DotNetNuke Web Farm Support

Dan Caron

Version 1.0.0
Last Updated: June 20, 2006
Category: Configuration

DotNetNuke Web Farm Support

Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results of the use of this document remains with the user.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.

DOTNETNUKE™

DotNetNuke Web Farm Support

Abstract

In order to clarify the intellectual property license granted with contributions of software from any person or entity (the "Contributor"), Perpetual Motion Interactive Systems Inc. must have a Contributor License Agreement on file that has been signed by the Contributor.

Contents

DotNetNuke Web Farm Support 1

Introduction 1

Supported Configuration 1

Alternative Configuration 3

Components Affected by Web Farm Support 4

How to Enable Web Farm Support 7

Additional Information 11

Appendix A: Document History 12

DotNetNuke Web Farm Support

Introduction

In order to deliver the server uptime that is required by many client service level agreements, hosting providers and enterprises often architect their environment to utilize a web farm configuration. The redundancy offered by a web farm allows them to take servers offline for patching and upgrades without affecting the uptime of their clients. In version 3.1, DotNetNuke introduced support for web farm configurations.

An important item to note in a shared hosting environment is the high density of sites on single web server. It is expected that there may be 1-2000 individual sites running the DotNetNuke application on a single web server. This factor has significant implications on the web farm support strategy.

Supported Configuration

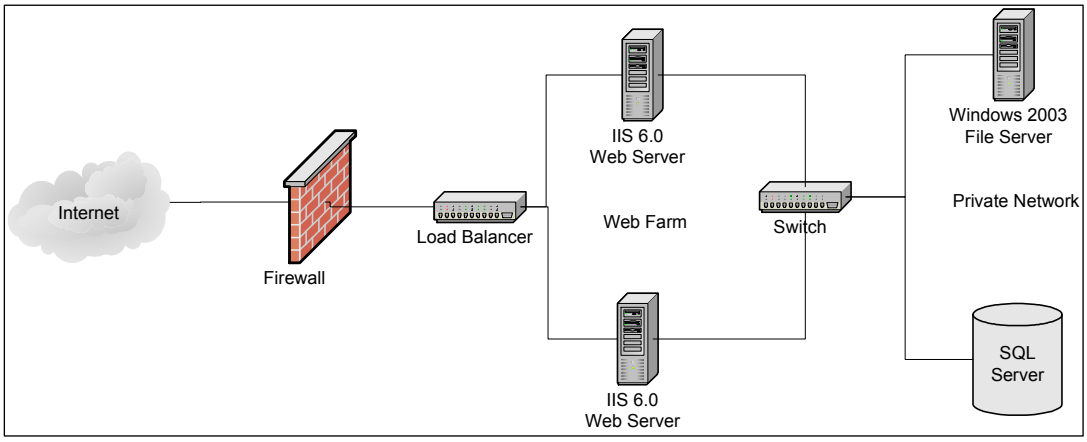
The web farm configuration that DotNetNuke initially supports involves two or more front end web servers ("web-heads") whose IIS website root directories are mapped to a common UNC share on a remote file server. The UNC share contains the application source code as well as any static content for the individual sites. Each web server connects to a remote database server for dynamic application content. This configuration allows any of the web servers to be removed from the farm without interrupting service for the web users browsing the sites.

Helpful details on the supported configuration can be found at the following URL:

http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/webapp/iis/remstorg.mspx

This URL includes details on how to configure and optimize your web servers and file server for the best performance.

Fig. 1 - Component Relationship Diagram for Supported Configuration



Alternative Configuration

The primary supported method above will provide the best reliability in terms of keeping files in synch among the machines in the farm. However, in some alternative configurations the servers may be configured to use file replication services. In this case, there is no central file server. Instead, each web server has its own IIS directory mapped to a local folder. This configuration requires a slightly different setup in DotNetNuke that is detailed below.

Note about file replication services: Keeping the files in synch between multiple web servers can be very challenging in situations where the hosted web sites engage in heavy file-changing activity. If you must use file replication services, consult with your file replication services vendor to ensure it will meet your requirements for web server synchronization.

Components Affected by Web Farm Support

The following core areas have been enhanced to enable web farm support:

Caching

The DotNetNuke core framework provides an API for handling items in the cache. There are core methods exposed for adding, removing and getting items from the cache. We have modified these methods to encapsulate any changes into the API that module developers already use. The end result should be that any 3rd party modules that use the core API for managing the cache will automatically support web farm cache synchronization.

The Caching API (DotNetNuke.Common.Utilities.DataCache) has been enhanced to use the provider model design pattern. By following this design pattern, DotNetNuke can accommodate a wide array of web farm configurations by allowing the caching mechanism to be easily replaced.

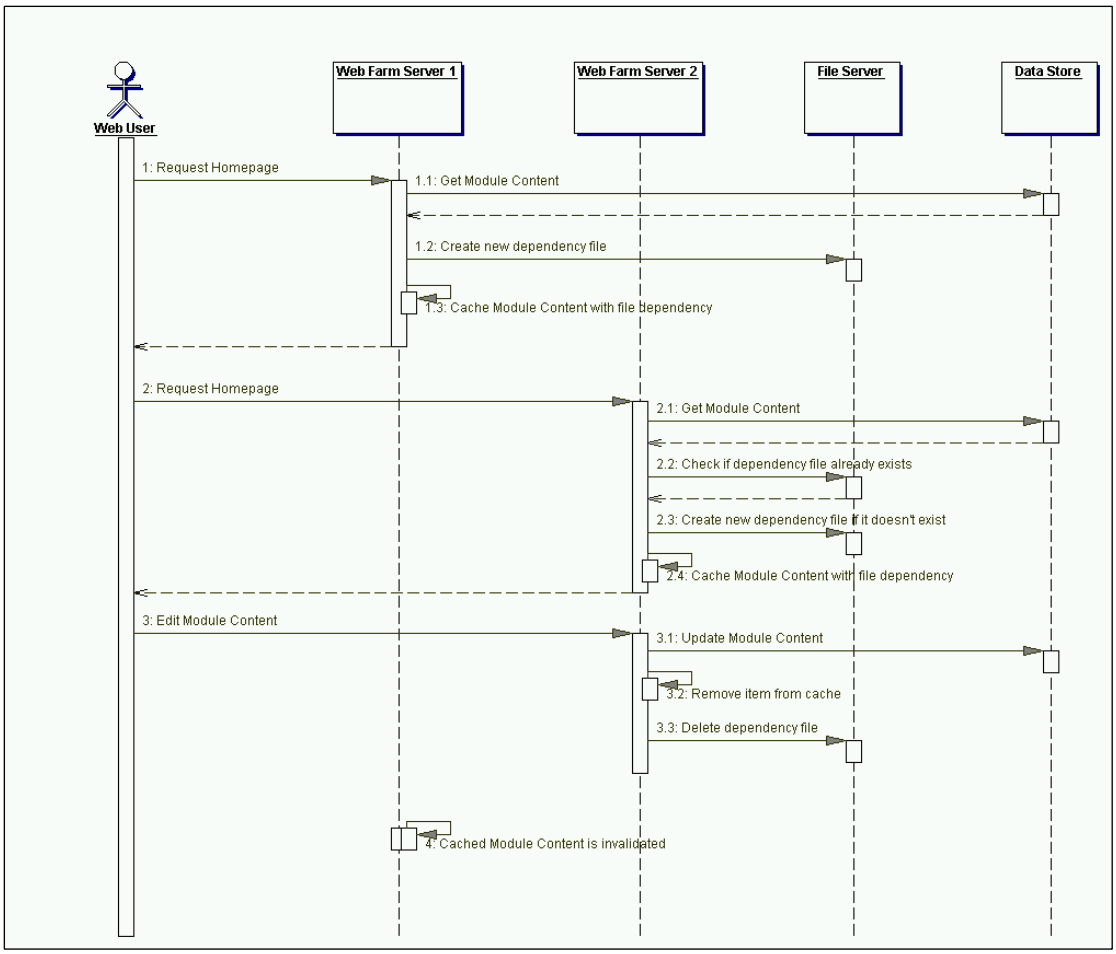
There are two implementations of the provider that are included with the default installation:

- *1. File-Dependency Based Caching Provider*. In this provider implementation, if web farm support is enabled, just before an item is added to the cache, a file is created in the "/Portals/_default/Cache" directory. That file represents a file dependency for the specific cached item (the filename is a Base64 encoded string [minus the "/" character] that is derived from the cache key). The filename needs to be encoded in case the cache key is of a different character set that is incompatible with file naming conventions.

  When another server adds the same item to the cache, it checks to see if the dependency file has already been created. If it has not been created, it creates the dependency file. It then adds the item to the cache with a file dependency on the file that was created. When any web server removes the item in the cache, the file is removed as well. When the dependency file is removed, all web servers with the same item in the cache will invalidate those cached items and remove them from the cache. A scheduled task cleans up the cache directory periodically (default is every 2 hours), deleting any cache dependency files that have no keys in memory any longer (on the machine that the scheduled task is currently running on) and were created

more than "x" hours ago (perhaps 2 hours). This prevents Server A from deleting a recently cached item that is being used by Server B.

Fig. 2 - File Dependency Cache Invalidation - Sequence Diagram



- *2. Broadcast/Polling Based Caching Provider*. This implementation of the provider uses a database to broadcast messages to the other servers. Each server in the web farm polls the database every 30 seconds to check for any cached items that need to be removed from the cache.

This implementation of the API is for use in configurations where a central file server is not used (i.e. using file replication services instead) or in configurations

where “more database activity” is preferred over “more file input/output activity”. Note, enabling this implementation of the caching provider in a dense hosting environment may cause heavy of database activity. Keep this in mind when choosing the implementation of the caching provider.

Scheduler

One essential core component of DotNetNuke is the task scheduler. The scheduler runs tasks on the web server regularly which help to maintain a DotNetNuke instance. Scheduled tasks include the Search Indexer, Purge Users Online, Event Log Buffer Purge, Site Log Purge, Resource Installer, and File System Synchronizer. In a web farm, some of these tasks will need to be run on every web server, while others will only need to be run on a single web server. For instance, since the Search Indexer indexes content in the database, it does not need to run on every web server. Conversely, the Purge Users Online task needs to run on every server because it is responsible for moving data from web server memory to the database.

By default, all enabled scheduled tasks will run all tasks on all servers. The web farm enhancements for the scheduler allow each scheduled task to be assigned to a specific server, or to all servers.

The Scheduler itself, and the associated stored procedures and tables, have been modified to account for a new ServerName setting. When the web server initiates the scheduler to check for tasks to run, it will get all scheduled tasks where the ServerName is the same as the web server’s ServerName, along with all tasks where the ServerName is null (signifying “all” servers).

Logging Provider

Another essential core component of DotNetNuke is the logging provider. The Logging Provider is responsible for logging system events, exceptions and informational messages. The logging solution is implemented using the provider model. The previous default core implementation of the logging API was the XML Logging Provider. This provider uses the file system to store log entries in XML format. This solution works well

in a single server scenario, however there is more opportunity for file locking and contention in a web farm environment.

A new database logging provider ("DBLoggingProvider") implementation has been included and is not the default logging provider. This new implementation of the logging API utilizes a database as a persistence medium. This reduces file locking/contention that can occur in a web farm environment.

How to Enable Web Farm Support

Step 1.

A setting has been added to the web.config file that specifies whether web farm support is enabled or not. The new setting is stored in the "/configuration/appSettings/EnableWebFarmSupport" node. The default value is "false".

Set this value to "true" to enable web farm support.

Step 2.

Next, enable the DBLoggingProvider. Verify the DBLoggingProvideris the default Logging Provider by opening web.config. Look at the "defaultProvider" attribute of this node: "configuration/dotnetnuke/logging". Verify the value of the "defaultProvider" attribute is "DBLoggingProvider".

Save web.config. The logging provider section of web.config should look like this below:

```
<logging defaultProvider="DBLoggingProvider">
    <providers>
        <clear />
        <add name="XMLLoggingProvider"
            type="DotNetNuke.Services.Log.EventL
            configfilename="LogConfig.xml.resour
            providerPath="~\Providers\LoggingPro
        <add name="DBLoggingProvider"
            type="DotNetNuke.Services.Log.EventL
            providerPath="~\Providers\LoggingPro
    </providers>
</logging>
```

Step 3.

Next, the machine keys in web.config must match across all web servers in the farm. Open web.config on one machine and find the following XML node: "configuration/appSettings". There are two application settings defined here for the machine keys:

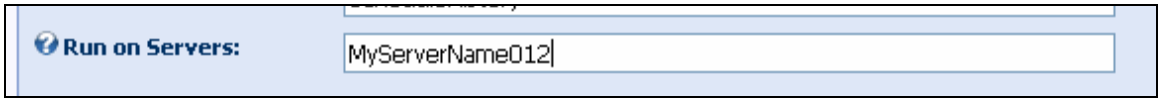
MachineValidationKey – ensure all servers in the farm have the same value for this key
MachineDecryptionKey – ensure all servers in the farm have the same value for this key

Step4.

The scheduler must be configured to run certain tasks on "all" machines, and other tasks on "certain" machines. By default, all enabled scheduled tasks will run on all machines in the web farm.

Log in to your DotNetNuke site as a SuperUser. Go to the Host menu and select "Schedule".

Edit the scheduled task "**DotNetNuke.Services.Scheduling.PurgeScheduleHistory, DOTNETNUKE**" by clicking on the edit icon next to the task. In the textbox labeled "Run on Servers", enter one server name that this task should be run on.



Repeat this step for each of the following scheduled tasks:
-"DotNetNuke.Services.Log.EventLog.SendLogNotifications, DOTNETNUKE"
-"DotNetNuke.Services.Search.SearchEngineScheduler, DOTNETNUKE"
-"DotNetNuke.Modules.Admin.ResourceInstaller.InstallResources, DOTNETNUKE"
-"DotNetNuke.Services.FileSystem.SynchronizeFileSystem, DOTNETNUKE"

Step 5.

Depending on which of the two configurations you are using, the following steps will need to be performed to enable web farm support.

- **Supported Configuration** (using a central file server)

- A scheduled task needs to be enabled. Log in as a SuperUser, go to the Host menu and select "Schedule". Edit the scheduled task "DotNetNuke.Services.Cache.PurgeCache, DOTNETNUKE" by clicking on the edit icon next to the scheduled task. Select the checkbox labeled "Schedule Enabled" so it is checked. Click "Update".
- Verify the FileBasedCachingProvider is the default Caching Provider by opening web.config. Look at the "defaultProvider" attribute of this node: "configuration/dotnetnuke/caching". Verify the value of the "defaultProvider" attribute is "FileBasedCachingProvider".

1. Save web.config. The caching provider section of web.config should look like this below:

```
<caching defaultProvider="FileBasedCachingProvider">
    <providers>
        <clear />
            <add name="FileBasedCachingProvider" type="Do
            <add name="BroadcastPollingCachingProvider" t
    </providers>
</caching>
```

- **Alternative Configuration** (using file replication services)

- Open the following text file in Notepad: "$root\Providers\CachingProviders\BroadcastPollingCachingProvider\Providers\SQLDataProvider\ 03.01.00.SqlDataProvider"
- Highlight the entire contents of the file and copy it to your clipboard
- Log in to your DotNetNuke installation as a SuperUser, go to the Host menu and select "SQL".
- Paste the contents of your clipboard into the textbox on the SQL page.
- Select the "Run as Script" checkbox so it is checked.

- Click “Execute”
- Schedule Enabled” so it is checked. Click “Update”.
- Change the default Caching Provider to “BroadcastPollingCachingProvider” by opening web.config. Look at the “defaultProvider” attribute of this node: ”configuration/dotnetnuke/caching”. Change the value of the “defaultProvider” attribute to “BroadcastPollingCachingProvider”.
- Save web.config. The caching provider section should look like this below:

```
<caching defaultProvider="BroadcastPollingCachingProvider">
    <providers>
        <clear />
            <add name="FileBasedCachingProvider" type="DotNetN
            <add name="BroadcastPollingCachingProvider" type="
    </providers>
</caching>
```

Web Farm Support is now enabled.

Additional Information

The DotNetNuke Portal Application Framework is constantly being revised and improved. To ensure that you have the most recent version of the software and this document, please visit the DotNetNuke website at:

http://www.dotnetnuke.com

The following additional websites provide helpful information about technologies and concepts related to DotNetNuke:

DotNetNuke Community Forums
http://www.dotnetnuke.com/tabid/795/Default.aspx

Microsoft® ASP.Net
http://www.asp.net

Open Source
http://www.opensource.org/

W3C Cascading Style Sheets, level 1
http://www.w3.org/TR/CSS1

Errors and Omissions

If you discover any errors or omissions in this document, please email marketing@dotnetnuke.com. Please provide the title of the document, the page number of the error and the corrected content along with any additional information that will help us in correcting the error.

Appendix A: Document History

Version	Last Update	Author(s)	Changes
1.0.0	Aug 16, 2005	Shaun Walker	• Applied new template