**How To Secure Nginx with Let's Encrypt on Ubuntu 22.04**

### Introduction

Let’s Encrypt is a Certificate Authority (CA) that provides an accessible way to obtain and install free [TLS/SSL certificates](https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs), thereby enabling encrypted HTTPS on web servers. It simplifies the process by providing a software client, Certbot, that attempts to automate most (if not all) of the required steps. Currently, the entire process of obtaining and installing a certificate is fully automated on both Apache and Nginx.

We will use Certbot to obtain a free SSL certificate for Nginx on Ubuntu 22.04 and set up your certificate to renew automatically.

We will use a separate Nginx server configuration file instead of the default. Create a new Nginx server block files for each domain because it helps to avoid common mistakes and maintains the default files as a fallback configuration.

## Prerequisites

1. One Ubuntu 22.04 server including a sudo enabled non-**root** user and a firewall.
2. A registered domain. It can be created using domain creator www.hostinger.com
3. Create A record for pointing that IP address to the domain.
4. Install Nginx
   * 1. sudo apt-get update
     2. sudo apt install nginx -y
     3. sudo service start nginx

**Step 1 — Installing Certbot**

Certbot recommends using their snap package for installation. Snap packages work on nearly all Linux distributions, but they require that you’ve installed snap first in order to manage snap packages. Ubuntu 22.04 comes with support for snaps out of the box, so you can start by making sure your snap core is up to date:

1. sudo snap install core; sudo snap refresh core

If you’re working on a server that previously had an older version of certbot installed, you should remove it before going any further:

1. sudo apt remove certbot

After that, you can install the certbot package:

1. sudo snap install --classic certbot

Finally, you can link the certbot command from the snap install directory to your path, so you’ll be able to run it by just typing certbot. This isn’t necessary with all packages, but snaps tend to be less intrusive by default, so they don’t conflict with any other system packages by accident:

1. sudo ln -s /snap/bin/certbot /usr/bin/certbot

Now that we have Certbot installed, let’s run it to get our certificate.

## **Step 2– Setting Up Server Blocks**

When using the Nginx web server, server blocks (similar to virtual hosts in Apache) can be used to encapsulate configuration details and host more than one domain from a single server. We will set up a domain called **your\_domain**, but you should **replace this with your own domain name**.

Nginx on Ubuntu 22.04 has one server block enabled by default that is configured to serve documents out of a directory at /var/www/html. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying /var/www/html, let’s create a directory structure within /var/www for our **your\_domain** site, leaving /var/www/html in place as the default directory to be served if a client request doesn’t match any other sites.

Create the directory for **your\_domain** as follows, using the -p flag to create any necessary parent directories:

1. sudo mkdir -p /var/www/your\_domain/html

Next, assign ownership of the directory with the $USER environment variable:

1. sudo chown -R $USER:$USER /var/www/your\_domain/html

The permissions of your web roots should be correct if you haven’t modified your umask value, which sets default file permissions. To ensure that your permissions are correct and allow the owner to read, write, and execute the files while granting only read and execute permissions to groups and others, you can input the following command:

1. sudo chmod -R 755 /var/www/your\_domain

Next, create a sample index.html page using nano or your favorite editor:

1. nano /var/www/your\_domain/html/index.html

Inside, add the following sample HTML:

/var/www/your\_domain/html/index.html

<html>

<head>

<title>Welcome to your\_domain!</title>

</head>

<body>

<h1>Success! The your\_domain server block is working!</h1>

</body>

</html>

Save and close the file by pressing Ctrl+X to exit, then when prompted to save, Y and then Enter.

In order for Nginx to serve this content, it’s necessary to create a server block with the correct directives. Instead of modifying the default configuration file directly, let’s make a new one at /etc/nginx/sites-available/your\_domain:

1. sudo nano /etc/nginx/sites-available/your\_domain

Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:

/etc/nginx/sites-available/your\_domain

server {

listen 80;

listen [::]:80;

root /var/www/your\_domain/html;

index index.html index.htm index.nginx-debian.html;

server\_name your\_domain www.your\_domain;

location / {

try\_files $uri $uri/ =404;

}

}

Notice that we’ve updated the root configuration to our new directory, and the server\_name to our domain name.

Next, let’s enable the file by creating a link from it to the sites-enabled directory, which Nginx reads from during startup:

1. sudo ln -s /etc/nginx/sites-available/your\_domain /etc/nginx/sites-enabled/

**Note:** Nginx uses a common practice called symbolic links, or symlinks, to track which of your server blocks are enabled. Creating a symlink is like creating a shortcut on disk, so that you could later delete the shortcut from the sites-enabled directory while keeping the server block in sites-available if you wanted to enable it.

Two server blocks are now enabled and configured to respond to requests based on their listen and server\_name directives (you can read more about how Nginx processes these directives [here](https://www.digitalocean.com/community/tutorials/understanding-nginx-server-and-location-block-selection-algorithms)):

* your\_domain: Will respond to requests for your\_domain and www.your\_domain.
* default: Will respond to any requests on port 80 that do not match the other two blocks.

To avoid a possible hash bucket memory problem that can arise from adding additional server names, it is necessary to adjust a single value in the /etc/nginx/nginx.conf file. Open the file:

1. sudo nano /etc/nginx/nginx.conf

Find the server\_names\_hash\_bucket\_size directive and remove the # symbol to uncomment the line. If you are using nano, you can quickly search for words in the file by pressing CTRL and w.

**Note:** Commenting out lines of code – usually by putting # at the start of a line – is another way of disabling them without needing to actually delete them. Many configuration files ship with multiple options commented out so that they can be enabled or disabled, by toggling them between active code and documentation.

/etc/nginx/nginx.conf

...

http {

...

server\_names\_hash\_bucket\_size 64;

...

}

...

Save and close the file when you are finished.

Next, test to make sure that there are no syntax errors in any of your Nginx files:

1. sudo nginx -t

If there aren’t any problems, restart Nginx to enable your changes:

1. sudo systemctl restart nginx

Nginx should now be serving your domain name. You can test this by navigating to http://your\_domain, where you should see something like this:

**Step 3 — Obtaining an SSL Certificate**

Certbot provides a variety of ways to obtain SSL certificates through plugins. The Nginx plugin will take care of reconfiguring Nginx and reloading the config whenever necessary. To use this plugin, type the following:

1. sudo certbot --nginx -d example.com -d www.example.com

This runs certbot with the --nginx plugin, using -d to specify the domain names we’d like the certificate to be valid for.

When running the command, you will be prompted to enter an email address and agree to the terms of service. After doing so, you should see a message telling you the process was successful and where your certificates are stored:

Output

IMPORTANT NOTES:

Successfully received certificate.

Certificate is saved at: /etc/letsencrypt/live/your\_domain/fullchain.pem

Key is saved at: /etc/letsencrypt/live/your\_domain/privkey.pem

This certificate expires on 2022-06-01.

These files will be updated when the certificate renews.

Certbot has set up a scheduled task to automatically renew this certificate in the background.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

If you like Certbot, please consider supporting our work by:

\* Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate

\* Donating to EFF: https://eff.org/donate-le

Your certificates are downloaded, installed, and loaded, and your Nginx configuration will now automatically redirect all web requests to https://. Try reloading your website and notice your browser’s security indicator. It should indicate that the site is properly secured, usually with a lock icon. If you test your server using the  <https://www.ssllabs.com/ssltest/>, it will get an **A** grade.

Let’s finish by testing the renewal process.

## **Step 4— Verifying Certbot Auto-Renewal**

Let’s Encrypt’s certificates are only valid for ninety days. This is to encourage users to automate their certificate renewal process. The certbot package we installed takes care of this for us by adding a systemd timer that will run twice a day and automatically renew any certificate that’s within thirty days of expiration.

You can query the status of the timer with systemctl:

1. sudo systemctl status snap.certbot.renew.service

Output

○ snap.certbot.renew.service - Service for snap application certbot.renew

Loaded: loaded (/etc/systemd/system/snap.certbot.renew.service; static)

Active: inactive (dead)

TriggeredBy: ● snap.certbot.renew.timer

To test the renewal process, you can do a dry run with certbot:

1. sudo certbot renew --dry-run

If you see no errors, you’re all set. When necessary, Certbot will renew your certificates and reload Nginx to pick up the changes. If the automated renewal process ever fails, Let’s Encrypt will send a message to the email you specified, warning you when your certificate is about to expire