

Context aware HTTP cache

If you are looking for 5.2/5.3 Context aware HTTP cache, see the page [Context aware HTTP cache in eZ Publish 5.2-5.3](#)

- [Use case](#)
- [Feature](#)
- [Http cache clear](#)
- [Symfony reverse proxy](#)
- [Varnish](#)
- [User context hash](#)
- [Workflow](#)
- [User hash generation](#)
 - [User hash generation with Varnish 3](#)
 - [User hash generation with Varnish 4](#)
 - [New anonymous X-User-Hash](#)
 - [Default options for FOSHttpCacheBundle defined in eZ](#)
- [Credits](#)

This feature is available as of **eZ Publish 5.2**

Use case

Being based on Symfony 2, eZ Publish 5 uses HTTP cache from version 5.0 [extended with content awareness](#). However this cache management is only available for anonymous users due to HTTP restrictions.

It is of course possible to make HTTP cache vary thanks to the `Vary` response header, but this header can only be based on one of the request headers (e.g. `Accept-Encoding`). Thus, to make the cache vary on a specific context (e.g. a hash based on a user roles and limitations), this context must be present in the original request

Feature

As the response can vary on a request header, the base solution is to make the kernel do a sub-request in order to retrieve the user context hash (aka **user hash**). Once the *user hash* has been retrieved, it's injected in the original request in the `X-User-Hash` custom header, making it possible to vary the HTTP response on this header:

```
<?php
use Symfony\Component\HttpFoundation\Response;

// ...

// Inside a controller action
$response = new Response();
$response->setVary( 'X-User-Hash' );
```

This solution is implemented in Symfony reverse proxy (aka *HttpCache*) and is also accessible to dedicated reverse proxies like Varnish.

Note that sharing ESIs across SiteAccesses is not possible by design (see

[EZP-22535](#) - Cached ESI can not be shared across pages/siteaccesses due to "pathinfo" property

CLOSED

for technical details)

Vary by User

In cases where you need to deliver content uniquely to a given user, and tricks like using javascript and cookie values, `hinclude`, or disabling cache is not an option. Then remaining option is to vary response by cookie:

```
$response->setVary( 'Cookie' );
```

Unfortunately this is not optimal as it will by default vary by all cookies, including those set by add trackers, analytics tools, recommendation services, ... However as long as *your* application backend does not need these cookies, you can solve this by stripping everything but session cookie. Example for Varnish can be found in the default VCL examples in part dealing with User Hash, for single server setup this can easily be accomplished in Apache / Nginx as well.

Http cache clear

5.4

/

2014.11

As of v5.4 / v2014.11, usage of `FOSHttpCacheBundle` has been introduced, impacting the following features:

- Http cache purge
- User context hash

Varnish proxy client from `FOSHttpCache` lib is now used for clearing eZ Http cache, even when using Symfony `HttpCache`. A single `BAN` request is sent to registered purge servers, containing a `X-Location-Id` header. This header contains all Location IDs for which objects in cache need to be cleared.

Symfony reverse proxy

Symfony reverse proxy (aka `HttpCache`) is supported out of the box, all you have to do is to activate it.

Varnish

Please refer to [Using Varnish with eZ Publish Platform 5.4](#)

User context hash

`FOSHttpCacheBundle` *User Context feature* is used is activated by default.

As the response can vary on a request header, the base solution is to make the kernel do a sub-request in order to retrieve the context (aka **user context hash**). Once the *user hash* has been retrieved, it's injected in the original request in the `X-User-Hash` header, making it possible to vary the HTTP response on this header:

Name of the user hash header is configurable in `FOSHttpCacheBundle`. By default eZ Publish sets it to `**X-User-Hash**`.

```
<?php
use Symfony\Component\HttpFoundation\Response;

// ...

// Inside a controller action
$response = new Response();
$response->setVary( 'X-User-Hash' );
```

This solution is implemented in [Symfony reverse proxy \(aka *HttpCache*\)](#) and is also accessible to dedicated reverse proxies like Varnish.

Workflow

Please refer to [FOSHttpCacheBundle documentation on how user context feature works](#).

User hash generation

Please refer to [FOSHttpCacheBundle documentation on how user hashes are being generated](#).

eZ Publish already interferes in the hash generation process, by adding current user permissions and limitations. One can also interfere in this process by [implementing custom context provider\(s\)](#).

User hash generation with Varnish 3

Described behavior comes out of the box with Symfony reverse proxy, but it's of course possible to use Varnish to achieve the same.

```
# Varnish 3 style for eZ Publish 5.4 / 2014.11
# Our Backend - We assume that eZ Publish Web server listen on port 80 on the same
machine.
backend ezpublish {
    .host = "127.0.0.1";
    .port = "80";
}

# Called at the beginning of a request, after the complete request has been received
sub vcl_recv {

    # Set the backend
    set req.backend = ezpublish;

    # ...

    # Retrieve client user hash and add it to the forwarded request.
    call ez_user_hash;

    # If it passes all these tests, do a lookup anyway;
    return (lookup);
}

# Sub-routine to get client user hash, for context-aware HTTP cache.
# Don't forget to correctly set the backend host for the Curl sub-request.
sub ez_user_hash {

    # Prevent tampering attacks on the hash mechanism
    if (req.restarts == 0
        && (req.http.accept ~ "application/vnd.fos.user-context-hash"
            || req.http.x-user-context-hash
        )
    ) {
        error 400;
    }
}
```

```

    if (req.restarts == 0 && (req.request == "GET" || req.request == "HEAD")) {
        # Get User (Context) hash, for varying cache by what user has access to.
        # https://doc.ez.no/display/EZP/Context+aware+HTTP+cach

        # Anonymous user w/o session => Use hardcoded anonymous hash to avoid backend
lookup for hash
        if (req.http.Cookie !~ "eZSESSID" && !req.http.authorization) {
            # You may update this hash with the actual one for anonymous user
            # to get a better cache hit ratio across anonymous users.
            # Note: Then needs update every time anonymous user role assignments
change.
            set req.http.X-User-Hash = "38015b703d82206ebc01d17a39c727e5";
        }
        # Pre-authenticate request to get shared cache, even when authenticated
        else {
            set req.http.x-fos-original-url      = req.url;
            set req.http.x-fos-original-accept  = req.http.accept;
            set req.http.x-fos-original-cookie  = req.http.cookie;
            # Clean up cookie for the hash request to only keep session cookie, as
hash cache will vary on cookie.
            set req.http.cookie = ";" + req.http.cookie;
            set req.http.cookie = regsuball(req.http.cookie, ";" + ", ";"");
            set req.http.cookie = regsuball(req.http.cookie, "(eZSESSID[^=]*)=", "\1=");

            set req.http.cookie = regsuball(req.http.cookie, "[^ ][^;]*", "");
            set req.http.cookie = regsuball(req.http.cookie, "^[; ]+|[[; ]+$", "");

            set req.http.accept = "application/vnd.fos.user-context-hash";
            set req.url = "/_fos_user_context_hash";

            # Force the lookup, the backend must tell not to cache or vary on all
            # headers that are used to build the hash.

            return (lookup);
        }
    }

    # Rebuild the original request which now has the hash.
    if (req.restarts > 0
        && req.http.accept == "application/vnd.fos.user-context-hash"
    ) {
        set req.url      = req.http.x-fos-original-url;
        set req.http.accept = req.http.x-fos-original-accept;
        set req.http.cookie = req.http.x-fos-original-cookie;

        unset req.http.x-fos-original-url;
        unset req.http.x-fos-original-accept;
        unset req.http.x-fos-original-cookie;

        # Force the lookup, the backend must tell not to cache or vary on the
        # user hash to properly separate cached data.

        return (lookup);
    }
}

sub vcl_fetch {

```

```
# ...

if (req.restarts == 0
    && req.http.accept ~ "application/vnd.fos.user-context-hash"
    && beresp.status >= 500
) {
    error 503 "Hash error";
}

}

sub vcl_deliver {
    # On receiving the hash response, copy the hash header to the original
    # request and restart.
    if (req.restarts == 0
        && resp.http.content-type ~ "application/vnd.fos.user-context-hash"
        && resp.status == 200
    ) {
        set req.http.x-user-hash = resp.http.x-user-hash;

        return (restart);
    }

    # If we get here, this is a real response that gets sent to the client.

    # Remove the vary on context user hash, this is nothing public. Keep all
    # other vary headers.
    set resp.http.Vary = regsub(resp.http.Vary, "(?i),? *x-user-hash *", "");
    set resp.http.Vary = regsub(resp.http.Vary, "^, *", "");
    if (resp.http.Vary == "") {
        remove resp.http.Vary;
    }

    # Sanity check to prevent ever exposing the hash to a client.
```

```
    remove resp.http.x-user-hash;
}
```

User hash generation with Varnish 4

```
// Varnish 4 style - eZ 5.4+ / 2014.09+
// Complete VCL example

vcl 4.0;

// Our Backend - Assuming that web server is listening on port 80
// Replace the host to fit your setup
backend ezpublish {
    .host = "127.0.0.1";
    .port = "80";
}

// Called at the beginning of a request, after the complete request has been received
sub vcl_recv {

    // Set the backend
    set req.backend_hint = ezpublish;

    // ...

    // Retrieve client user hash and add it to the forwarded request.
    call ez_user_hash;

    // If it passes all these tests, do a lookup anyway.
    return (hash);
}

// Called when the requested object has been retrieved from the backend
sub vcl_backend_response {
    if (bereq.http.accept ~ "application/vnd.fos.user-context-hash"
        && beresp.status >= 500
    ) {
        return (abandon);
    }

    // ...
}

// Sub-routine to get client user hash, for context-aware HTTP cache.
sub ez_user_hash {

    // Prevent tampering attacks on the hash mechanism
    if (req.restarts == 0
        && (req.http.accept ~ "application/vnd.fos.user-context-hash"
            || req.http.x-user-hash
        )
    ) {
        return (synth(400));
    }

    if (req.restarts == 0 && (req.method == "GET" || req.method == "HEAD")) {
```

```

// Get User (Context) hash, for varying cache by what user has access to.
// https://doc.ez.no/display/EZP/Context+aware+HTTP+cache

// Anonymous user w/o session => Use hardcoded anonymous hash to avoid backend
lookup for hash
if (req.http.Cookie !~ "eZSESSID" && !req.http.authorization) {
    // You may update this hash with the actual one for anonymous user
    // to get a better cache hit ratio across anonymous users.
    // Note: You should then update it every time anonymous user rights
change.
    set req.http.X-User-Hash = "38015b703d82206ebc01d17a39c727e5";
}
// Pre-authenticate request to get shared cache, even when authenticated
else {
    set req.http.x-fos-original-url      = req.url;
    set req.http.x-fos-original-accept  = req.http.accept;
    set req.http.x-fos-original-cookie  = req.http.cookie;
    // Clean up cookie for the hash request to only keep session cookie, as
hash cache will vary on cookie.
    set req.http.cookie = ";" + req.http.cookie;
    set req.http.cookie = regsuball(req.http.cookie, "; +", ";");
    set req.http.cookie = regsuball(req.http.cookie, "(eZSESSID[^=]*)=", "\1=");

    set req.http.cookie = regsuball(req.http.cookie, "[^ ][^;]*", "");
    set req.http.cookie = regsuball(req.http.cookie, "^[; ]+|[[; ]+$", "");
    set req.http.accept = "application/vnd.fos.user-context-hash";
    set req.url = "/_fos_user_context_hash";

    // Force the lookup, the backend must tell how to cache/vary response
containing the user hash
    return (hash);
}
}

// Rebuild the original request which now has the hash.
if (req.restarts > 0
    && req.http.accept == "application/vnd.fos.user-context-hash"
) {
    set req.url      = req.http.x-fos-original-url;
    set req.http.accept = req.http.x-fos-original-accept;
    set req.http.cookie = req.http.x-fos-original-cookie;
    unset req.http.x-fos-original-url;
    unset req.http.x-fos-original-accept;
    unset req.http.x-fos-original-cookie;

    // Force the lookup, the backend must tell not to cache or vary on the
// user hash to properly separate cached data.

    return (hash);
}
}

sub vcl_deliver {
    // On receiving the hash response, copy the hash header to the original
// request and restart.
    if (req.restarts == 0
        && resp.http.content-type ~ "application/vnd.fos.user-context-hash"
    ) {
        set req.http.x-user-hash = resp.http.x-user-hash;
    }
}

```

```
    return (restart);
}

// If we get here, this is a real response that gets sent to the client.
// Remove the vary on context user hash, this is nothing public. Keep all
// other vary headers.
set resp.http.Vary = regsub(resp.http.Vary, "(?i),? *x-user-hash *", "");
set resp.http.Vary = regsub(resp.http.Vary, "^, *", "");
if (resp.http.Vary == "") {
    unset resp.http.Vary;
}

// Sanity check to prevent ever exposing the hash to a client.
unset resp.http.x-user-hash;
if (client.ip ~ debuggers) {
    if (obj.hits > 0) {
        set resp.http.X-Cache = "HIT";
        set resp.http.X-Cache-Hits = obj.hits;
    } else {
        set resp.http.X-Cache = "MISS";
    }
}
```

```
}  
}  
}
```

New anonymous X-User-Hash

The anonymous X-User-Hash is generated based on the *anonymous user, group and role*. The `38015b703d82206ebc01d17a39c727e5` hash that is provided in the code above will work only when these three variables are left unchanged. Once you change the default permissions and settings, the X-User-Hash will change and Varnish won't be able to effectively handle cache anymore.

In that case you need to find out the new anonymous X-User-Hash and change the VCL accordingly, else Varnish will return a no-cache header.

The easiest way to find the new hash is:

1. Connect to your server (*ssh* should be enough)
2. Add `<your-domain.com>` to your `/etc/hosts` file
3. Execute the following command:

```
curl -I -H "Accept: application/vnd.fos.user-context-hash" http://<your-domain.com>/_fos_user_context_hash
```

You should get a result like this:

```
HTTP/1.1 200 OK  
Date: Mon, 03 Oct 2016 15:34:08 GMT  
Server: Apache/2.4.18 (Ubuntu)  
X-Powered-By: PHP/7.0.8-0ubuntu0.16.04.2  
X-User-Hash: b1731d46b0e7a375a5b024e950fdb8d49dd25af85a5c7dd5116ad2a18cda82cb  
Cache-Control: max-age=600, public  
Vary: Cookie,Authorization  
Content-Type: application/vnd.fos.user-context-hash
```

4. Now, replace the existing X-User-Hash value with the new one:

```
# Note: This needs update every time anonymous user role assignments change.  
set req.http.X-User-Hash =  
"b1731d46b0e7a375a5b024e950fdb8d49dd25af85a5c7dd5116ad2a18cda82cb";
```

5. Restart the Varnish server and everything should work fine.

Default options for FOSHttpCacheBundle defined in eZ

The following configuration is defined in eZ by default for FOSHttpCacheBundle. You may override these settings.

```
fos_http_cache:
  proxy_client:
    # "varnish" is used, even when using Symfony HttpCache.
    default: varnish
    varnish:
      # Means http_cache.purge_servers defined for current SiteAccess.
      servers: [${http_cache.purge_servers$}]

  user_context:
    enabled: true
    # User context hash is cached during 10min
    hash_cache_ttl: 600
    user_hash_header: X-User-Hash
```

Credits

This feature is based on [Context aware HTTP caching post](#) by asm89.