over, you can use the reset command:

```
sudo ufw reset
```

This will disable UFW and delete any rules that were previously defined. Keep in mind that the default policies won’t change to their original settings, if you modified them at any point. This should give you a fresh start with UFW.

Conclusion

Your firewall is now configured to allow (at least) SSH connections. Be sure to allow any other incoming connections that your server, while limiting any unnecessary connections, so your server will be functional and secure.

To learn about more common UFW configurations, check out the UFW Essentials: Common Firewall Rules and Commands tutorial.

How to Use Ansible to Automate Initial Server Setup on Ubuntu 18.04

Written by Erika Heidi

This guide explains how to use Ansible to automate the intial server setup steps are described in Chapter 5 of this book.

Instead of manually adding a user and creating firewall rules, this tutorial demonstrates how to use Ansible to build an intial server configuration. Ansible will create a new user and grant it administrative privileges, and will also set up the UFW firewall to allow SSH traffic to the server.

Server automation now plays an essential role in systems administration, due to the disposable nature of modern application environments. Configuration management tools such as Ansible are typically used to streamline the process of automating server setup by establishing standard procedures for new servers while also reducing human error associated with manual setups.

Ansible offers a simple architecture that doesn’t require special software to be installed on nodes. It also provides a robust set of features and built-in modules which facilitate writing automation scripts.

This guide explains how to use Ansible to automate the steps contained in our Initial Server Setup Guide for Ubuntu 18.04 servers.

Prerequisites

In order to execute the automated setup provided by the playbook we’re discussing in this guide, you’ll need:

- One Ansible control node: an Ubuntu 18.04 machine with Ansible installed and configured to connect to your Ansible hosts using SSH keys. Make sure the control node has a regular user with sudo permissions and a firewall enabled, as explained in our Initial Server Setup guide. To set up Ansible, please follow our guide on How to Install and Configure Ansible on Ubuntu 18.04.
- One or more Ansible Hosts: one or more remote Ubuntu 18.04 servers.

Before proceeding, you first need to make sure your Ansible control node is able to connect and execute commands on your Ansible host(s). For a connection test, please check step 3 of How to Install and Configure Ansible on Ubuntu 18.04.

What Does this Playbook Do?

This Ansible playbook provides an alternative to manually running through the procedure outlined in the Ubuntu 18.04 initial server setup guide and the guide on setting up SSH keys on Ubuntu 18.04.

Running this playbook will perform the following actions on your Ansible hosts:

1. Install `aptitude`, which is preferred by Ansible as an alternative to the `apt` package manager.
2. Create the administrative group wheels and configure it for passwordless sudo.
3. Create a new sudo user.
4. Copy a local SSH public key and include it in the `authorized_keys` file for the new administrative user on the remote host.
5. Disable password-based authentication for the root user.
6. Install system packages.
7. Configure the UFW firewall to only allow SSH connections and deny any other requests.

Once the playbook has finished running, you'll have a new user which you can use to log in to the server.

How to Use this Playbook

The first thing you'll need to do is obtain the initial server setup playbook and its dependencies from the do-community/ansible-playbooks repository. We'll clone this repository to a local folder inside the Ansible control node.

If this is your first time using the `do-community/ansible-playbooks` repository, you should start by cloning the repository to your controller node with:

```
cd ~
git clone https://github.com/do-community/ansible-playbooks.git
cd ansible-playbooks
```

In case you have cloned this repository before while following a different guide, access your existing `ansible-playbooks` copy and run a `git pull` command to make sure you have updated contents:

```
cd ~/ansible-playbooks
git pull
```

The files we're interested in are located inside the `setup_ubuntu1804` folder, which has the following structure:

```
setup_ubuntu1804
├── playbook.yml
└── vars
    └── default.yml
```

Here is what each of these files are:

- `vars/default.yml`: Variable file for customizing playbook settings.
- `playbook.yml`: The playbook file, containing the tasks to be executed on the remote server(s).

We'll edit the playbook's variable file to customize its values. Access the `setup_ubuntu1804` directory and open the `vars/default.yml` file using your command line editor of choice:

```
cd setup_ubuntu1804
nano vars/default.yml
```

This file contains a few variables that require your attention:

vars/default.yml

```
---
create_user: sammy
copy_local_key: "{{ lookup('file', lookup('env','HOME') +
'/.ssh/id_rsa.pub') }}"
sys_packages: [ 'curl', 'vim', 'git', 'ufw']
```

The following list contains a brief explanation of each of these variables and how you might want to change them:

- `create_user`: The name of the sudo user that will be created. In this example, we will be using sammy.
- `copy_local_key`: The path to a local SSH public key file that should be copied to the remote server and added as `authorized_key` for the new sudo user. The default value uses the [`lookup`] plugin to obtain the full path to the default public key for the current system user at the Ansible control node.
- `sys_packages`: An array containing the list of packages you wish to install on your hosts as part of your initial server setup. In this

example, we are going to make sure the packages curl, vim, git and ufw are present.

Once you're done updating the variables inside vars/default.yml, save and close this file. If you used nano, do so by pressing CTRL + X, Y, then ENTER.

You're now ready to run this playbook on one or more servers. Most playbooks are configured to be executed on every server in your inventory, by default. We can use the -l flag to make sure that only a subset of servers, or a single server, is affected by the playbook. We can also use the -u flag to specify which user on the remote server we're using to connect and execute the playbook commands on the remote hosts.

To execute the playbook only on **server1**, connecting as **root**, you can use the following command:

```
ansible-playbook playbook.yml -l server1 -u root
```

You will get output similar to this:

Output

```
PLAY [all]
*******************************************************************
**********************************************************

TASK [Gathering Facts]
*******************************************************************
**********************************************
ok: [server1]

TASK [Install Prerequisites]
*******************************************************************
****************************************
changed: [server1]

TASK [Make sure we have a 'wheel' group]
*******************************************************************
****************************
changed: [server1]

TASK [Allow 'wheel' group to have passwordless sudo]
*******************************************************************
****************
changed: [server1]

TASK [Create a new regular user with sudo privileges]
```

```
********************************************************************
***************
changed: [server1]

TASK [Set authorized key for remote user]
********************************************************************
***************************
changed: [server1]

TASK [Disable password authentication for root]
********************************************************************
*********************
changed: [server1]

TASK [Update apt]
********************************************************************
***************************************************
changed: [server1]

TASK [Install required system packages]
********************************************************************
*****************************
ok: [server1]

TASK [UFW - Allow SSH connections]
********************************************************************
**********************************
```

```
changed: [server1]

TASK [UFW - Deny all other incoming traffic by default]
*********************************************************************
*************
changed: [server1]

PLAY RECAP
*********************************************************************
***********************************************************
server1                  : ok=11   changed=9    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
```

Note: For more information on how to run Ansible playbooks, check our Ansible Cheat Sheet Guide.

Once the playbook execution is finished, you'll be able to log in to the server with:

```
ssh sammy@server_host_or_IP
```

Remember to replace **sammy** with the user defined by the `create_user` variable, and **server_host_or_IP** with your server's hostname or IP address.

In case you have changed the `copy_local_key` variable to point to a custom SSH key (not your current system user's one), you'll need to provide an extra parameter specifying the location of its private key counterpart when connecting via SSH as the new user:

```
ssh sammy@server_host_or_IP -i
~/.ssh/ansible_controller_key
```

After logging in to the server, you can check the UFW firewall's active rules to confirm that it's properly configured:

```
[environment second]
sudo ufw status
```

You should get output similar to this:

Output

```
[environment second]
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
```

This means that the UFW firewall has successfully been enabled. Since this was the last task in the playbook, it confirms that the playbook was fully executed on this server.

The Playbook Contents

You can find the initial server setup playbook featured in this tutorial in the ansible-playbooks repository, within the DigitalOcean Community Playbooks. To copy or download the script contents directly, click the Raw button towards the top of each script.

The full contents of the playbook as well as its associated files are also included here for your convenience.

vars/default.yml

The `default.yml` variable file contains values that will be used within the playbook tasks, such as the name of the user that will be created and the packages that should be installed as part of the initial server setup.

vars/default.yml

```
---
create_user: sammy
copy_local_key: "{{ lookup('file', lookup('env','HOME') + '/.ssh/id_rsa.pub') }}"
sys_packages: [ 'curl', 'vim', 'git', 'ufw']
```

playbook.yml

The `playbook.yml` file is where all tasks from this setup are defined. It starts by defining the group of servers that should be the target of this setup (`all`), after which it uses `become: true` to define that tasks should be executed with privilege escalation (`sudo`) by default. Then, it includes the `vars/default.yml` variable file to load configuration options.

playbook.yml

```
---
- hosts: all
  become: true
  vars_files:
    - vars/default.yml

  tasks:
    - name: Install Prerequisites
      apt: name=aptitude update_cache=yes state=latest force_apt_get

  # Sudo Group Setup
    - name: Make sure we have a 'wheel' group
      group:
        name: wheel
        state: present

    - name: Allow 'wheel' group to have passwordless sudo
     lineinfile:
        path: /etc/sudoers
        state: present
        regexp: '^%wheel'
        line: '%wheel ALL=(ALL) NOPASSWD: ALL'
        validate: '/usr/sbin/visudo -cf %s'

  # User + Key Setup
```

```
    - name: Create a new regular user with sudo privileges
      user:
        name: "{{ create_user }}"
        state: present
        groups: wheel
        append: true
        create_home: true
        shell: /bin/bash

    - name: Set authorized key for remote user
      authorized_key:
        user: "{{ create_user }}"
        state: present
        key: "{{ copy_local_key }}"

    - name: Disable password authentication for root
      lineinfile:
        path: /etc/ssh/sshd_config
        state: present
        regexp: '^#?PermitRootLogin'
        line: 'PermitRootLogin prohibit-password'

  # Install Packages
    - name: Update apt
      apt: update_cache=yes

    - name: Install required system packages
      apt: name={{ sys_packages }} state=latest

 # UFW Setup
    - name: UFW - Allow SSH connections
      ufw:
        rule: allow
        name: OpenSSH

    - name: UFW - Deny all other incoming traffic by default
      ufw:
        state: enabled
        policy: deny
        direction: incoming
```

Feel free to modify this playbook or include new tasks to best suit your individual needs within your own workflow.

Conclusion

Automating the initial server setup can save you time, while also making sure your servers will follow a standard configuration that can be improved and customized to your needs. With the distributed nature of modern applications and the need for more consistency between different staging environments, automation like this becomes a necessity.

In this guide, we demonstrated how to use Ansible for automating the initial tasks that should be executed on a fresh server, such as creating a non-root user with sudo access, enabling UFW and disabling remote password-based root login.

If you'd like to include new tasks in this playbook to further customize your initial server setup, please refer to our introductory Ansible guide Configuration Management 101: Writing Ansible Playbooks. You can also check our guide on How to Use Ansible Roles to Abstract your Infrastructure Environment.

How to Use Ansible to Install and Set Up LAMP on Ubuntu 18.04

Written by Erika Heidi

This guide explains how to use Ansible to automate setting up a Linux, Apache, MySQL, and PHP server, commonly referred to as the LAMP stack. It is a companion to Chapter 8, How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04.

Instead of manually running `apt` commands to set up the LAMP stack, this tutorial demonstrates how to use Ansible to configure a server. Ansible will install all the required LAMP packages, create and enable a new VirtualHost for Apache to use, secure MySQL, set up the UFW firewall to allow traffic to the server, and finally add a test PHP script to demonstrate that all the components of the LAMP stack are working correctly.

Server automation now plays an essential role in systems administration, due to the disposable nature of modern application environments. Configuration management tools such as Ansible are typically used to streamline the process of automating server setup by establishing standard procedures for new servers while also reducing human error associated with manual setups.

Ansible offers a simple architecture that doesn’t require special software to be installed on nodes. It also provides a robust set of features and built-in modules which facilitate writing automation scripts.

This guide explains how to use Ansible to automate the steps contained in our guide on How To Install Linux, Apache, MySQL and PHP (LAMP) on Ubuntu 18.04. A “LAMP” stack is a group of open-source software that is typically installed together to enable a server to host dynamic websites and web apps. This term is actually an acronym which represents the Linux operating system, with the Apache web server. The site data is stored in a MySQL database, and dynamic content is processed by PHP.

Prerequisites

In order to execute the automated setup provided by the playbook we’re discussing in this guide, you’ll need:

- One Ansible control node: an Ubuntu 18.04 machine with Ansible installed and configured to connect to your Ansible hosts using SSH keys. Make sure the control node has a regular user with sudo permissions and a firewall enabled, as explained in our Initial Server Setup guide. To set up Ansible, please follow our guide on How to Install and Configure Ansible on Ubuntu 18.04.
- One or more Ansible Hosts: one or more remote Ubuntu 18.04 servers previously set up following the guide on How to Use Ansible to Automate Initial Server Setup on Ubuntu 18.04.

Before proceeding, you first need to make sure your Ansible control node is able to connect and execute commands on your Ansible host(s). For a connection test, please check step 3 of How to Install and Configure Ansible on Ubuntu 18.04.

What Does this Playbook Do?

This Ansible playbook provides an alternative to manually running through the procedure outlined in our guide on How To Install Linux, Apache, MySQL and PHP (LAMP) on Ubuntu 18.04.

Running this playbook will perform the following actions on your Ansible hosts:

1. Install `aptitude`, which is preferred by Ansible as an alternative to the `apt` package manager.
2. Install the required LAMP packages.
3. Create a new Apache `VirtualHost` and set up a dedicated document root for that.
4. Enable the new `VirtualHost`.
5. Disable the default Apache website, when the disable_default variable is set to `true`.
6. Set the password for the MySQL root user.
7. Remove anonymous MySQL accounts and the test database.
8. Set up UFW to allow HTTP traffic on the configured port (`80` by default).
9. Set up a PHP test script using the provided template.

Once the playbook has finished running, you will have a web PHP environment running on top of Apache, based on the options you defined within your configuration variables.

How to Use this Playbook

The first thing we need to do is obtain the LAMP playbook and its dependencies from the do-community/ansible-playbooks repository. We need to clone this repository to a local folder inside the Ansible Control Node.

In case you have cloned this repository before while following a different guide, access your existing `ansible-playbooks` copy and run a `git pull` command to make sure you have updated contents:

```
cd ~/ansible-playbooks
git pull
```

If this is your first time using the `do-community/ansible-playbooks` repository, you should start by cloning the repository to your home folder with:

```
cd ~
git clone https://github.com/do-community/ansible-playbooks.git
cd ansible-playbooks
```

The files we're interested in are located inside the `lamp_ubuntu1804` folder, which has the following structure:

```
lamp_ubuntu1804
├── files
│   ├── apache.conf.j2
│   └── info.php.j2
├── vars
│   └── default.yml
├── playbook.yml
└── readme.md
```

Here is what each of these files are:

- `files/info.php.j2`: Template file for setting up a PHP test page on the web server's root
- `files/apache.conf.j2`: Template file for setting up the Apache VirtualHost.
- `vars/default.yml`: Variable file for customizing playbook settings.
- `playbook.yml`: The playbook file, containing the tasks to be executed on the remote server(s).
- `readme.md`: A text file containing information about this playbook.

We'll edit the playbook's variable file to customize the configurations of both MySQL and Apache. Access the `lamp_ubuntu1804` directory and open the `vars/default.yml` file using your command line editor of choice:

```
cd lamp_ubuntu1804
nano vars/default.yml
```

This file contains a few variables that require your attention:

vars/default.yml

```
---
mysql_root_password: "mysql_root_password"
app_user: "sammy"
http_host: "your_domain"
http_conf: "your_domain.conf"
http_port: "80"
disable_default: true
```

The following list contains a brief explanation of each of these variables and how you might want to change them:

- mysql_root_password: The desired password for the root MySQL account.
- app_user: A remote non-root user on the Ansible host that will be set as the owner of the application files.
- http_host: Your domain name.
- http_conf: The name of the configuration file that will be created within Apache.
- http_port: HTTP port for this virtual host, where 80 is the default.
- disable_default: Whether or not to disable the default website that comes with Apache.

Once you're done updating the variables inside vars/default.yml, save and close this file. If you used nano, do so by pressing CTRL + X, Y, then ENTER.

You're now ready to run this playbook on one or more servers. Most playbooks are configured to be executed on every server in your inventory, by default. We can use the -l flag to make sure that only a subset of servers, or a single server, is affected by the playbook. We can also use the -u flag to specify which user on the remote server we're using to connect and execute the playbook commands on the remote hosts.

To execute the playbook only on **server1**, connecting as **sammy**, you can use the following command:

```
ansible-playbook playbook.yml -l server1 -u sammy
```

You will get output similar to this:

Output

```
PLAY [all]
********************************************************************
**************************************
TASK [Gathering Facts]
********************************************************************
**************************************ok: [server1]

TASK [Install prerequisites]
********************************************************************
**************************************ok: [server1] =>
(item=aptitude)

...

TASK [UFW - Allow HTTP on port 80]
********************************************************************
**************************************
changed: [server1]

TASK [Sets Up PHP Info Page]
********************************************************************
**************************************
changed: [server1]

RUNNING HANDLER [Reload Apache]
```

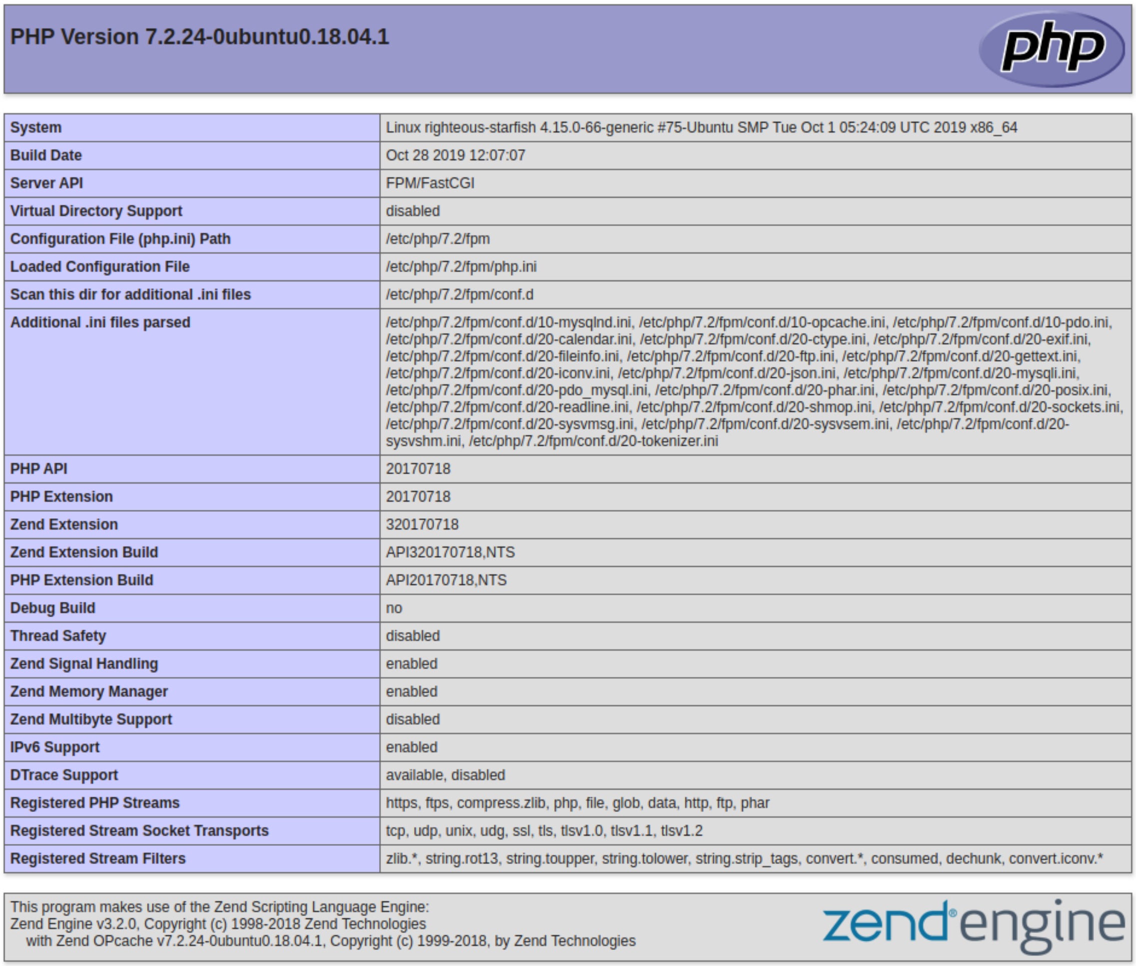
```
*******************************************************************
**************************************
changed: [server1]

RUNNING HANDLER [Restart Apache]
*******************************************************************
**************************************
changed: [server1]

PLAY RECAP
*******************************************************************
**************************************
server1                : ok=15    changed=11    unreachable=0
failed=0     skipped=0     rescued=0     ignored=0
```

Note: For more information on how to run Ansible playbooks, check our Ansible Cheat Sheet Guide.

When the playbook is finished running, go to your web browser and access the host or IP address of the server, as configured in the playbook variables, followed by `/info.php`:

`http://`**`server_host_or_IP`**`/info.php`

You will see a page like this:



PHP Version 7.2.24-0ubuntu0.18.04.1

System	Linux righteous-starfish 4.15.0-66-generic #75-Ubuntu SMP Tue Oct 1 05:24:09 UTC 2019 x86_64
Build Date	Oct 28 2019 12:07:07
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/fpm
Loaded Configuration File	/etc/php/7.2/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/fpm/conf.d
Additional .ini files parsed	/etc/php/7.2/fpm/conf.d/10-mysqlnd.ini, /etc/php/7.2/fpm/conf.d/10-opcache.ini, /etc/php/7.2/fpm/conf.d/10-pdo.ini, /etc/php/7.2/fpm/conf.d/20-calendar.ini, /etc/php/7.2/fpm/conf.d/20-ctype.ini, /etc/php/7.2/fpm/conf.d/20-exif.ini, /etc/php/7.2/fpm/conf.d/20-fileinfo.ini, /etc/php/7.2/fpm/conf.d/20-ftp.ini, /etc/php/7.2/fpm/conf.d/20-gettext.ini, /etc/php/7.2/fpm/conf.d/20-iconv.ini, /etc/php/7.2/fpm/conf.d/20-json.ini, /etc/php/7.2/fpm/conf.d/20-mysqli.ini, /etc/php/7.2/fpm/conf.d/20-pdo_mysql.ini, /etc/php/7.2/fpm/conf.d/20-phar.ini, /etc/php/7.2/fpm/conf.d/20-posix.ini, /etc/php/7.2/fpm/conf.d/20-readline.ini, /etc/php/7.2/fpm/conf.d/20-shmop.ini, /etc/php/7.2/fpm/conf.d/20-sockets.ini, /etc/php/7.2/fpm/conf.d/20-sysvmsg.ini, /etc/php/7.2/fpm/conf.d/20-sysvsem.ini, /etc/php/7.2/fpm/conf.d/20-sysvshm.ini, /etc/php/7.2/fpm/conf.d/20-tokenizer.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
    with Zend OPcache v7.2.24-0ubuntu0.18.04.1, Copyright (c) 1999-2018, by Zend Technologies

phpinfo page

Because this page contains sensitive information about your PHP environment, it is recommended that you remove it from the server by running an `rm -f /var/www/info.php` command once you have finished setting it up.

The Playbook Contents

You can find the LAMP server setup featured in this tutorial in the `lamp_ubuntu1804` folder inside the DigitalOcean Community Playbooks repository. To copy or download the script contents directly, click the Raw button towards the top of each script.

The full contents of the playbook as well as its associated files are also included here for your convenience.

vars/default.yml

The `default.yml` variable file contains values that will be used within the playbook tasks, such as the password for the MySQL root account and the domain name to configure within Apache.

vars/default.yml

```
---
mysql_root_password: "mysql_root_password"
app_user: "sammy"
http_host: "your_domain"
http_conf: "your_domain.conf"
http_port: "80"
disable_default: true
```

files/apache.conf.j2

The `apache.conf.j2` file is a Jinja 2 template file that configures a new Apache VirtualHost. The variables used within this template are defined in the `vars/default.yml` variable file.

files/apache.conf.j2

```
<VirtualHost *:{{ http_port }}>
   ServerAdmin webmaster@localhost
   ServerName {{ http_host }}
   ServerAlias www.{{ http_host }}
   DocumentRoot /var/www/{{ http_host }}
   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined

   <Directory /var/www/{{ http_host }}>
         Options -Indexes
   </Directory>

   <IfModule mod_dir.c>
       DirectoryIndex index.php index.html index.cgi index.pl
index.xhtml index.htm
   </IfModule>

</VirtualHost>
```

files/info.php.j2

The `info.php.j2` file is another Jinja template, used to set up a test PHP script in the document root of the newly configured LAMP server.

files/info.php.j2

```
<?php

phpinfo();
```

playbook.yml

The `playbook.yml` file is where all tasks from this setup are defined. It starts by defining the group of servers that should be the target of this setup (`all`), after which it uses `become: true` to define that tasks should be executed with privilege escalation (`sudo`) by default. Then, it includes the `vars/default.yml` variable file to load configuration options.

playbook.yml

```
---
- hosts: all
  become: true
  vars_files:
    - vars/default.yml

  tasks:
    - name: Install prerequisites
      apt: name={{ item }} update_cache=yes state=latest
force_apt_get=yes
      loop: [ 'aptitude' ]

  #Apache Configuration
    - name: Install LAMP Packages
      apt: name={{ item }} update_cache=yes state=latest
      loop: [ 'apache2', 'mysql-server', 'python3-pymysql', 'php',
'php-mysql', 'libapache2-mod-php' ]

    - name: Create document root
      file:
        path: "/var/www/{{ http_host }}"
        state: directory
        owner: "{{ app_user }}"
        mode: '0755'
    - name: Set up Apache virtualhost
      template:
        src: "files/apache.conf.j2"
        dest: "/etc/apache2/sites-available/{{ http_conf }}"
      notify: Reload Apache

    - name: Enable new site
      shell: /usr/sbin/a2ensite {{ http_conf }}
      notify: Reload Apache

    - name: Disable default Apache site
      shell: /usr/sbin/a2dissite 000-default.conf
      when: disable_default
      notify: Reload Apache

  # MySQL Configuration
    - name: Sets the root password
      mysql_user:
        name: root
        password: "{{ mysql_root_password }}"
        login_unix_socket: /var/run/mysqld/mysqld.sock

    - name: Removes all anonymous user accounts
      mysql_user:
        name: ''
        host_all: yes
        state: absent
        login_user: root
        login_password: "{{ mysql_root_password }}"

    - name: Removes the MySQL test database
      mysql_db:
        name: test
        state: absent
        login_user: root
        login_password: "{{ mysql_root_password }}"

  # UFW Configuration
    - name: "UFW - Allow HTTP on port {{ http_port }}"
      ufw:
        rule: allow
        port: "{{ http_port }}"
        proto: tcp

  # PHP Info Page
    - name: Sets Up PHP Info Page
      template:
        src: "files/info.php.j2"
        dest: "/var/www/{{ http_host }}/info.php"

  handlers:
    - name: Reload Apache
      service:
        name: apache2
        state: reloaded

    - name: Restart Apache
      service:
        name: apache2
        state: restarted
```

Feel free to modify these files to best suit your individual needs within your own workflow.

Conclusion

In this guide, we used Ansible to automate the process of installing and setting up a LAMP environment on a remote server. Because each individual typically has different needs when working with MySQL databases and users, we encourage you to check out the official Ansible documentation for more information and use cases of the `mysql_user` Ansible module.

If you'd like to include other tasks in this playbook to further customize your server setup, please refer to our introductory Ansible guide Configuration Management 101: Writing Ansible Playbooks.

How to Use Ansible to Install and Set Up LEMP on Ubuntu 18.04

Written by Erika Heidi

This guide explains how to use Ansible to automate setting up a Linux, Nginx, MySQL, and PHP server, commonly referred to as the LEMP stack. It is a companion to Chapter 10, How To Install Linux, Nginx, MySQL, PHP (LEMP stack) on Ubuntu 18.04.

Instead of manually running `apt` commands to set up the LEMP stack, this tutorial demonstrates how to use Ansible to configure a server. Ansible will install all the required LEMP packages, create an Nginx configuration using a template, secure MySQL, set up the UFW firewall to allow traffic to the server, and finally add a test PHP script to demonstrate that all the components of the LEMP stack are working correctly.

Server automation now plays an essential role in systems administration, due to the disposable nature of modern application environments. Configuration management tools such as Ansible are typically used to streamline the process of automating server setup by establishing standard procedures for new servers while also reducing human error associated with manual setups.

Ansible offers a simple architecture that doesn’t require special software to be installed on nodes. It also provides a robust set of features and built-in modules which facilitate writing automation scripts.

This guide explains how to use Ansible to automate the steps contained in our guide on How To Install Linux, Nginx, MySQL and PHP (LEMP) on

Ubuntu 18.04. The LEMP software stack is a group of software that can be used to serve dynamic web pages and web applications. This is an acronym that describes a Linux operating system, with an Nginx (pronounced like “Engine-X”) web server. The backend data is stored in the MySQL database and the dynamic processing is handled by PHP.

Prerequisites

In order to execute the automated setup provided by the playbook we’re discussing in this guide, you’ll need:

- One Ansible control node: an Ubuntu 18.04 machine with Ansible installed and configured to connect to your Ansible hosts using SSH keys. Make sure the control node has a regular user with sudo permissions and a firewall enabled, as explained in our Initial Server Setup guide. To set up Ansible, please follow our guide on How to Install and Configure Ansible on Ubuntu 18.04.
- One or more Ansible Hosts: one or more remote Ubuntu 18.04 servers previously set up following the guide on How to Use Ansible to Automate Initial Server Setup on Ubuntu 18.04.

Before proceeding, you first need to make sure your Ansible control node is able to connect and execute commands on your Ansible host(s). For a connection test, please check step 3 of How to Install and Configure Ansible on Ubuntu 18.04.

What Does this Playbook Do?

This Ansible playbook provides an alternative to manually running through the procedure outlined in our guide on How To Install Linux, Nginx, MySQL, PHP (LEMP stack) on Ubuntu 18.04.

Running this playbook will perform the following actions on your Ansible hosts:

1. Install `aptitude`, which is preferred by Ansible as an alternative to the `apt` package manager.
2. Install the required LEMP packages.
3. Set up the Nginx configuration file using the provided template.
4. Enable the new Nginx configuration and disable the default one.
5. Set the password for the MySQL root user.
6. Remove anonymous MySQL accounts and the test database.
7. Set up UFW to allow HTTP traffic on the configured port (`80` by default).
8. Set up a PHP test script using the provided template.

Once the playbook has finished running, you will have a web PHP environment running on top of Nginx, based on the options you defined within your configuration variables.

How to Use this Playbook

The first thing we need to do is obtain the LEMP playbook and its dependencies from the do-community/ansible-playbooks repository. We need to clone this repository to a local folder inside the Ansible Control Node.

In case you have cloned this repository before while following a different guide, access your existing `ansible-playbooks` copy and run a `git pull` command to make sure you have updated contents:

```
cd ~/ansible-playbooks
git pull
```

If this is your first time using the `do-community/ansible-playbooks` repository, you should start by cloning the repository to your home folder with:

```
cd ~
git clone https://github.com/do-community/ansible-playbooks.git
cd ansible-playbooks
```

The files we're interested in are located inside the `lemp_ubuntu1804` folder, which has the following structure:

```
lemp_ubuntu1804
├── files
│   ├── info.php.j2
│   └── nginx.conf.j2
├── vars
│   └── default.yml
├── playbook.yml
└── readme.md
```

Here is what each of these files are:

- `files/info.php.j2`: Template file for setting up a PHP test page on the web server's root

- `files/nginx.conf.j2`: Template file for setting up the Nginx server. directory.
- `vars/default.yml`: Variable file for customizing playbook settings.
- `playbook.yml`: The playbook file, containing the tasks to be executed on the remote server(s).
- `readme.md`: A text file containing information about this playbook.

We'll edit the playbook's variable file to customize the configurations of both MySQL and Nginx. Access the `lemp_ubuntu1804` directory and open the `vars/default.yml` file using your command line editor of choice:

```
cd lemp_ubuntu1804
nano vars/default.yml
```

This file contains a few variables that require your attention:

vars/default.yml

```
---
mysql_root_password: "mysql_root_password"
http_host: "your_domain"
http_conf: "your_domain.conf"
http_port: "80"
```

The following list contains a brief explanation of each of these variables and how you might want to change them:

- `mysql_root_password`: The desired password for the root MySQL account.
- `http_host`: The host name or IP address for this web server.
- `http_conf`: The name of the configuration file to be created inside `/etc/nginx/sites-available`, typically set to the host or application name for easier identification.
- `http_port`: The port Nginx will use to serve this site. This is port `80` by default, but if you want to serve your site or application on a different port, enter it here.

Once you’re done updating the variables inside `vars/default.yml`, save and close this file. If you used `nano`, do so by pressing `CTRL + X`, `Y`, then `ENTER`.

You’re now ready to run this playbook on one or more servers. Most playbooks are configured to be executed on everyserver in your inventory, by default. We can use the-lflag to make sure that only a subset of servers, or a single server, is affected by the playbook. We can also use the-u` flag to specify which user on the remote server we’re using to connect and execute the playbook commands on the remote hosts.

To execute the playbook only on **server1**, connecting as **sammy**, you can use the following command:

```
ansible-playbook playbook.yml -l server1 -u sammy
```

You will get output similar to this:

Output

```
PLAY [all]
*******************************************************************
**********************************************************

TASK [Gathering Facts]
*******************************************************************
**********************************************
ok: [server1]

TASK [Install Prerequisites]
*******************************************************************
****************************************
changed: [server1] => (item=aptitude)

...

TASK [UFW - Allow HTTP on port 80]
*******************************************************************
**********************************
changed: [server1]

TASK [Sets Up PHP Info Page]
*******************************************************************
****************************************
changed: [server1]
```

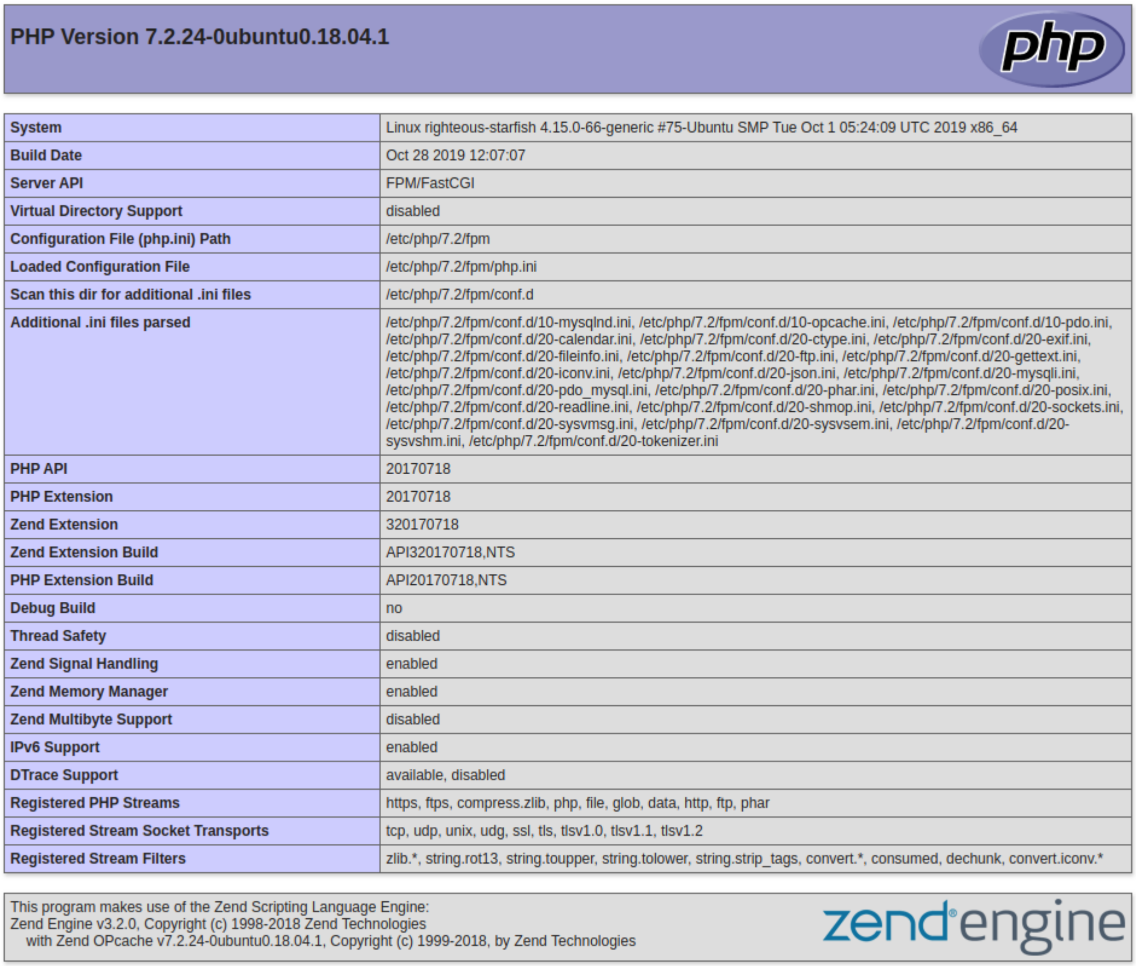
```
RUNNING HANDLER [Reload Nginx]
********************************************************************
**************************************
changed: [server1]

PLAY RECAP
********************************************************************
***********************************************************
server1         : ok=12   changed=9    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Note: For more information on how to run Ansible playbooks, check our Ansible Cheat Sheet Guide.

When the playbook is finished running, go to your web browser and access the host or IP address of the server, as configured in the playbook variables, followed by `/info.php`:

`http://`**`server_host_or_IP`**`/info.php`

You will see a page like this:



PHP Version 7.2.24-0ubuntu0.18.04.1

System	Linux righteous-starfish 4.15.0-66-generic #75-Ubuntu SMP Tue Oct 1 05:24:09 UTC 2019 x86_64
Build Date	Oct 28 2019 12:07:07
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/fpm
Loaded Configuration File	/etc/php/7.2/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/fpm/conf.d
Additional .ini files parsed	/etc/php/7.2/fpm/conf.d/10-mysqlnd.ini, /etc/php/7.2/fpm/conf.d/10-opcache.ini, /etc/php/7.2/fpm/conf.d/10-pdo.ini, /etc/php/7.2/fpm/conf.d/20-calendar.ini, /etc/php/7.2/fpm/conf.d/20-ctype.ini, /etc/php/7.2/fpm/conf.d/20-exif.ini, /etc/php/7.2/fpm/conf.d/20-fileinfo.ini, /etc/php/7.2/fpm/conf.d/20-ftp.ini, /etc/php/7.2/fpm/conf.d/20-gettext.ini, /etc/php/7.2/fpm/conf.d/20-iconv.ini, /etc/php/7.2/fpm/conf.d/20-json.ini, /etc/php/7.2/fpm/conf.d/20-mysqli.ini, /etc/php/7.2/fpm/conf.d/20-pdo_mysql.ini, /etc/php/7.2/fpm/conf.d/20-phar.ini, /etc/php/7.2/fpm/conf.d/20-posix.ini, /etc/php/7.2/fpm/conf.d/20-readline.ini, /etc/php/7.2/fpm/conf.d/20-shmop.ini, /etc/php/7.2/fpm/conf.d/20-sockets.ini, /etc/php/7.2/fpm/conf.d/20-sysvmsg.ini, /etc/php/7.2/fpm/conf.d/20-sysvsem.ini, /etc/php/7.2/fpm/conf.d/20-sysvshm.ini, /etc/php/7.2/fpm/conf.d/20-tokenizer.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
    with Zend OPcache v7.2.24-0ubuntu0.18.04.1, Copyright (c) 1999-2018, by Zend Technologies

phpinfo page

Because this page contains sensitive information about your PHP environment, it is recommended that you remove it from the server by running an `rm -f /var/www/info.php` command once you have finished setting it up.

The Playbook Contents

You can find the LEMP server setup featured in this tutorial in the `lemp_ubuntu1804` folder inside the DigitalOcean Community Playbooks repository. To copy or download the script contents directly, click the Raw button towards the top of each script.

The full contents of the playbook as well as its associated files are also included here for your convenience.

vars/default.yml

The `default.yml` variable file contains values that will be used within the playbook tasks, such as the password for the MySQL root account and the domain name to configure within Nginx.

vars/default.yml

```
---
mysql_root_password: "mysql_root_password"
http_host: "your_domain"
http_conf: "your_domain.conf"
http_port: "80"
```

files/nginx.conf.j2

The `nginx.conf.j2` file is a Jinja 2 template file that configures the Nginx web server. The variables used within this template are defined in the `vars/default.yml` variable file.

files/nginx.conf.j2

```
server {
        listen {{ http_port }};
        root /var/www/html;
        index index.php index.html index.htm index.nginx-debian.html;
        server_name {{ http_host }};

        location / {
                try_files $uri $uri/ =404;
        }

        location ~ \.php$ {
                include snippets/fastcgi-php.conf;
                fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;
        }

        location ~ /\.ht {
                deny all;
        }
}
```

files/info.php.j2

The `info.php.j2` file is another Jinja template, used to set up a test PHP script in the document root of the newly configured LEMP server.

files/info.php.j2

```
<?php
phpinfo();
```

playbook.yml

The `playbook.yml` file is where all tasks from this setup are defined. It starts by defining the group of servers that should be the target of this setup (`all`), after which it uses `become: true` to define that tasks should be executed with privilege escalation (`sudo`) by default. Then, it includes the `vars/default.yml` variable file to load configuration options.

playbook.yml

```
---
- hosts: all
  become: true
  vars_files:
   - vars/default.yml

 tasks:
   - name: Install Prerequisites
     apt: name={{ item }} update_cache=yes state=latest
force_apt_get=yes
     loop: [ 'aptitude' ]

   - name: Install LEMP Packages
     apt: name={{ item }} update_cache=yes state=latest
     loop: [ 'nginx', 'mysql-server', 'python3-pymysql', 'php-fpm',
'php-mysql' ]

# Nginx Configuration

   - name: Sets Nginx conf file
     template:
       src: "files/nginx.conf.j2"
       dest: "/etc/nginx/sites-available/{{ http_conf }}"

   - name: Enables new site
     file:
       src: "/etc/nginx/sites-available/{{ http_conf }}"
       dest: "/etc/nginx/sites-enabled/{{ http_conf }}"
       state: link
     notify: Reload Nginx

   - name: Removes "default" site
     file:
       path: "/etc/nginx/sites-enabled/default"
       state: absent
     notify: Reload Nginx

# MySQL Configuration

   - name: Sets the root password
     mysql_user:
       name: root
       password: "{{ mysql_root_password }}"
       login_unix_socket: /var/run/mysqld/mysqld.sock

   - name: Removes all anonymous user accounts
     mysql_user:
       name: ''
       host_all: yes
       state: absent
       login_user: root
       login_password: "{{ mysql_root_password }}"
   - name: Removes the MySQL test database
     mysql_db:
       name: test
       state: absent
       login_user: root
       login_password: "{{ mysql_root_password }}"

# UFW Configuration

   - name: "UFW - Allow HTTP on port {{ http_port }}"
     ufw:
       rule: allow
       port: "{{ http_port }}"
       proto: tcp

# Sets Up PHP Info Page

   - name: Sets Up PHP Info Page
     template:
       src: "files/info.php.j2"
       dest: "/var/www/html/info.php"

# Handlers

 handlers:
   - name: Reload Nginx
     service:
       name: nginx
       state: reloaded

   - name: Restart Nginx
     service:
       name: nginx
       state: restarted
```

Feel free to modify these files to best suit your individual needs within your own workflow.

Conclusion

In this guide, we used Ansible to automate the process of installing and setting up a LEMP environment on a remote server. Because each individual typically has different needs when working with MySQL databases and users, we encourage you to check out the official Ansible documentation for more information and use cases of the `mysql_user` Ansible module.

If you'd like to include other tasks in this playbook to further customize your server setup, please refer to our introductory Ansible guide Configuration Management 101: Writing Ansible Playbooks.

How To Acquire a Let’s Encrypt Certificate Using Ansible on Ubuntu 18.04

Written by Jamie Scaife

In this chapter you will learn how to use Ansible to automate setting up Letsencrypt TLS certificates. This chapter is a companion to Chapters 11 and 12, where you learned how to set up and install a Letsencrypt TLS certificate for your Apache and Nginx servers respectively.

The author selected the Electronic Frontier Foundation to receive a donation as part of the Write for DOnations program.

Modern infrastructure management is best done using automated processes and tools. Acquiring a Let’s Encrypt certificate using the standard Certbot client is quick and easy, but is generally a task that has to be done manually when commissioning servers. This is manageable for an individual server setup, but can become tedious when deploying a larger fleet.

Using a configuration management tool such as Ansible to acquire a certificate makes this task completely automatic and reproducible. If you ever have to rebuild or update your server, you can just run your Ansible playbook, rather than having to manually carry out the steps again.

In this tutorial, you’ll write an Ansible playbook to acquire a Let’s Encrypt certificate automatically for an Ansible host machine.

Prerequisites

To complete this tutorial, you will need:

- Two Ubuntu 18.04 servers set up by following the Initial Server Setup with Ubuntu 18.04, including a sudo non-root user.

The first server will be used as your Ansible server, which we will call Ansible server throughout this tutorial. This is where Ansible will run to send the commands to the host machine. Alternatively, you can use your local machine or any other machine that has your Ansible inventory configured as your Ansible server.

On your Ansible server, you'll need:

- A correctly configured Ansible installation that is able to connect to your Ansible hosts by following How To Install and Configure Ansible on Ubuntu 18.04.

The second server will be used as your Ansible host, which we will call the host machine throughout this tutorial. This is the machine that you wish to configure and issue certificates on. This machine will also run a web server to serve the certificate issuance validation files.

On your host machine, you'll need:

- A domain name that you are eligible to acquire a TLS certificate for, with the required DNS records configured to point to your Ansible host machine. In this particular example, the playbook will acquire a certificate valid for **`your-domain`** and `www.`**`your-domain`**, however it can be adjusted for other domains or subdomains if required.
- A web server that is accessible from the internet over port `80` (HTTP), for example by following steps 1, 2, and 3 of How To Install

the Apache Web Server on Ubuntu 18.04. This could also be an Nginx server, or any other suitable web server software.

Once you have these ready, log in to your Ansible server as your non-root user to begin.

Step 1 — Configuring the Settings for the Let’s Encrypt Ansible Module

Ansible has a built-in module named `letsencrypt`, which allows you to acquire valid TLS certificates using the ACME (Automated Certificate Management Environment) protocol.

In this first step, you will add a host variables configuration file to define the configuration variables that are required to use the module.

Note: The `letsencrypt` module has been renamed to `acme_certificate` as of Ansible 2.6. The `letsencrypt` name is now an alias of `acme_certificate`, so will still work, but you way wish to use `acme_certificate` instead, to ensure future-proofness of your playbooks. You can check your Ansible version using `ansible --version`. As of the writing of this tutorial, the Ubuntu 18.04 Apt repositories don’t support `acme_certificate` yet.

Firstly, create the `host_vars` Ansible directory on your Ansible server:

```
sudo mkdir /etc/ansible/host_vars
```

Next, create a new file in the `/etc/ansible/host_vars` directory with the name of your Ansible host machine. In this example, you’ll use `host1` as the name of the host:

```
sudo nano /etc/ansible/host_vars/host1
```

The following sample configuration includes everything you need to get started, including: the validation method and server address, an email address to receive certificate expiry reminders to, and the directories where your Let's Encrypt keys and certificates will be saved.

Copy the sample configuration into the file:

/etc/ansible/host_vars/host1

```
---

acme_challenge_type: http-01

acme_directory: https://acme-v02.api.letsencrypt.org/directory

acme_version: 2

acme_email: certificate-reminders@your-domain

letsencrypt_dir: /etc/letsencrypt

letsencrypt_keys_dir: /etc/letsencrypt/keys

letsencrypt_csrs_dir: /etc/letsencrypt/csrs

letsencrypt_certs_dir: /etc/letsencrypt/certs

letsencrypt_account_key: /etc/letsencrypt/account/account.key

domain_name: your-domain
```

Save and close the file when you've finished.

Adjust the domain name and email address as required. You can use any email address—it doesn't have to be the one on **`your-domain`**.

Some of the directory/file paths defined may not actually exist on your server yet. This is OK; the first part of the playbook will be to create these directories and assign the relevant permissions.

You've added the required configuration variables to your Ansible inventory file. Next, you will begin writing the playbook to acquire a certificate.

Step 2 — Creating the Let's Encrypt Directories and Account Key

In this step, you'll write the Ansible tasks that you'll use to create the required Let's Encrypt directories, assign the correct permissions, and generate a Let's Encrypt account key.

Firstly, create a new playbook named `letsencrypt-issue.yml` on your Ansible server in a new directory of your choice, for example `/home/user/ansible-playbooks`:

```
cd ~
mkdir ansible-playbooks
cd ansible-playbooks
nano letsencrypt-issue.yml
```

Before you can start writing Ansible tasks, you'll need to specify the hosts and associated settings. Adjust the following according to how you referred to your hosts in the prerequisite tutorial. Then add the following to the top of the file:

```
[label letsencrypt-issue.yml]
---
- hosts: "host1"
  tasks:
```

Now you can begin writing the required tasks, the first of which is to create the file system directories required to store the Let’s Encrypt files. Add the following Ansible task to the file after the previous content:

```
[label letsencrypt-issue.yml]
...
  - name: "Create required directories in /etc/lets
    file:
      path: "/etc/letsencrypt/{{ item }}"
      state: directory
      owner: root
      group: root
      mode: u=rwx,g=x,o=x
    with_items:
    - account
    - certs
    - csrs
    - keys
```

This Ansible task will create the `account`, `certs`, `csrs`, and `keys` directories in `/etc/letsencrypt`, which is where the files required for acquiring certificates will be stored.

You set the owner of the directories to `root` and apply the permissions `u=rwx,g=x,o=x` so that only `root` has read and write access to them. This is recommended as the directories will contain private keys, certificate signing requests (CSRs), and signed certificates, which should be kept confidential.

Next, the Let’s Encrypt account key needs to be created. You’ll use this to identify yourself to the Let’s Encrypt service.

Add the following task to your playbook:

```
[label letsencrypt-issue.yml]
...
  - name: "Generate a Let's Encrypt account key"
    shell: "if [ ! -f {{ letsencrypt_account_key }}
```

The account key doesn’t need to be re-created every time you renew the certificates, so you also add a check for an existing key `if [ ! -f {{ letsencrypt_account_key }} ];`, to make sure that it isn’t overwritten.

You’ll continue to work in `letsencrypt-issue.yml` in the next step, so don’t close this file yet.

You’ve created your playbook and set up the initial configuration and tasks in order to prepare for acquiring your Let’s Encrypt certificate. Next, you will add further tasks for the private key and CSR generation.

Step 3 — Generating Your Private Key and Certificate Signing Request

In this step, you’ll write the playbook tasks to generate the required private key and certificate signing request.

The first task in this section will generate the required private key for your certificate. Add the following to the end of your playbook that you started writing in Step 2:

```
[label letsencrypt-issue.yml]
```

```
[label letsencrypt-issue.yml]
...
  - name: "Generate Let's Encrypt private key"
    shell: "openssl genrsa 4096 | sudo tee /etc/let
```

Subdomains on the same domain will all be added to the same certificate through the use of [Subject Alternate Names (SANs)](), so you only need to generate one private key for now.

You’ll use the next task to generate a Certificate Signing Request (CSR) for the certificate that you want to acquire. This is submitted to Let’s Encrypt in order for them to validate and issue each certificate.

Add the following to the end of the playbook:

```
[label letsencrypt-issue.yml]
...
  - name: "Generate Let's Encrypt CSR"
    shell: "openssl req -new -sha256 -key /etc/lets
    args:
      executable: /bin/bash
```

This task generates a CSR for your domain, with the `www` subdomain added to the certificate as a SAN.

You’ll continue to work in `letsencrypt-issue.yml` in the next step, so don’t close this file yet.

You’ve written the Ansible tasks to generate the private key and CSR for your certificate. Next, you’ll work on the tasks that will begin the validation and issuance process.

Step 4 — Starting the ACME Validation Process

In this step, you’ll write a task to submit the Certificate Signing Request to Let’s Encrypt using the outputted files from the task documented in Step 3. This will return some `challenge` files, which you’ll need to serve on your web server in order to prove ownership of the domain name and subdomain for which you’re requesting a certificate.

The following task will submit the CSR for **`your-domain`**. Add it to the end of your playbook:

```
[label letsencrypt-issue.yml]
...
  - name: "Begin Let's Encrypt challenges"
    letsencrypt:
      acme_directory: "{{ acme_directory }}"
      acme_version: "{{ acme_version }}"
      account_key_src: "{{ letsencrypt_account_key
      account_email: "{{ acme_email }}"
      terms_agreed: 1
      challenge: "{{ acme_challenge_type }}"
      csr: "{{ letsencrypt_csrs_dir }}/{{ domain_na
      dest: "{{ letsencrypt_certs_dir }}/{{ domain_
      fullchain_dest: "{{ letsencrypt_certs_dir }}/
      remaining_days: 91
    register: acme_challenge_your_domain
```

This task makes wide usage of the variables that you configured in Step 1. It registers a variable containing the ACME challenge files that you'll use in the next step. You'll need to manually adjust the name of the variable to contain **your-domain**, but with all . characters replaced with a _, as dots cannot be used in a variable name. For example, the variable for example.com would become acme_challenge_example_com.

You'll continue to work in letsencrypt-issue.yml in the next step, so don't close this file yet.

You've written a task to submit your CSR to Let's Encrypt. Next, you will add a task to implement the ACME challenge files for finalization of the certificate validation process.

Step 5 — Implementing the ACME Challenge Files

In this step, you will write an Ansible task to read and implement the ACME challenge files. These files prove that you're eligible to acquire a certificate for the requested domains and subdomains.

The ACME challenge files must be served on a web server listening on port 80, at the /.well-known/acme-challenge/ path for the domain or subdomain that you're requesting a certificate for. For example, in order to validate the certificate request for www.**your-domain**, the ACME challenge file will need to be accessible over the internet at the following path: http://www.**your-domain**/.well-known/acme-challenge.

The method for serving these files at the required destinations will vary significantly depending on your current web server setup. However, in this guide, we will assume that you have a web server (as per the prerequisite tutorial) configured to serve files out of the /var/www/html directory. Therefore you may need to adjust the task accordingly in order to be compatible with your own web server setup.

Firstly, add the following task that creates the .well-known/acme-challenge/ directory structure required to serve the files to the end of your playbook:

```
[label letsencrypt-issue.yml]
...
  - name: "Create .well-known/acme-challenge direct
    file:
      path: /var/www/html/.well-known/acme-challeng
      state: directory
      owner: root
      group: root
      mode: u=rwx,g=rx,o=rx
```

Make sure to adjust the path accordingly if you are using a directory other than /var/www/html to serve files with your web server.

Next, you'll implement the ACME challenge files that were saved into the acme_challenge_**your-domain** variable in Step 4 with the following task:

```
[label letsencrypt-issue.yml]
...
  - name: "Implement http-01 challenge files"
    copy:
```

```
copy:
  content: "{{ acme_challenge_your_domain['chal
  dest: "/var/www/html/{{ acme_challenge_your_d
  owner: root
  group: root
  mode: u=rw,g=r,o=r
with_items:
- "{{ domain_name }}"
- "www.{{ domain_name }}"
```

Note that you need to manually adjust the `acme_challenge_`**`your_domain`** variable name in the task to be set to the name of your ACME challenge variable, which is `acme_challenge_` followed by your domain name, but with all `.` characters replaced with `_`. This Ansible task copies the ACME validation files from the variable into the `.well-known/acme-challenge` path on your web server. This will allow Let's Encrypt to retrieve them in order to verify the ownership of the domain and your eligibility to acquire a certificate.

You'll continue to work in `letsencrypt-issue.yml` in the next step, so don't close this file yet.

You've written the Ansible tasks required to create the ACME validation directory and files. Next, you will complete the ACME verification process and acquire the signed certificate.

Step 6 — Acquiring Your Certificate

In this step, you’ll write a task to trigger Let’s Encrypt to verify the ACME challenge files that you submitted, which will allow you to acquire your signed certificate(s).

The following task validates the ACME challenge files that you implemented in Step 5 and saves your signed certificates to the specified paths. Add it to the end of your playbook:

```
[label letsencrypt-issue.yml]
...
  - name: "Complete Let's Encrypt challenges"
    letsencrypt:
      acme_directory: "{{ acme_directory }}"
      acme_version: "{{ acme_version }}"
      account_key_src: "{{ letsencrypt_account_key
      account_email: "{{ acme_email }}"
      challenge: "{{ acme_challenge_type }}"
      csr: "{{ letsencrypt_csrs_dir }}/{{ domain_na

      dest: "{{ letsencrypt_certs_dir }}/{{ domain_
      chain_dest: "{{ letsencrypt_certs_dir }}/chai
      fullchain_dest: "{{ letsencrypt_certs_dir }}/
      data: "{{ acme_challenge_your_domain }}"
```

Similarly to Step 4, this task makes use of the variables that you configured in Step 1. Once the task has completed, it will save the signed certificate to the specified paths, allowing you to begin using it for your application or service.

Note that you'll need to manually adjust the `data` value in the task to be set to the name of your ACME challenge variable, similarly to Step 5.

Following is the full playbook showing each of the tasks you've added:

```
[label letsencrypt-issue.yml]
- hosts: "host1"
  tasks:

  - name: "Create required directories in /etc/lets
    file:
      path: "/etc/letsencrypt/{{ item }}"
      state: directory
      owner: root
      group: root
      mode: u=rwx,g=x,o=x
    with_items:
    - account
    - certs
    - csrs
    - keys


  - name: "Generate a Let's Encrypt account key"
    shell: "if [ ! -f {{ letsencrypt_account_key }}

  - name: "Generate Let's Encrypt private key"
    shell: "openssl genrsa 4096 | sudo tee /etc/let
```

```
- name: "Generate Let's Encrypt CSR"
  shell: "openssl req -new -sha256 -key /etc/lets
  args:
    executable: /bin/bash

- name: "Begin Let's Encrypt challenges"
  letsencrypt:
    acme_directory: "{{ acme_directory }}"
    acme_version: "{{ acme_version }}"
    account_key_src: "{{ letsencrypt_account_key
    account_email: "{{ acme_email }}"
    terms_agreed: 1
    challenge: "{{ acme_challenge_type }}"
    csr: "{{ letsencrypt_csrs_dir }}/{{ domain_na
    dest: "{{ letsencrypt_certs_dir }}/{{ domain_
    fullchain_dest: "{{ letsencrypt_certs_dir }}/
    remaining_days: 91
  register: acme_challenge_your_domain


- name: "Create .well-known/acme-challenge direct
  file:
    path: /var/www/html/.well-known/acme-challeng
    state: directory
    owner: root
    group: root
    mode: u=rwx,g=rx,o=rx

- name: "Implement http-01 challenge files"
  copy:
    content: "{{ acme_challenge_your_domain['chal
    dest: "/var/www/html/{{ acme_challenge_your_d
    owner: root
    group: root
    mode: u=rw,g=r,o=r
  with_items:
  - "{{ domain_name }}"
  - "www.{{ domain_name }}"

- name: "Complete Let's Encrypt challenges"
  letsencrypt:
    acme_directory: "{{ acme_directory }}"
    acme_version: "{{ acme_version }}"
    account_key_src: "{{ letsencrypt_account_key
    account_email: "{{ acme_email }}"
    challenge: "{{ acme_challenge_type }}"

    csr: "{{ letsencrypt_csrs_dir }}/{{ domain_na
    dest: "{{ letsencrypt_certs_dir }}/{{ domain_
    chain_dest: "{{ letsencrypt_certs_dir }}/chai
    fullchain_dest: "{{ letsencrypt_certs_dir }}/
    data: "{{ acme_challenge_your_domain }}"
```

Save and close your file when you’re finished.

You’ve added the task to complete the ACME challenges and acquire your signed certificate. Next, you’ll run the playbook against your Ansible host machine in order to run all of the actions.

Step 7 — Running Your Playbook

Now that you’ve written the playbook and all of the required tasks, you can run it against your Ansible host machine to issue the certificate.

From your Ansible server, you can run the playbook using the `ansible-playbook` command:

```
ansible-playbook letsencrypt-issue.yml
```

This will run the playbook, one task at a time. You’ll see output similar to the following:

Output

```
PLAY [host1]
********************************************************************
***************

TASK [Gathering Facts]
********************************************************************
*****
ok: [host1]

TASK [Create required directories in /etc/letsencrypt]
*****************************************
changed: [host1] => (item=account)
changed: [host1] => (item=certs)
changed: [host1] => (item=csrs)
changed: [host1] => (item=keys)

TASK [Generate a Let's Encrypt account key]
****************************************************
changed: [host1]

TASK [Generate Let's Encrypt private key]
******************************************************
changed: [host1]

TASK [Generate Let's Encrypt CSR]
**************************************************************
```

```
changed: [host1]

TASK [Begin Let's Encrypt challenges]
**********************************************************
changed: [host1]

TASK [Create .well-known/acme-challenge directory]
*********************************************
changed: [host1]

TASK [Implement http-01 challenge files]
*******************************************************
changed: [host1] => (item=your-domain)
changed: [host1] => (item=www.your-domain)

TASK [Complete Let's Encrypt challenges]
*******************************************************
changed: [host1]

PLAY RECAP
********************************************************************
*****************
host1                      : ok=9    changed=8    unreachable=0
failed=0
```

If any errors are encountered while the playbook is running, these will be outputted for your review.

Once the playbook has finished, your valid Let’s Encrypt certificate will be saved to the `/etc/letsencrypt/certs` directory on your host machine. You can then use this, along with the private key in `/etc/letsencrypt/keys`, to secure connections to your web server, mail server, etc.

Let’s Encrypt certificates are valid for 90 days by default. You will receive renewal reminders via email to the address that you specified in Step 1. To renew your certificate, you can run the playbook again. Make sure to double check that any services using your certificate have picked up the new one, as sometimes you may need to manually install it, move it to a particular directory, or restart the service for it to properly adopt the new certificate.

In this step, you ran your playbook which issued your valid Let’s Encrypt certificate.

Conclusion

In this article you wrote an Ansible playbook to request and acquire a valid Let’s Encrypt certificate.

As a next step, you can look into using your new playbook to issue certificates for a large fleet of servers. You could even create a central ACME validation server that can issue certificates centrally and distribute them out to web servers.

Finally, if you’d like to learn more about the ACME specification and Let’s Encrypt project, you may wish to review the following links:

- Let’s Encrypt website and documentation.
- Ansible acme_certificate Module.
- RFC8555 - Automated Certificate Management Environment (ACME).

You may also like to view some other relevant Ansible tutorials:

- How To Use Ansible: A Reference Guide.
- How To Automate Server Setup with Ansible on Ubuntu 18.04.
- How To Use Vault to Protect Sensitive Ansible Data on Ubuntu 16.04.

How To Install Git on Ubuntu 18.04

Written by Lisa Tagliaferri

This chpater explains how to install Git on an Ubuntu 18.04 server. It covers how to install Git using the default package that comes with Ubuntu, as well as how to install Git from source code. Once you have Git installed, the guide explains how to configure Git so that it knows your name and email address, which it uses in commit logs to track who changed a file or files.

An earlier version of this tutorial was written by Brennen Bearnes.

Version control systems are increasingly indispensable in modern software development as versioning allows you to keep track of your software at the source level. You can track changes, revert to previous stages, and branch to create alternate versions of files and directories.

One of the most popular version control systems currently available is Git. Many projects' files are maintained in a Git repository, and sites like GitHub, GitLab, and Bitbucket help to facilitate software development project sharing and collaboration.

In this guide, we will demonstrate how to install and configure Git on an Ubuntu 18.04 server. We will cover how to install the software in two different ways, each of which have their own benefits depending on your specific needs.

Prerequisites

In order to complete this tutorial, you should have a non-root user with `sudo` privileges on an Ubuntu 18.04 server. To learn how to achieve this setup, follow our manual initial server setup guide or run our automated script.

With your server and user set up, you are ready to begin.

Installing Git with Default Packages

Ubuntu's default repositories provide you with a fast method to install Git. Note that the version you install via these repositories may be older than the newest version currently available. If you need the latest release, consider moving to the next section of this tutorial to learn how to install and compile Git from source.

First, use the apt package management tools to update your local package index. With the update complete, you can download and install Git:

```
sudo apt update
sudo apt install git
```

You can confirm that you have installed Git correctly by running the following command:

```
git --version
```

Output

```
git version 2.17.1
```

With Git successfully installed, you can now move on to the Setting Up Git section of this tutorial to complete your setup.

Installing Git from Source

A more flexible method of installing Git is to compile the software from source. This takes longer and will not be maintained through your package manager, but it will allow you to download the latest release and will give you some control over the options you include if you wish to customize.

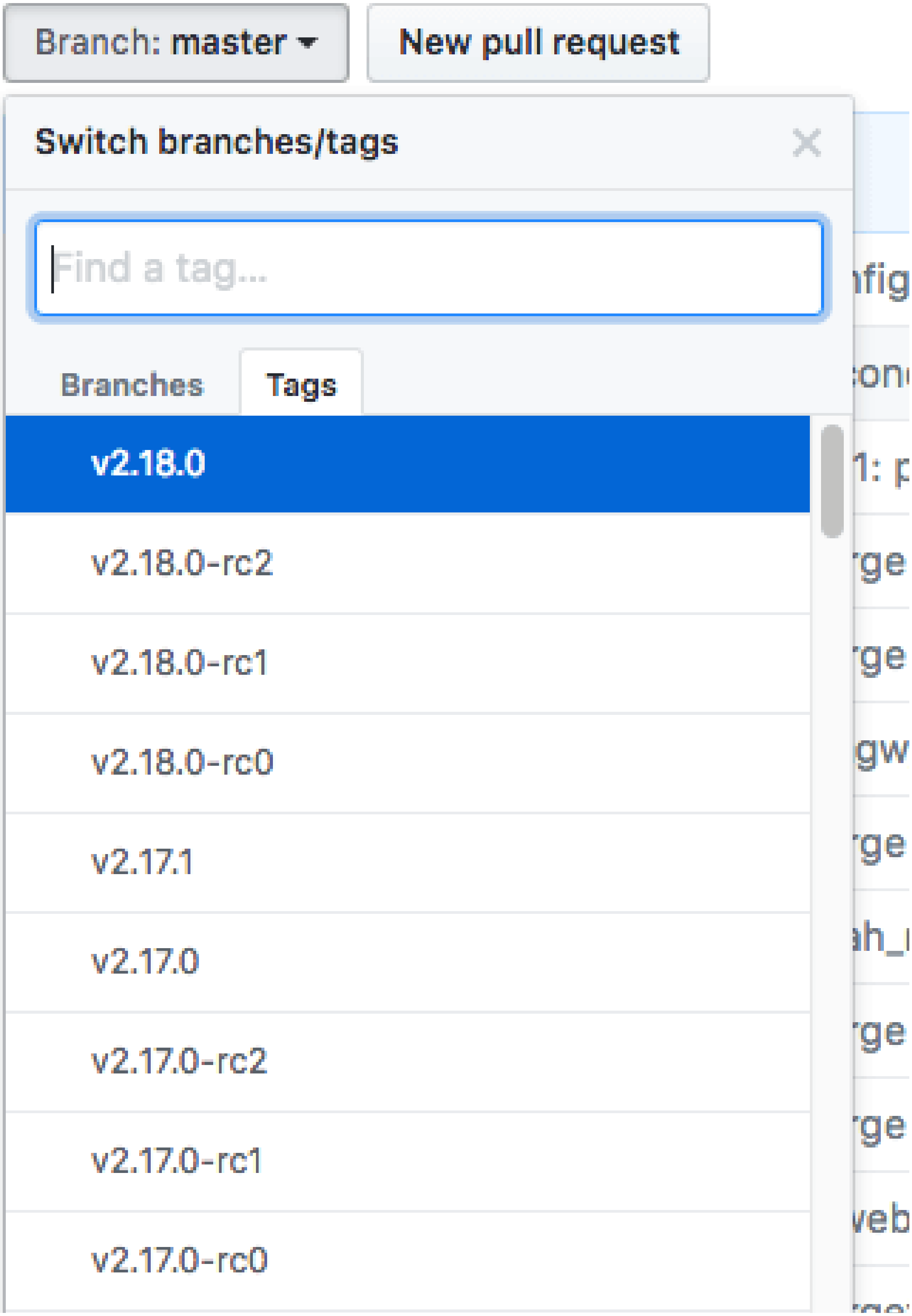
Before you begin, you need to install the software that Git depends on. This is all available in the default repositories, so we can update our local package index and then install the packages.

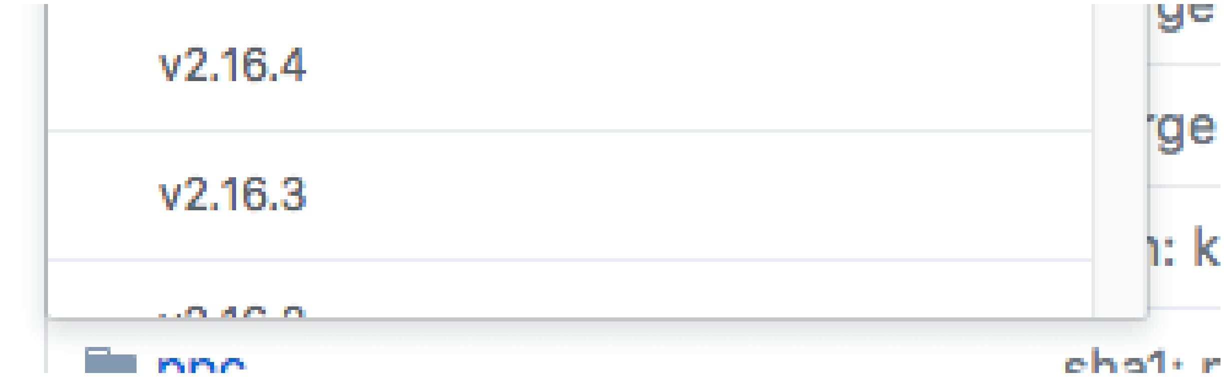
```
sudo apt update
sudo apt install make libssl-dev libghc-zlib-dev
libcurl4-gnutls-dev libexpat1-dev gettext unzip
```

After you have installed the necessary dependencies, you can go ahead and get the version of Git you want by visiting the Git project's mirror on GitHub, available via the following URL:

```
https://github.com/git/git
```

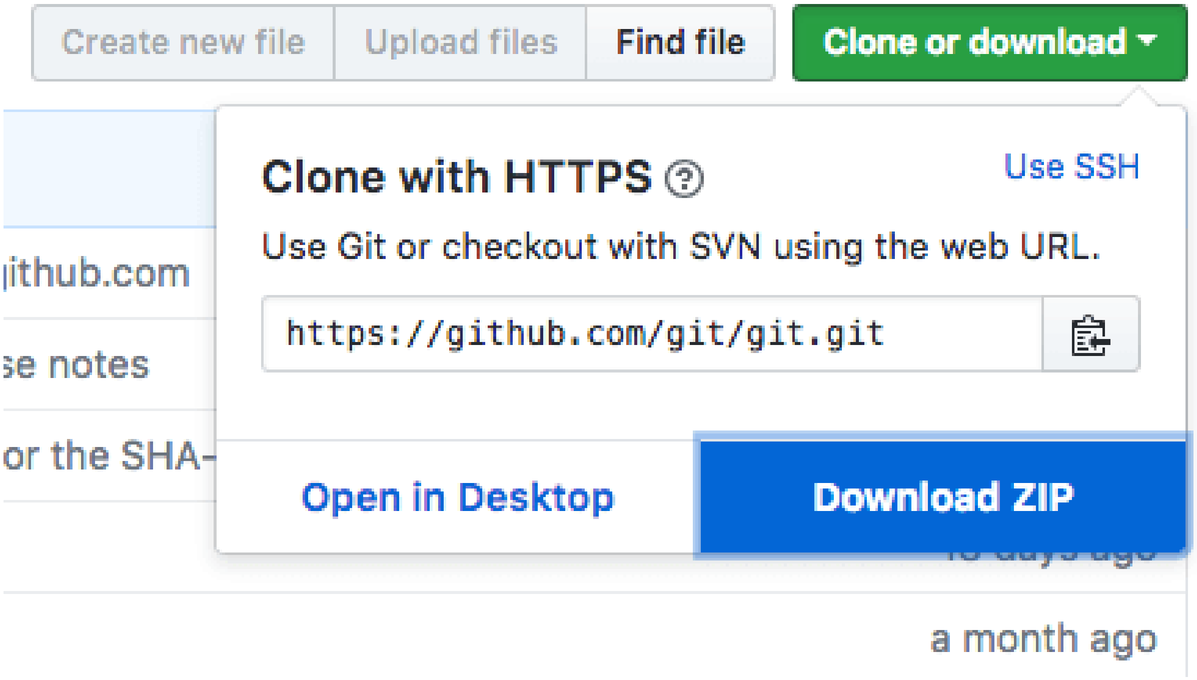
From here, be sure that you are on the `master` branch. Click on the Tags link and select your desired Git version. Unless you have a reason for downloading a release candidate (marked as rc) version, try to avoid these as they may be unstable.


Branch: master
New pull request
Switch branches/tags
Find a tag...
Branches
Tags
v2.18.0
v2.18.0-rc2
v2.18.0-rc1
v2.18.0-rc0
v2.17.1
v2.17.0
v2.17.0-rc2
v2.17.0-rc1
v2.17.0-rc0



git change branch select tags

Next, on the right side of the page, click on the Clone or download button, then right-click on Download ZIP and copy the link address that ends in `.zip`.



Use Git or checkout with SVN using the web URL.

```
https://github.com/git/git.git
```

right-click on download zip to copy url

Back on your Ubuntu 16.04 server, move into the `tmp` directory to download temporary files.

```
cd /tmp
```

From there, you can use the `wget` command to install the copied zip file link. We'll specify a new name for the file: `git.zip`.

```
wget
https://github.com/git/git/archive/v2.18.0.zip -O
git.zip
```

Unzip the file that you downloaded and move into the resulting directory by typing:

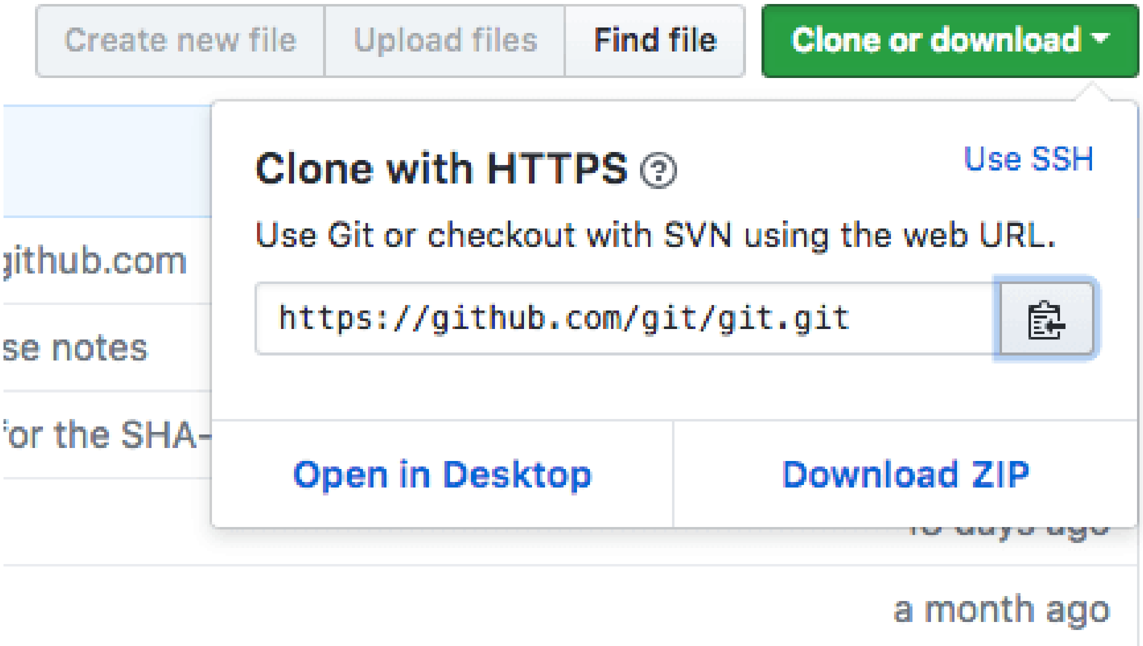
```
unzip git.zip
cd git-*
```

Now, you can make the package and install it by typing these two commands:

```
make prefix=/usr/local all
sudo make prefix=/usr/local install
```

To ensure that the install was successful, you can type `git --version` and you should receive relevant output that specifies the current installed version of Git.

Now that you have Git installed, if you want to upgrade to a later version, you can clone the repository, and then build and install. To find the URL to use for the clone operation, navigate to the branch or tag that you want on the project's GitHub page and then copy the clone URL on the right side:



Clone with HTTPS

Use Git or checkout with SVN using the web URL.

```
https://github.com/git/git.git
```

git copy URL

At the time of writing, the relevant URL is:

```
https://github.com/git/git.git
```

Change to your home directory, and use `git clone` on the URL you just copied:

```
cd ~
git clone https://github.com/git/git.git
```

This will create a new directory within your current directory where you can rebuild the package and reinstall the newer version, just like you did above. This will overwrite your older version with the new version:

```
cd git
make prefix=/usr/local all
sudo make prefix=/usr/local install
```

With this complete, you can be sure that your version of Git is up to date.

Setting Up Git

Now that you have Git installed, you should configure it so that the generated commit messages will contain your correct information.

This can be achieved by using the `git config` command. Specifically, we need to provide our name and email address because Git embeds this information into each commit we do. We can go ahead and add this information by typing:

```
git config --global user.name "Your Name"
git config --global user.email
"youremail@domain.com"
```

We can see all of the configuration items that have been set by typing:

```
git config --list
```

Output

```
user.name=Your Name

user.email=youremail@domain.com

...
```

The information you enter is stored in your Git configuration file, which you can optionally edit by hand with a text editor like this:

```
nano ~/.gitconfig
```

~/.gitconfig contents

```
[user]

  name = Your Name

  email = youremail@domain.com
```

There are many other options that you can set, but these are the two essential ones needed. If you skip this step, you’ll likely see warnings when you commit to Git. This makes more work for you because you will then have to revise the commits you have done with the corrected information.

Conclusion

You should now have Git installed and ready to use on your system.

To learn more about how to use Git, check out these articles and series:

- How To Use Git Effectively
- How To Use Git Branches
- An Introduction to Open Source

How To Use Git Effectively

Written by Jason Kurtz

In this chapter you will learn how to use Git to manage your workspace environment. You will also learn how to add an existing project to a Git workspace. Once you are using Git with your project, this chapter explains how to create commit file changes to Git, along with how to add commit messages that annotate your changes. Finally, this chapter explains how you can push changes to a remote server so that you can collaborate with others on a project.

This article assumes that you have git installed and that your global configuration settings (namely username and email) are properly set. If this is not the case, please refer to the git introduction tutorial.

Git is a very useful piece of software to help streamline development for programming projects. It comes with no language requirements nor file structure requirements, leaving it open for the developers to decide how they want to structure their workflow.

Before using git for your development, it’s a good idea to plan out your workflow. The workflow decision is typically based on the size and scale of your project. To gain a basic understanding of git for now, a simple, single-branch workflow will suffice. By default, the first branch on any git project is called “master”. In a following tutorial in this series, you will learn how to create other branches.

Let’s create our first project and call it “testing”. (If you already have a project that you want to import to git you can skip down to that section.)

Creating your workspace

Just like you want to have a good, clean work environment, the same idea applies to where you do your coding, especially if you're going to contribute to a number of projects at the same time. A good suggestion might be to have a folder called git in your home directory which has subfolders for each of your individual projects.

The first thing we need to do is create our workspace environment:

```
user@host ~ $ mkdir -p ~/git/testing ; cd
~/git/testing
```

The above commands will accomplish two things: 1) It creates a directory called "git" in our home directory and then creates a subdirectory inside of that called "testing" (this is where our project will actually be stored). 2) It brings us to our project's base directory.

Once inside that directory, we need to create a few files that will be in our project. In this step, you can either follow along and create a few dummy files for testing purposes or you can create files/directories you wish that are going to be part of your project.

We are going to create a test file to use in our repository:

```
user@host ~/git/testing $ touch file
```

Once all your project files are in your workspace, you need to start tracking your files with git. The next step explains that process.

Converting an existing project into a workspace environment

Once all the files are in your git workspace, you need to tell git that you want to use your current directory as a git environment.

```
user@host ~/git/testing $ git init
Initialized empty Git repository in
/home/user/git/testing/.git/
```

Once your have initialized your new empty repository, you can add your files.

The following will add all files and directories to your newly created repository:

```
user@host ~/git/testing $ git add .
```

In this case, no output is good output. Unfortunately, git does not always inform you if something worked.

Every time you add or make changes to files, you need to write a commit message. The next section describes what a commit message is and how to write one.

Creating a commit message

A commit message is a short message explaining the changes that you've made. It is required before sending your coding changes off (which is called a push) and it is a good way to communicate to your co-developers what to expect from your changes. This section will explain how to create them.

Commit messages are generally rather short, between one and two sentences explaining what your change did. It is good practice to commit each individual change before you do a push. You can push as many commits as you like. The only requirement for any commit is that it involves at least one file and it has a message. A push must have at least one commit.

Continuing with our example, we are going to create the message for our initial commit:

```
user@host ~/git/testing $ git commit -m "Initial
Commit" -a
[master (root-commit) 1b830f8] initial commit
 0 files changed
 create mode 100644 file
```

There are two important parameters of the above command. The first is -m, which signifies that our commit message (in this case “Initial Commit”) is going to follow. Secondly, the -a signifies that we want our commit message to be applied to all added or modified files. This is okay for the first commit, but generally you should specify the individual files or directories that we want to commit.

We could have also done:

```
user@host ~/git/testing $ git commit -m "Initial
Commit" file
```

To specify a particular file to commit. To add additional files or directories, you just add a space separated list to the end of that command.

Pushing changes to a remote server

Up until this point, we have done everything on our local server. That’s certainly an option to use git locally, if you want to have any easy way to have version control of your files. If you want to work with a team of developers, however, you’re going to need to push changes to a remote server. This section will explain how to do that.

The first step to being able to push code to a remote server is providing the URL where the repository lives and giving it a name. To configure a remote repository to use and to see a list of all remotes (you can have more than one), type the following:

```
user@host ~/git/testing $ git remote add origin
ssh://git@git.domain.tld/repository.git
user@host ~/git/testing $ git remote -v
origin  ssh://git@git.domain.tld/repository.git
(fetch)
origin  ssh://git@git.domain.tld/repository.git
(push)
```

The first command adds a remote, called “origin”, and sets the URL to ssh://git@git.domain.tld/repository.git.

You can name your remote whatever you’d like, but the URL needs to point to an actual remote repository. For example, if you wanted to push code to GitHub, you would need to use the repository URL that they provide.

Once you have a remote configured, you are now able to push your code.

You can push code to a remote server by typing the following:

```
user@host ~/git/testing $ git push origin master
Counting objects: 4, done.
Delta compression using up to 2 threads.
```

```
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 266 bytes, done.
Total 3 (delta 1), reused 1 (delta 0)
To ssh://git@git.domain.tld/repository.git
   0e78fdf..e6a8ddc  master -> master
```

"git push" tells git that we want to push our changes, "origin" is the name of our newly-configured remote server and "master" is the name of the first branch.

In the future, when you have commits that you want to push to the server, you can simply type "git push".

I hope this article provided you with a basic understanding of how git can be used effectively for a team of developers. The next article in this series will provide a more in-depth analysis of git branches and why they are so effective.

By Jason Kurtz

How To Install Jenkins on Ubuntu 18.04

Written by Melissa Anderson and Kathleen Juell

This chapter demonstrates how to install Jenkins on a Linux server. Jenkins is a Continuous Integration tool that helps automate repetitive tasks like testing and deploying software. You will learn how to install Jenkins from the upstream Jenkins project's software repository, and how to run it on a server. Once Jenkins is running, you'll also learn how to configure the correct firewall rules to allow access to it, along with how to configure the administrative user and plugins.

Introduction

Jenkins is an open-source automation server that automates the repetitive technical tasks involved in the continuous integration and delivery of software. Jenkins is Java-based and can be installed from Ubuntu packages or by downloading and running its web application archive (WAR) file — a collection of files that make up a complete web application to run on a server.

In this tutorial, you will install Jenkins by adding its Debian package repository, and using that repository to install the package with `apt`.

Prerequisites

To follow this tutorial, you will need:

- One Ubuntu 18.04 server configured with a non-root sudo user and firewall by following the Ubuntu 18.04 initial server setup guide. We recommend starting with at least 1 GB of RAM. See Choosing the Right Hardware for Masters for guidance in planning the capacity of a production Jenkins installation.
- Java 8 installed, following our guidelines on installing specific versions of OpenJDK on Ubuntu 18.04.

Step 1 — Installing Jenkins

The version of Jenkins included with the default Ubuntu packages is often behind the latest available version from the project itself. To take advantage of the latest fixes and features, you can use the project-maintained packages to install Jenkins.

First, add the repository key to the system:

```
wget -q -O -
https://pkg.jenkins.io/debian/jenkins.io.key |
sudo apt-key add -
```

When the key is added, the system will return `OK`. Next, append the Debian package repository address to the server's `sources.list`:

```
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-
stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
```

When both of these are in place, run `update` so that `apt` will use the new repository:

```
sudo apt update
```

Finally, install Jenkins and its dependencies:

```
sudo apt install jenkins
```

Now that Jenkins and its dependencies are in place, we'll start the Jenkins server.

Step 2 — Starting Jenkins

Let's start Jenkins using `systemctl`:

```
sudo systemctl start jenkins
```

Since `systemctl` doesn't display output, you can use its `status` command to verify that Jenkins started successfully:

```
sudo systemctl status jenkins
```

If everything went well, the beginning of the output should show that the service is active and configured to start at boot:

Output

```
● jenkins.service - LSB: Start Jenkins at boot time
   Loaded: loaded (/etc/init.d/jenkins; generated)
   Active: active (exited) since Mon 2018-07-09 17:22:08 UTC; 6min
ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 0 (limit: 1153)
   CGroup: /system.slice/jenkins.service
```

Now that Jenkins is running, let's adjust our firewall rules so that we can reach it from a web browser to complete the initial setup.

Step 3 — Opening the Firewall

By default, Jenkins runs on port `8080`, so let's open that port using `ufw`:

```
sudo ufw allow 8080
```

Check `ufw`'s status to confirm the new rules:

```
sudo ufw status
```

You will see that traffic is allowed to port `8080` from anywhere:

Output

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
8080                       ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
8080 (v6)                  ALLOW       Anywhere (v6)
```

Note: If the firewall is inactive, the following commands will allow OpenSSH and enable the firewall:

```
sudo ufw allow OpenSSH
sudo ufw enable
```

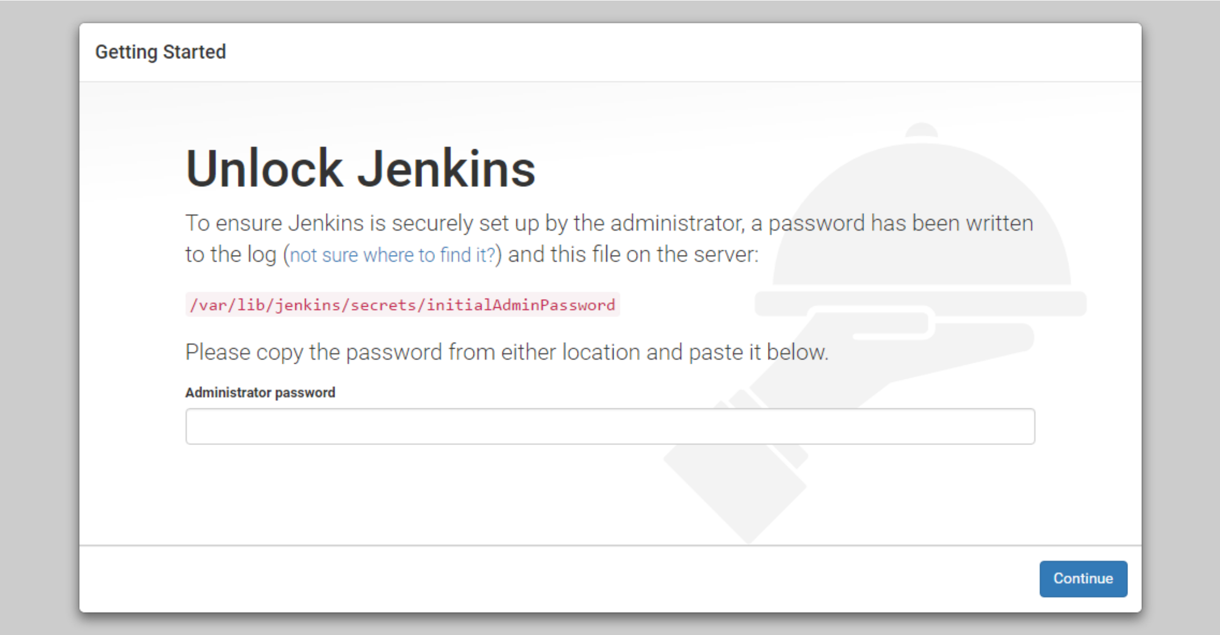
With Jenkins installed and our firewall configured, we can complete the initial setup.

Step 4 — Setting Up Jenkins

To set up your installation, visit Jenkins on its default port, `8080`, using your server domain name or IP address:

```
http://your_server_ip_or_domain:8080
```

You should see the Unlock Jenkins screen, which displays the location of the initial password:



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

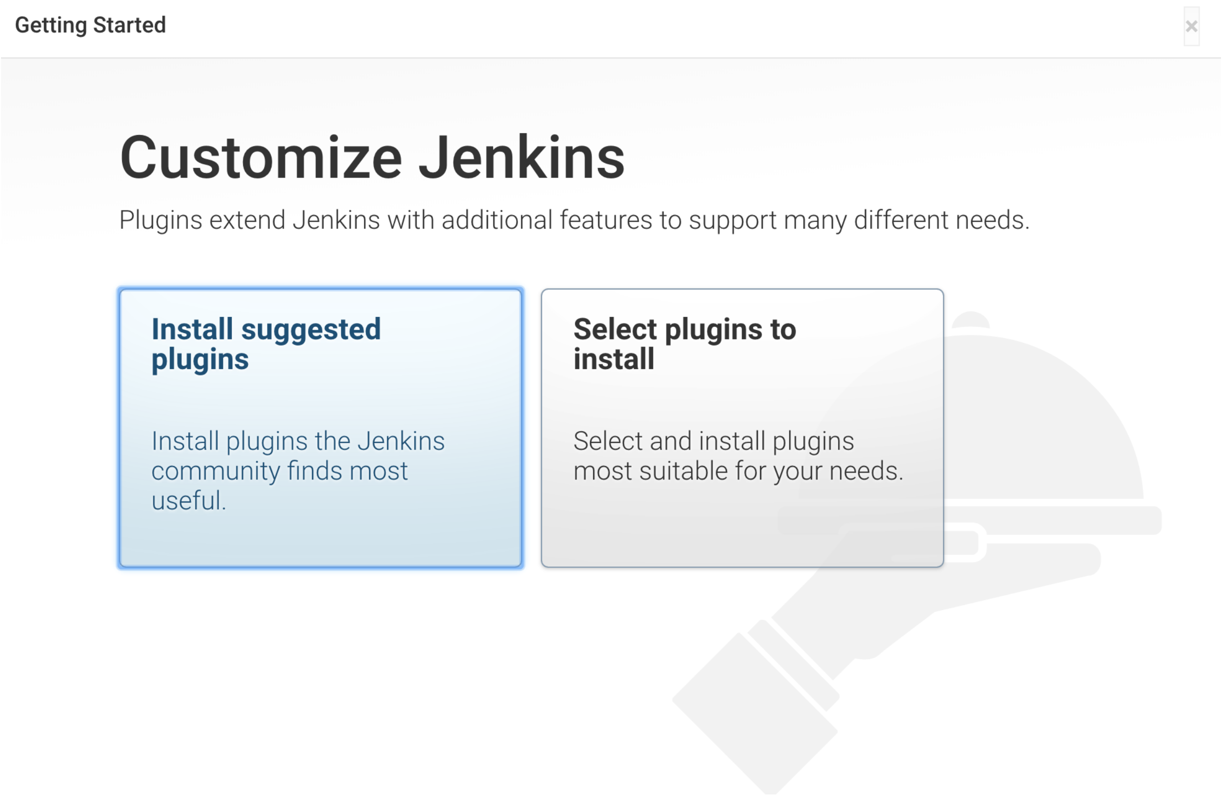
Unlock Jenkins screen

In the terminal window, use the `cat` command to display the password:

```
sudo cat
/var/lib/jenkins/secrets/initialAdminPassword
```

Copy the 32-character alphanumeric password from the terminal and paste it into the Administrator password field, then click Continue.

The next screen presents the option of installing suggested plugins or selecting specific plugins:


Getting Started

Customize Jenkins

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

Install suggested plugins

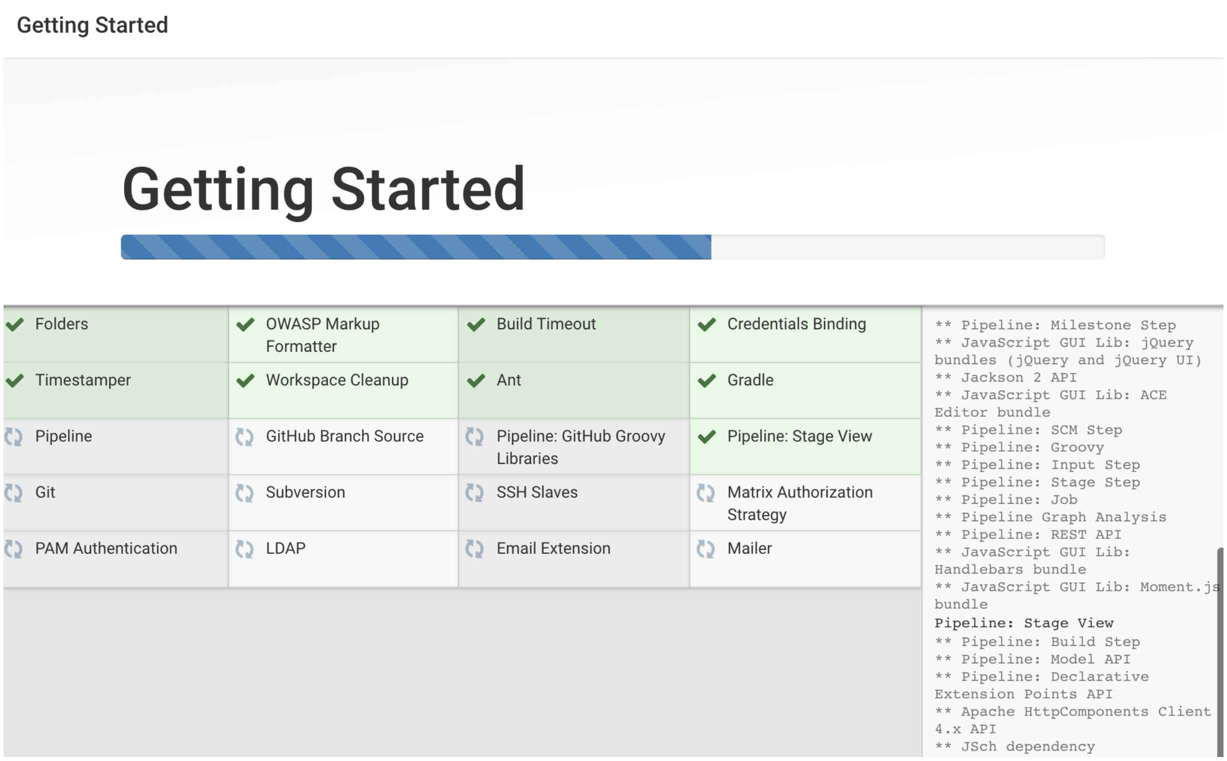
Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Customize Jenkins Screen

We'll click the Install suggested plugins option, which will immediately begin the installation process:


Getting Started

Getting Started

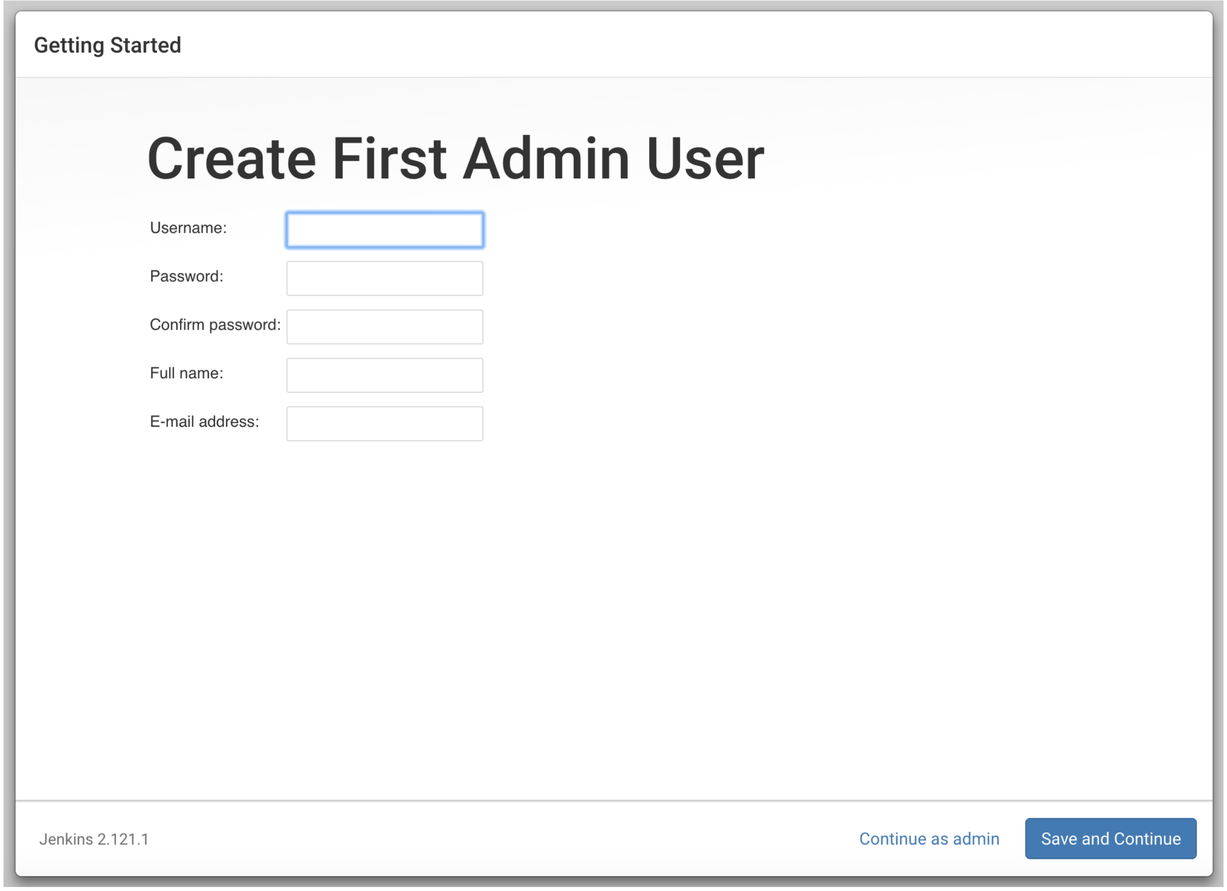
✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ Timestamper	✓ Workspace Cleanup	✓ Ant	✓ Gradle
⟳ Pipeline	⟳ GitHub Branch Source	⟳ Pipeline: GitHub Groovy Libraries	✓ Pipeline: Stage View
⟳ Git	⟳ Subversion	⟳ SSH Slaves	⟳ Matrix Authorization Strategy
⟳ PAM Authentication	⟳ LDAP	⟳ Email Extension	⟳ Mailer

```
** Pipeline: Milestone Step
** JavaScript GUI Lib: jQuery
bundles (jQuery and jQuery UI)
** Jackson 2 API
** JavaScript GUI Lib: ACE
Editor bundle
** Pipeline: SCM Step
** Pipeline: Groovy
** Pipeline: Input Step
** Pipeline: Stage Step
** Pipeline: Job
** Pipeline Graph Analysis
** Pipeline: REST API
** JavaScript GUI Lib:
Handlebars bundle
** JavaScript GUI Lib: Moment.js
bundle
Pipeline: Stage View
** Pipeline: Build Step
** Pipeline: Model API
** Pipeline: Declarative
Extension Points API
** Apache HttpComponents Client
4.x API
** JSch dependency
```

Jenkins Getting Started Install Plugins Screen

When the installation is complete, you will be prompted to set up the first administrative user. It's possible to skip this step and continue as `admin` using the initial password we used above, but we'll take a moment to create the user.

Note: The default Jenkins server is NOT encrypted, so the data submitted with this form is not protected. When you're ready to use this installation, follow the guide How to Configure Jenkins with SSL Using an Nginx Reverse Proxy on Ubuntu 18.04. This will protect user credentials and information about builds that are transmitted via the web interface.



Getting Started

Create First Admin User

Username:

Password:

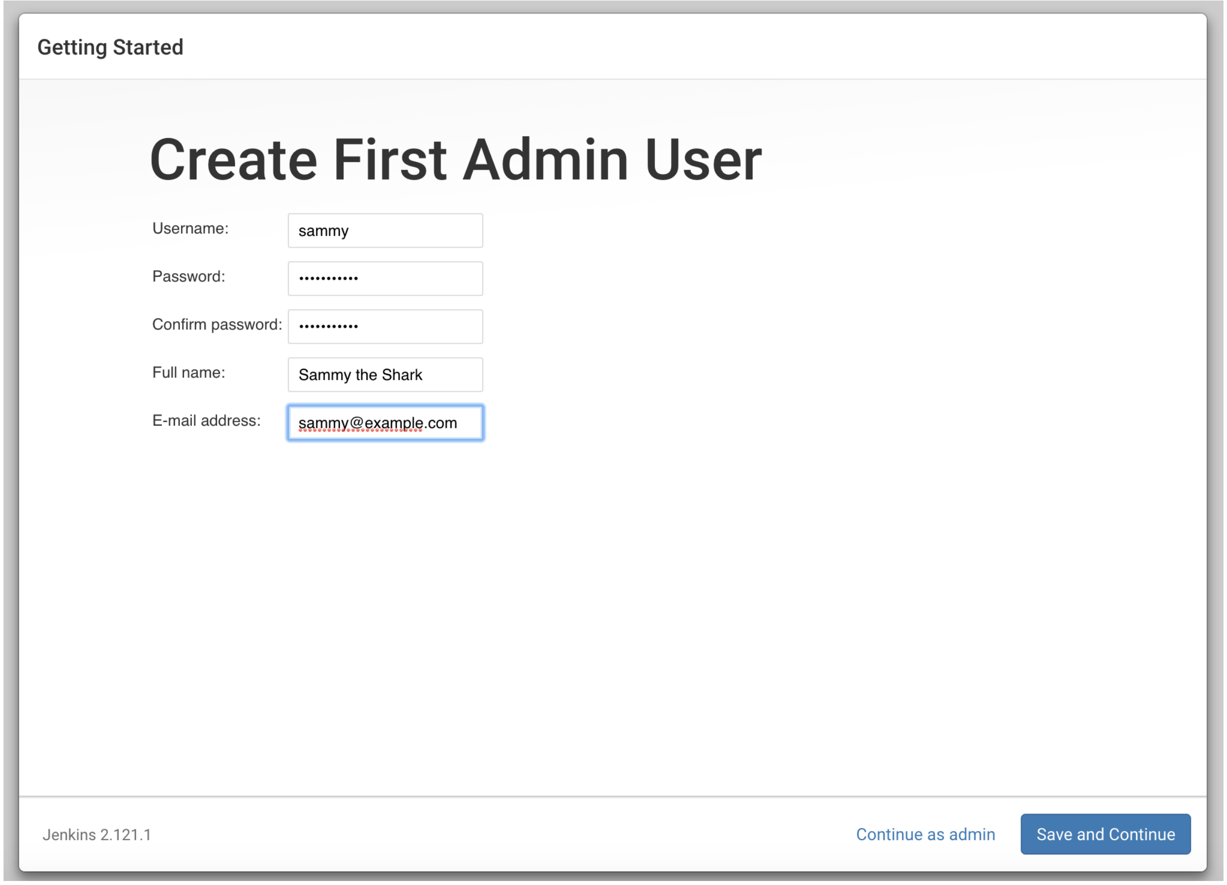
Confirm password:

Full name:

E-mail address:

Jenkins Create First Admin User Screen

Enter the name and password for your user:



Getting Started

Create First Admin User

Username: sammy

Password: •••••••••••

Confirm password: •••••••••••

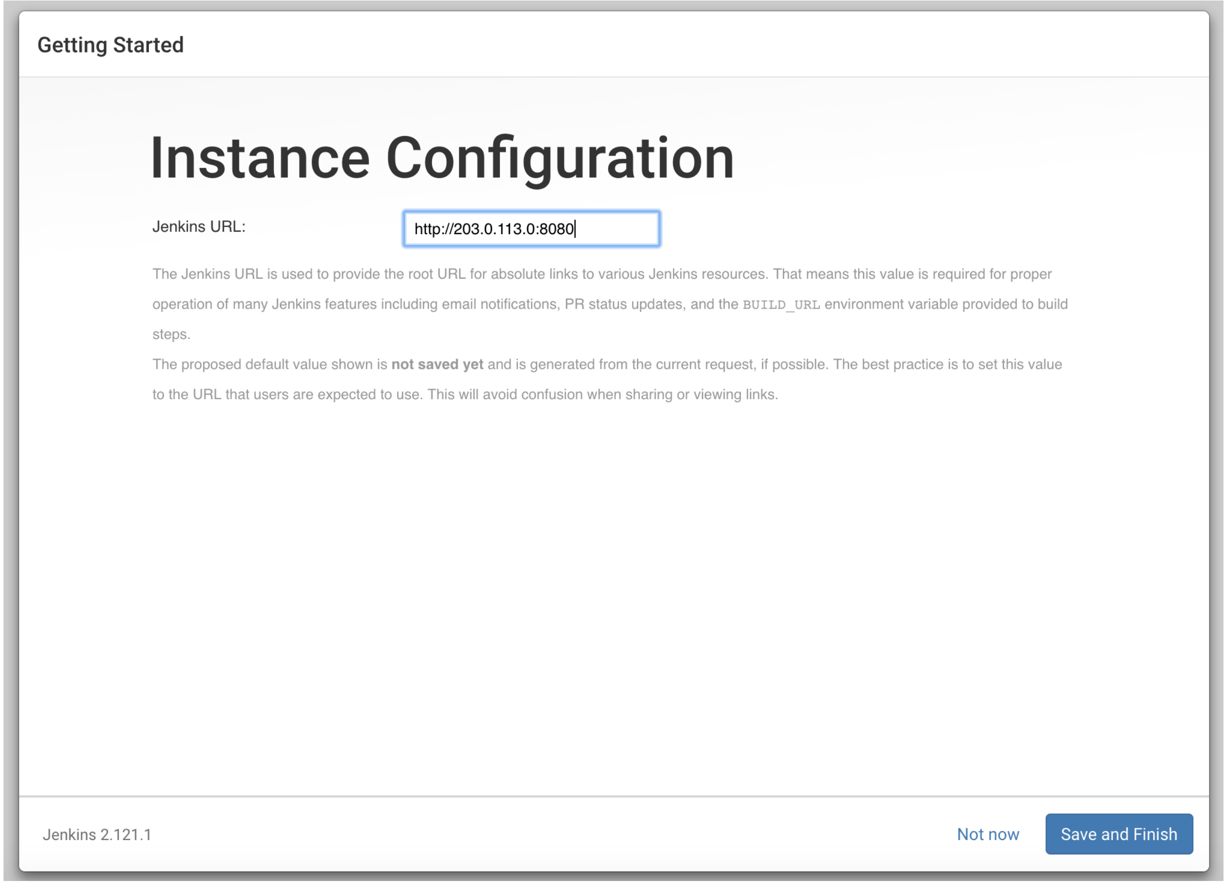
Full name: Sammy the Shark

E-mail address: sammy@example.com

Jenkins 2.121.1

Jenkins Create User

You will see an Instance Configuration page that will ask you to confirm the preferred URL for your Jenkins instance. Confirm either the domain name for your server or your server’s IP address:



Getting Started

Instance Configuration

Jenkins URL: http://203.0.113.0:8080

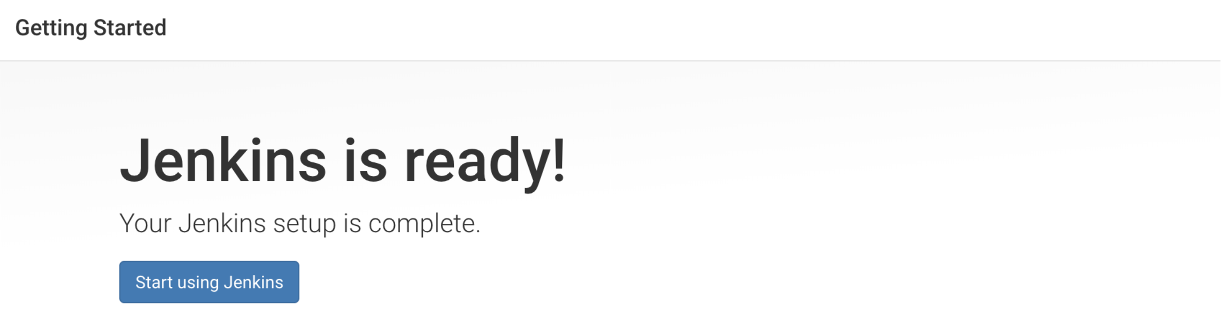
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.121.1

Jenkins Instance Configuration

After confirming the appropriate information, click Save and Finish. You will see a confirmation page confirming that “Jenkins is Ready!”:



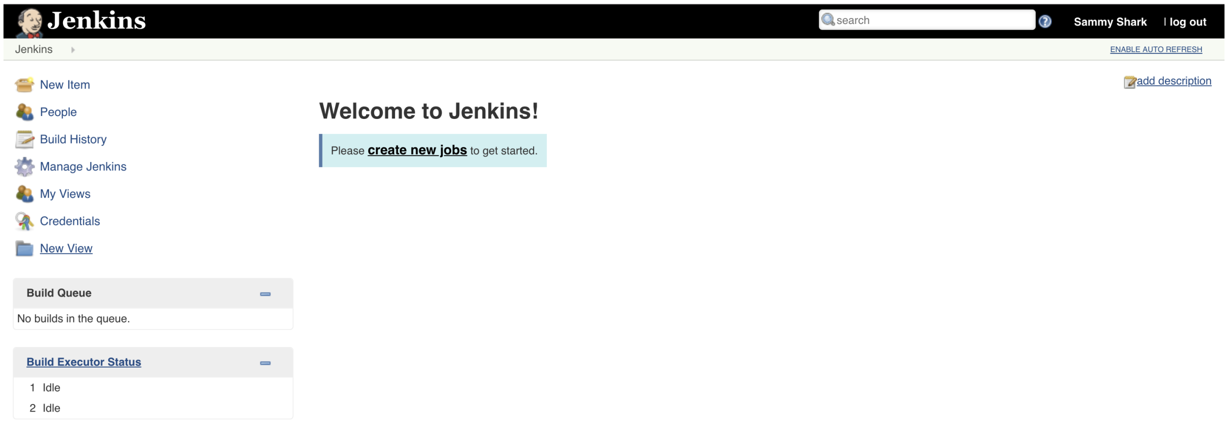
Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

Jenkins is ready screen

Click Start using Jenkins to visit the main Jenkins dashboard:



Welcome to Jenkins!

Please **create new jobs** to get started.

No builds in the queue.

Welcome to Jenkins Screen

At this point, you have completed a successful installation of Jenkins.

Conclusion

In this tutorial, you have installed Jenkins using the project-provided packages, started the server, opened the firewall, and created an administrative user. At this point, you can start exploring Jenkins.

When you’ve completed your exploration, if you decide to continue using Jenkins, follow the guide How to Configure Jenkins with SSL Using an Nginx Reverse Proxy on Ubuntu 18.04 to protect your passwords, as well as any sensitive system or product information that will be sent between your machine and the server in plain text.

How To Configure Jenkins with SSL Using an Nginx Reverse Proxy on Ubuntu 18.04

Written by Melissa Anderson and Kathleen Juell

In Chapters 11 and 12 you learned how to secure web servers with Letsencrypt. In this chapter you will place Jenkins behind an Nginx server, and secure it using a Letsencrypt TLS certificate. This configuration is more secure than running Jenkins on its own because it protects sensitive data like usernames and passwords when you are logging into Jenkins.

By default, Jenkins comes with its own built-in Winstone web server listening on port `8080`, which is convenient for getting started. It's also a good idea, however, to secure Jenkins with SSL to protect passwords and sensitive data transmitted through the web interface.

In this tutorial, you will configure Nginx as a reverse proxy to direct client requests to Jenkins.

Prerequisites

To begin, you'll need the following: - One Ubuntu 18.04 server configured with a non-root sudo user and firewall, following the Ubuntu 18.04 initial server setup guide. - Jenkins installed, following the steps in How to Install Jenkins on Ubuntu 18.04 - Nginx installed, following the steps in How to Install Nginx on Ubuntu 18.04 - An SSL certificate for a domain provided by Let's Encrypt. Follow How to Secure Nginx with Let's Encrypt on Ubuntu 18.04 to obtain this certificate. Note that you will need a registered domain name that you own or control. This tutorial will use the domain name example.com throughout.

Step 1 — Configuring Nginx

In the prerequisite tutorial How to Secure Nginx with Let's Encrypt on Ubuntu 18.04, you configured Nginx to use SSL in the `/etc/nginx/sites-available/example.com` file. Open this file to add your reverse proxy settings:

```
sudo nano /etc/nginx/sites-available/example.com
```

In the `server` block with the SSL configuration settings, add Jenkins-specific access and error logs:

/etc/nginx/sites-available/example.com

```
. . .
server {
        . . .
        # SSL Configuration
        #
        listen [::]:443 ssl ipv6only=on; # managed by Certbot
        listen 443 ssl; # managed by Certbot
        access_log            /var/log/nginx/jenkins.access.log;
        error_log             /var/log/nginx/jenkins.error.log;
        . . .
        }
```

Next let's configure the proxy settings. Since we're sending all requests to Jenkins, we'll comment out the default `try_files` line, which would otherwise return a 404 error before the request reaches Jenkins:

/etc/nginx/sites-available/example.com

```
. . .
           location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                # try_files $uri $uri/ =404;        }
. . .
```

Let's now add the proxy settings, which include: - `proxy_params`: The `/etc/nginx/proxy_params` file is supplied by Nginx and ensures that important information, including the hostname, the protocol of the client request, and the client IP address, is retained and available in the log files. - `proxy_pass`: This sets the protocol and address of the proxied server, which in this case will be the Jenkins server accessed via `localhost` on port `8080`. - `proxy_read_timeout`: This enables an increase from Nginx's 60 second default to the Jenkins-recommended 90 second value. - `proxy_redirect`: This ensures that responses are correctly rewritten to include the proper host name.
Be sure to substitute your SSL-secured domain name for **`example.com`** in the `proxy_redirect` line below:

/etc/nginx/sites-available/example.com

```
Location /
. . .
           location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                # try_files $uri $uri/ =404;
                include /etc/nginx/proxy_params;
                proxy_pass          http://localhost:8080;
                proxy_read_timeout  90s;
                # Fix potential "It appears that your reverse proxy
setup is broken" error.
                proxy_redirect      http://localhost:8080
https://example.com;
```

Once you've made these changes, save the file and exit the editor. We'll hold off on restarting Nginx until after we've configured Jenkins, but we can test our configuration now:

```
sudo nginx -t
```

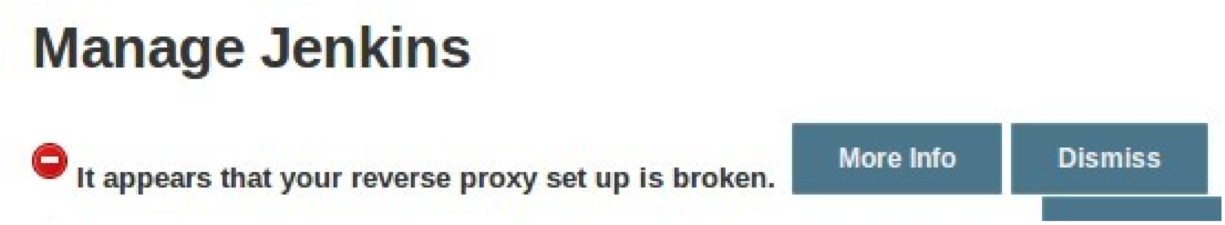
If all is well, the command will return:

Output

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

If not, fix any reported errors until the test passes.

Note: If you misconfigure the `proxy_pass` (by adding a trailing slash, for example), you will get something similar to the following in your Jenkins Configuration page.



Manage Jenkins

It appears that your reverse proxy set up is broken.

Jenkins error: Reverse proxy set up is broken

If you see this error, double-check your `proxy_pass` and `proxy_redirect` settings in the Nginx configuration.

Step 2 — Configuring Jenkins

For Jenkins to work with Nginx, you will need to update the Jenkins configuration so that the Jenkins server listens only on the `localhost` interface rather than on all interfaces (`0.0.0.0`). If Jenkins listens on all interfaces, it’s potentially accessible on its original, unencrypted port (`8080`).

Let’s modify the `/etc/default/jenkins` configuration file to make these adjustments:

```
sudo nano /etc/default/jenkins
```

Locate the `JENKINS_ARGS` line and add `--httpListenAddress=127.0.0.1` to the existing arguments:

/etc/default/jenkins

```
. . .
JENKINS_ARGS="--webroot=/var/cache/$NAME/war --httpPort=$HTTP_PORT
--httpListenAddress=127.0.0.1"
```

Save and exit the file.

To use the new configuration settings, restart Jenkins:

```
sudo systemctl restart jenkins
```

Since `systemctl` doesn’t display output, check the status:

```
sudo systemctl status jenkins
```

You should see the `active (exited)` status in the `Active` line:

Output

```
● jenkins.service - LSB: Start Jenkins at boot time
   Loaded: loaded (/etc/init.d/jenkins; generated)
   Active: active (exited) since Mon 2018-07-09 20:26:25 UTC; 11s
ago
     Docs: man:systemd-sysv-generator(8)
  Process: 29766 ExecStop=/etc/init.d/jenkins stop (code=exited,
status=0/SUCCESS)
  Process: 29812 ExecStart=/etc/init.d/jenkins start (code=exited,
status=0/SUCCESS)
```

Restart Nginx:

```
sudo systemctl restart nginx
```

Check the status:

```
sudo systemctl status nginx
```

Output

```
● nginx.service - A high performance web server and a reverse proxy
server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled;
vendor preset: enabled)
   Active: active (running) since Mon 2018-07-09 20:27:23 UTC; 31s
ago
     Docs: man:nginx(8)
  Process: 29951 ExecStop=/sbin/start-stop-daemon --quiet --stop --
retry QUIT/5 --pidfile /run/nginx.pid (code=exited,
status=0/SUCCESS)
  Process: 29963 ExecStart=/usr/sbin/nginx -g daemon on;
master_process on; (code=exited, status=0/SUCCESS)
  Process: 29952 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on;
master_process on; (code=exited, status=0/SUCCESS)
 Main PID: 29967 (nginx)
```

With both servers restarted, you should be able to visit the domain using either HTTP or HTTPS. HTTP requests will be redirected automatically to HTTPS, and the Jenkins site will be served securely.

Step 3 — Testing the Configuration

Now that you have enabled encryption, you can test the configuration by resetting the administrative password. Let’s start by visiting the site via

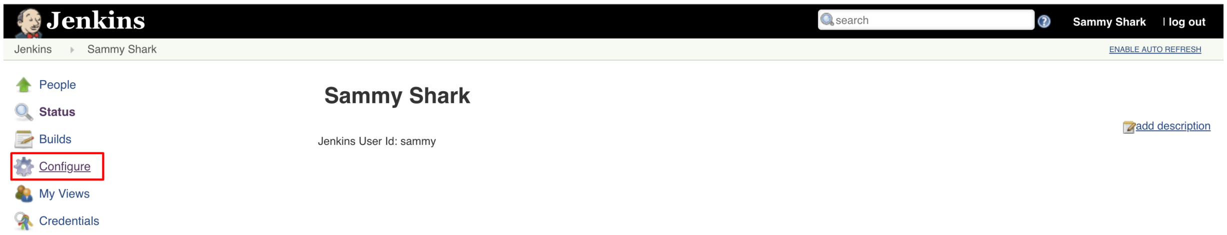
HTTP to verify that you can reach Jenkins and are redirected to HTTPS.

In your web browser, enter `http://example.com`, substituting your domain for `example.com`. After you press `ENTER`, the URL should start with `https` and the location bar should indicate that the connection is secure.

You can enter the administrative username you created in [How To Install Jenkins on Ubuntu 18.04]() in the User field, and the password that you selected in the Password field.

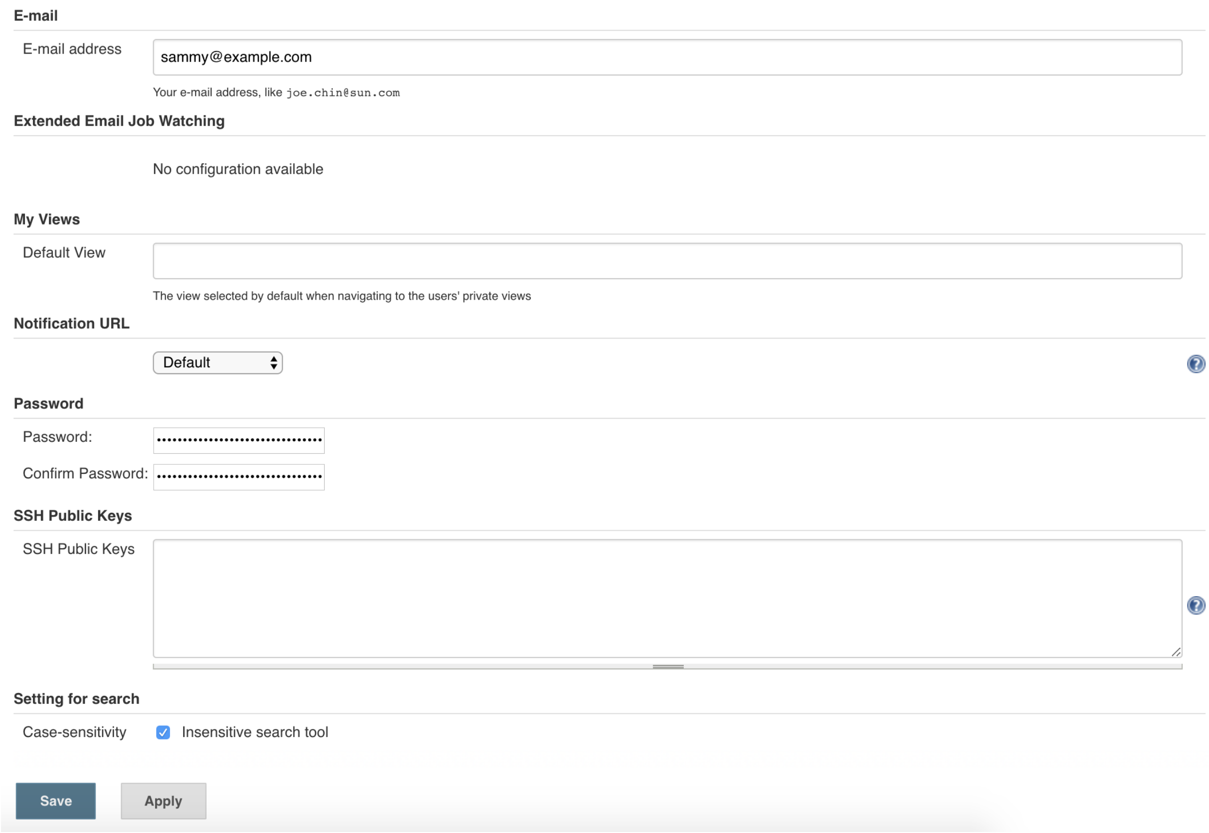
Once logged in, you can change the password to be sure it's secure.

Click on your username in the upper-right-hand corner of the screen. On the main profile page, select Configure from the list on the left side of the page:



Navigate to Jenkins password page

This will take you to a new page, where you can enter and confirm a new password:



E-mail

E-mail address: sammy@example.com

Your e-mail address, like joe.chin@sun.com

Extended Email Job Watching

No configuration available

My Views

Default View:

The view selected by default when navigating to the users' private views

Notification URL

Default

Password

Password: ••••••••••••••••••••••••••••••••

Confirm Password: ••••••••••••••••••••••••••••••••

SSH Public Keys

SSH Public Keys:

Setting for search

Case-sensitivity: ☑ Insensitive search tool

Jenkins create password page

Confirm the new password by clicking Save. You can now use the Jenkins web interface securely.

Conclusion

In this tutorial, you configured Nginx as a reverse proxy to Jenkins' built-in web server to secure your credentials and other information transmitted via the web interface. Now that Jenkins is secure, you can learn how to set up a continuous integration pipeline to automatically test code changes. Other resources to consider if you are new to Jenkins are the Jenkins project's "Creating your first Pipeline" tutorial or the library of community-contributed plugins.