Apache - Server Multiple Site Setup

What you'll need

- A server running Rocky 9 Linux
- Know the vi text editor, here's a handy tutorial.
- Basic knowledge of installing and running web services
- For those looking for a similar setup for Nginx, examine this guide.

Rocky 9 Installation

- Minimal Installation .ISO

```
dnf -y upgrade
dnf install epel-release -y
dnf -y upgrade
```

Install required system packages

```
dnf install bind-utils bzip2 cups cifs-utils enscript ftp gdb ghostscript java-1.8.0-openjdk-headless java-11-openjdk-headless krb5-workstation ksh lftp lrzsz lsof libnsl lzop mariadb-server mlocate mutt ncompress net-tools net-snmp net-snmp-utils net-tools nfs-utils nmap nvme-cli openldap-clients openssh-clients psmisc realmd rsync samba-client strace sysstat tcpdump telnet telnet-server tmux unix2dos vim vim-enhanced vsftpd wget xfsdump vsftpd htop mc rsyslog rsyslog-doc postfix dbus-daemon s-nail git composer -y
```

Install MsSQL ODBC

```
curl https://packages.microsoft.com/config/rhel/8/prod.repo > /etc/yum.repos.d/msprod.repo
dnf remove mssql-tools unixODBC-utf16-devel
dnf install mssql-tools unixODBC-devel -y
```

Install Apache

You will likely need other packages for your website, such as PHP, database, or other packages. Installing PHP along with `http` will get you the most recent version from the Rocky Linux repositories.

Just remember that you may need modules, such as `php-bcmath` or `php-mysqlind`. Your web application specifications will dictate what you need. You can install these when needed. For now, you will install `http` and PHP, as those are almost a forgone conclusion:

From the command-line run:

```
dnf -y install httpd php mod_ssl openssh
```

Add the REMI php upgrades to Rocky

```
dnf install https://rpms.remirepo.net/enterprise/remi-release-9.rpm -y
dnf module reset php -y
```

Look to see which version you wish to install

```
dnf module list php -y
```

We will install the latest 8.0 if not php is installed to be able to upgrade and install php-sqlsrv

```
dnf module install php:remi-8.2 -y
```

Add more of the php tools

```
dnf -y install php-dbg php-pear php-ldap php-odbc php-pgsql php-bcmath php-gd php-pdo php-intl php-json php-enchant php-pecl-apcu php-mbstring php-devel php-snmp php-embedded php-pecl-zip php-fpm php-mysqlnd php-opcache php-dba php-process php-gmp php-common php-soap php-xml php-cli php-gd php-mysql
dnf -y install fail2ban
```

Installed the MySql to MsSQL connection

```
sudo yum install php-sqlsrv -y
```

Verify the installation afterwards

```
php -v
```

Add extra directories

This method uses a couple of additional directories, which do not currently exist on the system. You need to add two directories in */etc/httpd/* called "sites-available" and "sites-enabled."

From the command-line enter:

```
mkdir -p /etc/httpd/sites-available /etc/httpd/sites-enabled
```

This will create both needed directories.

You also need a directory where our sites are going to be. This can be anywhere, but a good way to keep things organized is to create a "sub-domains" directory. Put this in /var/www: `mkdir /var/www/sub-domains/` to decrease complexity.

MySQL Configuration

This to configure mariadb database

Create users, Run “mysql” and execute the fallowing statements: (**change passwords as needed**)

```
CREATE USER 'webftp'@'%' IDENTIFIED BY 'XXXXXXXX';
GRANT all ON *.* TO 'webftp'@'%' WITH GRANT OPTION;
CREATE USER 'webftp'@'localhost' IDENTIFIED BY 'XXXXXXXX';
GRANT all ON *.* TO 'webftp'@'localhost' WITH GRANT OPTION;
FLUSH PRIVILEGES;
Quit
```

Add below “[mysqld]” in “vi /etc/my.cnf.d/mariadb-server.cnf”

```
#Custom
performance_schema = ON
tmpdir = /run/mariadb
thread_cache_size = 4
table_open_cache = 16384
expire_logs_days = 2
table_definition_cache = 8384
sql_mode = ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
```

```
query_cache_type = 0
query_cache_size = 0
query_cache_limit = 128M
query_cache_strip_comments = 1

tmp_table_size = 512M
max_heap_table_size = 512M

max_connections = 512
max_allowed_packet = 24M
sort_buffer_size = 24M
join_buffer_size = 48M

innodb_buffer_pool_size = 4G
innodb_buffer_pool_instances = 4
innodb_use_native_aio = 1
innodb_flush_log_at_trx_commit = 0
innodb_file_per_table
innodb_log_file_size = 512M

log_bin = /var/log/mariadb/mariadb.log
expire_logs_days = 2
```

Run this to reload for changes

```
systemctl daemon-reload
```

Enable MySql to restart on boot

```
systemctl enable --now mariadb.service
```

Configuration

You also need to add a line to the bottom of the `httpd.conf` file. To do this, enter:

```
vi /etc/httpd/conf/httpd.conf
```

and go to the bottom of the file and add:

```
IncludeOptional sites-enabled/*.conf
```

Our actual configuration files will be in */etc/httpd/sites-available* and you will symlink to them in */etc/httpd/sites-enabled*.

Why do you do this?

Say you have 10 websites all running on the same server on different IP addresses. Say that site B has some major updates, and you have to make changes to the configuration for that site. Say also that something goes wrong with the changes made, and when you restart `httpd` to read in the changes, `httpd` does not start. Not only will the site you were working on not start, but neither will the rest of them. With this method, you can remove the symbolic link for the site that caused the problem, and restart `httpd`. It will start working again, and you fix the broken site's configuration.

It takes the pressure off, knowing the telephone will not ring with some upset customer or boss because a service is off-line.

The site configuration

The other benefit of this method is that it allows us to fully specify everything outside the default `httpd.conf` file. The default `httpd.conf` file loads the defaults, and your site configurations do everything else. Great, right? Plus again, it makes troubleshooting a broken site configuration less complex.

Say you have a website that loads a wiki. You will need a configuration file, which makes the site available on port 80.

If you want to serve the website with SSL/TLS (and face it, in most cases you do), you need to add another (nearly the same) section to that file to enable port 443.

You can examine that below in the Configuration `https` using An SSL/TLS certificate section.

You first need to create this configuration file in *sites-available*:

```
vi /etc/httpd/sites-available/com.wiki.www
```

The configuration file content will look something like this:

```
<VirtualHost *:80>
    ServerName your-server-hostname
    ServerAdmin username@rockylinux.org
    DocumentRoot /var/www/sub-domains/your-server-hostname/html
    DirectoryIndex index.php index.htm index.html
```

```
        Alias /icons/ /var/www/icons/
        # ScriptAlias /cgi-bin/ /var/www/sub-domains/your-server-hostname/cgi-bin/

    CustomLog "/var/log/httpd/your-server-hostname-access_log" combined
    ErrorLog  "/var/log/httpd/your-server-hostname-error_log"

        <Directory /var/www/sub-domains/your-server-hostname/html>
                Options -ExecCGI -Indexes
                AllowOverride None

                Order deny,allow
                Deny from all
                Allow from all

                Satisfy all
        </Directory>
</VirtualHost>
```

When created, you need to write (save) it with Shift+:+W+Q.

In the example, loading the wiki site happens from the "html" subdirectory of *your-server-hostname*, which means that the path you created in */var/www* (above) will need some additional directories to satisfy this:

```
mkdir -p /var/www/sub-domains/your-server-hostname/html
```

This will create the entire path with a single command. Next you want to install your files to this directory that will actually run the website. This might be something you made yourself, or an installable web application (in this case a wiki) that you downloaded.

Copy your files to the path you created:

```
cp -Rf wiki_source/* /var/www/sub-domains/your-server-hostname/html/
```

Configuration `https` using an SSL/TLS certificate

As stated earlier, every web server created these days *should* be running with SSL/TLS (the secure socket layer).

This process starts by generating a private key and CSR (certificate signing request) and submitting the CSR to the certificate authority to buy the SSL/TLS certificate. The process of generating these keys is somewhat extensive.

If you are not familiar with SSL/TLS key generation examine: Generating SSL Keys

You can also use this alternate process, using an SSL certificate from Let's Encrypt

Placement of the SSL/TLS keys and certificates¶

Since you have your keys and certificate files, you need to place them logically in your file system on the web server. As you have seen with the example configuration file, you are placing your web files in `/var/www/sub-domains/your-server-hostname/html`.

You want to place your certificate and key files with the domain, but outside of the document root, which in this case is the *html* folder.

You never want to risk exposing your certificates and keys to the web. That would be bad!

Instead, you will create a directory structure for our SSL/TLS files, outside the document root:

```
mkdir -p /var/www/sub-domains/your-server-hostname/ssl/{ssl.key,ssl.crt,ssl.csr}
```

If you are new to the "tree" syntax for making directories, what the above says is:

"Make a directory called "ssl" and make three directories inside called ssl.key, ssl.crt, and ssl.csr."

Just a note ahead of time: Storing the certificate signing request (CSR) file in the tree is not necessary, but it simplifies some things. If you ever need to re-issue the certificate from a different provider, having a stored copy of the CSR is a good idea. The question becomes where can you store it so that you will remember, and storing it within the tree of your website is logical.

Assuming that you have named your key, csr, and crt (certificate) files with the name of your site, and that you have them stored in */root*, you will copy them up to their locations:

```
cp /root/com.wiki.www.key /var/www/sub-domains/your-server-hostname/ssl/ssl.key/
cp /root/com.wiki.www.csr /var/www/sub-domains/your-server-hostname/ssl/ssl.csr/
cp /root/com.wiki.www.crt /var/www/sub-domains/your-server-hostname/ssl/ssl.crt/
```

The site configuration - `https`

Once you have generated your keys and purchased the SSL/TLS certificate, you can move forward with the website configuration using your keys.

For starters, break down the beginning of the configuration file. For instance, even though you still want to listen on port 80 (standard `http` port) for incoming requests, you do not want any of those requests to actually go to port 80.

You want them to go to port 443 (or "`http` secure", better known as SSL/TLS or `https`). Our port 80 configuration section will be minimal:

```
<VirtualHost *:80>
        ServerName your-server-hostname
        ServerAdmin username@rockylinux.org
        Redirect / https://your-server-hostname/
</VirtualHost>
```

What this says is to send any regular web request to the `https` configuration instead. The apache "Redirect" option shown is temporary. When testing is complete and you can see that the site is running as expected, you can change this to "Redirect permanent."

A permanent redirect will teach the search engines, and soon any traffic to your site that comes from search engines will go only to port 443 (`https`) without hitting port 80 (`http`) first.

Next, you need to define the `https` part of the configuration file:

```
<VirtualHost *:80>
        ServerName your-server-hostname
        ServerAdmin username@rockylinux.org
        Redirect / https://your-server-hostname/
</VirtualHost>
<Virtual Host *:443>
        ServerName your-server-hostname
        ServerAdmin username@rockylinux.org
        DocumentRoot /var/www/sub-domains/your-server-hostname/html
        DirectoryIndex index.php index.htm index.html
        Alias /icons/ /var/www/icons/
        # ScriptAlias /cgi-bin/ /var/www/sub-domains/your-server-hostname/cgi-bin/


    CustomLog "/var/log/`http`d/your-server-hostname-access_log" combined
    ErrorLog  "/var/log/`http`d/your-server-hostname-error_log"
```

```
        SSLEngine on
        SSLProtocol all -SSLv2 -SSLv3 -TLSv1
        SSLHonorCipherOrder on
        SSLCipherSuite
EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+a
RSA+SHA384
:EECDH+aRSA+SHA256:EECDH+aRSA+RC4:EECDH:EDH+aRSA:RC4:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PS
K:!SRP:!DSS

        SSLCertificateFile /var/www/sub-domains/your-server-hostname/ssl/ssl.crt/com.wiki.www.crt
        SSLCertificateKeyFile /var/www/sub-domains/your-server-hostname/ssl/ssl.key/com.wiki.www.key
        SSLCertificateChainFile /var/www/sub-domains/your-server-
hostname/ssl/ssl.crt/your_providers_intermediate_certificate.crt

        <Directory /var/www/sub-domains/your-server-hostname/html>
                Options -ExecCGI -Indexes
                AllowOverride None

                Order deny,allow
                Deny from all
                Allow from all

                Satisfy all
        </Directory>
</VirtualHost>
```

So, breaking down this configuration further, after the normal portions of the configuration and down to the SSL/TLS section:

- SSLEngine on - says to use SSL/TLS
- SSLProtocol all -SSLv2 -SSLv3 -TLSv1 - says to use all available protocols, except those with vulnerabilities. You should research periodically the protocols currently acceptable for use.
- SSLHonorCipherOrder on - this deals with the next line regarding the cipher suites, and says to deal with them in the order shown. This is another area where reviewing the cipher suites should occur periodically.
- SSLCertificateFile - is exactly what it says: the newly purchased and applied certificate file and its location
- SSLCertificateKeyFile - the key you generated when creating your certificate signing request

- SSLCertificateChainFile - the certificate from your certificate provider, often called the intermediate certificate

Take everything live and if no errors exist when starting the web service, and if going to your website reveals `https` without errors, you are ready to go.

Taking it live

Remember that our *httpd.conf* file is including */etc/httpd/sites-enabled* at the end of the file. When `httpd` restarts, it will load whatever configuration files are in that *sites-enabled* directory. Thing is, all of our configuration files are in *sites-available*.

That is by design, so that you can remove things when or if `httpd` fails to restart. To enable our configuration file, you need to create a symbolic link to that file in *sites-enabled* and start or restart the web service. To do this, you use this command:

```
ln -s /etc/httpd/sites-available/your-server-hostname /etc/httpd/sites-enabled/
```

This will create the link to the configuration file in *sites-enabled*.

Now just start `httpd` with `systemctl start httpd`. Or restart it if it is already running: `systemctl restart httpd`, and assuming the web service restarts, you can now go and do some testing on your site.

Migrating from old server to new another

Copy the data

```
rsync -chavzP --stats -e ssh root@sfl-web-004:/opt/ /opt/
rsync -chavzP --stats -e ssh root@sfl-web-004:/var/www/html/ /var/www/html/
rsync -chavzP --stats -e ssh root@sfl-web-004:/etc/httpd/sites-available/ /etc/httpd/sites-available/
rsync -chavzP --stats -e ssh root@sfl-web-004:/etc/httpd/sites-enabled/ /etc/httpd/sites-enabled/
rsync -chavzP --stats -e ssh root@sfl-web-004:/etc/pki/tls/private/ /etc/pki/tls/private/
rsync -chavzP --stats -e ssh root@sfl-web-004:/etc/pki/tls/certs/ /etc/pki/tls/certs/
```

Restore the MySQL databases with a specific date your want to use.

```
cd /opt/backups/byhour
gunzip 2024-05-21_21.*
2024-05-21_21.hdrwp01.sql
2024-05-21_21.llc.sql
2024-05-21_21.mcwp.sql
2024-05-21_21.mfbforum.sql
2024-05-21_21.mfbwp.sql
2024-05-21_21.ogforum.sql
2024-05-21_21.ogwp.sql
2024-05-21_21.sign.sql
mysql -f -e "CREATE DATABASE hdrwp01;"
mysql -f -e "CREATE DATABASE llc;"
mysql -f -e "CREATE DATABASE mcwp;"
mysql -f -e "CREATE DATABASE mfbforum;"
mysql -f -e "CREATE DATABASE mfbwp;"
mysql -f -e "CREATE DATABASE ogforum;"
mysql -f -e "CREATE DATABASE ogwp;"
mysql -f hdrwp01 < 2024-05-21_21.hdrwp01.sql
mysql -f llc < 2024-05-21_21.llc.sql;
mysql -f mcwp < 2024-05-21_21.mcwp.sql
mysql -f mfbforum < 2024-05-21_21.mfbforum.sql
mysql -f mfbwp < 2024-05-21_21.mfbwp.sql
mysql -f ogforum < 2024-05-21_21.ogforum.sql
mysql -f ogwp < 2024-05-21_21.ogwp.sql
mysql -f sign < 2024-05-21_21.sign.sql
```

SAMBA Configuration

This is to setup samba on the pc to transfer file more easily.

```
dnf -y samba samba-common samba-client
```

Backup the original setup file

```
mv /etc/samba/smb.conf /etc/samba/smb.conf.bak
```

Edit the config file with vi or nano

```
vi /etc/samba/smb.conf
```

Change and add the following, leave the rest as is

```
[global]
        workgroup = ONLING
        security = user
        server string = Samba Server %v
        netbios name = sfl-web-004
        map to guest = bad user
        dns proxy = no
        ntlm auth = true

[web]
        comment = apache
        path =  /var/www/html
        browsable =yes
        writable = yes
        guest ok = yes
        read only = no
        force user = apache
        force group = apache
```

Revision #1
Created 2 September 2024 22:29:44 by Steve Ling
Updated 2 September 2024 22:36:05 by Steve Ling