

Apache

Copyright Notice

SFL Services LLC has prepared this document for use only by their staff, agents, customers and prospective customers. Companies, names and data used as examples in this document are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of SFL Services LLC, who reserve the right to change specifications and other information contained herein without prior notice. The reader should consult SFL Services LLC to determine whether any such changes have been made.

Licensing and Warranty

The terms and conditions governing the licensing of SFL Services LLC software consist solely of those set forth in the written contracts between SFL Services LLC and its customers. Except as expressly provided for in the warranty provisions of those written contracts, no representation or other affirmation of fact contained in this document, including but not limited to statements regarding capacity, suitability for use or performance of products described herein, shall be deemed to be a warranty by SFL Services LLC for any purpose, or give rise to any liability of SFL Services LLC whatsoever.

Liability

In no event shall SFL Services LLC be liable for any incidental, indirect, special or consequential damages whatsoever (including but not limited to lost profits) arising out of or related to this document or the information contained in it, even if SFL Services LLC had been advised, knew or should have known of the possibility of such damages, and even if they had acted negligently.

- [Apache - Health Check](#)
- [Apache - Enabling HTTPS w/Certificate](#)
- [Apache - Host Visual Studio .NET 8 Deployments](#)
- [Apache - Self Signed HTTPS Certificates](#)
- [Apache - Setting up an SSL secured Webserver w/Rocky Linux](#)
- [Apache - How to create a multi-domain SSL certificate](#)
- [Apache - Certbot SSL Certificate](#)
- [Apache - Server Multiple Site Setup](#)

Apache - Health Check

Server Health Commands

Check TLS

```
openssl s_client -connect <IP_ADDRESS>:8443 -tls1  
nmap -sV --script ssl-enum-ciphers -p 8443 <IP_ADDRESS>
```

Connections onto Server

```
netstat -anp | grep 'tcp|udp' | awk '{print $5}' | cut -d: -f1 | sort | uniq -c | sort -n
```

```
netstat -n | grep :80 | wc -l
```

```
netstat -n | grep :80 | grep SYN | wc -l
```

Benchmarking

```
ab -k -n500 -c100 -H "Accept-Encoding: gzip,deflate" http://hammerdownrange.com/
```

Apache - Enabling HTTPS w/Certificate

1. Getting the required software

For an SSL encrypted web server you will need a few things. Depending on your install you may or may not have OpenSSL and mod_ssl, Apache's interface to OpenSSL. Use yum to get them if you need them.

```
yum install mod_ssl openssl
```

Yum will either tell you they are installed or will install them for you.

2. Generate a self-signed certificate

Using OpenSSL we will generate a self-signed certificate. If you are using this on a production server you are probably likely to want a key from a Trusted Certificate Authority, but if you are just using this on a personal site or for testing purposes a self-signed certificate is fine. To create the key you will need to be root so you can either su to root or use sudo in front of the commands

```
# Generate private key
openssl genrsa -out ca.key 2048

# Generate CSR
openssl req -new -key ca.key -out ca.csr

# Generate Self Signed Key
openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
```

```
# Copy the files to the correct locations
cp ca.crt /etc/pki/tls/certs
cp ca.key /etc/pki/tls/private/ca.key
cp ca.csr /etc/pki/tls/private/ca.csr
```

WARNING: Make sure that you **copy** the files and do not **move** them if you use SELinux. Apache will complain about missing certificate files otherwise, as it cannot read them because the certificate files do not have the right SELinux context.

If you have moved the files and not copied them, you can use the following command to correct the SELinux contexts on those files, as the correct context definitions for /etc/pki/* come with the bundled SELinux policy.

```
restorecon -RvF /etc/pki
```

Then we need to update the Apache SSL configuration file

```
vi +/SSLCertificateFile /etc/httpd/conf.d/ssl.conf
```

Change the paths to match where the Key file is stored. If you've used the method above it will be

```
SSLCertificateFile /etc/pki/tls/certs/ca.crt
```

Then set the correct path for the Certificate Key File a few lines below. If you've followed the instructions above it is:

```
SSLCertificateKeyFile /etc/pki/tls/private/ca.key
```

Quit and save the file and then restart Apache

```
/etc/init.d/httpd restart
```

All being well you should now be able to connect over https to your server and see a default Centos page. As the certificate is self signed browsers will generally ask you whether you want to accept the certificate.

3. Setting up the virtual hosts

Just as you set [VirtualHosts](#) for http on port 80 so you do for https on port 443. A typical [VirtualHost](#) for a site on port 80 looks like this

```
<VirtualHost *:80>
    <Directory /var/www/vhosts/yoursite.com/httpdocs>
        AllowOverride All
    </Directory>
    DocumentRoot /var/www/vhosts/yoursite.com/httpdocs
    ServerName yoursite.com
</VirtualHost>
```

To add a sister site on port 443 you need to add the following at the top of your file

```
NameVirtualHost *:443
```

and then a [VirtualHost](#) record something like this:

```
<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/ca.crt
    SSLCertificateKeyFile /etc/pki/tls/private/ca.key
    <Directory /var/www/vhosts/yoursite.com/httpsdocs>
        AllowOverride All
    </Directory>
    DocumentRoot /var/www/vhosts/yoursite.com/httpsdocs
    ServerName yoursite.com
</VirtualHost>
```

Restart Apache again using

```
/etc/init.d/httpd restart
```

4. Configuring the firewall

You should now have a site working over https using a self-signed certificate. If you can't connect you may need to open the port on your firewall. To do this amend your iptables rules:

```
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
/sbin/service iptables save
iptables -L -v
```


Apache - Host Visual Studio .NET 8 Deployments

Introduction

This is to prep the Linux server to host Visual Studio code.

This is for either RedHat or Rocky Linux installations.

For this configuration we are installing on a Rocky Linux server.

Preparation

Make sure that the server is up to date

```
dnf update -y
```

Configuration

Install the ASP .NET Core Runtime

```
dnf install aspnetcore-runtime-8.0 -y
```

Verification

```
dotnet --info
```

This should bring back the installation results

```
Host:
  Version:      8.0.8
  Architecture: x64
  Commit:       08338fcaa5
  RID:          rocky.9-x64
.NET SDKs installed:
  No SDKs were found.
.NET runtimes installed:
  Microsoft.AspNetCore.App 8.0.8 [/usr/lib64/dotnet/shared/Microsoft.AspNetCore.App]
  Microsoft.NETCore.App 8.0.8 [/usr/lib64/dotnet/shared/Microsoft.NETCore.App]
Other architectures found:
  None
Environment variables:
  DOTNET_ROOT      [/usr/lib64/dotnet]
global.json file:
  Not found
Learn more:
  https://aka.ms/dotnet/info
Download .NET:
  https://aka.ms/dotnet/download
```

Setup the Project

You will need to SSH to the Linux server and navigate to the deployment folder.

In our case the project is an API project that is called MyFFLBookAPI

Once in the folder you can start the project this way

```
cd /var/www/html/api
dotnet MyFFLBookAPI.dll
```

```
[/var/www/html/api]# dotnet MyFFLBookAPI.dll
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key {3cacc8f4-baca-4475-9498-610b5e187653} may be persisted to storage in
unencrypted form.
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /var/www/html/api
```

As you can see the server started on localhost port 5000. If you wish you can change the port to another if needed.

Simply use the Ctrl+C key to shutdown the project

Change Port

You will need to add an entry for Kestrel to redirect the port after the allowed hosts entry

```
"AllowedHosts": "*",
  "Kestrel": {
    "Endpoints": {
      "Http": {
        "Url": "http://*:8081"
      }
    }
  }
}
```

Now to test restart the project

```
dotnet MyFFLBookAPI.dll
```

```
[/var/www/html/api]# dotnet MyFFLBookAPI.dll
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:8081
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /var/www/html/api
```

Now you can see that is started on the new port 8081

Setup a Service

First you will have to create a unit file for the service by doing the following

```
vi /lib/systemd/system/myfflbookapi.service
```

then for the entries within the unit service file

```
[Unit]
Description=MyFFLBookAPI

[Service]
WorkingDirectory=/var/www/html/api
ExecStart=/usr/bin/dotnet /var/www/html/api/MyFFLBookAPI.dll
Restart=always
```

```
RestartSec=10
KillSignal=SIGINT
SyslogIdentifier=myfflbookapi
User=root
Environment=ASPNETCORE_ENVIRONMENT=Production
```

[Install]

```
WantedBy=multi-user.target
```

Once configured you can test the service

To start the service do the following

```
systemctl start myfflbookapi
```

Then verify the status of the service

```
systemctl status myfflbookapi
```

Results of the above command

```
* myfflbookapi.service - MyFFLBookAPI
   Loaded: loaded (/usr/lib/systemd/system/myfflbookapi.service; disabled; preset: disabled)
   Active: active (running) since Sat 2024-08-31 10:14:38 EDT; 6s ago
     Main PID: 59786 (dotnet)
        Tasks: 28 (limit: 408004)
       Memory: 40.5M
          CPU: 515ms
      CGroup: /system.slice/myfflbookapi.service
              └─59786 /usr/bin/dotnet /var/www/html/api/MyFFLBookAPI.dll
Aug 31 10:14:38 sfl-web-001.onling.com systemd[1]: Started MyFFLBookAPI.
Aug 31 10:14:38 sfl-web-001.onling.com myfflbookapi[59786]: info: Microsoft.Hosting.Lifetime[14]
Aug 31 10:14:38 sfl-web-001.onling.com myfflbookapi[59786]:   Now listening on: http://[::]:8081
Aug 31 10:14:38 sfl-web-001.onling.com myfflbookapi[59786]: info: Microsoft.Hosting.Lifetime[0]
Aug 31 10:14:38 sfl-web-001.onling.com myfflbookapi[59786]:   Application started. Press Ctrl+C to shut down.
Aug 31 10:14:38 sfl-web-001.onling.com myfflbookapi[59786]: info: Microsoft.Hosting.Lifetime[0]
Aug 31 10:14:38 sfl-web-001.onling.com myfflbookapi[59786]:   Hosting environment: Production
Aug 31 10:14:38 sfl-web-001.onling.com myfflbookapi[59786]: info: Microsoft.Hosting.Lifetime[0]
Aug 31 10:14:38 sfl-web-001.onling.com myfflbookapi[59786]:   Content root path: /var/www/html/api
```

As you can see the service is running.

Now you can set the service to auto start on boot

```
systemctl enable myfflbookapi
```

It will auto create a link for boot

```
Created symlink /etc/systemd/system/multi-user.target.wants/myfflbookapi.service ->
/usr/lib/systemd/system/myfflbookapi.service.
```

Redirecting Ports

in the "/etc/httpd/sites-available" folder you will need to create a file for the website, in our case we will be using "sflservicesllc.io.conf"

```
cd /etc/httpd/sites-available
```

Then create the following file

```
vi sflservicesllc.io.conf
```

Add the following in the file

```
<VirtualHost *:80>

    ServerName sflservicesllc.io
    ServerAlias www.sflservicesllc.io
    ProxyPreserveHost On
    ProxyPass / https://localhost:8081/
    ProxyPassReverse / https://localhost:8081/

    ErrorLog logs/API_error_log
    TransferLog logs/API_access_log
    RewriteEngine on
    RewriteCond %{SERVER_NAME} =sflservicesllc.io
    RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]

</VirtualHost>

<VirtualHost *:443>

    SSLProxyEngine On
    SSLProxyVerify none
    SSLProxyCheckPeerCN off
    SSLProxyCheckPeerName off
    SSLProxyCheckPeerExpire off
    #SSLProxyCACertificateFile

    ServerName sflservicesllc.io
```

```
ServerAlias www.sflservicesllc.io
ProxyPreserveHost On
ProxyPass / https://localhost:8081/
ProxyPassReverse / https://localhost:8081/
```

```
ErrorLog logs/ssl_API_error_log
TransferLog logs/ssl_API_access_log
```

```
</VirtualHost>
```

Now create a link to allow Apache to start the website

```
cd /etc/httpd/sites-enabled
ln -s /etc/httpd/sites-available/sflservicesllc.io.conf
```

Then restart the Apache web service

```
systemctl restart httpd
```

Other Materials

You can also find on the Microsoft web site how to install on different versions other than the ones mentioned here

[Install the .NET SDK or the .NET Runtime on RHEL and CentOS Stream](#)

Apache - Self Signed HTTPS Certificates

If you are starting to migrate your web servers over to Linux (or have already done so) and are looking to serve those pages up over secure http (aka https), you're going to need to know how to make this happen. Although https does will not guarantee security for your web server, it is a solid first step in the process. Configuring Apache for https on CentOS isn't difficult, but there are a few steps. Let's walk through the process, so you can start serving your pages up to your clients/customers more confidently.

This walkthrough will use CentOS/Rocky and work with a self-signed certificate. The self-signed option works great for testing purposes. For your official business websites, you'll want to purchase an SSL certificate from us (contact sales and we'll get you going). we'll also assume you already have Apache running on the server.

Installing and using OpenSSL

The first step in the process is the installation of OpenSSL and the generating of the certificate to be used. To install OpenSSL, open a terminal window and issue the command:

```
sudo yum install mod_ssl openssl -y
```

Issuing the above command will pick up all the necessary dependencies; installing OpenSSL on CentOS can be done with a single command.

Now we generate the SSL key with the following commands:

Generate private key

```
sudo openssl genrsa -out ca.key 2048
```

Generate CSR

```
sudo openssl req -new -key ca.key -out ca.csr
```

Generate Self Signed Key

```
sudo openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
```

Now we need to copy the newly generated files to the correct locations with the following commands:

```
sudo cp ca.crt /etc/pki/tls/certs  
sudo cp ca.key /etc/pki/tls/private/ca.key  
sudo cp ca.csr /etc/pki/tls/private/ca.csr
```

When you issue the command to generate the CSR, you will be asked a number of questions for the key (such as Country Name, State or Province, Locality, Organization Name, Organizational Unit, Common Name, Email Address, etc.). OpenSSL will also require you to enter a challenge password for the CSR.

The next step requires the editing of the `/etc/httpd/conf.d/ssl.conf` file. Open that file for editing and locate and change the following lines:

```
SSLCertificateFile /etc/pki/tls/certs/localhost.crt
```

changes to:

```
SSLCertificateFile /etc/pki/tls/certs/ca.crt
```

```
SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
```

changes to:

```
SSLCertificateKeyFile /etc/pki/tls/private/ca.key
```

Finally, restart the Apache daemon with the command:

```
sudo systemctl restart httpd
```

Create a virtual host

Let's create a virtual host that makes use of SSL. To do this we'll create the necessary directories with the following commands:

```
sudo mkdir -p /var/www/html/web
sudo mkdir -p /etc/httpd/sites-available
sudo mkdir -p /etc/httpd/sites-enabled
```

I'm using "web" as an example. You can use whatever name you like/need.

Next we must edit the `httpd.conf` file, so that it becomes aware of the sites-enabled directory. To do this, open up `/etc/httpd/conf/httpd.conf` and add the following line to the bottom of the file:

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

Save and close that file.

Now we need to create our virtual host file. We'll do this in `/etc/httpd/sites-available/web.conf`. Again, swap "web.conf" with the name of your virtual host. In that file we'll add the following contents (customize as needed):

```
<VirtualHost *:443>
    [ServerAdmin email@address]
    [DocumentRoot "/var/www/html/web/"]
    [ServerName website]
    [ServerAlias web]
    [ErrorLog /var/www/html/web/error.log]

    <Directory "/var/www/html/web/">
        [DirectoryIndex index.html index.php]
        [Options FollowSymLinks]
        [AllowOverride All]
        [Require all granted]
    </Directory>
</VirtualHost>
```

Save and close that file.

In order for Apache to be aware of the new virtual host, we must create a symbolic link, from sites-available to sites-enabled, with the command:

```
sudo ln -s /etc/httpd/sites-available/web.conf /etc/httpd/sites-enabled/web.conf
```

Restart Apache with the command:

```
sudo systemctl restart httpd
```

Your virtual host should now be visible to the server. All you have to do is add content to the */var/www/html/web* directory and you're good to go.

A quick test

That's all there is to the setup of https on Apache with CentOS. You can do a quick test by pointing a browser to `https://IP_OF_SERVER`. You should receive a security warning (since we are using a self-signed certificate). Okay that warning and Apache will serve up your site using https. Point your browser to `https://IP_OF_SERVER`. to visit the newly created virtual host. Depending on what type of site you are serving up, you might have to do a bit of extra work with that particular platform.

Apache - Setting up an SSL secured Webserver w/Rocky Linux

This guide will explain how to set up a site over https. The tutorial uses a self signed key so will work well for a personal website or testing purposes. This is provided as is so proceed at your own risk and take backups!

1. Getting the required software

For an SSL encrypted web server you will need a few things. Depending on your install you may or may not have OpenSSL and mod_ssl, Apache's interface to OpenSSL. Use yum to get them if you need them.

```
yum install mod_ssl openssl
```

Yum will either tell you they are installed or will install them for you.

2. Generate a self-signed certificate

Using OpenSSL we will generate a self-signed certificate. If you are using this on a production server you are probably likely to want a key from a Trusted Certificate Authority, but if you are just using this on a personal site or for testing purposes a self-signed certificate is fine. To create the key you will need to be root so you can either su to root or use sudo in front of the commands

```
# Generate private key
openssl genrsa -out ca.key 2048

# Generate CSR
openssl req -new -key ca.key -out ca.csr

# Generate Self Signed Key
openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt

# Copy the files to the correct locations
cp ca.crt /etc/pki/tls/certs
cp ca.key /etc/pki/tls/private/ca.key
cp ca.csr /etc/pki/tls/private/ca.csr
```

WARNING: Make sure that you **copy** the files and do not **move** them if you use SELinux. Apache will complain about missing certificate files otherwise, as it cannot read them because the certificate files do not have the right SELinux context.

If you have moved the files and not copied them, you can use the following command to correct the SELinux contexts on those files, as the correct context definitions for `/etc/pki/*` come with the bundled SELinux policy.

```
restorecon -RvF /etc/pki
```

Then we need to update the Apache SSL configuration file

```
vi +/SSLCertificateFile /etc/httpd/conf.d/ssl.conf
```

Change the paths to match where the Key file is stored. If you've used the method above it will be

```
SSLCertificateFile /etc/pki/tls/certs/ca.crt
```

Then set the correct path for the Certificate Key File a few lines below. If you've followed the instructions above it is:

```
SSLCertificateKeyFile /etc/pki/tls/private/ca.key
```

Quit and save the file and then restart Apache

```
/etc/init.d/httpd restart
```

All being well you should now be able to connect over https to your server and see a default Centos page. As the certificate is self signed browsers will generally ask you whether you want to accept the certificate.

3. Setting up the virtual hosts

Just as you set [VirtualHosts](#) for http on port 80 so you do for https on port 443. A typical [VirtualHost](#) for a site on port 80 looks like this

```
<VirtualHost *:80>
  <Directory /var/www/vhosts/yoursite.com/httpdocs>
    AllowOverride All
  </Directory>
  DocumentRoot /var/www/vhosts/yoursite.com/httpdocs
  ServerName yoursite.com
</VirtualHost>
```

To add a sister site on port 443 you need to add the following at the top of your file

```
NameVirtualHost *:443
```

and then a [VirtualHost](#) record something like this:

```
<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/ca.crt
    SSLCertificateKeyFile /etc/pki/tls/private/ca.key
    <Directory /var/www/vhosts/yoursite.com/httpsdocs>
        AllowOverride All
    </Directory>
    DocumentRoot /var/www/vhosts/yoursite.com/httpsdocs
    ServerName yoursite.com
</VirtualHost>
```

Restart Apache again using

```
/etc/init.d/httpd restart
```

4. Configuring the firewall

You should now have a site working over https using a self-signed certificate. If you can't connect you may need to open the port on your firewall. To do this amend your iptables rules:

```
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
/sbin/service iptables save
iptables -L -v
```

Apache - How to create a multi-domain SSL certificate

Introduction

By default, an SSL certificate is valid for only a single domain name.

Using wildcards, you can match all the subdomains with a single certificate. However, a wildcard certificate is valid only for the subdomains, and not for the main domain: so, a certificate for `*.example.com` is valid for <http://www.example.com> , foo.example.com and bar.example.com, but not for <http://example.com> .

If you need to match both the main domain and subdomains, or even different domains (i.e. <http://example.com> and <http://example.net>), you need a **multi-domain SSL certificate**.

How to create the multi-domain SSL certificate

To create the multi-domain SSL certificate you need the **openssl** libraries and application on your PC.

Basically, the commands to create a **multi-domain SSL certificate** are almost the same to create a [single-domain certificate](#).

In this case it's required to generate a **Certificate Signing Request** (CSR) using a customized version of the OpenSSL configuration file, including in it the list of domain names (SubjectAltName) and, optionally, IP addresses.

Customize the *openssl.conf* file

Make a copy of the *openssl.conf* file (usually located in */etc/ssl/openssl.cnf*) into the working directory. You can name this file *openssl_copy.cnf*, for example.

```
cp /etc/ssl/openssl.cnf /etc/ssl/openssl_copy.cnf
```

Then open the new file with a text editor

```
vi /etc/ssl/openssl_copy.cnf
```

Search for the *[req]* section

```
/ [ req ]
```

Uncomment the *req_extensions* line removing the hash (#) on the first column:

```
req_extensions = v3_req # The extensions to add to a certificate request
```

Then search for the *[v3_req]* section

```
/ [ v3_req ]
```

add the *subjectAltName* parameter

```
subjectAltName = @alt_names
```

Finally, add at the end of the file a new section *[alt_names]* that contains all the domain names and/or IP addresses you want to include in the SSL certificate:

```
[ alt_names ]
DNS.1 = example.com
DNS.2 = www.example.com
DNS.3 = *.third.example.com
DNS.4 = example.net
DNS.5 = *.example.net
IP.1 = 1.2.3.4
IP.2 = 5.6.7.8
```

In this example, the SSL certificate will be valid for <http://example.com> , <http://www.example.com> , all the subdomains of third.example.com (but not for third.example.com itself), and <http://example.net> including *all* its subdomains (only the third-levels).

The certificate will be valid also for IP addresses *1.2.3.4* and *5.6.7.8*: it could be useful if the server is accessible directly via the IP address, instead of using a domain name.

Create the Certificate Signing Request (CSR)

Now you can create the key and the CSR file:

```
openssl req -newkey rsa:2048 -nodes -keyout ca.key -out ca.csr -config /etc/ssl/openssl_copy.cnf
```

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:OH
Locality Name (eg, city) [Default City]:LOVELAND
Organization Name (eg, company) [Default Company Ltd]:SFL SERVICES LLC
Organizational Unit Name (eg, section) []:IT DEPT
Common Name (eg, your name or your server's hostname) []:docs.sflservicesllc.com
Email Address []:sales@sflservicesllc.com
```

Please enter the following 'extra' attributes to be sent with your certificate request

```
A challenge password []:
An optional company name []:
```

This will create a 2048-bits key: if you need longer keys, change *rsa:2048* with the value you prefer.

You can verify the CSR file content to be sure the multiple domain names have been included:

```
openssl req -text -noout -in ca.csr
```

Finally, you can create the self-signed multi-domain SSL certificate:

```
openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt -extensions v3_req -extfile
/etc/ssl/openssl_copy.cnf
```

Copy the certificates to the default location

```
cp ca.crt /etc/pki/tls/certs
cp ca.key /etc/pki/tls/private/ca.key
cp ca.csr /etc/pki/tls/private/ca.csr
```

Apache configuration

The last step is the virtual host configuration on **Apache**:

```
<VirtualHost 1.2.3.4:443>
  ServerName www.example.com
  DocumentRoot /www
  ErrorLog logs/www.example.com-error.log
  CustomLog logs/www.example.com-access.log
  combined SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/ca.crt
  SSLCertificateKeyFile /etc/pki/tls/private/ca.key
</VirtualHost>
```

You can then restart Apache to make the changes effective.

From <https://www.wizlab.it/code/apache-create-multi-domain-ssl-certificate.html>

Nice to have <https://serverfault.com/questions/648534/accidentally-removed-localhost-crt-ssl-in-centos-6-what-can-i-do>

Apache - Certbot SSL Certificate

Introduction

The Certbot Package is not included in Rocky Linux's base repository by default. In order to obtain it, we must install the EPEL (Extra Packages for Enterprise Linux) repository. This repository provides additional software packages through open-source efforts. In addition to certbot, we must also install "mod_ssl," which is a security module for Apache to support SSL/TLS protocols.

Prerequisites

You can now install the **Certbot package** and its **dependencies** for Rocky Linux with the following commands

```
sudo dnf install epel-release -y
sudo dnf install mod_ssl -y
sudo dnf install certbot python3-certbot-apache -y
```

Install SSL Certificate for Apache httpd

Upon completion of the installation, you will be able to get a Let's Encrypt SSL certificate. Certbot offers various methods for obtaining an SSL Certificate, you may use one of the following commands.

Simple method to complete all sites that are configured

```
sudo certbot --apache
```

Alternately, you can use the "-d" flag with this command directly to specify multiple domains

```
certbot --apache -d http://website.com
```

When you run the above command, you will be prompted for a series of questions which you must answer in order to deploy the certificate successfully. In order to make things easier for beginners, I have separated each prompt into different boxes.

In order to verify the certificate, Let's Encrypt it will ask you to enter your email address

Also you will need to accept the following terms and conditions

After your first certificate is issued, you will be asked to share your email address to receive updates on new/campaigns with the Electronic Frontier Foundation. The decision is yours to make "Y or N"

Depending on your web server configuration, it will list your domains and ask which one you want to activate HTTPS for. You can select '1' or '2'. However, if you want all domains to begin using HTTPS, press ENTER

Certificate Automatic Renewal

Let's Encrypt certificates are generally valid for 90 days, so you need to renew them manually after that time. The following command needs to be run to renew the certificate.

```
sudo certbot renew --dry-run
```

However, we can automate the renewal process using cron jobs. In your crontab file, add the following entry:

```
0 0 1 * * /usr/bin/certbot renew >/dev/null 2>&1
```

Delete Certificate

If you wish to delete the certificate, use the following command

```
sudo certbot delete
```

Conclusion

We hope this article has helped you understand how to Secure Apache with SSL in Rocky Linux 9.1 step by step. You can also get help from Let's Encrypt's [community](#) site if you encounter any

issues. Drop me your feedback/comments. Feel free to share this article with others if you like it.

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 httpd 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-mysqlnd`. 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 openssl
```

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-debug 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 following 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/httpd/your-server-hostname-access_log" combined
    ErrorLog "/var/log/httpd/your-server-hostname-error_log"
```

SSL Engine on

SSLProtocol all -SSLv2 -SSLv3 -TLSv1

SSLHonorCipherOrder on

SSLCipherSuite

EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+aRSA+SHA384
:EECDH+aRSA+SHA256:EECDH+aRSA+RC4:EECDH:EDH+aRSA:RC4:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!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:

- SSL Engine 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
```