Introduction

The Apache web server is a popular method for serving websites on the internet. As of 2019, it is estimated to serve 29% of all active websites and it offers robustness and flexibility for developers. Using Apache, an administrator can set up one server to host multiple domains or sites off of a single interface or IP by using a matching system.

Each domain or individual site — known as a **virtual host** — that is configured using Apache will direct the visitor to a specific directory holding that site's information. This is done without 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. The basic unit that describes an individual site or domain is called a virtual hostIn this guide, we will walk you through how to set up Apache virtual hosts on an Ubuntu 20.04 server. 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.

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 the **apt** package manager:

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

If you would like more detailed instructions as well as firewall setup, please refer to our guide How To Install the Apache Web Server on Ubuntu 20.04.

For the purposes of this guide, our configuration will make a virtual host for **example.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 test values. This will allow you to validate 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 the virtual host we plan on making.

Within this directory, 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 site, we're going to make our directory as follows. If you are using actual domain or alternate values, swap out the highlighted text for this.

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

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
```

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 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 a Demo Page for the Virtual Host

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

For demonstration purposes, we'll make an **index.html** page this site.

Let's begin with **example.com**. We can open up an **index.html** file in a text editor, in this case we'll use nano:

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

Within this file, create an HTML document that indicates the site it is connected to, like the following:

```
<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 (in nano, press CTRL + X then Y then ENTER) when you are finished.

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 our domain.

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

Start by copying the default 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
```

With comments removed, the file will look similar to this:

```
<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>
```

Within this file, we will customize the items 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 our 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
```

At this point, save and close the file.

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'll be using the **a2ensite** tool to enable each of our sites. If you would like to read more about this script, you can refer to the **a2ensite** documentation.

```
sudo a2ensite example.com.conf
sudo a2ensite test.com.conf
```

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

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

When you are finished, you need to restart Apache to make these changes take effect and use **systemctl status** to verify the success of the restart.

```
sudo systemctl restart apache2
sudo systemctl status apache2
```

Your server should now be set up to serve your website.

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 local computer though, and only 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 server followed by the domain you want to use to reach that server.

Using the domains used in this guide, and replacing your server IP for the **your_server_IP** text, your file should look like this:

```
127.0.0.1 localhost
127.0.1.1 guest-desktop
your_server_IP example.com
```

Save and close the file.

This will direct any requests for **example.com** on our computer and send them to our server. This is what we want if we are not actually the owners of this domain in order to test our virtual host.

Step Seven — Test your Results

Now that you have your virtual host configured, you can test your setup by going to the domain that you configured in your web browser:

```
http://example.com
```

You should see a page that looks like this:

If this site work as expected, you've successfully configured a virtual host on a 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 no longer necessary.

If you need to access this long term, consider adding a domain name for each site you need and setting it up to point to your server .

DOWNLOAD AS PDF

Join The Conversation

Please Log In to post.