Web Servers

- NGINX
- Apache

NGINX

Install

Install nginx by running the commands below

```
sudo apt update && sudo apt upgrade
sudo apt install nginx
```

Configure SSL

> SSL certificates are limited, see the LetsEncrypt documentation on SSL rate limits for more information. Take notice of the section about renewals - to avoid regenerating certificates during testing, run `sudo certbot certonly --dry-run -d domain.com -d www.domain.com`

Before we pass any traffic, we should configure SSL for any domains we want to serve on this host. To use LetsEncrypt and Certbot to do this, run the commands below.

Certbot installation instructions have changed

The new method of installation, as explained in the above link, is using `snap`. I very much dislike snap, because I've been on systems with limited resources and have experienced snap causing poor performance, especially when installing larger applications with it.

In any case, the new method of installation is below. `certbot` is an apt package, but the official instructions do not recommend to use that apt package (so why is it there o.o)

If you installed with apt, remove certbot first, then reinstall with snap.

```
sudo apt remove certbot
sudo snap install --classic certbot
sudo certbot certonly --nginx
```

Reading the output generated, we can see where our certificates were created. **Take note of these paths**, you will need to refer to these certificates within your `/etc/nginx/nginx.conf`

```
Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/git.shaunreed.com/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/git.shaunreed.com/privkey.pem
This certificate expires on 2022-04-25.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.
```

There are a few benefits to using Certbot. Your certificates will automatically be renewed when nearing expiration, and it even configures nginx for you automatically.

Instead of using the default configuration Certbot creates, you can make one yourself. Below, we create our own nginx configuration from scratch which still uses Certbot to manage SSL certificates.

Basic NGINX Settings

A virtual host in NGINX serves content based on settings found within `/etc/nginx/nginx.conf` , we can use these settings to do things like handle SSL and pass traffic to other hosts if using a specific sub domain.

These settings can be modified to suit the needs of a basic host serving one page or application.Below, we route traffic to a docker container running on a localhost port.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains
  server {
    listen 80;
    server_name domain.com www.domain.com;
    return 301 https://www.domain.com$request_uri;

  }
```

```
  # SSL - domain.com
  server {
    server_name domain.com www.domain.com;
    server_tokens off;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/domain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/domain.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # Pass to container
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }

  }

}
```

Multiple Domains

If serving multiple domains over SSL on one host, see the configuration below for a basic example. It should look fairly similar to the above.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains
```

```
server {
  listen 80;
  server_name domain.com www.domain.com;
  return 301 https://www.domain.com$request_uri;
}

server {
  listen 80;
  server_name domain2.com www.domain2.com;
  return 301 https://www.domain2.com$request_uri;
}

# SSL - domain
server {
  server_name domain.com www.domain.com;
  server_tokens off;
  listen 443 ssl;
  ssl_certificate /etc/letsencrypt/live/domain.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/domain.com/privkey.pem;
  include /etc/letsencrypt/options-ssl-nginx.conf;
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

  # Pass to container
  location / {
    include proxy_params;
    proxy_pass http://localhost:1234/;
  }

}

# SSL - domain2
server {
  server_name domain2.com www.domain2.com;
  server_tokens off;
  listen 443 ssl;
  ssl_certificate /etc/letsencrypt/live/domain2.com/fullchain.pem; # managed by Certbot
  ssl_certificate_key /etc/letsencrypt/live/domain2.com/privkey.pem; # managed by Certbot
  include /etc/letsencrypt/options-ssl-nginx.conf;
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
```

```
    # Pass to second container
    location / {
      include proxy_params;
      proxy_pass http://localhost:4321/;
    }

  }
}
```

Above, we serve two different applications running on different ports depending on the url request.

Custom logging

Sometimes, especially when hosting multiple domains on one box, you may want to separate the NGINX logs based on return code *and* host / domain name referenced. Below we see a `nginx.conf` which enables this feature -

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains for first server
  server {
    listen 80;
    server_name firstsite.com www.firstsite.com;
    return 301 https://www.firstsite.com$request_uri;
  }

  # Redirect root domains for second server
  server {
    listen 80;
    server_name secondsite.com;
    return 301 https://secondsite.com$request_uri;
  }
```

```
# Map the 100-200 error codes to $oks
map $status $oks {
  ~^[1-2] 1;
  default 0;
}

# Map the 400-500 error codes to $errs
map $status $errs {
  ~^[4-5] 1;
  default 0;
}

# Map the 300 error codes to $redir
map $status $redir {
  ~^[3] 1;
  default 0;
}

# SSL - firstsite
server {
  server_name firstsite.com www.firstsite.com;
  server_tokens off;
  listen 443 ssl;
  ssl_certificate /etc/letsencrypt/live/firstsite.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/firstsite.com/privkey.pem;

  # Configure Server-wide logging
  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;

  # Configure logs for this domain
  access_log /var/log/nginx/firstsite.log;

  # Configure return-specific logging
  access_log /var/log/nginx/firstsite.access combined if=$oks;
  access_log /var/log/nginx/firstsite.error combined if=$errs;
  access_log /var/log/nginx/firstsite.redir combined if=$redir;

  # Pass to firstsite container
```

```
    location / {
      include proxy_params;
      proxy_pass http://localhost:4321/;
    }
   }



  # SSL - secondsite
  server {
    server_name secondsite.com;
    server_tokens off;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/secondsite.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/secondsite.com/privkey.pem;

    # Configure Server-wide logging (to create one log to monitor with fail2ban, ossec, etc)
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    # Configure logs for this domain
    access_log /var/log/nginx/secondsite.log;

    # Configure return specific logginng if they match the statuses we mapped
    access_log /var/log/nginx/secondsite.access combined if=$oks;
    access_log /var/log/nginx/secondsite.error combined if=$errs;
    access_log /var/log/nginx/secondsite.redir combined if=$redir;

    # Pass to second container
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }

  }
}
```

Now after running `systemctl reload nginx` you should notice there are logs for each domain being created according to your organization above. It can be useful for smaller setups, but would quickly get out of hand with large amounts of traffic, I'd imagine.

No SSL

This is at the bottom of the page for a reason. This should only be used for testing, but you can get away with a very simple nginx configuration if you don't use SSL encryption -

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains
  server {
    listen 80;
    server_name domain.com www.domain.com;

    # Pass to container port
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }

  }
}
```

Apache

Changing your Apache Server Root Directory

Your Apache installation determines what content to serve based on the DocumentRoot Default directory.

```
DocumentRoot Default = /var/www/html


-document root configured in the following files:
sudo vim /etc/apache2/sites-available/000-default.conf
sudo vim /etc/apache2/apache2.conf
```

Application Port Settings

After installing Apache, there will be a default configuration file located in `/etc/apache/enabled-sites/` that contains some VirtualHost configuration settings. It can be modified using Apache modules to allow Apache to serve our application as our root domain. To do this, we will need to enable one of the following Apache mods using `sudo a2enmod <module name>`

So, we run..

```
sudo a2enmod proxy_http
sudo a2enmod mod_proxy
```

After these two modules install, set up your VirtualHost configuration within the /etc/apache/enabled-sites/ directory as follows:

```
<VirtualHost *:80>
    ServerAdmin me@mydomain.com
    ServerAlias mydomain.com
    ServerName www.mydomain.com
    ProxyPreserveHost On

    # setup the proxy
    <Proxy *>
        Order allow,deny
        Allow from all
    </Proxy>
    ProxyPass / http://www.mydomain.com:8888/
```

```
    ProxyPassReverse / http://www.mydomain.com:8888/
</VirtualHost>
```

To apply these settings, run `sudo systemctl restart apache2.service` to restart Apache and reload your new configuration. If this command returns an error, look closer at your config settings and be sure you enabled the necessary modules.