GetCache Server User Guide

version 1.2.0

Introduction

GetCache is an in-memory key-value store developed with .Net 4.5 very simple to use and manage.

GetCache is a distributed in-memory cache that supports data sharding on multiple nodes and data replication. Nodes can be added and removed to the cluster without causing service interruptions.

GetCache stores any type of data, objects can be serialized using JSON or XML, row object like images or videos can be stored as byte arrays. GetCache supports automatic data expiration.

GetCache server installation

Get the software

GetCache can be downloaded from the website www.getcache.net or via NuGet.

Examining the downloaded package we see that it has this folder structure:

- *dist*
    - *client*
    - *config*
    - *deploy*
    - *doc*
    - *log*

The *dist* folder contains the executable and DLL files needed for the program. The folder also includes the configuration file *GetCache.Server.exe.config* which will be described below.

The *config* folder contains the files config.xml and nodelist.xml described below.

The *client* folder contains the client libraries.

The *doc* folder contains the documentation files.

The *log* folder will contains the log file *server.log*.

The *deploy* folder will contains user's deployed modules (i.e. Robots)

Start the server

Run the program *GetCahce.Server.exe* as administrator.
Check the *server.log* file under *log* folder. The log will show messages similar to these:

```
2013-08-09 11:06:12,623 INFO  GetCache.Server.Node.ServerNode - Node Hydra-6 started.
2013-08-09 11:06:12,641 INFO  GetCache.Server.Node.Manager.NodeManager - Current node
registered [ Name = Hydra-6; NodeGuid = 568c1e3b-5de9-48cb-910d-f953490f1a3f; Host =
localhost; ComPort = 1010; ShardingIndex = [ 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12;
13; 14; 15; 16; 17; 18; 19; 20; 21; 22; 23; 24; 25; 26; 27; 28; 29; 30; 31; 32; 33; 34;
35; 36; 37; 38; 39; 40; 41; 42; 43; 44; 45; 46; 47; 48; 49; 50; 51; 52; 53; 54; 55; 56;
57; 58; 59; 60; 61; 62; 63;  ]; Role = PEER; StartDate = 09/08/2013 11.06.12; PingTime =
01/01/0001 00.00.00; WcfHost = localhost; WcfPort = 9292; WcfPortTcp = 8282;  ]
2013-08-09 11:06:12,642 INFO  GetCache.Server.Node.Manager.NodeManager - Node manager
started.
2013-08-09 11:06:12,662 INFO  GetCache.Server.Node.Host.InternalServerHost - Opening
detected for InternalServerHost
2013-08-09 11:06:12,679 INFO  GetCache.Server.Node.Host.InternalServerHost - Opened
detected for InternalServerHost
2013-08-09 11:06:12,682 INFO  GetCache.Server.Node.Host.RemoteServerHost - Opening
detected for RemoteServerHost
2013-08-09 11:06:12,702 INFO  GetCache.Server.Node.Host.RemoteServerHost - Opened
detected for RemoteServerHost
2013-08-09 11:06:12,705 INFO  GetCache.Server.Node.Service.CleanerService - Cleaner
service start with dueTime 60 seconds and period 60 seconds
2013-08-09 11:06:12,705 INFO  GetCache.Server.Node.Service.PingService - Ping service
start with period 20 seconds
```

GetCache server configuration

Server configuration

The server node is configured by the config.xml file. This is an example of configuration file:

```
<?xml version="1.0"?>
<NodeConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <NodeName></NodeName>
  <WcfServerPort>9292</WcfServerPort>
  <WcfServerPortTcp>8282</WcfServerPortTcp>
  <WcfServerHost>localhost</WcfServerHost>
  <WcfInternalServerPort>1010</WcfInternalServerPort>
  <WcfInternalServerHost>localhost</WcfInternalServerHost>
  <CleanerServiceDueTime>60</CleanerServiceDueTime>
  <CleanerServicePeriod>60</CleanerServicePeriod>
  <FailureCount>3</FailureCount>
  <PingPeriod>10</PingPeriod>
  <EnableRemoteMonitor>true</EnableRemoteMonitor>
</NodeConfiguration>
```

The *NodeName* field is not mandatory because the server generates it randomly.

The *WcfServerHost* is the hostname (or IP address) used by the server to create a service host for incoming client request.
The *WcfServerPort* and *WcfServerPortTcp* are the http port and tcp port used by the service host for receiving client's communications.

The WcfInternalServerHost and *WcfInternalServerPort* represent the hostname and port used by the node for receiving cluster nodes communications.

The *CleanerServiceDueTime* represents the due time seconds of the Cleaner Service, the *CleanerServicePeriod* indicates the recurrence period of the Cleaner Service.

The *FailureCount* indicates the number of communications failure before a node of the cluster should be considered not available.

The *PingPeriod* represents the recurrence period of the Ping Service.

The *EnableRemoteMonitor* is used to turn on or off the ability to access the Monitor API.

Cluster configuration

The cluster nodes configuration file is the *nodelist.xml* file. The file contains a list of *GetCacheNode* elements, each one defining a node with a unique couple of host and port. *NodeGuid* element can have a zeros values because node get it asking the other nodes.

This is an example of configuration for a cluster of two node running on the same localhost:

Node 1 has *WcfInternalServerPort* equal to 1010 and *nodelist.xml* file

```
<?xml version="1.0"?>
<ArrayOfGetCacheNode xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <GetCacheNode>
       <NodeGuid>00000000-0000-0000-0000-000000000000</NodeGuid>
       <Host>localhost</Host>
       <ComPort>1012</ComPort>
  </GetCacheNode>
</ArrayOfGetCacheNode>
```

Node 2 has *WcfInternalServerPort* equal to 1012 and *nodelist.xml* file

```
<?xml version="1.0"?>
<ArrayOfGetCacheNode xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <GetCacheNode>
        <NodeGuid>00000000-0000-0000-0000-000000000000</NodeGuid>
        <Host>localhost</Host>
        <ComPort>1010</ComPort>
  </GetCacheNode>
</ArrayOfGetCacheNode>
```

The important point is that a node list must be valorized with the *WcfInternalServerHost* and *WcfInternalServerPort* of the others nodes of the cluster.

It's not necessary that the list is completed because a node determines who are all the nodes of the cluster asking the cluster configuration to the members of the list.
For example, Node 1 can not know anything of Node 2, the nodelist.xml of Node 1 can be empty and after Node 2 starts, Node 2 ping Node 1 and join the cluster.

Log configuration

GetCache use Log4Net to write logs and the Log4Net configuration is defined in the *GetCache.Server.exe.config* file.

GetCache server console command

After the server starts it shows a prompt > and typing help you can see the list of commands.

Here an example of usage:

```
GetCache server node Wasat-9 start.
Type help for commands list.
>put mykey "hello world!"
>get mykey
mykey =>  hello world!
>keys
Keys list:
       mykey
>count
Keys Count = 1
>show node
Current node: [ Name = Wasat-9; NodeGuid = 28393010-9b31-4357-b878-3a611a1ecc95;
 Host = localhost; ComPort = 1010; ShardingIndex = [ 0; 1; 2; 3; 4; 5; 6; 7; 8;
9; 10; 11; 12; 13; 14; 15; 16; 17; 18; 19; 20; 21; 22; 23; 24; 25; 26; 27; 28; 2
9; 30; 31; 32; 33; 34; 35; 36; 37; 38; 39; 40; 41; 42; 43; 44; 45; 46; 47; 48; 4
9; 50; 51; 52; 53; 54; 55; 56; 57; 58; 59; 60; 61; 62; 63;  ]; Role = PEER; Star
tDate = 09/08/2013 14.17.54; PingTime = 09/08/2013 14.18.36; WcfHost = localhost
; WcfPort = 9292; WcfPortTcp = 8282;  ]
>
```

GetCache clustering model

Cluster model

GetCache does not use Master - Slave configuration, so every node of the cluster assumes the role of a PEER, because no slaves are present. Every node can move data or ask to store replicated data to the others nodes.

When a new node joins the cluster, all the nodes recalculate the assigned sharding indexes and if necessary move the data that belong to a particular index to another node. Replicated data are not moved.

Data sharding vs. replicated data

Clients can send data to the server using two persistence methods (the word “persistence” is used improperly because GetCache stores the data in memory): Single or Replicated method.

The Single method (used by default) asks to store the data only in one node that is determined by a data sharding algorithm. This method makes it possible to distribute the load of memory on multiple servers.

The Replicated method is used to store the data in all the nodes of the cluster. This method can be used to maintain the important data, such as session data of a customer of a web application, on all GetCache nodes in order to access the data in high availability.

This two options are mutually exclusive.

The choice of which method to use is made by the client and may be different from object to object.

Ghost node

If a node is removed from the cluster, its “ghost” remains in the nodes cluster in-memory configuration for some time. This period can be calculated considering the *PingPeriod* and multiply it by the *FailureCount*.
During this period server nodes and clients continue to work correctly.

When the time is passed the remaining nodes remove the ghost from the in-memory configuration and reassign the sharding indexes (moving data if necessary).

If a new node joins the cluster or if the removed node restarts during this period the cluster and

the clients continue to work correctly, but the sharding indexes will be reassigned to the cluster nodes twice, when the new node joins the cluster and after when the ghost disappears.

GetCache client

GetCache is implemented partially in a client, and partially in a server.

Clients determine how to send object to particular node using a client-based hashing algorithm, which chooses a node in based on the input key and the assigned sharding indexes of the nodes. Clients also determine what to do when it cannot connect a node, and how to fetch keys from the nodes.

The nodes understand how to receive items, how to store them (Single vs. Replicated) and how to expire them.

GetCache Robots

Robots are a feature introduced with GetCache 1.1.0 that allows the server to execute user's developed components to pre-populate the storage.

The typical use of the cache is to read the data from the cache if it is present otherwise fetch it from a data-source and then put it in cache.
Using GetCache Robots, users may forget to retrieve data from the data-source because the robots care about retrieve the data and keep them in the cache.

See GetCache Robots User Manual for more details.

GetCache Monitor API

Monitor API is introduced with GetCache 1.2.0, this API provides an access to the cluster nodes to perform monitoring.
External applications can use these API to check the nodes states and to perform commands. It's possible for example to produce a remote console application that can send console commands to the nodes.
Monitor API offers also the possibility to upload robot module and deploy it to the server node.

See GetCache Monitor API User Manual for more details.