eZ Publish cluster unleashed

Using the eZ Publish cluster in your projects

Bertrand Dunogier, eZ Systems

eZ®
The information sharing company

Goal and history

One eZ Publish on multiple servers, for:

- Performance
- Redundancy

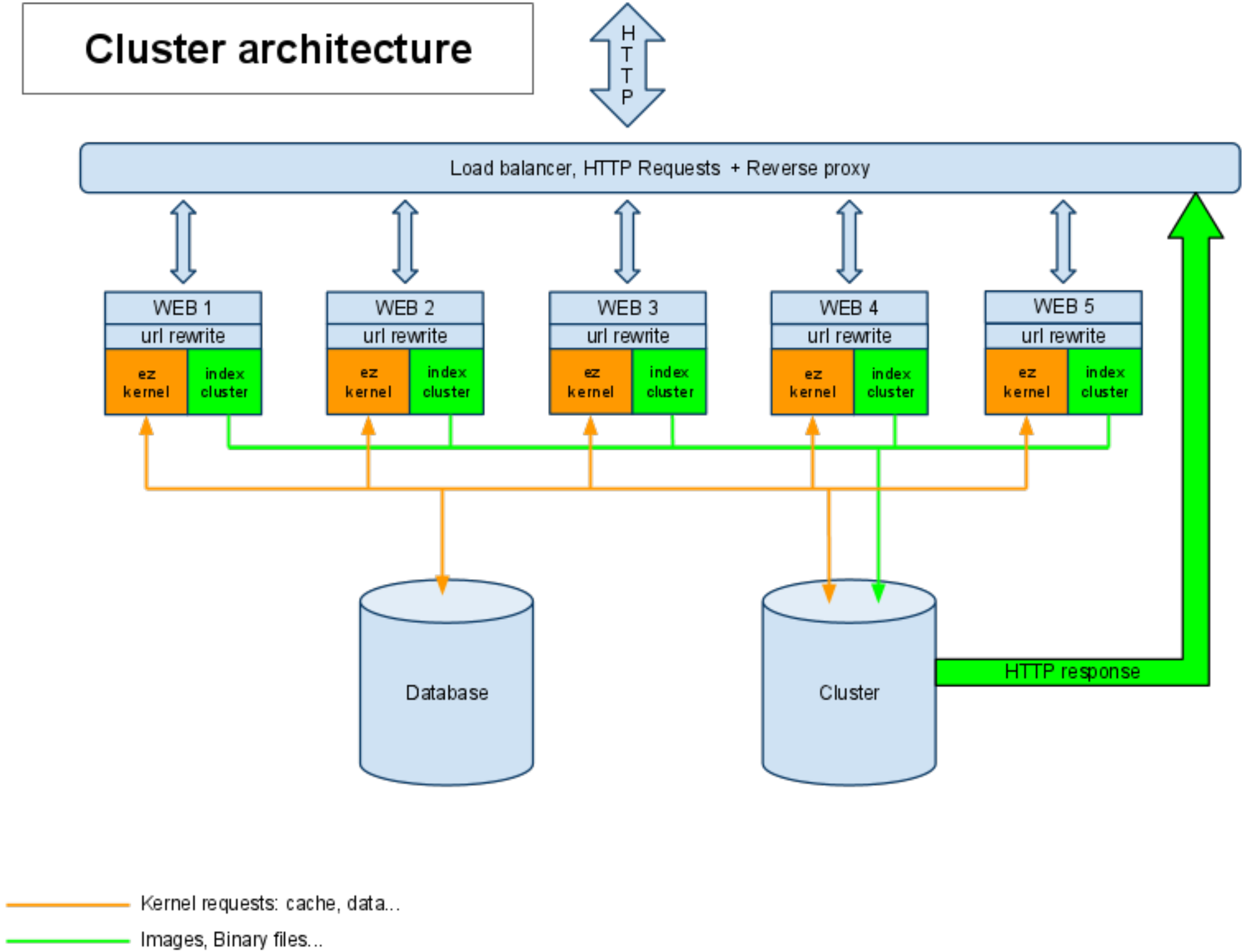
Created for eZ Publish 3.8, vastly improved since then:

- 3.10: huge schema & feature improvements
- 4.2: new eZDFS cluster handler for NFS
- 4.2: Stalecaching mechanism for eZDB and eZDFS

Matches the shared database for filesystem operations:

- Inserting new data, expiring/deleting obsolete data
- serving files through HTTP

Cluster architecture



Cluster handlers

eZ DB	eZ DFS
From eZ Publish 3.8	From eZ Publish 4.2
Stores metadata in DB	Stores metadata in DB
Stores data in DB (BLOBs)	Stores data on NFS
Easy to setup	Requires an NFS server
Huge DB, harder to maintain	Maintenance is much easier

HTTP vs Internal calls

HTTP

- Requests done through index_cluster.php
- Serves files directly
- A reverse proxy caches images & binary files

INTERNAL / KERNEL CALLS

- Reads/writes cache items
- Stores binary files

Configuration #1: file.ini

A cluster handler must be configured in a file.ini override:

```
1. [ClusteringSettings]
2. FileHandler=eZDFSFileHandler
3.
4. # DFS Cluster Handler settings
5. [eZDFSClusteringSettings]
6. MountPointPath=var/nfs
7. DBBackend=eZDFSFileHandlerMySQLBackend
8. DBHost=localhost
9. DBName=labs_ezpublish_trunk_cluster
10.DBUser=root
11.DBPassword=root
```

Configuration #2: database

- eZDB and eZDFS both require database tables
- The same DB server can be used
- Performance wise it is better to use another server
- Schemas are available in the cluster driver PHP files

Configuration #3: clusterize

- Local files must be moved to the cluster
- This is done using bin/php/clusterize.php
- This script will copy/move images & binary files from the local filesystem to the cluster
- It must only be executed once for the whole cluster

Configuration #4: mod_rewrite

- URL rewriting is required to "stream" clusterized binary files to visitors:
  - Images
  - Binary files

- They only affect known path for these types of files

- To stream different files, new rules must be added

```
RewriteEngine On
Rewriterule ^/var/([^/]+/)?storage/original/.* /index_cluster.php [L]
Rewriterule ^/var/([^/]+/)?storage/images/.*   /index_cluster.php [L]

# Standard eZ Publish rewrite rules
# ...
```

Configuration #5: cluster index

The cluster index handles binary files HTTP requests

- It doesn't use the whole kernel in order to perform better

- One custom file is edited, and this file uses the default ones:
    1. index_cluster.php (custom)
    2. index_image.php (default)
    3. index_image_<clusterhandler>.php (default)

- Values must of course match file.ini's

Configuration #5: cluster index

Index example:

```
1. define( 'STORAGE_BACKEND',     'dfsmysql'         );
2. define( 'STORAGE_HOST',        'localhost'        );
3. define( 'STORAGE_PORT',        3306               );
4. define( 'STORAGE_USER',        'root'             );
5. define( 'STORAGE_PASS',        'root'             );
6. define( 'STORAGE_DB',          'ezpublish_dfs_cluster' );
7. define( 'MOUNT_POINT_PATH',    'var/nfs'          );
8.
9. include_once( 'index_image.php' );
```

How cache is handled

- Remote, clusterized files are copied locally
- Before using a local file, eZ always checks if it is newer than the remote one
- When a cache file is generated, it is stored on the cluster, and re-used by other nodes
- expiry.php provides an intermediate method

API: reading clusterized files

```
$path = '/var/ezwebin_site/path/to/my/file.txt';
```

```
// read a text file, "instance" syntax
$contents = eZClusterFileHandler::instance( $path )->fetchContents();

// read a text file, "static" syntax
$contents = eZClusterFileHandler::instance()->fileFetchContents( $path );

// fetch a file (a binary one for example) to disk
$path = 'var/ezflow_site/storage/original/video/961d8a65efffdd93708cc23bc6398953.flv';
$handler = eZClusterFileHandler::instance( $path )->fetchUnique();

// ... use the file ... then delete it
$handler->deleteLocal( $uniquePath );

// reading metadata
$file = eZClusterFileHandler::instance( $someFile );
echo $file->size();
echo $file->mtime();
```

API: writing a file to the cluster

Again, the native PHP I/O API isn't cluster aware !

```
1. // stores the contents of the $contents variable to cluster
2. $path = 'var/ezwebin_site/file.txt';
3. $handler = eZClusterFileHandler::instance( $path );
4. $handler->storeContents( $contents, 'text', 'text/plain' );
5. // alternative "static" method: fileStoreContents()
6.
7. // stores the contents of a local image as a clusterized file
8. $imagePath = 'var/ezwebin_site/path/image.png';
9. $handler = eZClusterFileHandler::instance();
10.$handler->fileStore( $imagePath, 'image', true, 'image/png' );
```

API: view cache API !

- ViewCache uses advanced methods of the cluster API
- These method handle:
  - cache read and/or generation
  - concurrency
  - stale caching
- It can technically be used to manage custom cache !
- Next: introducing the processCache() method

API: The processCache() method

- can be used for custom developements too !
- It will let you implement caching on custom modules/views
- It uses:
  - a generate callback that generates the dynamic content
  - a retrieve callback that retrieves valid cached content

API: The processCache() method

- can be used for custom developements too !
- It will let you implement caching on custom modules/views
- It uses:
  - a generate callback that generates the dynamic content
  - a retrieve callback that retrieves valid cached content

processCache(): view

```
1. $Module = $Params["Module"];
2. $cacheFilePath = eZCachedModule::cacheFilePath( $Params );
3. $cacheFile = eZClusterFileHandler::instance( $cacheFilePath );
4. $Result = $cacheFile->processCache(
5.     array( 'eZCachedModule', 'retrieve' ),
6.     array( 'eZCachedModule', 'generate' ),
7.     null,
8.     null,
9.     compact( 'Params' ) );

1. return $Result;
```

processCache(): handler

```
1.  class eZCachedModule
2.  {
3.      public static function generate( $file, $args )
4.      {
5.          extract( $args );
6.
7.          $tpl = templateInit();
8.          $tpl->setVariable( 'value', $Params['Value'] );

1.          $Result['content'] = $tpl->fetch( "design:cachedmodule/view.tpl" );
2.          $Result['path'] = array( array( 'url' => '/cachedmodule/view/',
3.                                          'text' => 'Cached module testing' ) );
4.          $retval = array( 'content' => $Result,
5.                           'scope'   => 'cachedmodule' );
6.          return $retval;
7.      }

1.      public static function retrieve( $file, $mtime, $args )
2.      {
3.          return include( $file );
4.      }
5.
6.      public static function cacheFilePath( $params )
7.      {
8.          $currentSiteAccess = $GLOBALS['eZCurrentAccess']['name'];
9.          $cacheFile = $params['Value'] . '.php'
10.         $cachePath = eZDir::path( array( eZSys::cacheDirectory(),
11.             'cachedmodule', $currentSiteAccess, $cacheFile ) );
12.         return $cachePath;
13.     }
14. }
```

Questions ?

Thank you for listening

Documentation

http://ez.no/doc/ez_publish/technical_manual/4_x/features/clustering

Contact

bd@ez.no
share.ez.no ;)