A Raspberry Pi Web Server (2012 to 2023).

I bought my first Raspberry Pi in 2012. One of my first Raspberry Pi projects was building a web server to host a single static website. That server ran with the same hardware and system-software for 9 years, 24/7, with almost no interruption. The only failures were peripherals – an SD memory card, and a power supply.

During that period the site itself was much expanded and regularly updated.   In 2018 it was cited as "the greenest website tested" in a talk by Tom Greenwood to the London Wordpress Meetup Conference, and in 2021 he used it as an example in his book "Sustainable Web Design".   That book, and exchanges with Tom Greenwood influenced me in subsequent work. The home page now carries a website carbon badge certifying that it is *"Cleaner than 98% of pages tested".*   Furthermore the typical power consumption of the Raspberry Pi (5 watts) is low.

It was time for a rebuild in 2021, with a newer Raspberry Pi (Version 3) and the latest Pi OS software. The basic configuration remained the same, with minor changes to configuration files and their locations.

1. However 'chrooting' was adopted for the site files. This is a security improvement denying the web site 'owner' access via ftp to anything other than his own site files. Here is the method of achieving this

Create the user, <siteowner>. Create directories /home/<siteowner>/ftp (the ftp root) and home/<siteowner>/ftp/files (to contain the site).

Deny writeability to the ftp root, using terminal command ***sudo chmod a-w /home/<username>/ftp****.*

Edit the vsftpd configuration file /etc/vsftpd.conf to include the following directives ***anonymous_enable =NO ; local_enable=YES ; write_enable=YES ; chroot local_user = YES ; local_root =/home/<username>/ftp***

Insert the root definition within the nginx server block (root /home/<username>/ftp/files)

2. Another change (which I should have made much sooner) was to define the browser cache settings. Website html files which might be edited without name change should not be cached.

So insert ***expires $expires;*** after the root statement in the nginx server block and then insert a 'map' for $expires such as the following which caches all images but does not cache html files.

map $sent_http_content_type $expires { default off;

text/html epoch;

~image/ max; }

Place this just ahead of the server block.

3. Finally I changed the domain from http to https. The instructions on 'Letsencrypt' and related 'certbot' pages were faultless and easy to follow. In my case I needed to use ***certbot --expand*** to add an alternative version of the domain name. Redirection of http requests worked straight away. (Port 443 needs to be opened for https on the router – but Port 80 for http should not be closed.