

Memory caching



- [Recommendations based on type of deployment](#)
 - [Small/Private home server](#)
 - [Organizations with single-server](#)
 - [Organizations with clustered setups](#)
 - [Additional notes for Redis vs. APCu on memory caching](#)
- [APCu](#)
- [Redis / Valkey using the Nextcloud KeyValueCollection](#)
 - [KeyValueCollection configuration in Nextcloud \(config.php\)](#)
 - [Connecting over TLS](#)
 - [Connecting over UNIX socket](#)
- [Redis using the phpredis PHP extension](#)
 - [Redis configuration in Nextcloud \(config.php\)](#)
 - [Connecting to single Redis server over TCP](#)
 - [Connecting to single Redis server over TLS](#)
 - [Connecting to Redis cluster over TLS](#)
 - [Connecting to single Redis server over UNIX socket](#)
 - [Using the Redis session handler](#)
- [Memcached](#)
 - [Memcached configuration in Nextcloud \(config.php\)](#)
- [Cache Directory location](#)
- [Cache Key Prefix for Redis or Memcached](#)

You can significantly improve your Nextcloud server performance with memory caching, where frequently-requested objects are stored in memory for faster retrieval. There are two types of caches to use: a PHP opcode cache, which is commonly called *opcache*, and data cache for your web server, commonly called "memcache".



If you do not install and enable a local memcache you will see a warning on your Nextcloud admin page. **A memcache is not required. You may safely ignore the warning if you prefer.** If you enable only a distributed cache (`memcache.distributed`) in your `config.php` and not a local cache (`memcache.local`) you will still see the cache warning.

A **PHP opcode** stores compiled PHP scripts, so they don't need to be re-compiled every time they are called. PHP bundles the Zend OPcache in core since version 5.5, so you don't need to install an opcode manually.

Data caching is supplied by the user. Nextcloud supports multiple memory caching backends, so you can choose the type of memcache that best fits your needs. The supported caching backends are:

- [APCu](#) (APCu 4.0.6 and up required).
A local cache for systems.
- [Redis](#) (4.0.0 and up required); [Valkey](#) and [KeyDB](#) are expected to work as Redis-compatible backends.



Automated/formal testing currently only occurs against Redis Open Source using the redis PHP extension and against Valkey using the Nextcloud keyvalue cache provider. KeyDB is expected to work as compatible backends as well, but are not formally tested.

For local and distributed caching, as well as transactional file locking.

- [Memcached](#)
For distributed caching.

Data caches, or memcaches, must be explicitly configured in Nextcloud by installing and enabling your desired cache, and then adding the appropriate entry to `config.php` (See [Configuration Parameters](#) for an overview of all possible config parameters).

Recommendations based on type of deployment

You may use both a local and a distributed cache. Recommended caches are APCu and Redis or Valkey. After installing and enabling your chosen memcache (data cache), verify that it is active by running

“ ”

See specific cache configuration options under the appropriate section further down.

Small/Private home server

Only use APCu:

```
'memcache.local' => '\OC\Memcache\APCu',
```

Organizations with single-server

Use Redis for everything except local memcache:

```
'memcache.local' => '\OC\Memcache\APCu',
'memcache.distributed' => '\OC\Memcache\Redis',
'memcache.locking' => '\OC\Memcache\Redis',
'redis' => [
    'host' => 'localhost',
    'port' => 6379,
],
```

Organizations with clustered setups

Use APCu for local cache and either Redis cluster ...:

```
'memcache.local' => '\OC\Memcache\APCu',
'memcache.distributed' => '\OC\Memcache\Redis',
'memcache.locking' => '\OC\Memcache\Redis',
'redis.cluster' => [
    'seeds' => [ // provide some/all of the cluster servers to bootstrap discovery, port
required
        'cache-cluster:7000',
        'cache-cluster:7001',
```

```
],  
],
```

... or Memcached cluster ...:

```
'memcache.local' => '\0C\Memcache\APCu',  
'memcache.distributed' => '\0C\Memcache\Memcached',  
'memcache.locking' => '\0C\Memcache\Memcached',  
'memcached_servers' => [  
    [ 'server1.example.com', 11211 ],  
    [ 'server2.example.com', 11211 ],  
],
```

... for distributed and locking caches.



If you run multiple web servers and enable a distributed cache in your `config.php` (`memcache.distributed`) or a file locking provider (`memcache.locking`) you need to make sure that they are referring to the very same memcache server/cluster and not to `localhost` or a unix socket.

Additional notes for Redis vs. APCu on memory caching

APCu is faster for local caching than Redis. If you have enough memory, use APCu for Memory Caching and Redis for File Locking. If you are low on memory, use Redis for both.

APCu

APCu is a data cache, and it is available in most Linux distributions. On Red Hat/CentOS/Fedora systems install `php-pear-apcu`. On Debian/Ubuntu/Mint systems install `php-apcu`.

After restarting your Web server, add this line to your `config.php` file:

```
'memcache.local' => '\0C\Memcache\APCu',
```

Refresh your Nextcloud admin page, and the cache warning should disappear.

Depending on your installation size and the number of users and interactions with the system you may want to adapt the `apc.shm_size` setting in your `php.ini`. The default value is 32M which is usually too low for Nextcloud. A good starting point is 128M. If you have a lot of users and/or a lot of apps installed you may want to increase this value further. Keep in mind that this memory needs to be available in your system's memory and kept in mind when sizing the amount of workers on your server.

A frequently resetting cache can lead to unexpected slow downs when the cache is being cleared and refilled.

There is an admin check trying to detect too low memory sizing, but make sure to monitor the APCu cache status to see if the cache is full and if you need to increase the size. [APCu provides a script](#) that can help with this, otherwise the [serverinfo app](#) in Nextcloud can also show the APCu cache status.

Redis / Valkey using the Nextcloud KeyValueCache

“ ”

The KeyValueCache was added with Nextcloud 34.0.2.

The `KeyValueCache` cache is a brand-independent cache backend for key-value stores like [Valkey](#) or [Redis](#). These are excellent modern memcaches to use for distributed caching, and as a key-value store for [Transactional File Locking](#) because they guarantee that cached objects are available for as long as they are needed.

Unlike the `phpredis` backend described in the next section, the `KeyValueCache` does not require a PHP extension: it is built on the [predis](#) library which is bundled with Nextcloud. You only need to install and run a Valkey or Redis compatible server and configure it in `config.php`.

“ ”

When the cache server is not located on the same machine the performance may be impacted due to network overhead compared to the `php-redis` based

backend. In those cases it may be preferable to use the `php-redis` backend instead.

On Debian/Ubuntu/Mint, install `valkey-server` or `redis-server`. On CentOS and Fedora, install `valkey` or `redis`. Depending on your distribution the server might not start automatically, so you may need to use your service manager to start the server, and to launch it at boot as a daemon.

You can verify that the daemon is running with `ps ax`:

```
ps ax | grep valkey
22203 ? Ssl    0:00 /usr/bin/valkey-server 127.0.0.1:6379
```

Add the appropriate entries to your `config.php`, and refresh your Nextcloud admin page.

KeyValueCache configuration in Nextcloud (config.php)

For best performance, use the `KeyValueCache` for file locking by adding this:

```
'memcache.locking' => '\OC\Memcache\KeyValueCache',
```

Additionally, you should use it for the distributed server cache:

```
'memcache.distributed' => '\OC\Memcache\KeyValueCache',
```

Furthermore, you could use it for the local cache like so, but it's not recommended (see warning below):

```
'memcache.local' => '\OC\Memcache\KeyValueCache',
```



Using a key-value store server for local cache on a multi-server setup can cause issues. Also, even on a single-server setup, APCu (see section above) should be faster.

When using the `keyvalue` cache for any of the above cache settings, you also need to specify the `memcache.kvstore` configuration in `config.php`. Three topologies are supported: a single server (`server`), a Sentinel managed replication set (`sentinel`), and a server cluster (`seeds`). Configure

exactly one of them.

The following options are available when using a single server (all but `server` are optional):

```
'memcache.locking' => '\0C\Memcache\KeyValueCache',
'memcache.distributed' => '\0C\Memcache\KeyValueCache',
'memcache_customprefix' => 'mynextcloudprefix',
'memcache.kvstore' => [
  'server' => [
    'host' => 'cache-host.example.com',
    'port' => 6379,
    // Protocol used to connect. One of 'tcp', 'tls' or 'unix'.
    // When omitted it is derived from the host (a leading '/' means 'unix').
    'protocol' => 'tcp',
  ],
  'user'      => 'nextcloud', // only sent when the server uses ACLs
  'password'  => 'password',
  'dbindex'   => 0,
  'timeout'   => 1.5,
  'read_timeout' => 1.5,
  'persistent' => false, // keep the connection open across requests
],
```

The same optional parameters are available for the Sentinel and cluster topologies below and are applied to every node.

For a Sentinel managed replication setup, provide the name of the monitored service and one seed entry per Sentinel node (each seed uses the same format as the `server` entry above):

```
'memcache.kvstore' => [
  'sentinel' => [
    'service' => 'mymaster',
    'seeds' => [
      ['host' => 'sentinel1.example.com', 'port' => 26379],
      ['host' => 'sentinel2.example.com', 'port' => 26379],
      ['host' => 'sentinel3.example.com', 'port' => 26379],
    ],
  ],
],
```

For a cluster, provide some or all of the cluster nodes to bootstrap discovery:

```
'memcache.kvstore' => [
  'seeds' => [
    ['host' => 'cache-cluster.example.com', 'port' => 7000],
    ['host' => 'cache-cluster.example.com', 'port' => 7001],
  ],
],
```



Selecting a numbered database with `dbindex` is only supported for single servers and Sentinel setups; clusters always use database 0, the support for numbered databases added with Valkey v9 is pending [upstream](#).

Connecting over TLS

To connect via TCP over TLS, set the `protocol` of the server (or of each seed) to `tls` and provide the SSL context ([SSL context options](#)):

```
'memcache.kvstore' => [
  'server' => [
    'host' => 'cache-host.example.com',
    'port' => 6379,
    'protocol' => 'tls',
  ],
  'ssl_context' => [
    'local_cert' => '/certs/cache.crt',
    'local_pk' => '/certs/cache.key',
    'cafile' => '/certs/ca.crt',
  ],
],
```

Connecting over UNIX socket

If you want to connect to a server configured to listen on a Unix socket (which is recommended if the server is running on the same system as Nextcloud), pass the socket path as the host. A leading slash in the host is automatically detected as a Unix socket, so no `protocol` or `port` is needed:

```
'memcache.kvstore' => [  
  'server' => [  
    'host' => '/run/valkey/valkey.sock',  
  ],  
],
```

Update the server configuration (for example `/etc/valkey/valkey.conf`) accordingly: uncomment the Unix socket options and ensure the socket path matches your Nextcloud configuration.

Be sure to set the right permissions on the socket so that your web server can read and write to it. For this you typically have to add the web server user to the `valkey` (or `redis`) group:

```
usermod -a -G valkey www-data
```

And modify the `unixsocketperm` setting of the server configuration accordingly:

```
unixsocketperm 770
```

You might need to restart your web server and the cache server for the changes to take effect.

Redis using the phpredis PHP extension

Redis is an excellent modern memcache to use for distributed caching, and as a key-value store for [Transactional File Locking](#) because it guarantees that cached objects are available for as long as they are needed.

Nextcloud uses the PhpRedis PHP extension. This extension provides an API for communicating with Redis-compatible key-value stores. Redis Open Source is the formally tested backend; Valkey and KeyDB are expected to work as compatible backends as well.

The Redis PHP module must be version 2.2.6+. If you are running a Linux distribution that does not package the supported versions of this module, or does not package Redis at all, see [Memcached](#).

On Debian/Ubuntu/Mint, install `redis-server` (or equivalent) and `php-redis`. Package names vary by distribution and backend.

On CentOS and Fedora, install `redis` (or equivalent) and `php-pecl-redis`. It will not start automatically, so you must use your service manager to start the `redis` server, and to launch it at boot as a daemon.

You can verify that the Redis daemon is running with `ps ax`:

```
ps ax | grep redis
22203 ? Ssl    0:00 /usr/bin/redis-server 127.0.0.1:6379
```

Restart your Web server, add the appropriate entries to your `config.php`, and refresh your Nextcloud admin page.

Redis configuration in Nextcloud (config.php)

For best performance, use Redis for file locking by adding this:

```
'memcache.locking' => '\OC\Memcache\Redis',
```

Additionally, you should use Redis for the distributed server cache:

```
'memcache.distributed' => '\OC\Memcache\Redis',
```

Furthermore, you could use Redis for the local cache like so, but it's not recommended (see warning below):

```
'memcache.local' => '\OC\Memcache\Redis',
```



Using Redis for local cache on a multi-server setup can cause issues. Also, even on a single-server setup, APCu (see section above) should be faster.

When using Redis for any of the above cache settings, you also need to specify either the `redis` or `redis.cluster` configuration in `config.php`.

The following options are available to configure when using a single redis server (all but `host` and `port` are optional. For the latter two, see the next sections):

```
'memcache.locking' => '\OC\Memcache\Redis',
'memcache.distributed' => '\OC\Memcache\Redis',
'memcache.local' => '\OC\Memcache\Redis' ,
'memcache_customprefix' => 'mynextcloudprefix',
```

```
'redis' => [
  // 'host'      => see connection parameters below
  // 'port'      => see connection parameters below
  'user'        => 'nextcloud',
  'password'    => 'password',
  'dbindex'     => 0,
  'timeout'     => 1.5,
  'read_timeout' => 1.5,
],
```

The following options are available to configure when using a redis cluster (all but `seeds` are optional):

```
'memcache.locking' => '\OC\Memcache\Redis',
'memcache.distributed' => '\OC\Memcache\Redis',
'memcache.local' => '\OC\Memcache\Redis' ,
'memcache_customprefix' => 'mynextcloudprefix',
'redis.cluster' => [
  'seeds' => [ // provide some/all of the cluster servers to bootstrap discovery, port
required
    'cache-cluster:7000',
    'cache-cluster:7001',
    'cache-cluster:7002',
    'cache-cluster:7003',
    'cache-cluster:7004',
    'cache-cluster:7005'
  ],
  'failover_mode'    => \RedisCluster::FAILOVER_ERROR,
  'timeout'          => 0.0,
  'read_timeout'     => 0.0,
  'user'             => 'nextcloud',
  'password'         => 'password',
  'dbindex'          => 0,
],
```



The port is required as part of the server URL. However, it is not necessary to list all servers: for example, if all servers are load balanced via the same DNS name,

only that server name is required.

Connecting to single Redis server over TCP

To connect to a remote or local Redis server over TCP use:

```
'redis' => [  
  'host' => 'redis-host.example.com',  
  'port' => 6379,  
],
```

Connecting to single Redis server over TLS

To connect via TCP over TLS, add the following configuration:

```
'redis' => [  
  'host' => 'tls://127.0.0.1',  
  'port' => 6379,  
  'ssl_context' => [  
    'local_cert' => '/certs/redis.crt',  
    'local_pk' => '/certs/redis.key',  
    'cafile' => '/certs/ca.crt',  
    'verify_peer_name' => false,  
  ],  
],
```

Connecting to Redis cluster over TLS

To connect via TCP over TLS, add the following configuration:

```
'redis.cluster' => [  
  'seeds' => [ // provide some/all of the cluster servers to bootstrap discovery, port  
               required  
    'cache-cluster:7000',
```

```
'cache-cluster:7001',  
],  
'ssl_context' => [  
  'local_cert' => '/certs/redis.crt',  
  'local_pk' => '/certs/redis.key',  
  'cafile' => '/certs/ca.crt',  
  'verify_peer_name' => false,  
],  
],
```

Connecting to single Redis server over UNIX socket

If you want to connect to Redis configured to listen on an Unix socket (which is recommended if Redis is running on the same system as Nextcloud) use this example `config.php` configuration:

```
'redis' => [  
  'host'      => '/run/redis/redis-server.sock',  
  'port'      => 0,  
],
```

Only "host" and "port" variables are required, the other ones are optional.

Update the redis configuration in `/etc/redis/redis.conf` accordingly: uncomment Unix socket options and ensure the "socket" and "port" settings match your Nextcloud configuration.

Be sure to set the right permissions on redis.sock so that your webserver can read and write to it. For this you typically have to add the webserver user to the redis group:

```
usermod -a -G redis www-data
```

And modify the `unixsocketperm` of the `redis.conf` accordingly:

```
unixsocketperm 770
```

You might need to restart apache and redis for the changes to take effect:

```
systemctl restart apache2  
systemctl restart redis-server
```

Redis is very configurable; consult [the Redis documentation](#) to learn more.

Using the Redis session handler

If you are using Redis for locking and/or caching, you may also wish to use Redis for session management. Redis can be used for centralized session management across multiple Nextcloud application servers, unlike the standard files handler. If you use the Redis handler, though, you *MUST* ensure that session locking is enabled. As of this writing, the Redis session handler does *NOT* enable session locking by default, which can lead to session corruption in some Nextcloud apps that make heavy use of session writes such as Talk. In addition, even when session locking is enabled, if the application fails to acquire a lock, the Redis session handler does not currently return an error. Adding the following settings in your `php.ini` file will prevent session corruption when using Redis as your session handler: :

```
redis.session.locking_enabled=1
redis.session.lock_retries=-1
redis.session.lock_wait_time=10000
```

More information on configuration of phpredis session handler can be found on the [PhpRedis GitHub page](#)

Memcached

Memcached is a reliable oldtimer for shared caching on distributed servers, and performs well with Nextcloud with one exception: it is not suitable to use with [Transactional File Locking](#) because it does not store locks, and data can disappear from the cache at any time (Redis is the best memcache for this).

“ ”

Be sure to install the **memcached** PHP module, and not memcache, as in the following examples. Nextcloud supports only the **memcached** PHP module.

Setting up Memcached is easy. On Debian/Ubuntu/Mint install `memcached` and `php-memcached`. The installer will automatically start `memcached` and configure it to launch at startup.

On Red Hat/CentOS/Fedora install `memcached` and `php-pecl-memcached`. It will not start automatically, so you must use your service manager to start `memcached`, and to launch it at boot as a daemon.

You can verify that the Memcached daemon is running with `ps ax`:

```
ps ax | grep memcached
19563 ? Sl 0:02 /usr/bin/memcached -m 64 -p 11211 -u memcache -l
127.0.0.1
```

Restart your Web server, add the appropriate entries to your `config.php`, and refresh your Nextcloud admin page.

Memcached configuration in Nextcloud (config.php)

This example uses APCu for the local cache, Memcached as the distributed memcache, and lists all the servers in the shared cache pool with their port numbers:

```
'memcache.local' => '\OC\Memcache\APCu',
'memcache.distributed' => '\OC\Memcache\Memcached',
'memcache.locking' => '\OC\Memcache\Memcached',
'memcached_servers' => [
    [ 'server0.example.com', 11211 ],
    [ 'server1.example.com', 11211 ],
    [ 'server2.example.com', 11211 ],
],
```

Cache Directory location

The cache directory defaults to `data/$user/cache` where `$user` is the current user. You may use the `'cache_path'` directive in `config.php` (See [Configuration Parameters](#)) to select a different location.

Cache Key Prefix for Redis or Memcached

By default, Nextcloud generates a semi-unique prefix for cache keys using information like instance ID, version etc. to mitigate the problem of collisions when using the same cache for multiple Nextcloud instances. If you want to make sure to prevent collisions altogether, you can use the

following setting to define your custom prefix:

```
'memcache_customprefix' => 'mynextcloudprefix',
```

This also allows you to create ACLs in Redis and limit the keys specific users can access (e.g. if you want to isolate specific Nextcloud instances when using the same cache). This may be security-relevant for you.

📄 #3

📄 cocobook-sync 📄 2026-10-28 16:26:59 UTC

📄 cocobook-sync 📄 2026-10-29 01:09:16 UTC