Caching

Caching

A cache is a collection of processed data that is kept on hand and reused in order to avoid costly repeated database queries. Totara 2.4 saw the implementation of MUC, the Moodle Universal Cache. This new system allows certain functions of Totara (e.g. string fetching) take advantage of different installed cache services (e.g. files, ram, memcached).

In future versions of Totara we will continue expanding the number of Totara functions that use MUC, which will continue improving performance, but you can already start using it to improve your site.

General approach to performance testing

Here is the general strategy you should be taking:

1. Build a test environment that is as close to your real production instance as possible (e.g. hardware, software, networking, etc.).
2. Make sure to remove as many uncontrolled variables as you can from this environment (e.g. other services).
3. Use a tool to place a realistic, but simulated and repeatable load on your server. (e.g. jmeter or selenium).
4. Decide on a way to measure performance of the server by capturing data (ram, load, time taken, etc.).
5. Run your load and measure a baseline performance result.
6. Change one variable at a time, and rerun the load to see if performance gets better or worse. Repeat as necessary.
7. When you discover settings that result in a consistent performance improvement, apply to your production site.

Cache configuration in Totara

Since Totara 2.4, Totara has provided a caching plugin framework to give administrators the ability to control where Totara stores cached data. For most Totara sites the default configuration should be sufficient and it is not necessary to change the configuration. For larger Totara sites with multiple servers, administrators may wish to use memcached, mongodb or other systems to store cache data. The cache plugin screen provides administrators with the ability to configure what cache data is stored where.

Caching in Totara is controlled by what is known as the Moodle Universal Cache, commonly referred to as MUC.

This document explains briefly what MUC is before proceeding into detail about the concepts and configuration options it offers.

The basic cache concepts in Totara

Caching in Totara isn't as complex as it first appears. A little background knowledge will go a long way in understanding how cache configuration works.

Cache types

There are three basic types (sometimes referred to as modes) of caches in Totara. These are:

Cache type	Description
Application cache	Application cache is by far the most commonly used cache type in code. Its information is shared by all users and its data persists between requests. Information stored here is usually cached for one of two reasons: • It is required information for the majority of requests and saves us a one or more database interactions • It is information that is accessed less frequently but is resource intensive to generate By default this information is stored in an organised structure within your Totara data directory.

On this page

Session cache	Session cache is just like the PHP session that you will already be familiar with, in fact it uses the PHP session by default. You may be wondering why we have this cache type at all, but the answer is simple. MUC provides a managed means of storing, and removing information that is required between requests. It offers developers a framework to use rather than having to reinvent the wheel and ensures that we have access to a controlled means of managing the cache as required. It's important to note that this isn't a frequently used cache type as by default session cache data is stored in the PHP session and the PHP session is stored in the database. Uses of the session cache type are limited to small datasets as we don't want to bloat sessions and thus put strain on the database.
Request cache	Data stored in the request cache type only persists for the lifetime of the request. If you're a PHP developer think of it like a managed static variable. This is by far the least used of the cache types, uses are often limited to information that will be accessed several times within the same request, usually by more than area of code. Cached information is stored in memory by default.

Cache types and multiple-server systems

If you have a system with multiple front-end web servers, the application cache must be shared between the servers. In other words, you cannot use fast local storage for the application cache, but must use shared storage or some other form of shared cache such as a shared memcache.

The same applies to session cache, unless you use a sticky sessions mechanism to ensure that within a session, users always access the same front-end server.

Cache back-ends

Cache back-ends are where data actually gets stored. These include things like the file system, php session, Memcached, and memory.

By default just file system, php session, and memory are used within Totara.

It is not required that a site has access to any other systems such a Memcached. Instead that is something you are responsible for installing and configuring yourself.

When cache back-ends are mentioned think of systems outside of Totara that can be used to store data, such as the MongoDB server, the Memcache server, and similar server applications.

Cache stores

Cache stores are a plugin type within Totara. They facilitate connecting Totara to the cache back-ends discussed above.

Totara ships with the three defaults mentioned above as well as Memcache, Memcached, and MongoDB.

The code for these is located within **cache/stores** in your Totara directory root.

Within Totara you can configure as many cache stores as your architecture requires. If you have several Memcache servers for instance, you can create an cache store instance for each.

Totara by default contains three cache store instances that get used when you've made no other configuration.

- A file store instance is created which gets used for all application caches (data is stored in your sitedata directory)
- A session store instance is created which gets used for all session caches (data is stored in the PHP session, which by default is stored in your database)
- A static memory store instance is created which gets used for all request cache types (data exists in memory for just the lifetime of a request)

How caches work in code

Caches are created in code and are used by the developer to store data they see a need to cache.

The developer does not get any say in where the data gets cached. They must specify the following information when creating a cache to use.

- The type of cache they require
- The area of code this cache will belong to (the API if you will)
- The name of the cache, something they make up to describe in one word what the cache stores

There are several optional requirements and settings they can specify as well. Importantly though, they can't choose which cache back-end to use, they can only choose the type of cache they want from the three detailed above.

How it ties together

This is best described in relation to roles played in an organisation.

- The System Administrator installs the cache back-ends you wish to use. Memcache, XCache, APC and so on. Totara doesn't know about these, they are outside of Totara's scope and purely the responsibility of your System Administrator.
- The Totara Site Administrator creates a cache store instance in Totara for each back end the site will make use of. There can be one or more cache stores instances for each back-end. Some back-ends like Memcached have settings to create separated spaces within one back-end.
- The developer has created caches in code and is using them to store data. They don't know anything about how you will use your caches, they just create a cache and tell Totara what type it is best for it.
- The Totara Site Administrator creates a mapping between a cache store instance and a cache. That mapping tells Totara to use the back-end you specify to store the data the developer wants cached.

In addition to that you can take things further still.

- You can map many caches to a single cache store instance
- You can map multiple cache store instances to a single cache with priority (primary ... final)
- You can map a cache store instance to be the default store used for all caches of a specific type that don't otherwise have specific mappings

If this is the first time you are reading about the MUC this probably sounds pretty complex but don't worry it will be discussed in better detail as we work through how to configure the caching in Totara.

Advanced concepts

These concepts are things that most sites will not need to know or concern themselves about.

You should only start looking here if you are looking to maximise performance on large sites running over clustered services with shared cache back-ends, or on multi-site architecture again where information is being shared between sites.

Locking

The idea of locking is nothing new, it is the process of controlling access in order to avoid concurrency issues.

MUC has a second type of plugin, a cache lock plugin that gets used when caches require it. To date no caches have required it. A cache by nature is volatile and any information that is absolutely mission critical should be a more permanent data store likely the database.

Nonetheless there is a locking system that cache definitions can require within their options and that will be applied when interacting with a cache store instance.

Sharing

Every bit of data that gets stored within a cache has a calculated unique key associated with it.

By default part of that key is the site identifier making any content stored in the cache specific to the site that stored it. For most sites this is exactly what you want.

However in some situations its beneficial to allow multiple sites, or somehow linked sites to share cached data.

Of course not all caches can be shared, however some certainly can and by sharing you can further reduce load and increase performance by maximising resource use.

This is an advanced feature, if you choose to configure sharing please do so carefully.

To make use of sharing you need to first configure identical cache store instances in the sites you want to share information, and then on each site set the sharing for the cache to the same value.

Available settings options are as follows:

- **Sites with the same site ID:** This is the default, it allows for sites with the same site ID to share cached information. It is the most restrictive but is going to work for all caches. All other options carry an element of risk in that you have to ensure the information in the cache is applicable to all sites that will be accessing it.

- **Sites running the same version:** All sites accessing the back-end that have the same Totara version can share the information this cache has stored in the cache store.
- **Custom key:** For this you manually enter a key to use for sharing. You must then enter the exact same key into the other sites you want to share information.
- **Everyone:** The cached data is accessible to all other sites accessing the data. This option puts the ball entirely in the Totara administrators court.

As an example if you had several Totara sites all the same version running on server with APC installed you could decide to map the language cache to the APC store and configure sharing for all sites running the same version.

The language cache for sites on the same version is safe to share in many situations, it is used on practically every page, and APC is extremely fast. These three points may result in a small performance boost for your sites.

It is important to consider with the language cache that by sharing it between sites any language customisations will also be shared.

Cache configuration settings

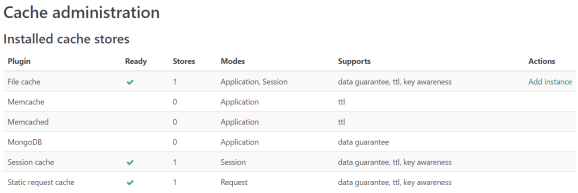
Cache configuration provides links to all of the actions you can perform to configure caching for your site's requirements.

Accessing the cache configuration screen

The cache configuration screen can only be accessed by users with the **site:config** capability. By default this is only administrators.

Once logged in the configuration screen can be found in *Site Administration* > *Plugins* > *Caching* > *Configuration*.

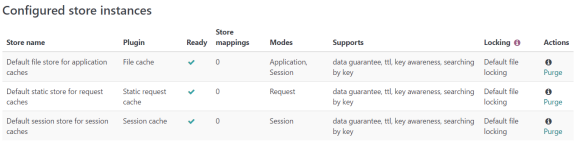
Installed cache stores



This is showing you a list of cache store plugins that you have installed.

For each plugin you can quickly see whether it is ready to be used (any php requirements have been met), how many store instances already exist on this site, the cache types that this store can be used for, what features it supports (advanced) and any actions you can perform relating to this store.

Session cache and static request cache do not support having multiple instances.

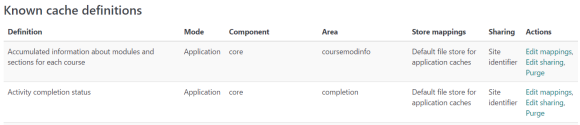


Here you get a list of the cache store instances on this site.

Column	Description
Store name	The name given to this cache store instance when it is created so that you can recognise it. It can be anything you want and is only used so that you can identify the store instance.
Plugin	The cache store plugin of which this is an instance of.
Ready	A tick gets shown when all PHP requirements have been met as well as any connection or set-up requirements have been verified.
Store mappings	The number of caches this store instance has been mapped to explicitly. Does not include any uses through default mappings (discussed below).

Modes	The modes that this cache store instance can serve.
Supports	The features supported by this cache store instance.
Locking	Locking is a mechanism that restricts access to cached data to one process at a time to prevent the data from being overwritten. The locking method determines how the lock is acquired and checked.
Actions	Any actions that can be performed against this cache store instance.

Known cache definitions


Known cache definitions

Definition	Mode	Component	Area	Store mappings	Sharing	Actions
Accumulated information about modules and sections for each course	Application	core	coursemodinfo	Default file store for application caches	Site identifier	Edit mappings, Edit sharing, Purge
Activity completion status	Application	core	completion	Default file store for application caches	Site identifier	Edit mappings, Edit sharing, Purge

The idea of a cache definition hasn't been discussed here yet. It is something controlled by the developer. When they create a cache they can do so in two ways, the first is by creating a cache definition.

This is essentially telling Totara about the cache they've created. The second way is to create an Adhoc cache. Developers are always encouraged to use the first method. Only caches with a definition can be mapped and further configured by the admin. Adhoc caches will make use of default settings only.

Typically Adhoc caches are only permitted in situations where the cache is small and configuring it beyond defaults would provide no benefit to administrators.

For each cache shown here you get the following information:

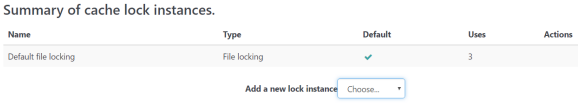
Column	Description
Definition	A concise description of this cache.
Mode	The cache type this cache is designed for.
Component	The code component the cache is associated with.
Area	The area of code this cache is serving within the component.
Store mappings	The store or stores that will be used for this cache.
Sharing	How is sharing configured for this site.
Actions	Any actions that can be performed on the cache. Typically you can edit the cache store instance mappings, edit sharing, and purge the cache.

You'll also find at the bottom of this table a link title **Rescan definitions**. Clicking this link will cause Totara to go off an check all core components, and installed plugins looking for changes in the cache definitions.

This happens by default during upgrade, and if a new cache definition is encountered. However should you find yourself looking for a cache that isn't there this may be worth a try.

It is also handy for developers as it allows them to quickly apply changes when working with caches. It is useful when tweaking cache definitions to find what works best.

Summary of cache lock instances


Summary of cache lock instances.

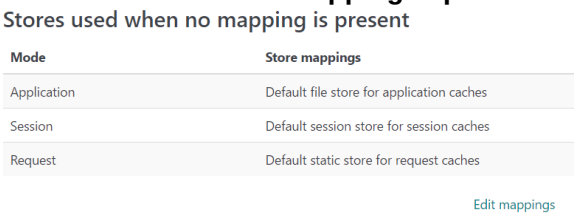
Name	Type	Default	Uses	Actions
Default file locking	File locking	✔	3	

Add a new lock instance: Choose...

As mentioned above cache locking is an advanced concept in MUC.

The table here shows information on the configured locking mechanisms available to MUC. By default just a single locking mechanism is available, file locking. At present there are no caches that make use of this.

Stores used when no mapping is present



This table quickly shows which cache store instances are going to be used for cache types if there are no specific mappings in place.

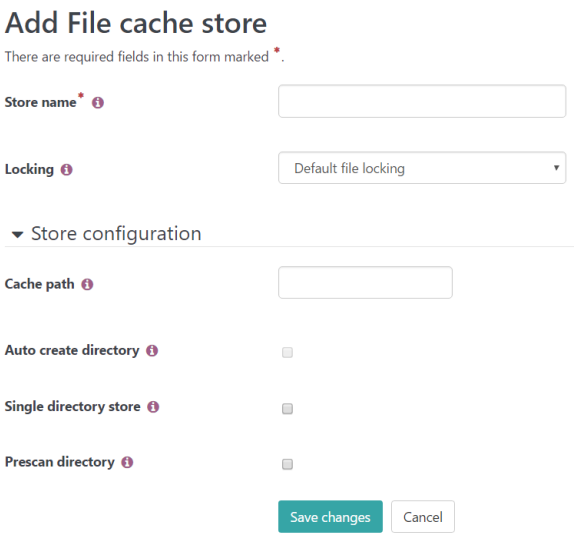
To simplify that, this show the default cache stores for each type.

At the bottom you will notice there is a link **Edit mappings** that takes you to a page where you can configure this.

Adding cache store instances

The default configuration is going to work for all sites, however you may be able to improve your sites performance by making use of various caching back-ends and techniques. The first thing you are going to want to do is add cache store instances configured to connect to/use the cache back-ends you've set up.

File cache



When on the cache configuration screen within the **Installed cache stores** table you should be able to see the File cache plugin, click **Add instance** to start the process of adding a file cache store instance.

When creating a file cache there is in fact only one required param, the store name. The store name is used to identify the file store instance in the configuration interface and must be unique to the site. It can be anything you want, but we would advise making it something that describes you intended use of the file store.

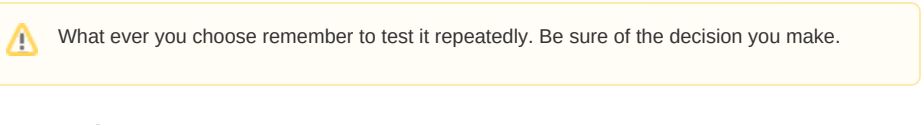
The following properties can also be specified, customising where the file cache will be located, and how it operates.

Setting	Description	Notes
Cache path	Allows you to specify a directory to use when storing cache data on the file system. Of course the user the webserver is running as must have read/write access to this directory. By default (blank) the sitedata directory will be used.	
Auto create directory	If enabled when the cache is initialised if the specified directory does not exist Totara will create it. If this is specified and the directory does not exist the cache will be deemed not ready and will not be used.	
Single directory store	By default the file store will create a subdirectory structure to store data in. The first 3 characters of the data key will be used as a directory. This is useful in avoiding file system limits it the file system has a maximum number of files per	

	directory. By enabling this option the file cache will not use subdirectories for storage of data. This leads to a flat structure but one that is more likely hit file system limits. Use with care.	
Prescan directory	One of the features the file cache provides is to prescan the storage directory when the cache is first used. This leads to faster checks of files at the expense of an in-depth read.	

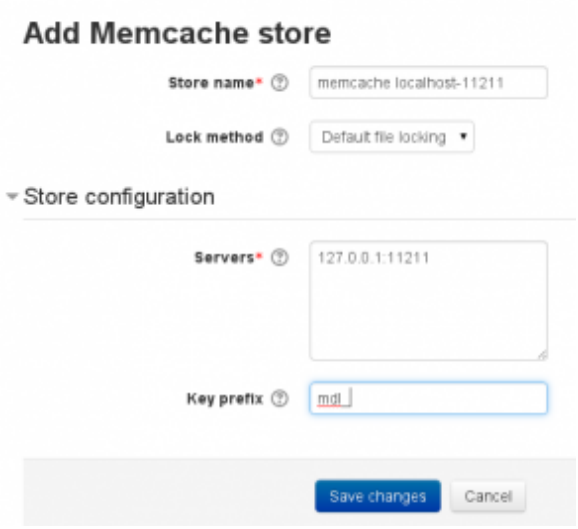
The file cache store is the default store used for application caches and by default the sitedata directory gets used for the cache. File access can be a taxing resource in times of increased load and the following are some ideas about configuring alternative file stores in order to improve performance.

- Check if there is a faster file system available for use on your server. Perhaps you have an SSD installed but are not using it for your sitedata directory because space is a premium. You should consider creating a directory or small partition on your SSD and creating a file store to use that instead of your Totara data directory.
- If you've not got a faster drive available for use, but you do have plenty of free space. Something that may be worth giving a shot would be to create a small partition on the drive you've got installed that uses a performance orientated file system. Many Linux installations these days for example use EXT4, a nice file system but one that has overheads due to the likes of journaling.
- Creating a partition and using a file system that has been optimised for performance may give you that little boost you are looking for. Remember caches are designed to be volatile and choosing a file system for a cache is different decision to choosing a file system for your server.
- Finally if you're ready to go to lengths and have an abundance of memory on your server you could consider creating a ramdisk/tmpfs and pointing a file store at that. Purely based in memory, it is volatile exactly like the cache is, and file system performance just isn't going to get much better than this.



What ever you choose remember to test it repeatedly. Be sure of the decision you make.

Memcache



Before you can add a Memcache store instance you must first have a Memcached server you can access and have the Memcache php extension installed and enabled on your web server.

Like the file store you must provide a the store name. It is used to identify the store instance in the configuration interface and must be unique to the site.

For a Memcache store you must also enter the Memcache server, or servers you wish it to make use of. Servers should be added one per line and each line can contain one to three properties separated by colons.

1. The URL or IP address of the server (required)
2. The port the server is listening on (optional)
3. The weight to give this server (optional)

For example, if you had two Memcached instances running on your server, one configured for the default port, and one configured for 11212 you would use the following:

```
127.0.0.1
127.0.0.1:11212
```

Optionally you can also specify a key prefix to use. What you enter here will prefixed to all keys before accessing the server and can be used to effectively partition the Memcache space in a recognisable way. This can be handy if you have a management tool for you Memcached server that you use to inspect what is stored there.

Important implementation notes

The Memcache extension does not provide a means of deleting a set or entries. Either a single entry is deleted, or all entries are deleted. Because of this it is important to note that when you purge a Memcache store within Totara it deletes all entries in the Memcache back-end. Not just those relating to Totara. For that reason it is highly recommended to use dedicated Memcached servers and to not configure any other software to use those servers. Doing so may lead to performance depreciation and adverse effects.

Likewise if you want to use Memcache for caching and for sessions in Totara it is essential to use two Memcached servers. One for sessions, and one for caching. Otherwise a cache purge in Totara will purge your sessions!

Memcached

Like the Memcache store you must first have a Memcached server you can access and have the Memcached php extension installed and enabled on your server.

Also like the Memcache store there are two required parameters in configuring a Memcached store, **Store name** and **Server**. There are also several optional parameters you can set when creating a Memcached store.

Setting	Description	Notes
Store name	It is used to identify the store instance in the configuration interface and must be unique to the site.	-

Servers	The servers you wish this cache store use. See below for details.
	
	Servers should be added one per line and each line can contain 1 to 3 properties separated by colons.
	
	• The URL or IP address of the server (required)
	• The port the server is listening on (optional)
	• The weight to give this server (optional)
		For example, if you had two Memcached instances running on your server, one configured for the default port, and one configured for 11212 you would use the following:
		
		`127.0.0.1`
		`127.0.0.1:11212`

Use compression	Defaults to true, but can be disabled if you wish.	-
Use serialiser	Allows your to select which serialiser gets used when communicating with the Memcache server.	By default the Memcached extension and PHP only provide one serialised, however there is a couple of others available for installation if you go looking for them. One for example is the igbinary found at https://github.com/igbinary/igbinary.
Prefix key	Allows you to set some characters that will be prefixed to all keys before interacting with the server.	-
Hash method		Please see the PHP manual for more

	The hash method provided by the Memcached extension is used by default here. However you can select to use an alternative if you wish.	information on the options available. If you wish to you can also override the default hash function PHP uses within your php.ini.
Buffer writes	Disabled by default. Turning on buffered writes will minimise interaction with the Memcached server by buffering io operations. The downside to this is that on a system with any concurrency there is a good chance multiple requests will end up generating the data because no one had pushed it to the Memcached server when they first requested it. Enabling this can be advantageous for caches that are only accessed in capability controlled areas for example where multiple interaction is taking a toll on network resources or such. But that is definitely on the extreme tweaking end of the scale.	

Important implementation notes

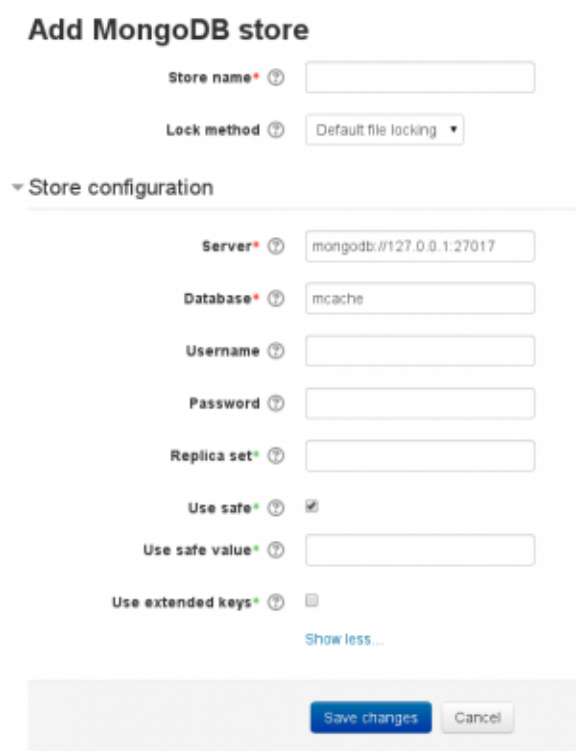
The Memcached extension does not provide a means of deleting a set or entries. Either a single entry is deleted, or all entries are deleted.

Because of this it is important to note that when you purge a Memcached store within Totara it deletes all entries in the Memcached server. Not just those relating to Totara.

For that reason it is highly recommended to use dedicated Memcached servers and to not configure any other software to use the same servers. Doing so may lead to performance depreciation and adverse affects.

Likewise if you want to use Memcached for caching and for sessions in Totara it is essential to use two Memcached servers. One for sessions, and one for caching. Otherwise a cache purge in Totara will purge your sessions!

MongoDB



MongoDB is an open source document orientated NoSQL database. Check out their website www.mongodb.org for more information.

Setting	Description	Notes

Store name	Used to identify the store instance in the configuration interface and must be unique to the site.	-
Server	This is the connection string for the server you want to use. Multiple servers can be specified using a comma-separated list.	-
Database	The name of the database to make use of.	-
Username	The username to use when making a connection.	-
Password	The password of the user being used for the connection.	-
Replica set	The name of the replica set to connect to. If this is given the master will be determined by using the ismaster database command on the seeds, so the driver may end up connecting to a server that was not even listed.	-
Use safe	If enabled the use safe option will be used during insert, get, and remove operations. If you've specified a replica set this will be forced on anyway.	-
Use safe value	You can choose to provide a specific value for use safe. This will determine the number of servers that operations must be completed on before they are deemed to have been completed.	-
Use extended keys	If enabled full key sets will be used when working with the plugin. This isn't used internally yet but would allow you to easily search and investigate the MongoDB plugin manually if you so choose. Turning this on will add an small overhead so should only be done if you require it.	-

Redis

Redis is open-source in-memory data structure store that can be used for caching or as a database. You can find out more in the Redis website.

Setting	Description	Notes
Server	This sets the hostname or IP address of the Redis server to use.	-
Password	This sets the password of the Redis server	If you want to use Redis as a session store you need to set the correct values in the config file (**$CFG->session_redis_***). You can find more details in the **config-dist.php** file.
Key prefix	This prefix is used for all key names on the Redis server. If you only have one Totara instance using this server, you can leave this value default.	Due to key length restrictions, a maximum of five characters is permitted.
Use serializer	Specifies the serialiser to use for serialising. The valid serialisers are either the default PHP serialiser or the igbinary serialiser. The latter is supported only when it has been configured correctly on the system (see more below in the **Using the igbinary serialiser** section).	-

Using the igbinary serialiser

The igbinary serialiser stores data structures in compact binary form and savings can be significant for storing serialised data.

The igbinary serialiser is not part of a standard PHP distribution but can be installed optionally. You can get it from GitHub or PECL.

Redis cache store

When igbinary is installed you can choose the serialiser when configuring the Redis cache store. It defaults to the standard PHP serialiser but can be switched to igbinary.

Static cache store

The static cache store automatically makes use of the igbinary serialiser when it is installed. There's no setting or config option to activate or deactivate it.

Redis session handler

If igbinary is installed and **$CFG->session_redis_serializer_use_igbinary** is set to true the Redis session handler uses igbinary for serialising the data.

Mapping a cache to a store instance

Mapping a store instance to a cache tells Totara to use that store instance when the cache is interacted with. This allows the Totara administrator to control where information gets stored and to most importantly optimise performance of your site by making the most of the resources available to your site.

To set a mapping first browse to the cache configuration screen.

1. Proceed to find the **Known cache definitions** table and within it find the cache you'd like to map.
2. In the actions column select the link for **Edit mappings**.
3. The screen you are presented allows you to map one or more cache store instances to be used by this cache.
4. You are presented with several dropdowns that allow you to map one or more cached. All mapped caches get interacted with.
    a. The **Primary store** is the store that will be used first when interacting with the cache.
    b. The **Final mapped store** will be the last cache store interacted with.

How this interaction occurs is documented below.

If no stores are mapped for the cache then the default stores are used. Have a look at the section below for information on changing the default stores.

If a single store instance is mapped to the cache the following occurs:

- Getting data from the cache Totara asks the cache to get the data.
- The cache attempts to get it from the store.
- If the store has it it gives it to the cache, and the cache gives it to Totara so that it can use the data.
- If the store doesn't have it the a fail is returned and Totara will have to generate the data and will most likely then send it to the cache.
- Storing data in the cache Totara will ask the cache to store some data, and the cache will give it to the cache store.

If multiple store instances are mapped to the cache the following occurs:

- Getting data from a store Totara asks the cache to get the data.
- The cache attempts to get it from the first store.
- If the first store has it then it returns the data to the cache and the cache returns it to Totara.
- If the first store doesn't have the data then it attempts to get the data from the second store.
- If the second store has it it returns it to the first store that then stores it itself before returning it to the cache.
- If it doesn't then the next store is used.
- This continue until either the data is found or there are no more stores to check.
- Storing data in the cache Totara will ask the cache to store some data, the cache will give it to every mapped cache store for storage.

The main advantage to assigning multiple stores is that you can introduce cache redundancy. Of course this introduces an overhead so it should only be used when actually required. The following is an example of when mapping multiple stores can provide an advantage.

Scenario

Problem: You have a web server that has a Totara site as well as other sites. You also have a Memcache server that is used by several sites including Totara. Memcache has a limited size cache, that when full and requested to store more information frees space by dropping the least used cache entries. You want to use Memcache for your Totara site because it is fast, however you are aware that it may introduce more cache misses because it is a heavily used Memcache server.

Solution: To get around this you map two stores to caches you wish to use Memcache. You make Memcache the primary store, and you make the default file store the final cache store.

Explanation: By doing this you've created redundancy, when something is requested Totara first tries to get it from Memcache (the fastest store) and if its not there it proceeds to check the file cache.

Just a couple more points of interest:

- Mapping multiple caches will introduce overhead, the more caches mapped the more overhead.
- Consider the cache stores you are mapping to, if data remains there once set then there is no point mapping any further stores after it. This technique is primarily valuable in situations where data is not guaranteed to remain after being set.

- Always test your configuration. Enable the display of performance information and then watch which stores get used when interacting with Totara in such a way as to trigger the cache.

Setting the stores that get used when no mapping is present

This is really setting the default stores that get used for a cache type when there is not a specific mapping that has been made for it.

1. To set a mapping first browse to the cache configuration screen.
2. Proceed to find the **Stores used when no mapping is present** table.
3. After the table you will find a link **Edit mappings**, click this.

On the screen you are presented with you can select one store for each cache type to use when a cache of the corresponding type gets initialised and there is not an explicit mapping for it.

On this interface the drop downs only contain store instances that are suitable for mapping to the type. Not all instances will necessarily be shown. If you have a store instance you don't see then it is not suitable for all the cache definitions that exist.

You will not be able to make that store instance the default, you will instead need to map it explicitly to each cache you want/can use it for.

Configuring caching for your site

This is where it really gets tricky, and unfortunately there is no step-by-step guide to this.

How caching can be best configured for a site depends on the site and the resources available to it.

Below are some points to help optimise cache performance:

- Plan it. It's a complex thing. Understand your site, understand your system, and really think how users will be using it all.
- If you've got a small site the gains aren't likely to be significant, if you've got a large site getting this right can lead to a substantial boost in performance.
- When looking at cache back-ends really research the advantages and disadvantages of each. Keep your site in mind when thinking about them. Depending upon your site you may find that no one cache back-end is going to meet the entire needs of your site and that you will benefit from having a couple of back-ends at your disposal.
- Things aren't usually as simple as installing a cache back-end and then using it. Pay attention to configuration and try to optimise it for your system. Test it separately and have an understanding of its performance before tell Totara about it. The cache allows you to shift load off the database and reduce page request processing. If for instance you have memcache installed but your connection has not been optimised for it you may well find yourself in a losing situation before you even tell Totara about the memcache server.
- When considering your default store instances keep in mind that they must operate with data sets of varying sizes and frequency. For a large site really your best bet is to look at each cache definition and map it to a store that is best suited for the data it includes and the frequency of access.
- Again when mapping store instances to caches really think about the cache you are mapping and make a decision based upon what you understand of your site and what you know about the cache.
- Test your configuration. If you can stress test it even better! If you turn on performance information Totara will also print cache access information at the bottom of the screen. You can use this to visually check the cache is being used as you expect, and it will give you an indication of where misses etc are occurring.
- Keep an eye on your back-end. Totara doesn't provide a means of monitoring a cache back-end and that is certainly something you should keep an eye on. Memcache for instance drops least used data when full to make room for new entries. APC on the other hand stops accepting data when full. Both will impact your performance if full and you're going to encounter misses. However APC when full is horrible, but it is much faster.