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 14.04 VPS. During this process, you’ll learn how to serve different content to different visitors depending on which domains they are requesting.

## ****PREREQUISITES****

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

After these steps are complete, we can get started.

For the purposes of this guide, my 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.

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

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

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

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

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

<html>

  <head>

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

  </head>

  <body>

    <h1>Success!  The example.com virtual host is working!</h1>

  </body>

</html>

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

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

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

<html>

  <head>

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

  </head>

  <body>

    <h1>Success!  The test.com virtual host is working!</h1>

  </body>

</html>

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

Open the new file in your editor with root privileges:

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

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

<VirtualHost \*:80>

    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html

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

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

</VirtualHost>

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](mailto: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](http://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 DocumentRootdirective to reflect the directory we created:

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

In total, our virtualhost file should look like this:

<VirtualHost \*:80>

    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>

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

Open the new file with root privileges in your editor:

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

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:

<VirtualHost \*:80>

    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>

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

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

sudo service apache2 restart

You will most likely receive a message saying something similar to:

 \* Restarting web server apache2

 AH00558: apache2: Could not reliably determine the server’s fully qualified domain name, using 127.0.0.1. Set the ‘ServerName’ directive globally to suppress this message

This is a harmless message that does not affect our site.

## ****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

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:

127.0.0.1   localhost

127.0.1.1   guest-desktop

111.111.111.111 example.com

111.111.111.111 test.com

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](http://example.com/)

You should see a page that looks like this:

http://35.185.152.13/wp-content/uploads/2017/11/2.png

Likewise, if you can visit your second page:

[http://test.com](http://test.com/)

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

http://35.185.152.13/wp-content/uploads/2017/11/1.png

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.

## ****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.

There is no software limit on the number of domain names Apache can handle, so feel free to make as many as your server is capable of handling.