Introduction

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 discuss how to install an Apache web server on your Ubuntu 16.04 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 configure 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 16.04](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-16-04).

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

Step 1: Install Apache

Apache is available within Ubuntu’s default software repositories, so we will install it using conventional package management tools.

We will begin by updating the local package index to reflect the latest upstream changes. Afterwards, we can install the apache2 package:

* sudo apt-get update
* sudo apt-get install apache2

Copy

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

Step 2: Adjust the Firewall

Before we can test Apache, we need to modify our firewall 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. We can use these profiles to simplify the process of enabling or disabling access to Apache through our firewall.

We can list the ufw application profiles by typing:

* sudo ufw app list

Copy

You should get a listing 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)

For our purposes, we will allow incoming traffic for the **Apache Full** profile by typing:

* sudo ufw allow 'Apache Full'

Copy

You can verify the change by typing:

* sudo ufw status

Copy

You should see HTTP traffic allowed in the displayed output:

Output

Status: active

To Action From

-- ------ ----

OpenSSH ALLOW Anywhere

Apache Full ALLOW Anywhere

OpenSSH (v6) ALLOW Anywhere (v6)

Apache Full (v6) ALLOW Anywhere (v6)

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

Step 3: Check your Web Server

At the end of the installation process, Ubuntu 16.04 starts Apache. 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:

* sudo systemctl status apache2

Copy

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 Fri 2017-05-19 18:30:10 UTC; 1h 5min ago

Docs: man:systemd-sysv-generator(8)

Process: 4336 ExecStop=/etc/init.d/apache2 stop (code=exited, status=0/SUCCESS)

Process: 4359 ExecStart=/etc/init.d/apache2 start (code=exited, status=0/SUCCESS)

Tasks: 55

Memory: 2.3M

CPU: 4.094s

CGroup: /system.slice/apache2.service

├─4374 /usr/sbin/apache2 -k start

├─4377 /usr/sbin/apache2 -k start

└─4378 /usr/sbin/apache2 -k start

May 19 18:30:09 ubuntu-512mb-nyc3-01 systemd[1]: Stopped LSB: Apache2 web server.

May 19 18:30:09 ubuntu-512mb-nyc3-01 systemd[1]: Starting LSB: Apache2 web server...

May 19 18:30:09 ubuntu-512mb-nyc3-01 apache2[4359]: \* Starting Apache httpd web server apache2

May 19 18:30:09 ubuntu-512mb-nyc3-01 apache2[4359]: AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message

May 19 18:30:10 ubuntu-512mb-nyc3-01 apache2[4359]: \*

May 19 18:30:10 ubuntu-512mb-nyc3-01 systemd[1]: Started LSB: Apache2 web server.

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 Apache.

You can access the default Apache landing page to confirm that the software is running properly. You can access this through your server’s domain name or IP address.

If you are using DigitalOcean and do not have a domain name set up for your server, you can follow our guide [how to set up a domain with DigitalOcean](https://digitalocean.com/community/articles/how-to-set-up-a-host-name-with-digitalocean) to set one up.

If you do not want to set up a domain name for your server, you can use your server’s public 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

Copy

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:

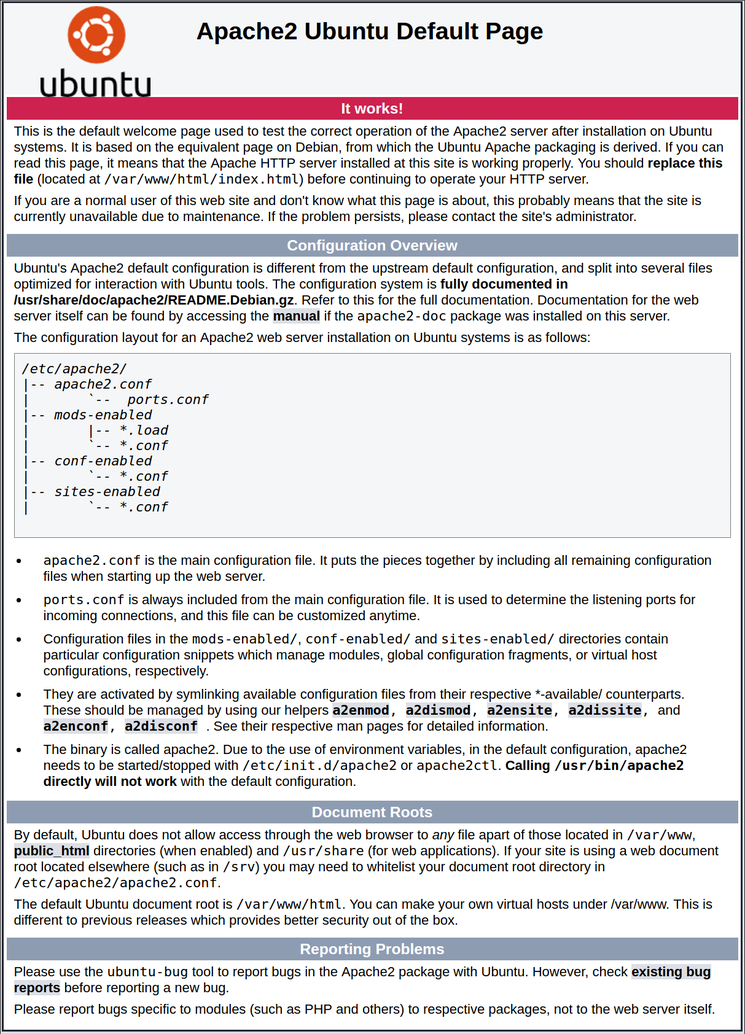
* sudo apt-get install curl
* curl -4 icanhazip.com

Copy

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

http://server\_domain\_or\_IP

You should see the default Ubuntu 16.04 Apache web page, which should look something like this:



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

Step 4: Manage the Apache Process

Now that you have your web server up and running, we can go over some basic management commands.

To stop your web server, you can type:

* sudo systemctl stop apache2

Copy

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

* sudo systemctl start apache2

Copy

To stop and then start the service again, type:

* sudo systemctl restart apache2

Copy

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

* sudo systemctl reload apache2

Copy

By default, Apache 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 apache2

Copy

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

* sudo systemctl enable apache2

Copy

Apache should now start automatically when the server boots again.

Step 5: Get Familiar with Important Apache Files and Directories

Now that you know how to manage the 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 (see below). 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.

# How To Set Up Apache Virtual Hosts on Ubuntu 16.04

Introduction

The Apache web server is the most popular way of serving web content on the internet. It accounts for more than half of all active websites on the internet and is extremely powerful and flexible.

Apache breaks its functionality and components into individual units that can be customized and configured independently. The basic unit that describes an individual site or domain is called a virtual host.

These designations allow the administrator to use one server to host multiple domains or sites off of a single interface or IP by using a matching mechanism. This is relevant to anyone looking to host more than one site off of a single VPS.

Each domain that is configured will direct the visitor to a specific directory holding that site’s information, never indicating that the same server is also responsible for other sites. This scheme is expandable without any software limit as long as your server can handle the load.

In this guide, we will walk you through how to set up Apache virtual hosts on an Ubuntu 16.04 VPS. During this process, you’ll learn how to serve different content to different visitors depending on which domains they are requesting.

Prerequisites

Before you begin this tutorial, you should [create a non-root user](https://www.digitalocean.com/community/articles/initial-server-setup-with-ubuntu-16-04) as described in steps 1-4 here.

You will also need to have Apache installed in order to work through these steps. If you haven’t already done so, you can get Apache installed on your server through apt-get:

* sudo apt-get update
* sudo apt-get install apache2

Copy

After these steps are complete, we can get started.

For the purposes of this guide, our configuration will make a virtual host for example.com and another for test.com. These will be referenced throughout the guide, but you should substitute your own domains or values while following along.

To learn [how to set up your domain names with DigitalOcean](https://www.digitalocean.com/community/articles/how-to-set-up-a-host-name-with-digitalocean), follow this link. If you do *not* have domains available to play with, you can use dummy values.

We will show how to edit your local hosts file later on to test the configuration if you are using dummy values. This will allow you to test your configuration from your home computer, even though your content won’t be available through the domain name to other visitors.

Step One — Create the Directory Structure

The first step that we are going to take is to make a directory structure that will hold the site data that we will be serving to visitors.

Our document root (the top-level directory that Apache looks at to find content to serve) will be set to individual directories under the /var/www directory. We will create a directory here for both of the virtual hosts we plan on making.

Within each of *these* directories, we will create a public\_html folder that will hold our actual files. This gives us some flexibility in our hosting.

For instance, for our sites, we’re going to make our directories like this:

* sudo mkdir -p /var/www/example.com/public\_html
* sudo mkdir -p /var/www/test.com/public\_html

Copy

The portions in red represent the domain names that we are wanting to serve from our VPS.

Step Two — Grant Permissions

Now we have the directory structure for our files, but they are owned by our root user. If we want our regular user to be able to modify files in our web directories, we can change the ownership by doing this:

* sudo chown -R $USER:$USER /var/www/example.com/public\_html
* sudo chown -R $USER:$USER /var/www/test.com/public\_html

Copy

The $USER variable will take the value of the user you are currently logged in as when you press **Enter**. By doing this, our regular user now owns the public\_html subdirectories where we will be storing our content.

We should also modify our permissions a little bit to ensure that read access is permitted to the general web directory and all of the files and folders it contains so that pages can be served correctly:

* sudo chmod -R 755 /var/www

Copy

Your web server should now have the permissions it needs to serve content, and your user should be able to create content within the necessary folders.

Step Three — Create Demo Pages for Each Virtual Host

We have our directory structure in place. Let’s create some content to serve.

We’re just going for a demonstration, so our pages will be very simple. We’re just going to make an index.html page for each site.

Let’s start with example.com. We can open up an index.html file in our editor by typing:

* nano /var/www/example.com/public\_html/index.html

Copy

In this file, create a simple HTML document that indicates the site it is connected to. My file looks like this:

/var/www/example.com/public\_html/index.html

<html>

<head>

<title>Welcome to Example.com!</title>

</head>

<body>

<h**1**>Success! The example.com virtual host is working!</h**1**>

</body>

</html>

Copy

Save and close the file when you are finished.

We can copy this file to use as the basis for our second site by typing:

* cp /var/www/example.com/public\_html/index.html /var/www/test.com/public\_html/index.html

Copy

We can then open the file and modify the relevant pieces of information:

* nano /var/www/test.com/public\_html/index.html

Copy

/var/www/test.com/public\_html/index.html

<html>

<head>

<title>Welcome to Test.com!</title>

</head>

<body> <h**1**>Success! The test.com virtual host is working!</h**1**>

</body>

</html>

Copy

Save and close this file as well. You now have the pages necessary to test the virtual host configuration.

Step Four — Create New Virtual Host Files

Virtual host files are the files that specify the actual configuration of our virtual hosts and dictate how the Apache web server will respond to various domain requests.

Apache comes with a default virtual host file called 000-default.conf that we can use as a jumping off point. We are going to copy it over to create a virtual host file for each of our domains.

We will start with one domain, configure it, copy it for our second domain, and then make the few further adjustments needed. The default Ubuntu configuration requires that each virtual host file end in .conf.

Create the First Virtual Host File

Start by copying the file for the first domain:

* sudo cp /etc/apache2/sites-available/000-default.conf /etc/apache2/sites-available/example.com.conf

Copy

Open the new file in your editor with root privileges:

* sudo nano /etc/apache2/sites-available/example.com.conf

Copy

The file will look something like this (I’ve removed the comments here to make the file more approachable):

/etc/apache2/sites-available/example.com.conf

<VirtualHost \*:8**0**>

ServerAdmin webmaster@localhost

DocumentRoot /var/www/html

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

</VirtualHost>

Copy

As you can see, there’s not much here. We will customize the items here for our first domain and add some additional directives. This virtual host section matches *any* requests that are made on port 80, the default HTTP port.

First, we need to change the ServerAdmin directive to an email that the site administrator can receive emails through.

ServerAdmin admin@example.com

After this, we need to *add* two directives. The first, called ServerName, establishes the base domain that should match for this virtual host definition. This will most likely be your domain. The second, called ServerAlias, defines further names that should match as if they were the base name. This is useful for matching hosts you defined, like www:

ServerName example.com

ServerAlias www.example.com

The only other thing we need to change for a basic virtual host file is the location of the document root for this domain. We already created the directory we need, so we just need to alter the DocumentRoot directive to reflect the directory we created:

DocumentRoot /var/www/example.com/public\_html

In total, our virtualhost file should look like this:

/etc/apache2/sites-available/example.com.conf

<VirtualHost \*:8**0**>

ServerAdmin admin@example.com

ServerName example.com

ServerAlias www.example.com

DocumentRoot /var/www/example.com/public\_html

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

</VirtualHost>

Copy

Save and close the file.

Copy First Virtual Host and Customize for Second Domain

Now that we have our first virtual host file established, we can create our second one by copying that file and adjusting it as needed.

Start by copying it:

* sudo cp /etc/apache2/sites-available/example.com.conf /etc/apache2/sites-available/test.com.conf

Copy

Open the new file with root privileges in your editor:

* sudo nano /etc/apache2/sites-available/test.com.conf

Copy

You now need to modify all of the pieces of information to reference your second domain. When you are finished, it may look something like this:

/etc/apache2/sites-available/test.com.conf

<VirtualHost \*:8**0**>

ServerAdmin admin@test.com

ServerName test.com

ServerAlias www.test.com

DocumentRoot /var/www/test.com/public\_html

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

</VirtualHost>

Copy

Save and close the file when you are finished.

Step Five — Enable the New Virtual Host Files

Now that we have created our virtual host files, we must enable them. Apache includes some tools that allow us to do this.

We can use the a2ensite tool to enable each of our sites like this:

* sudo a2ensite example.com.conf
* sudo a2ensite test.com.conf

Copy

Next, disable the default site defined in 000-default.conf:

* sudo a2dissite 000-default.conf

Copy

When you are finished, you need to restart Apache to make these changes take effect:

* sudo systemctl restart apache2

Copy

In other documentation, you may also see an example using the service command:

* sudo service apache2 restart

Copy

This command will still work, but it may not give the output you’re used to seeing on other systems, since it’s now a wrapper around systemd’s systemctl.

Step Six — Set Up Local Hosts File (Optional)

If you haven’t been using actual domain names that you own to test this procedure and have been using some example domains instead, you can at least test the functionality of this process by temporarily modifying the hosts file on your local computer.

This will intercept any requests for the domains that you configured and point them to your VPS server, just as the DNS system would do if you were using registered domains. This will only work from your computer though, and is simply useful for testing purposes.

Make sure you are operating on your local computer for these steps and not your VPS server. You will need to know the computer’s administrative password or otherwise be a member of the administrative group.

If you are on a Mac or Linux computer, edit your local file with administrative privileges by typing:

* sudo nano /etc/hosts

Copy

If you are on a Windows machine, you can [find instructions on altering your hosts file](http://support.microsoft.com/kb/923947) here.

The details that you need to add are the public IP address of your VPS server followed by the domain you want to use to reach that VPS.

For the domains that I used in this guide, assuming that my VPS IP address is 111.111.111.111, I could add the following lines to the bottom of my hosts file:

/etc/hosts

127.0.0.1 localhost

127.0.1.1 guest-desktop

111.111.111.111 example.com

111.111.111.111 test.com

Copy

This will direct any requests for example.com and test.com on our computer and send them to our server at 111.111.111.111. This is what we want if we are not actually the owners of these domains in order to test our virtual hosts.

Save and close the file.

Step Seven — Test your Results

Now that you have your virtual hosts configured, you can test your setup easily by going to the domains that you configured in your web browser:

http://example.com

You should see a page that looks like this:

Apache virt host example

Likewise, if you can visit your second page:

http://test.com

You will see the file you created for your second site:

Apache virt host test

If both of these sites work well, you’ve successfully configured **two** virtual hosts on the same server.

If you adjusted your home computer’s hosts file, you may want to delete the lines you added now that you verified that your configuration works. This will prevent your hosts file from being filled with entries that are not actually necessary.

If you need to access this long term, consider purchasing a domain name for each site you need and [setting it up to point to your VPS server](https://www.digitalocean.com/community/articles/how-to-set-up-a-host-name-with-digitalocean).

Conclusion

If you followed along, you should now have a single server handling two separate domain names. You can expand this process by following the steps we outlined above to make additional virtual hosts.