Clustering

Introduction

Clustering in eZ Platform refers to setting up your install with several web servers for handling more load and/or for failover.

Server setup overview

This diagram illustrates how clustering of eZ Platform is typically set up, the parts illustrate the different roles needed for a successful cluster setup. The number of web servers, Memcached servers, Solr servers, Varnish servers, Database servers, NFS servers, as well as whether some servers play several of these roles *(typically running Memcached across the web server)* is up to you and your performance needs.

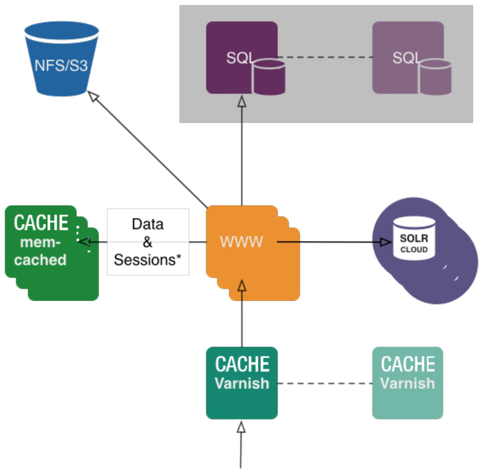
The minimal requirements are the following *(with what is currently supported in italics)*:

- Shared HTTP cache *(using Varnish)*
- Shared Persistence cache and Sessions *(using Memcached, or experimentally also Redis)*
- Shared Database *(using MySQL/MariaDB)*
- Shared Filesystem *(using NFS, or experimentally also S3)*

For further details on requirements, see Requirements page .

While this is not a complete list, further recommendations include:

- Using Solr for better search and better search performance
- Using a CDN for improved performance and faster ping time worldwide
- Using Active/Passive Database for failover
- In general: Make sure to use later versions of PHP and MySQL/MariaDB within what is supported for your eZ Platform version to get more performance out of each server.



Binary files clustering

eZ Platform supports multi-server by means of custom IO handlers. They will make sure that files are correctly synchronized among the multiple clients that might use the data.

In this topic:

Related:

- Overview of steps to set up Cluster:
    - Configure Persistence Cache
    - Set up Varnish
    - Configure sessions
- Performance

Configuration

Memcached must not be bound to the local address if clusters are in use, or user logins will fail. To avoid this, in `/etc/memcached.conf` take a look under `# Specify which IP address to listen on. The default is to listen on all IP addresses`

For development environments, change the address below to `-l 0.0.0.0`

For production environments, follow this more secure instruction from the memcached man:

> *-l <addr>*
> *Listen on <addr>; default to INADDR_ANY. <addr> may be specified as host:port. If you don't specify a port number, the value you specified with -p or -U is used. You may specify multiple addresses separated by comma or by using -l multiple times. This is an important option to consider as there is no other way to secure the installation. Binding to an internal or firewalled network interface is suggested.*

DFS IO Handler

What it is meant for

The DFS IO handler (`legacy_dfs_cluster`) can be used to store binary files on an NFS server. It will use a database to manipulate metadata, making up for the potential inconsistency of network based filesystems.

Configuration

You need to configure both metadata and binarydata handlers.

As the binarydata handler, create a new Flysystem local adapter configured to read/write to the NFS mount point on each local server. As metadata handler handler, create a dfs one, configured with a doctrine connection.

V1.8.0

Note: the default database install will now include the dfs table *in the same database*

For production, we strongly recommend creating the DFS table in its own database, using the `vendor/ezsystems/ezpublish-kernel/data/mysql/dfs_schema.sql` file.
In our example, we will use one named `dfs`.

```
# new doctrine connection for the dfs legacy_dfs_cluster
metadata handler.
doctrine:
    dbal:
        connections:
            dfs:
                driver: pdo_mysql
                host: 127.0.0.1
                port: 3306
                dbname: ezdfs
                user: root
                password: "rootpassword"
                charset: UTF8

# define the flysystem handler
oneup_flysystem:
    adapters:
        nfs_adapter:
            local:
                directory: "/<path to
nfs>/$var_dir$/$storage_dir$"

# define the ez handlers
ez_io:
    binarydata_handlers:
        nfs:
            flysystem:
                adapter: nfs_adapter
    metadata_handlers:
        dfs:
            legacy_dfs_cluster:
                connection: doctrine.dbal.dfs_connection

# set the application handlers
ezpublish:
    system:
        default:
            io:
                metadata_handler: dfs
                binarydata_handler: nfs
```

Customizing the storage directory

eZ Publish 5.x required the NFS adapter directory to be set to `$var_dir$/$storage_dir$` part for the NFS path. This is no longer required with eZ Platform, but the default prefix used to serve binary files will still match this expectation.

If you decide to change this setting, make sure you also set `io.url_prefix` to a matching value. If you set the NFS adapter's directory to "/path/to/nfs/storage", use this configuration so that the files can be served by Symfony:

```
ezpublish:
 system:
  default:
   io:
    url_prefix: "storage"
```

As an alternative, you may serve images from NFS using a dedicated web server. If in the example above, this server listens on http://static.example.com and uses `/path/to/nfs/storage` as the document root, configure `io.url_prefix` as follows:

```
ezpublish:
 system:
  default:
   io:
    url_prefix: "http://static.example.com/"
```

You can read more about that on Binary files URL handling.

Web server rewrite rules

The default eZ Platform rewrite rules will let image requests be served directly from disk. With native support, files matching `^/var/([^/]+/)?storage/images(-versioned)?/.*` have to be passed through `/web/app.php`.

In any case, this specific rewrite rule must be placed without the ones that "ignore" image files and just let the web server serve the files.

Apache

```
RewriteRule
^/var/([^/]+/)?storage/images(-versioned)?/.* /app.php
[L]
```

nginx

```
rewrite
"^/var/([^/]+/)?storage/images(-versioned)?/(.*)"
"/app.php" break;
```