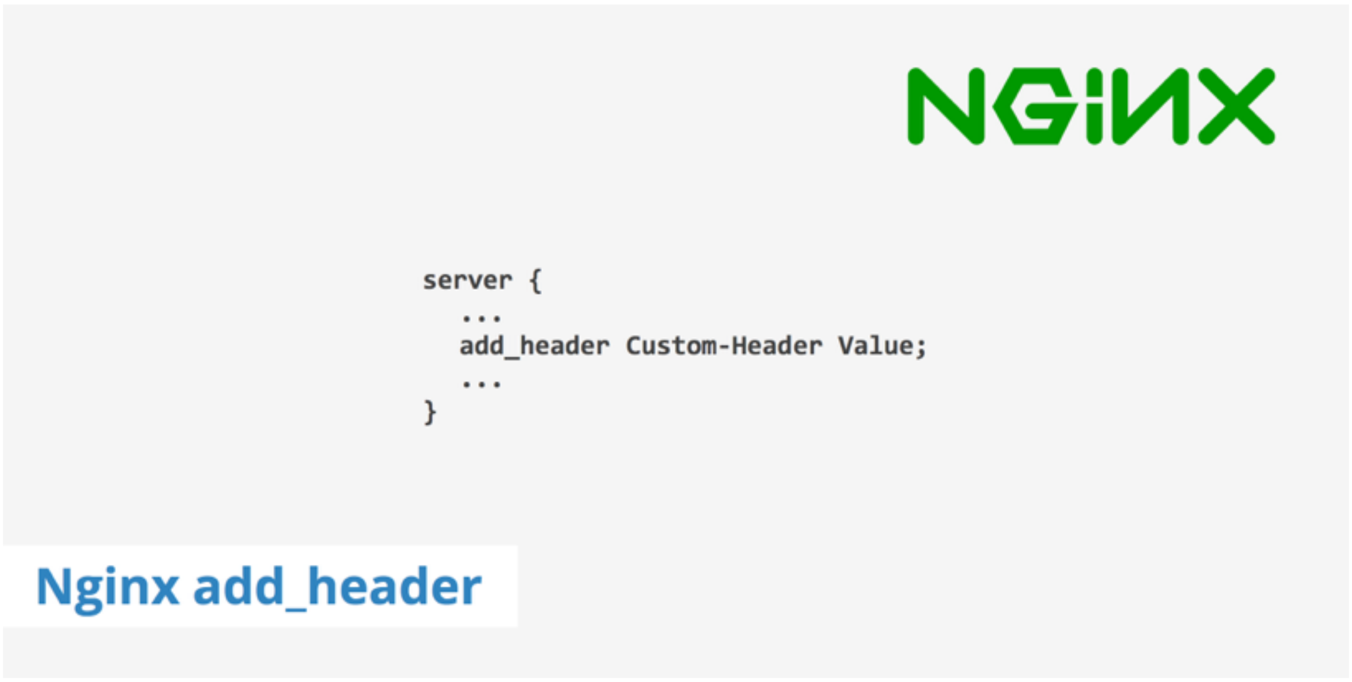
Using the Nginx add_header Directive

Updated: October 4, 2018



```
server {
   ...
   add_header Custom-Header Value;
   ...
}
```

What Is the Nginx add_header Directive?

The Nginx add_header directive allows you to define an arbitrary response header and value to be included in all response codes which are equal to: 200, 201, 204, 206, 301, 302, 303, 304, or 307. This can be defined from within your nginx.conf file by using the following snippet.

```
add_header Custom-Header Value;
```

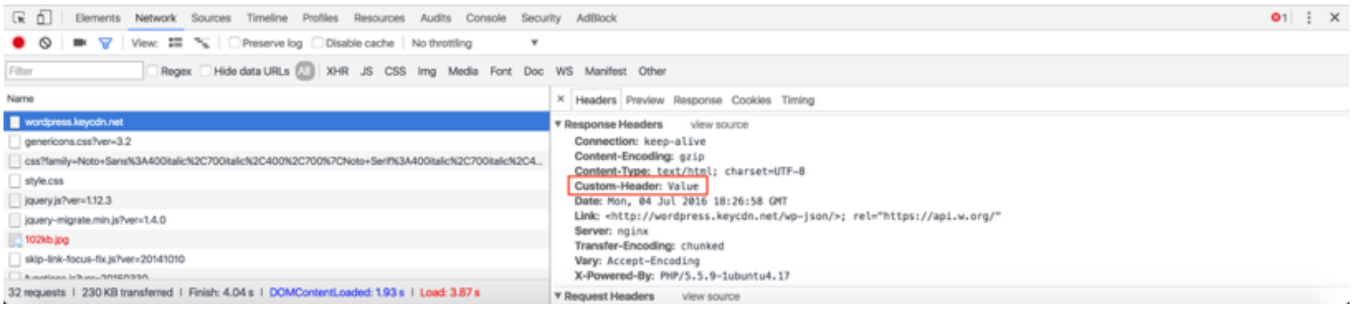
The Custom-Header portion corresponds to the name of your response header while the Value portion corresponds to what value you want the header to return. This directive can be defined either in a http, server, or location block.

How to Check If the Header Is Active

Once you have specified a custom header in your Nginx configuration file, save your changes and **reload the Nginx configuration** with the following command.

```
service nginx reload
```

Your custom header should now be active and delivered as a response header. There are a couple of ways to verify that the Nginx add_header has been properly set. The first method is to **check your response headers using Chrome dev tools**. To do this, simply open the Chrome inspect tool and navigate to the Network tab. Select your HTML document and check the Response Headers section to verify that your custom header was sent.



wordpress.keycdn.net
genericons.css?ver=3.2
css?family=Noto+Sans%3A400italic%2C700italic%2C400%2C700%7CNoto+Serif%3A400italic%2C700italic%2C4...
style.css
jquery.js?ver=1.12.3
jquery-migrate.min.js?ver=1.4.0
102kb.jpg
skip-link-focus-fix.js?ver=20141010

32 requests | 230 KB transferred | Finish: 4.04 s | DOMContentLoaded: 1.93 s | Load: 3.87 s

▼ Response Headers

```
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Custom-Header: Value
Date: Mon, 04 Jul 2016 18:26:58 GMT
Link: <http://wordpress.keycdn.net/wp-json/>; rel="https://api.w.org/"
Server: nginx
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Powered-By: PHP/5.5.9-1ubuntu4.17
```

▼ Request Headers

Additionally, you can also use cURL to check wether the custom header is being returned. Check out our Popular cURL examples article for a list of ways to use cURL. To check a particular URL's headers use the following command.

```
curl -I http://wordpress.keycdn.net/

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 04 Jul 2016 18:40:59 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.5.9-1ubuntu4.17
Link: <http://wordpress.keycdn.net/wp-json/>; rel="https://api.w.org/"
Custom-Header: Value
```

Purpose of Sending Custom Headers

Now that you know how to send a custom response header using the Nginx add_header method, you may be asking yourself – “what the purpose is of doing this?”. Custom headers can be used for **informational / debugging purposes**. For example, WordPress includes a header such as `X-Powered-By: PHP/5.5.9-1ubuntu4.17` to identify which version of PHP and Ubuntu your server is running.

Additionally, if you are using a CDN, you may have noticed a response header such as `X-Cache:HIT` or `X-Cache:MISS`. These custom headers are used for informational purposes so that the client knows wether the asset was delivered via cache or not.

You can also define **specific headers to be used solely for certain files or folders**. For example, if you do not want to cache a particular file, you can use a location block to define the file’s location and use the following add_header snippet.

```
add_header Cache-control no-cache;
```

This will tell the browser not to cache the particular asset(s) stored at the location defined. To learn more about caching headers read our complete HTTP Cache Headers guide.

What to Be Aware of When Using Nginx add_header

It is important to be aware of how exactly the Nginx add_header works in terms of hierarchical Nginx configuration structure. From the Nginx HTTP Headers Module documentation, it says:

> *There could be several add_header directives. These directives are*

> *inherited from the previous level if and only if there are no add_header directives defined on the current level.*

Therefore, let's say you have an http block and have specified the add_header directive within that block. Then, within the http block you have 2 server blocks – one for HTTP and one for HTTPs.

Let's say we **don't include** an add_header directive within the HTTP server block, however we **do include** an additional add_header within the HTTPs server block. In this scenario, the add_header directive defined in the http block will only be inherited by the HTTP server block as it **does not have any add_header directive defined on the current level**. On the other hand, the HTTPS server block will not inherit the add_header directive defined in the http block.

Server administrators should be aware of this when modifying their Nginx configuration. There are solutions to this such as using an alternative module such as ngx_headers_more or defining a common config snippet. To learn more about this Nginx add_headers limitation, read more here.

#PERFMATTERS

Free Test Account
Supercharge your Website with KeyCDN
HTTP/2 - Free SSL - RESTful API - Instant Purge