ReInflating

So you want to handle the text from a proxy or other app which has deflate'd the content?

Setup a log with debugging info

The first most valuable thing for diagnosing any issues is to create a compression log which will show you the Accept-Encoding (AE) permitted by the client (browser) along with the Transfer-Encoding (TE) and Content-Encoding (CE) for the request body with the total input byte count. We'll prefix each entry with the client ip/host and their request line. And the meat of the response to the client are the response size (bytes transmitted), response Transfer-Encoding and response Content-Encoding. Throughout these examples we'll be sharing the results as gathered with the following config snippet;

```
LogFormat "%h \"%r\" %>s %b (%O) TE:\"%{Transfer-Encoding}o\" CE:\"%{Content-Encoding}o\" <-- %I AE:\"%{Accept-Encoding}i\" TE:\"%{Transfer-Encoding}i\" CE:\"%{Content-Encoding}i\"" compression

CustomLog logs/compression.log compression
```

Configuring a local test server

The first thing is to provide a server to test against which deflates it's output. We can do this in the same instance of Apache, simply set up an alias to the htdocs for /deflated/
requests for the very same documents in /.

```
Alias /deflated "/path-to/htdocs"
<Location "/deflated">
    SetOutputFilter DEFLATE
</Location>
```

Requesting the same LICENSE.txt from both / and /deflated/ reveals the impact deflate can offer;

```
127.0.0.1 "GET /LICENSE.txt HTTP/1.1" 200 39858 (40146) TE:"-" CE:"-" <-- 412 AE:"gzip,deflate" TE:"-" CE:"-"
127.0.0.1 "GET /deflated/LICENSE.txt HTTP/1.1" 200 10629 (10964) TE:"-" CE:"gzip" <-- 418 AE:"gzip,deflate" TE:"-" CE:"-"
```

Note the response body has shrunk nearly to a quarter of it's original size.

Proxy requests to the local test server

For our first experiment (and we will keep it in our config as the LAST snippet, because this would override any other example below if it appears first), we will create a reverse-proxy (gateway) to our own server's content, everything requested from /proxy/ will actually be proxied from / of our server.

```
ProxyPass /proxy/ http://127.0.0.1/
ProxyPassReverse /proxy/ http://127.0.0.1/
```

The compression.log gets interesting here. The first request to be satisfied is our back-end proxied request from ourself, once it's completed the /proxy/ request can be completed. In this case, the responses are identical, we aren't manipulating the Content-Encoding as the request passes through the proxy;

```
127.0.0.1 "GET /LICENSE.txt HTTP/1.1" 200 39858 (40146) TE:"-" CE:"-" <-- 501 AE:"gzip,deflate" TE:"-" CE:"-"
127.0.0.1 "GET /proxy/LICENSE.txt HTTP/1.1" 200 39858 (40146) TE:"-" CE:"-" <-- 418 AE:"gzip,deflate" TE:"-" CE:"-"

127.0.0.1 "GET /deflated/LICENSE.txt HTTP/1.1" 200 10629 (10964) TE:"-" CE:"gzip" <-- 510 AE:"gzip,deflate" TE:"-" CE:"-"
127.0.0.1 "GET /proxy/deflated/LICENSE.txt HTTP/1.1" 200 10629 (10964) TE:"-" CE:"gzip" <-- 427 AE:"gzip,deflate" TE:"-" CE:"-"
```

Remove the request for compression

Now the quick method of disabling compression is to never request a compressed document from the back-end machine, which we can accomplish by unset'ting the Accept-Encoding header from the client.

```
ProxyPass /proxy/nodeflate/ http://127.0.0.1/deflated/
<Location "/proxy/nodeflate/">
    RequestHeader unset Accept-Encoding
</Location>
```

And lo, neither the backend or proxy request appears to support gzip encoding;

```
127.0.0.1 "GET /deflated/LICENSE.txt HTTP/1.1" 200 39858 (40169) TE:"-" CE:"-" <-- 479 AE:"-" TE:"-" CE:"-"
127.0.0.1 "GET /proxy/nodeflate/LICENSE.txt HTTP/1.1" 200 39858 (40169) TE:"-" CE:"-" <-- 428 AE:"-" TE:"-" CE:"
-"
```

Keep compression but inflate at the proxy

But if we want to continue to accept deflated content from the back end server for bandwidth conservation, it's possible to use mod_deflate to perform response re-inflation. We only want to do this if the back end server provided a response in Content-Encoding:gzip, or else we will be trying to unpack an apparently corrupt data stream;

```
FilterProvider gzinflate INFLATE resp=Content-Encoding $gzip

ProxyPass /proxy/reinflate/ http://127.0.0.1/deflated/
<Location "/proxy/reinflate/">
    FilterChain gzinflate
</Location>
```

Now when a request is made through /proxy/reinflate/, we will take the deflated back end content and explode it. Note that the (2nd) response to the client is nearly four times larger, again, than the backend request;

```
127.0.0.1 "GET /deflated/LICENSE.txt HTTP/1.1" 200 10629 (10964) TE:"-" CE:"gzip" <-- 510 AE:"gzip,deflate" TE:"
-" CE:"-"
127.0.0.1 "GET /proxy/reinflate/LICENSE.txt HTTP/1.1" 200 39858 (40219) TE:"chunked" CE:"-" <-- 428 AE:"gzip,
deflate" TE:"-" CE:"-"
```

Transform the inflated content

Let's take this one step further, and add on a transformation. There's an experimental filter mod_case_filter which upper-cases the response, and it's perfect for illustrating the transformation. We can't upper-case the deflated binary stream, so we'll have to inflate it, then transform it.

NOTE A one line bug in mod_case_filter.c that must be fixed in 2.2.6 Insert the line;

```
        apr_brigade_cleanup(pbbIn);
```

immediately before this line;

```
        return ap_pass_brigade(f->next, pbbOut);
```

and rebuild the module.

So using the example above, we'll build on it with another FilterProvider we'll name ucase, and apply both gzinflate and ucase in that order to all /proxy /ucase/ requests;

```
FilterProvider gzinflate INFLATE resp=Content-Encoding $gzip
FilterProvider ucase CaseFilter Content-Type $text/

ProxyPass /proxy/ucase/ http://127.0.0.1/deflated/
<Location "/proxy/ucase/">
    FilterChain +gzinflate +ucase
</Location>
```

And the resulting log shows that the response to the client remains undeflated, and if you inspect it in the browser, it's irritatingly in all caps;

```
127.0.0.1 "GET /deflated/LICENSE.txt HTTP/1.1" 200 10629 (10964) TE:"-" CE:"gzip" <-- 513 AE:"gzip,deflate" TE:"
-" CE:"-"
127.0.0.1 "GET /proxy/ucase/LICENSE.txt HTTP/1.1" 200 39858 (40219) TE:"chunked" CE:"-" <-- 427 AE:"gzip,
deflate" TE:"-" CE:"-"
```

Redeflate the transformed content

Building on those two examples, we'll take this one step further, and ensure the final response can in fact be deflated to the client, so we'll name our redeflate filter gzdeflate and apply all three filters, in specific order, to inflate, manipulate, and re-deflate the response;

```
FilterProvider gzinflate INFLATE resp=Content-Encoding $gzip
FilterProvider ucase CaseFilter Content-Type $text/
FilterProvider gzdeflate DEFLATE Content-Type $text/

ProxyPass /proxy/ucase/ http://127.0.0.1/deflated/
<Location "/proxy/ucase/">
    FilterChain +gzinflate +ucase +gzdeflate
</Location>
```

Note a fun side effect on deflation of upper-casing the document;

```
127.0.0.1 "GET /deflated/LICENSE.txt HTTP/1.1" 200 10629 (10964) TE:"-" CE:"gzip" <-- 514 AE:"gzip,deflate" TE:"
-" CE:"-"
127.0.0.1 "GET /proxy/ucase/LICENSE.txt HTTP/1.1" 200 9600 (9934) TE:"-" CE:"gzip" <-- 428 AE:"gzip,deflate"
TE:"-" CE:"-"
```

With only 26 alpha characters rather than 52 (upper and lower case) it compresses down more efficiently than the back end had provided it to us.

Forward proxies

We can actually do this for forward proxies as well. Here is an example proxy server which inflating every C-E:gzip compressed response and serves only plain text to the clients using our proxy, we save about 10%.

```
LogFormat "%h \"%r\" %>s %b (%O) TE:\"%{Transfer-Encoding}o\" CE:\"%{Content-Encoding}o\" <-- %I AE:\"%{Accept-
Encoding}i\" TE:\"%{Transfer-Encoding}i\" CE:\"%{Content-Encoding}i\"" compression

FilterProvider gzinflate INFLATE resp=Content-Encoding $gzip

<VirtualHost 192.168.0.254:80>
    ErrorLog logs/proxyerror.log
    CustomLog logs/proxyaccess.log common
    CustomLog logs/proxycompression.log compression
    ProxyRequests On
    ProxyVia On

    <Proxy http:*>
        FilterChain gzinflate
        Order deny,allow
        Deny from all
        Allow from 127. 10. 192.168.
    </Proxy>
</VirtualHost>
```