Hardening a Web Application with NGINX - Part 2

In these exercises, we'll see how to improve the capacity of the web application with caching, and we will show how caching can protect from total application failures.

If there's time, we'll finish up by using LetsEncrypt to obtain an SSL certificate for the application, and we will enable encryption with the goal of achieving an A+ score on the Qualys SSL Test.

Speed up the Web Application with Caching

Despite all your work so far to protect the application and to improve reliability, performance is still below what you might like. You'll use microcaching to increase the capacity of the web application.

NGINX is commonly used as a web cache, offloading requests from the upstream origin servers and responding directly from a local cache. Microcaching is an effective method for accelerating the delivery of dynamic, non-personalized content by caching it for very short periods of time. It's appropriate for things like news feeds, blog post lists, calendar entries or database reports.

First, rerun the **wrk** test with a higher level of concurrency (20).

wrk -c 20 -d 30 http://localhost/

Record the best-case requests-per-second for the home page, and the corresponding CPU utilization using **top**.

Configure a Simple Web Cache

Create a disk-based cache and configure NGINX Plus to cache all valid 200 status-code responses for 1 second. Implement a lock so that only one request at a time seeks to update the cache, and other requests use the existing (stale) content.

Edit the beginning of your wordpress.conf file as follows:

```
proxy_cache_path /tmp/cache keys_zone=cache:10m levels=1:2 inactive=3600s max_size=100m;

server {
    listen 80;

    proxy_cache cache;
    proxy_cache_valid 200 1s;
    proxy_cache_lock on;
    proxy_cache_use_stale updating;

    # ...
```

Reload NGINX (**service nginx reload)** and re-run the **wrk** benchmark. Observe the change in performance and CPU. What just happened?

Protect the WordPress Server from Total Failure

Let's consider the possibility that the origin web server fails completely. You'll see how to create a ghost website to hide this failure. This is not a reliable, production solution, but it's a fun demo!

NGINX can be configured to respond from the local cache, using stale content, when the origin server has failed completely or if it returns 5xx server errors:

```
    proxy_cache_use_stale updating error timeout http_500 http_503 http_504;
```

Make this configuration change (**wordpress.conf**), and reload NGINX (**service nginx reload).**

Fill the web cache as best you can:

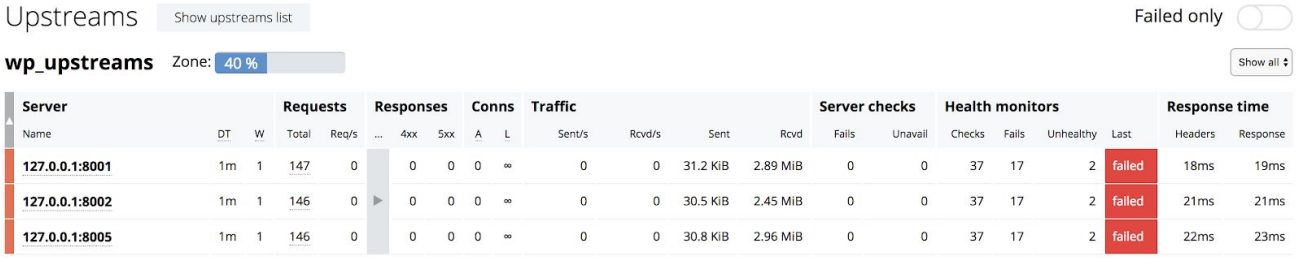
- Navigate round the site using your web browser
- Spider the site using **wget**. Note the use of the "`Accept-Encoding`" header, so that wget requests match requests issued by your browser:

```
cd /tmp
wget -pr -e robots=off --header "Accept-Encoding: gzip, deflate"
http://workshopXX.nginxtraining.com/
```

If you look in **/tmp/cache**, you'll see the cached content stored on disk. NGINX writes content to this location.

Now we kill the web server completely: **service apache2 stop**

Apache and Wordpress are no longer running on the host. NGINX's health check confirms that the upstream servers have failed:



Attempt to navigate round the website. If you have filled your cache appropriately, most or all of the resources should be served from your cache, even though they may be dynamic and have expired.

When would I use this?

This is a last-resort way to preserve a service in the event of a total failure. The feature is more useful when you are using NGINX as a CDN edge server, caching static content such as images and video files. It protects the service from intermittent connection errors to the origin server.

More tips to control and monitor Caching

If you have time: Configuring caching is a very complex task; NGINX tries to make it as easy as possible, but there are a lot of additional options.

If you want to clear NGINX's cache, you can safely delete the cache directory while NGINX is running: **rm -rf /tmp/cache**.

The NGINX Plus dashboard provides some basic statistics for the caches you have configured:

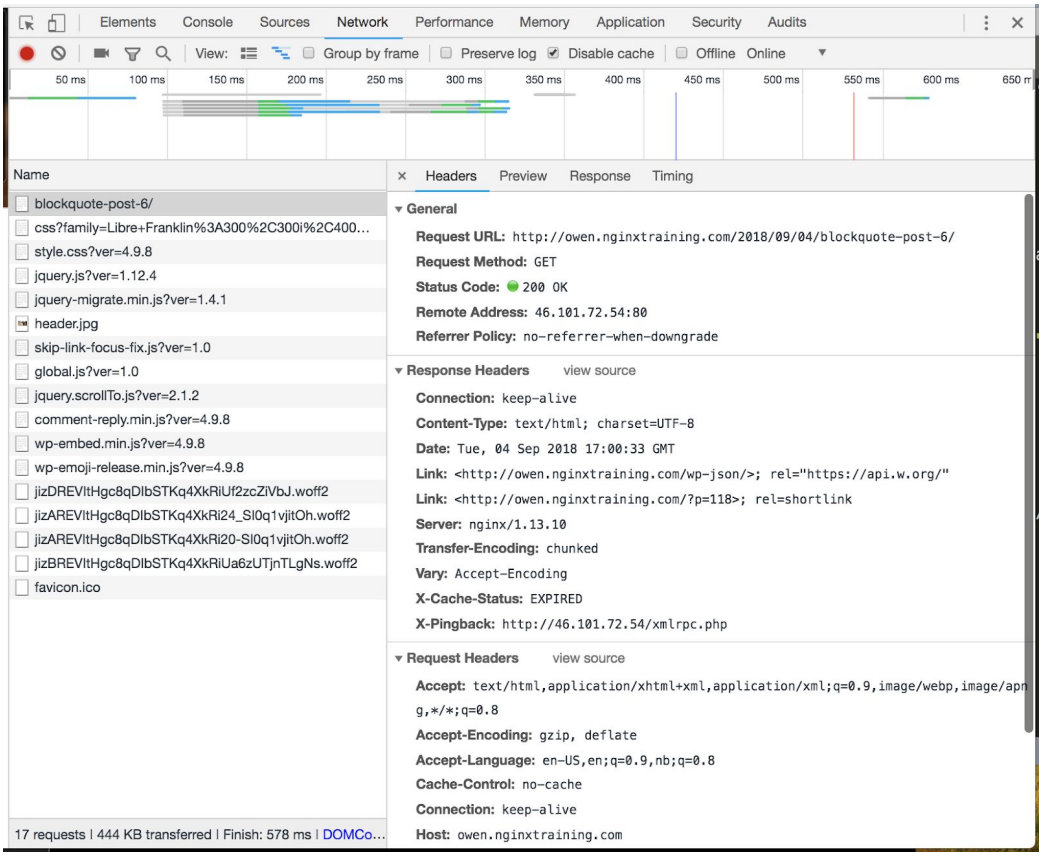
Caches

Zone	State	Memory usage	Max size	Used	Disk usage	Traffic			Hit Ratio
						Served	Written	Bypassed	
cache		1 %	100 MB	3.97 MB	4 %	13.1 GB	10.5 MB	10.5 MB	82%

You'll also find the following config useful when debugging your cache configuration:

```
add_header X-Cache-Status $upstream_cache_status;
```

This adds an 'X-Cache-Status' header to each response that indicates the cache status. You can inspect the values using the developer tools in your favourite browser:



Protect the site with SSL

Finally, we'll publish the WordPress site using HTTPS, using certificates from LetsEncrypt.

About LetsEncrypt

LetsEncrypt is an automated Certificate Authority that provides free, trusted SSL certificates. It uses a protocol called ACME to authenticate web sites and issue the certificates. There are dozens of ACME clients; we'll use acmetool in this example.

Install and Configure acmetool

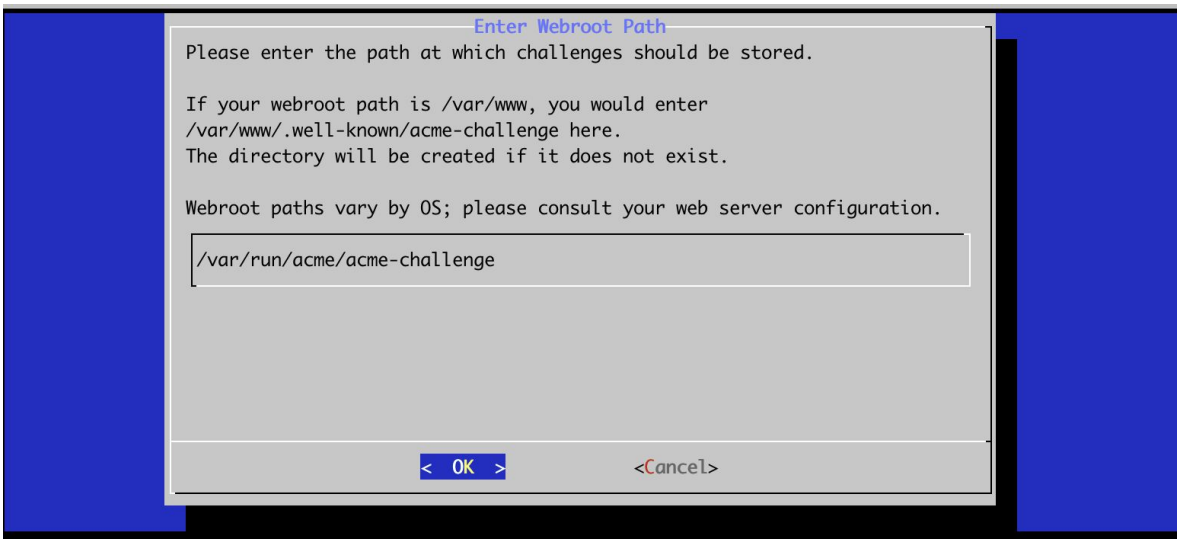
On your WordPress server, install acmetool and then run **acmetool quickstart:**

```
add-apt-repository -y ppa:hlandau/rhea
apt-get update
apt-get install -y acmetool

acmetool quickstart
```

In the quickstart configuration:

- Select **Live** certificates, and select **WEBROOT** mode
- Challenges should be stored: **/var/run/acme/acme-challenge** [continue despite warning]:



- Agree the Terms of Service; decline to give an email address; don't install the cron job.

Add the following to your **wordpress.conf** configuration, after the **`location / {...}`** block. Ask if you are unsure where to add this:

```
location /.well-known/acme-challenge/ {
      alias /var/run/acme/acme-challenge/;
}
```

Reload the NGINX configuration after making this change: **service nginx reload**

Get a LetsEncrypt Certificate using acmetool

When you request a certificate for **workshopXX.nginxtraining.com**, **acmetool** demonstrates to the LetsEncrypt service that you do own that domain. It does so by writing a temporary shared secret to the acme-challenge location.

Request a certificate for your domain:

```
acmetool want workshopXX.nginxtraining.com
```

If it works successfully, acmetool will store the private key and certificate chain in **/var/lib/acme/live**:

```
ls -l /var/lib/acme/live/workshopXX.nginxtraining.com/
total 20
-rw-r--r-- 1 root root 2171 Sep  4 17:08 cert
-rw-r--r-- 1 root root 1647 Sep  4 17:08 chain
-rw-r--r-- 1 root root 3818 Sep  4 17:08 fullchain
lrwxrwxrwx 1 root root   71 Sep  4 17:08 privkey ->
../../keys/t2hycpvtbltabrnl5bqvfwtqg7kl37q7v7bjp55tuyn7jsm7croa/privkey
-rw-r--r-- 1 root root   83 Sep  4 17:08 url
```

Configure NGINX Plus to use both HTTP and HTTPS

First, to keep things simple, we shall disable caching. Remove the following directives from wordpress.conf:

```
#proxy_cache_path

#proxy_cache cache;
#proxy_cache_valid 200 1s;
#proxy_cache_lock on;
#proxy_cache_use_stale updating;
```

We are then going to make three changes to our NGINX configuration:

- Configure the existing server to listen on **port 443** for SSL traffic, using the newly-obtained SSL keys

- Configure a new server listening on **port 80**, which immediately redirects requests to the SSL server using a 302 temporary redirect

- Add an **X-Forwarded-Proto** header to each request. Wordpress will use this to recognise the protocol, so that it can correctly generate https:// links.

The changes to your configuration are highlighted in **bold**:

```
# New virtual server - redirects to HTTPS
server {
    listen 80;
    return 302 https://$host$request_uri;
}

# Existing virtual server; change listen parameters, add SSL certificate and key
# and add the X-Forwarded-Proto header
server {
    listen 443 ssl;

    # Comment out or remove the 'listen 80'
    #listen 80;

    ssl_certificate /var/lib/acme/live/workshopXX.nginxtraining.com/fullchain;
    ssl_certificate_key /var/lib/acme/live/workshopXX.nginxtraining.com/privkey;

    location / {

        proxy_set_header X-Forwarded-Proto $scheme;

        ...
```

Reload your NGINX configuration after making the change.

Now try accessing your wordpress site. You should be immediately redirected to use SSL; any attempts to use HTTP are automatically upgraded to HTTPS.

You've enabled SSL encryption for your web application without any application changes!

Technical Notes

Note 1: This example redirects requests to HTTPS using a 302 (temporary redirect) status code. This is useful in a test environment where you may want to remove the SSL virtual server. In production, you should use a 301 (permanent redirect). Browsers will cache that instruction almost permanently, and automatically request HTTPS.

Note 2: Wordpress does not automatically detect the source protocol of requests. The **X-Forwarded-Proto** header informs WordPress, and our `setup.sh` script patched the wordpress code, in `wp-config.php`:

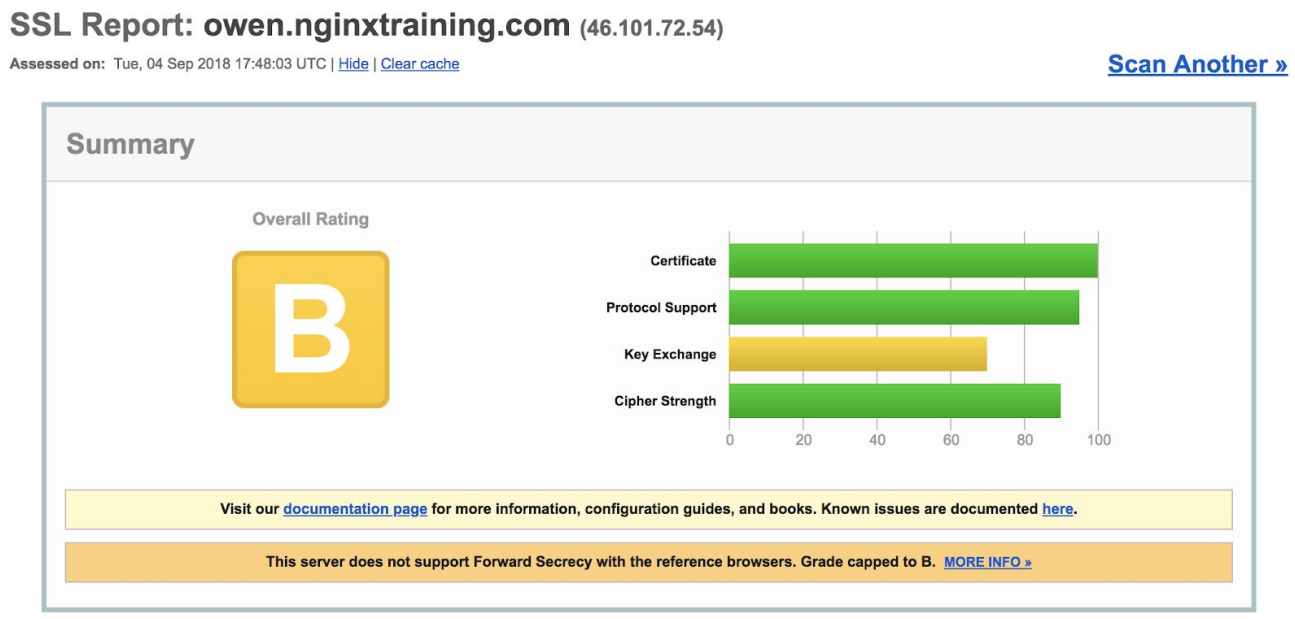
```
if ($_SERVER['HTTP_X_FORWARDED_PROTO'] == 'https')
    $_SERVER['HTTPS']='on';
```

In production, you may want to enable an SSL redirect plugin rather than patching the WordPress code directly.

Harden your SSL configuration

Qualys provide a comprehensive online testing tool that you can use to judge the security of your SSL configuration, and compatibility with a range of clients.

Go to https://www.ssllabs.com/ssltest, and test your website. The default NGINX configuration gives broad client compatibility, and should be sufficient to score a “B” with the ssllabs test:



Gaining an ‘A’ Ranking

You will need to enable Perfect Forward Secrecy (PFS) to get a higher ranking.

- Go to https://mozilla.github.io/server-side-tls/ssl-config-generator/
- Check your NGINX and OpenSSL version:

```
nginx -V
nginx version: nginx/1.XX.Y (nginx-plus-rZZ)
built by gcc ...
built with OpenSSL ...
```

- Generate a recommended SSL configuration, and add the following directives to your configuration:
    - `ssl_protocols`
    - `ssl_ciphers`
    - `ssl_prefer_server_ciphers`

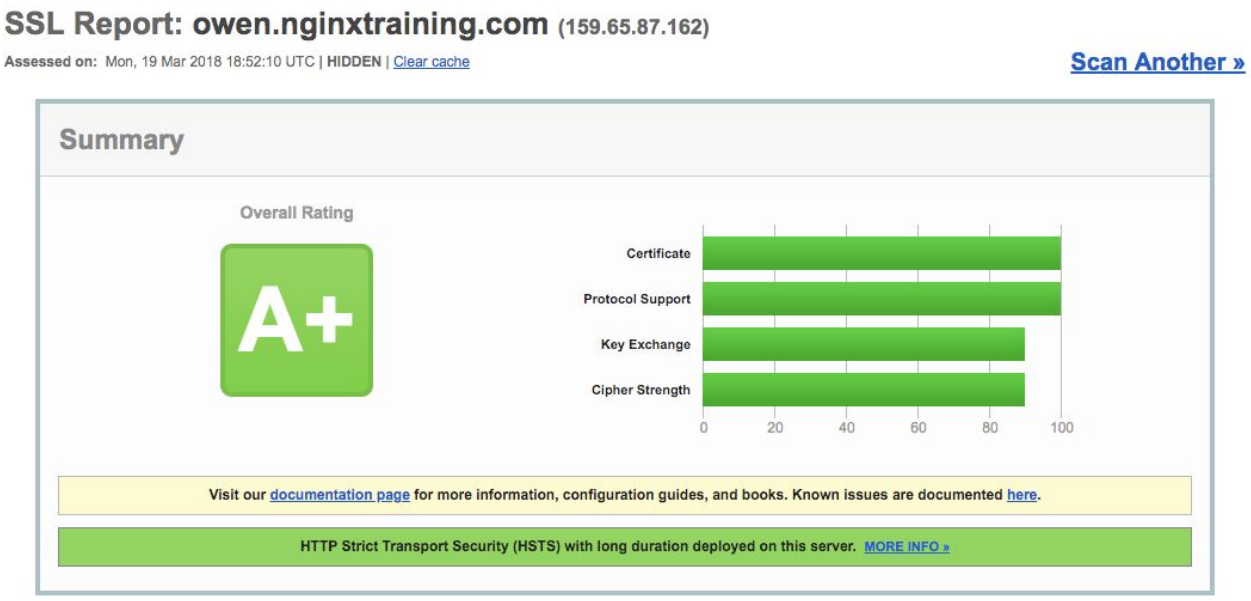
With a little work you should be able to achieve an A score on the SSL Labs test, but possibly at the expense of some client compatibility?

I want an A+ score!

If you want to get an A+ score on the SSLLabs test, you will need to enable [HSTS (HTTP Strict Transport Security)]:

```
add_header Strict-Transport-Security max-age=15768000;
```

Do this with care - see the blog post for some caveats!



What have we learnt?

*Firstly, **WELL DONE** for getting to the end of the NGINX Workshop! You deserve a beer!*

You have seen how NGINX can cache content from servers, even dynamic content such as a blog site, and deliver huge performance gains. Caching is a major use case for NGINX, and you can find more examples here: NGINX Blog - Cache Examples.

You have worked with LetsEncrypt to obtain a fully-validated SSL certificate, and configured NGINX to deliver our test application over SSL.

Finally, we hope you've gained some insight into what NGINX and NGINX Plus can do for you.