https://www.digitalocean.com/community/tutorials/how-to-set-up-apache-virtual-hosts-on-debian-8

How To Set Up Apache Virtual Hosts on Debian 8

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 you can customize independently. The basic unit that describes an individual site or domain is called a *virtual host*.

Using virtual hosts, you can use one server to host multiple domains or sites off of a single interface or IP by using a matching mechanism. You configure a request for a domain to direct the visitor to a specific directory holding that site's information. In other words, you can host more than one web site on a single server. This scheme is expandable without any software limit as long as your server can handle the load.

In this tutorial, you'll set up two Apache virtual hosts on a Debian 8 server, serving different content to visitors based on the domain they visit.

Prerequisites

To complete this tutorial, you will need:

- A Debian 8 server with a non-root user with sudo privileges. You can set up a user with these privileges in our Initial Server Setup with Debian 8 guide.

- Apache installed and configured, as shown in How To Install Linux, Apache, MySQL, PHP (LAMP) Stack on Debian 8.

  In this guide, we'll create virtual hosts for `example.com` and `test.com`, but you can substitute your own domains or values while following along. To point your domain names at your server, follow our tutorial How To Set Up a Host Name with DigitalOcean.

  If you don't have domains available to play with, you can use `example.com` and `test.com` and follow Step 5 of this tutorial to configure your local hosts file to map those domains to your server's IP address. This will allow you to test your configuration from your local computer.

Step 1 — Creating 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 for each of the virtual hosts we'll configure.

Within each of these directories, we'll create a folder called `public_html` that will hold the web pages we want to serve. This gives us a little more flexibility in how we deploy more complex web applications in the future; the `public_html` folder will hold web content we want to serve, and the parent folder can hold scripts or application code to support web content.

Create the directories using the following commands:

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

Since we created the directories with `sudo`, they are owned by our root user. If we want our regular user to be able to modify files in our web directories, we change the ownership, like 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 uses 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. Execute this command to change the permissions on the `/var/www` folder and its children:

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

If you're new to managing permissions on Linux, see this tutorial.

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. Let's create an HTML file for each site.

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

Step 2 — Creating Default Pages for Each Virtual Host

Let's create a simple `index.html` page for each site. This will help us ensure that our virtual hosts are configured properly later on.

Let's start with the page for `example.com`. Edit a new `index.html` file with the following command:

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

In this file, create a simple HTML document that indicates that the visitor is looking at `example.com`'s home page:

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

```
<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.

Now copy this file to the `test.com` site:

- `cp /var/www/example.com/public_html/index.html /var/www/test.com/public_html/index.html`

Then open the file in your editor:

- `nano /var/www/test.com/public_html/index.html`

Change the file so it references `test.com` instaed of `example.com`:

/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. You now have the pages necessary to test the virtual host configuration. Next, let's configure the virtual hosts.

Step 3 — Create New Virtual Host Files

Virtual host files 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 you can use as a jumping off point. Copy this file for the first domain:

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

Note: The default Apache configuration in Debian 8 requires that each virtual host file end in `.conf`.

Open the new file in your editor:

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

The file will look something like the following example, with some additional comments:

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

```
<VirtualHost *:80>

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

This virtual host matches *any* requests that are made on port `80`, the default HTTP port. Let's make a few changes to this configuration, and add a few new directives.

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

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

```
ServerAdmin admin@example.com
```

Next, we need to add two new directives. The first, called `ServerName`, establishes the base domain for this virtual host definition. The second, called `ServerAlias`, defines further names that should match as if they were the base name. This is useful for matching additional hosts you defined, so both `example.com`and `www.example.com` both work, provided both of these hosts point to this server's IP address.

Add these two directives to your configuration file, right after the `ServerAdmin` line:

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

```
<VirtualHost *:80>

        ServerAdmin webmaster@localhost
        ServerName example.com
        ServerAlias www.example.com
        DocumentRoot /var/www/html
...
```

Next, change the location of the document root for this domain by altering the `DocumentRoot` directive to point to the directory you created for this host:

```
DocumentRoot /var/www/example.com/public_html
```

Once you've made these changes, your file should look like this:

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

```
<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.

Then create the second configuration file by creating a copy of this file:

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

Open the new file in your editor:

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

Then change the relevant settings to reference your second domain. When you are finished, your file will look like this:

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

```
<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.

Now that we have created our virtual host files, we can enable them.

Step 4 — Enabling the New Virtual Host Files

You've created the folders and the virtual host configuration files, but Apache won't use them until you activate them. You can use the `a2ensite` tool to enable each of your sites.

Activate the first site:

- `sudo a2ensite example.com.conf`
-

You'll see the following output if there were no syntax errors or typos in your file:

```
Output
Enabling site example.com.
To activate the new configuration, you need to run:
  service apache2 reload
```

In order for your changes to take effect, you have to reload Apache. But before you do, enable the other site:

- `sudo a2ensite test.com.conf`
-

You'll see a similar message indicating the site was enabled:

```
Output
Enabling site test.com.
To activate the new configuration, you need to run:
  service apache2 reload
```

Next, disable the default site defined in `000-default.conf` by using the `a2dissite` command:

- `sudo a2dissite 000-default.conf`
-

Now, restart Apache:

- `sudo systemctl restart apache2`
-

The sites are now configured. Let's test them out. If you're using real domains configured to point to your server's IP address, you can skip the next step. But if your domains haven't propogated yet, or if you're just testing, read on to learn how to test this setup using your local computer.

Step 5 — Setting 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 only useful for testing purposes.

Make sure you follow these steps on your local computer, and not your VPS server. You will also need to know the local computer's administrative password or 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
```

If you're on Windows, open a Command Prompt with administrative privileges and type:

```
notepad %windir%\system32\drivers\etc\hosts
```

Once you have the file open, add a line that maps your server's public IP address to each domain name, as shown in the following example:

/etc/hosts

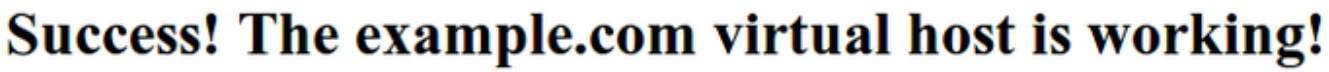
```
127.0.0.1   localhost
...

111.111.111.111 example.com
111.111.111.111 test.com
```

This will direct any requests for `example.com` and `test.com` on your computer and send them to your server at `111.111.111.111`.

Save and close the file. Now you can test out your setup. When you're confident things are working, remove the two lines from the file.

Step 6 — Testing 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. Visit the first site at `http://example.com` and you'll see a page that looks like this:



Likewise, if you can visit your second host at `http://test.com`, you'll see the file you created for your second site:

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

Note:If you adjusted your home computer's hosts file as shown in Step 5, 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

You now have a single server handling two separate domain names. You can expand this process by following these steps to add additional virtual hosts.