How To Set Up Apache Virtual Hosts on CentOS 7

PostedNovember 5, 2014 634k views Apache CentOS

Josh Barnett

Introduction

The Apache web server is the most popular way of serving web content on the Internet. It serves more than half of all of the Internet's active websites, and is extremely powerful and flexible.

Apache breaks down 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**. Virtual hosts allow one server to host multiple domains or interfaces by using a matching system. 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, without ever 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 traffic that all of the sites attract.

In this guide, we will walk through how to set up Apache virtual hosts on a CentOS 7 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 with this guide, there are a few steps that need to be completed first.

You will need access to a CentOS 7 server with a non-root user that has `sudo` privileges. If you haven't configured this yet, you can run through the CentOS 7 initial server setup guide to create this account.

You will also need to have Apache installed in order to configure virtual hosts for it. If you haven't already done so, you can use `yum` to install Apache through CentOS's default software repositories:

```
sudo yum -y install httpd
```

Next, enable Apache as a CentOS service so that it will automatically start after a reboot:

```
sudo systemctl enable httpd.service
```

After these steps are complete, log in as your non-root user account through SSH and continue with the tutorial.

Note: The example configuration in this guide will make one virtual host for `example.com` and another for `example2.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, follow this link.

If you do not have any real domains to play with, we will show you how to test your virtual host configuration with dummy values near the end of the tutorial.

Step One — Create the Directory Structure

First, we need to make a directory structure that will hold the site data to serve to visitors.

Our **document root** (the top-level directory that Apache looks at to find content to serve) will be set to individual directories in the `/var/www` directory. We will create a directory here for each of the virtual hosts that we plan on making.
Within each of these directories, we will create a `public_html` directory that will hold our actual files. This gives us some flexibility in our hosting.
We can make these directories using the `mkdir` command (with a `-p` flag that allows us to create a folder with a nested folder inside of it):

```
sudo mkdir -p /var/www/example.com/public_html
sudo mkdir -p /var/www/example2.com/public_html
```

Remember that the portions in red represent the domain names that we want to serve from our VPS.

Step Two — Grant Permissions

We now 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 with `chown`:

```
sudo chown -R $USER:$USER /var/www/example.com/public_html
sudo chown -R $USER:$USER /var/www/example2.com/public_html
```

The `$USER` variable will take the value of the user you are currently logged in as when you submit the command. 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 inside, 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 appropriate folders.

Step Three — Create Demo Pages for Each Virtual Host

Now that we have our directory structure in place, let's create some content to serve.
Because this is just for demonstration and testing, our pages will be very simple. We are just going to make an `index.html` page for each site that identifies that specific domain.
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 that the page is connected to. For this guide, the file for our first domain will look 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 template for our second site's `index.html` by typing:

```
cp /var/www/example.com/public_html/index.html
/var/www/example2.com/public_html/index.html
```
Now let's open that file and modify the relevant pieces of information:
```
nano /var/www/example2.com/public_html/index.html
```
```
<html>
  <head>
    <title>Welcome to Example2.com!</title>
  </head>
  <body>
    <h1>Success! The example2.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 what specify the configuration of our separate sites and dictate how the Apache web server will respond to various domain requests.

To begin, we will need to set up the directory that our virtual hosts will be stored in, as well as the directory that tells Apache that a virtual host is ready to serve to visitors. The `sites-available` directory will keep all of our virtual host files, while the `sites-enabled` directory will hold symbolic links to virtual hosts that we want to publish. We can make both directories by typing:
```
sudo mkdir /etc/httpd/sites-available
sudo mkdir /etc/httpd/sites-enabled
```
Note: This directory layout was introduced by Debian contributors, but we are including it here for added flexibility with managing our virtual hosts (as it's easier to temporarily enable and disable virtual hosts this way).

Next, we should tell Apache to look for virtual hosts in the `sites-enabled` directory. To accomplish this, we will edit Apache's main configuration file and add a line declaring an optional directory for additional configuration files:
```
sudo nano /etc/httpd/conf/httpd.conf
```
Add this line to the end of the file:
```
IncludeOptional sites-enabled/*.conf
```
Save and close the file when you are done adding that line. We are now ready to create our first virtual host file.

Create the First Virtual Host File

Start by opening the new file in your editor with root privileges:
```
sudo nano /etc/httpd/sites-available/example.com.conf
```
Note: Due to the configurations that we have outlined, all virtual host files *must* end in `.conf`.

First, start by making a pair of tags designating the content as a virtual host that is listening on port 80 (the default HTTP port):
```
<VirtualHost *:80>


</VirtualHost>
```

Next we'll declare the main server name, www.example.com. We'll also make a server alias to point to example.com, so that requests for www.example.com and example.com deliver the same content:

```
<VirtualHost *:80>
    ServerName www.example.com
    ServerAlias example.com
</VirtualHost>
```

Note: In order for the `www` version of the domain to work correctly, the domain's DNS configuration will need an A record or CNAME that points `www` requests to the server's IP. A wildcard (`*`) record will also work. To learn more about DNS records, check out our host name setup guide.

Finally, we'll finish up by pointing to the root directory of our publicly accessible web documents. We will also tell Apache where to store error and request logs for this particular site:

```
<VirtualHost *:80>

    ServerName www.example.com
    ServerAlias example.com
    DocumentRoot /var/www/example.com/public_html
    ErrorLog /var/www/example.com/error.log
    CustomLog /var/www/example.com/requests.log combined
</VirtualHost>
```

When you are finished writing out these items, you can save and close the file.

Copy First Virtual Host and Customize for Additional Domains

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 with `cp`:

```
sudo cp /etc/httpd/sites-available/example.com.conf /etc/httpd/sites-available/example2.com.conf
```

Open the new file with root privileges in your text editor:

```
sudo nano /etc/httpd/sites-available/example2.com.conf
```

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

```
<VirtualHost *:80>
    ServerName www.example2.com
    DocumentRoot /var/www/example2.com/public_html
    ServerAlias example2.com
    ErrorLog /var/www/example2.com/error.log
    CustomLog /var/www/example2.com/requests.log combined
</VirtualHost>
```

When you are finished making these changes, you can save and close the file.

Step Five — Enable the New Virtual Host Files

Now that we have created our virtual host files, we need to enable them so that Apache knows to serve them to visitors. To do this, we can create a symbolic link for each virtual host in the `sites-enabled` directory:

```
sudo ln -s /etc/httpd/sites-available/example.com.conf
/etc/httpd/sites-enabled/example.com.conf
sudo ln -s /etc/httpd/sites-available/example2.com.conf
/etc/httpd/sites-enabled/example2.com.conf
```

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

```
sudo apachectl restart
```

Step Six — Set Up Local Hosts File (Optional)

If you have been using example domains instead of actual domains to test this procedure, you can still test the functionality of your virtual hosts 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.

Note: Make sure that you are operating on your local computer for these steps and not your VPS server. You will need access to the administrative credentials for that computer.

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

```
sudo nano /etc/hosts
```

If you are on a Windows machine, you can find instructions on altering your hosts file here. The details that you need to add are the public IP address of your VPS followed by the domain that you want to use to reach that VPS:

```
127.0.0.1   localhost
127.0.1.1   guest-desktop
server_ip_address example.com
server_ip_address example2.com
```

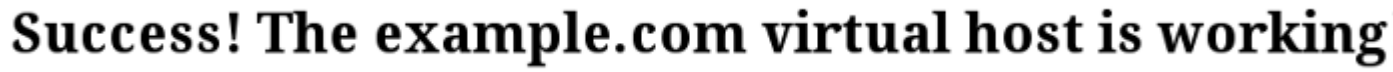
This will direct any requests for `example.com` and `example2.com` on our local computer and send them to our server at `server_ip_address`.

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:



Likewise, if you visit your other domains, you will see the files that you created for them.

If all of the sites that you configured work well, then you have successfully configured your new Apache virtual hosts on the same CentOS server.

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

Conclusion

At this point, you should now have a single CentOS 7 server handling multiple sites with separate domains. You can expand this process by following the steps we outlined above to make additional virtual hosts later. 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.