Using Varnish

eZ Publish 5 being built on top of Symfony 2, it uses standard HTTP cache headers. By default the Symfony 2 reverse proxy, written in PHP, is used to handle cache, but it can be easily replaced with any other reverse proxy like Varnish.

Configure Varnish

> Provided VCL example is given for **Varnish 3.x only**.
>
> It needs Varnish Curl Vmod to be installed.

The first thing is to configure Varnish to advertise ESI usage and to activate cache invalidation and Context aware HTTP cache.

ezpublish5.vcl

```
# Varnish 3.x - eZ Publish 5 - Complete VCL

import curl;

# Our Backend - We assume that eZ Publish Web server listens on port 80
backend ezpublish {
    .host = "127.0.0.1";
    .port = "80";
}

# ACL for purgers - based on IP.
# Provide here IP addresses that are allowed to send PURGE requests.
# PURGE requests will be sent by the backend. It is not a good idea to allow end-users
to invalidate caches.
acl purgers {
    "127.0.0.1";
    "192.168.0.0"/16;
}

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

    # Set the backend
    set req.backend = ezpublish;

    # Advertise ESI support to Symfony
    set req.http.Surrogate-Capability = "abc=ESI/1.0";

    # Add a unique header containing the client address (only for master request)
    # Please note that /_fragment URI can change in Symfony configuration
    if (!req.url ~ "^/_fragment") {
        if (req.http.x-forwarded-for) {
            set req.http.X-Forwarded-For = req.http.X-Forwarded-For + ", " +
client.ip;
        } else {
            set req.http.X-Forwarded-For = client.ip;
        }
    }

    # Trigger purge if needed.
    call ez_purge;
```

```
    # Normalize the Accept-Encoding headers
    if (req.http.Accept-Encoding) {
        if (req.http.Accept-Encoding ~ "gzip") {
            set req.http.Accept-Encoding = "gzip";
        } elsif (req.http.Accept-Encoding ~ "deflate") {
            set req.http.Accept-Encoding = "deflate";
        } else {
            unset req.http.Accept-Encoding;
        }
    }

    # Don't cache Authenticate & Authorization
    if (req.http.Authenticate || req.http.Authorization) {
        return(pass);
    }

    # Don't cache requests other than GET and HEAD.
    if (req.request != "GET" && req.request != "HEAD") {
        return (pass);
    }

    # Do a standard lookup on assets
    # Note that file extension list below is not extensive, so consider completing it
to fit your needs.
    if (req.request == "GET" && req.url ~
"\.(css|js|gif|jpe?g|bmp|png|tiff?|ico|img|tga|wmf|svg|swf|ico|mp3|mp4|m4a|ogg|mov|avi
|wmv|zip|gz|pdf|ttf|eot|wof)$") {
        return (lookup);
    }

    # 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);
}

# Called when the requested object has been retrieved from the backend
sub vcl_fetch {

    # Optimize to only parse the Response contents from Symfony
    if (beresp.http.Surrogate-Control ~ "ESI/1.0") {
        unset beresp.http.Surrogate-Control;
        set beresp.do_esi = true;
    }

    # Don't cache response with Set-Cookie
    if ( beresp.http.Set-Cookie ) {
        set beresp.ttl = 0s;
        return (hit_for_pass);
    }

    # Respect the Cache-Control=private header from the backend
    if (beresp.http.Cache-Control ~ "private") {
        set beresp.ttl = 0s;
        return (hit_for_pass);
    }
```

```
    return (deliver);
}

# Handle purge
# Only works with "multiple_http" purge method
sub ez_purge {

    # Handle purge
    # Only works with "multiple_http" purge method
    if (req.request == "PURGE") {
        if (!client.ip ~ purgers) {
            error 405 "Method not allowed";
        }

        if ( req.http.X-Location-Id == "*" ) {
            # Purge all locations
            ban( "obj.http.X-Location-Id ~ ^[0-9]+$" );
            error 200 "Purge all locations done.";
        } elseif ( req.http.X-Location-Id ) {
            # Purge location by its locationId
            ban( "obj.http.X-Location-Id == " + req.http.X-Location-Id );
            error 200 "Purge of content connected to the location id(" +
req.http.X-Location-Id + ") done.";
        }
    }
}

# 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 {

    if (req.request == "GET") {
        # Pre-authenticate request to get shared cache, even when authenticated
        if (req.http.Cookie !~ "is_logged_in=" ) {
            # User doesn't have "is_logged_in" cookie => Set a hardcoded anonymous
hash
            set req.http.X-User-Hash = "38015b703d82206ebc01d17a39c727e5";
        } else {
            # User is authenticated => fetch user hash
            curl.header_add("X-HTTP-Override: AUTHENTICATE");
            curl.header_add("Accept: application/vnd.ez.UserHash+text");
            curl.header_add("Cookie: " + req.http.Cookie);
            # Customize with real backend host
            # E.g. curl.get("http://www.metalfrance.net");
            curl.get("http://" + req.http.host + "/");
            if (curl.status() == 200) {
                set req.http.X-User-Hash = curl.header("X-User-Hash");
            }
        }
    }
```

```
}
```

You can of course add additional rules if you need.

Configure eZ Publish

Update your Virtual Host

> **New in eZ Publish 5.2 / 2013.07**
> The front controller can now be altered with environment variables, without the need to modify `index.php`.

By default your front controller (`index.php`) uses the built-in reverse proxy, `EzPublishCache`. In order to use Varnish, you need to deactivate it by commenting the line where `EzPublishCache` is instantiated:

my_virtualhost.conf

```
<VirthualHost *:80>
    # Configure your VirtualHost with rewrite rules and stuff

    # Force front controller NOT to use built-in reverse proxy.
    SetEnv USE_HTTP_CACHE 0
</VirtualHost>
```

Update YML configuration

ezpublish.yml

```
ezpublish:
    http_cache:
        # Use multiple_http method for purging content
        purge_type: multiple_http

    system:
        # Assuming that my_siteaccess_group your frontend AND backend siteaccesses
        my_siteaccess_group:
            http_cache:
                # Fill in your Varnish server(s) address(es).
                purge_servers: [http://my.varnish.server:6081]
```

Et voilà !