

ownCloud Server Administration Manual

version 10.0.7

The ownCloud developers

April 17, 2018

Contents

Table of Contents	1
Introduction	1
ownCloud Videos and Blogs	1
Target Audience	1
Release Notes	1
Changes in 10.0.7	2
Known issues	2
Changes in 10.0.6	2
Changes in 10.0.5	2
Technology preview for PHP 7.2 support	2
php-intl now is a hard requirement	2
Changed: Only allow a single active theme app	2
Removed old Dropbox external storage backend (Dropbox API v1)	2
Fixed: Only set CORS headers on WebDAV endpoint when Origin header is specified	2
Fixes and improvements for the Mail Template Editor	2
Known issues	3
Changes in 10.0.4	3
More granular sharing restrictions	3
Configurable solution for indistinguishable user display names	3
Added “occ files:scan” repair mode to repair filecache inconsistencies	3
Detailed mode for “occ security:routes”	3
Added mode of operations to differentiate between single-instance or clustered setup	3
Added occ dav:cleanup-chunks command to clean up expired uploads	3
Administrators can now exclude files from integrity check in config.php	4
Modification time value of files is now 64 bits long	4
Updated minimum supported browser versions	4
Known issues	4
10.0.3 resolved known issues	4
Known issues	4
Changes in 10.0.3	4
Changes in 10.0.1	5
Settings	6
Infrastructure	6
Known Issues	6
Converting the Database Type doesn't work	6
Installing the LDAP user backend will trigger the installation twice	6
SAML authentication only works for users synced with <code>occ user:sync</code>	6
The web UI prevents uninstalling apps marked as shipped, e.g., <code>user_ldap</code>	6
Moving files around in external storages outside of ownCloud will invalidate the metadata	6

Existing LDAP users only show up in the user management page and the share dialog after being synced	6
Error pages will not use the configured theme but will instead fall back to the community default	7
Changes in 10.0.0	7
Changes in 9.1	7
Changes in 9.0	9
Enterprise 9.0	10
Changes in 8.2	10
Changes in 8.1	10
Enterprise 8.1	11
Changes in 8.0	11
Manual LDAP Port Configuration	11
No Preview Icon on Text Files	11
Remote Federated Cloud Share Cannot be Reshared With Local Users	11
Manually Migrate Encryption Keys after Upgrade	11
Windows Server Not Supported	11
PHP 5.3 Support Dropped	12
Disable Apache Multiviews	12
ownCloud Does Not Follow Symlinks	12
No Commas in Group Names	12
Hebrew File Names Too Large on Windows	12
Google Drive Large Files Fail with 500 Error	12
Encrypting Large Numbers of Files	12
Enterprise 8.0	12
Sharepoint Drive SSL Not Verified	12
No Federated Cloud Sharing with Shibboleth	12
Direct Uploads to SWIFT do not Appear in ownCloud	12
SWIFT Objectstore Incompatible with Encryption App	12
application Store is Back	12
Changes in 7.0	13
Manual LDAP Port Configuration	13
LDAP Search Performance Improved	13
Protecting ownCloud on IIS from Data Loss	13
Antivirus Application Modes	13
“Enable Only for Specific Groups” Fails	13
Changes to File Previews	13
4GB Limit on SFTP Transfers	13
“Not Enough Space Available” on File Upload	14
No More Expiration Date On Local Shares	14
Zero Quota Not Read-Only	14
Enterprise 7.0	14

No Federated Cloud Sharing with Shibboleth	14
Windows Network Drive	14
Sharepoint Drive SSL	14
Shibboleth and WebDAV Incompatible	14
No SQLite	14
No Application Store	14
LDAP Home Connector Linux Only	14
What's New in ownCloud 10.0.7	14
Installation	15
System Requirements	15
Officially Recommended & Supported Options	15
Server	15
Mobile	15
Web Browser	15
Hypervisors	15
Desktop	16
Alternative (But Unsupported) Options	16
Web Server	16
Memory Requirements	16
Database Requirements	16
Deployment Recommendations	17
General Recommendations	17
ownCloud Administrators Must Have Command Line or Cron Access	17
Scenario 1: Small Workgroups and Departments	17
Recommended System Requirements	18
Components	18
Operating system	18
SSL Configuration	18
Load Balancer	18
Database	18
Backup	18
Authentication	19
Session Management	19
Memory Caching	19
Storage	19
ownCloud Edition	19
Scenario 2: Mid-Sized Enterprises	19
Recommended System Requirements	19
Components	19
Operating System	20
SSL Configuration	20
Load Balancer	20

Database	20
Backup	20
Authentication	20
Session Management	20
Memory Caching	20
Storage	20
ownCloud Edition	21
Scenario 3: Large Enterprises and Service Providers	21
Recommended System Requirements	21
Components	21
Operating system	21
SSL Configuration	21
Load Balancer	21
Database	21
Backup	22
Authentication	22
LDAP	22
Session Management	22
Caching	22
Storage	22
ownCloud Edition	22
Redis Configuration	22
Known Issues	22
Deadlocks When Using MariaDB Galera Cluster	22
Set wsrep_sync_wait to 1 on all Galera Cluster nodes	23
What the parameter does	23
Why enable it	23
References	23
Deployment Considerations	23
Hardware	23
Single Machine / Scale-Up Deployment	23
Scale-Out Deployment	24
What About NGINX / PHP-FPM?	24
A Single Master DB is Single Point of Failure, Does Not Scale	24
Software	24
Operating System	24
Web server	24
Relational Database	24
File Storage	25
Session Storage	25
Manual Installation on Linux	25
Prerequisites	26

Required	26
PHP Version	26
PHP Extensions	26
Database Extensions	27
Required For Specific Apps	27
Optional	27
Recommended	27
For Specific Apps	27
For Server Performance	27
For Preview Generation	28
For Command Line Processing	28
For MySQL/MariaDB	28
Install the Required Packages	28
On Ubuntu 16.04 LTS Server	28
On Ubuntu 14.04 LTS Server	29
Installing smbclient	29
Installing ssh2	30
Running Additional Apps?	30
Additional Extensions	30
RHEL (RedHat Enterprise Linux) 7.2	30
Required Extensions	30
Optional Extensions	30
SLES (SUSE Linux Enterprise Server) 12	30
Required Extensions	30
Optional Extensions	30
APCu	30
Redis	31
Install ownCloud	31
Configure Apache Web Server	31
Additional Apache Configurations	32
Multi-Processing Module (MPM)	32
Enable SSL	32
Run the Installation Wizard	33
Set Strong Directory Permissions	33
Managing Trusted Domains	33
Linux Package Manager Installation	34
Available Packages	34
Installing ownCloud Community Edition	34
What is the Correct Version?	35
Major Releases	35
Minor Releases	35
The Latest Stable Version	35

Installing ownCloud Enterprise Edition	35
Downgrading	35
Additional Guides and Notes	35
Archlinux	35
Mageia	35
Note for MySQL/MariaDB environments	35
Running ownCloud in a sub-directory	35
The Installation Wizard	35
Quick Start	36
In-Depth Guide	36
Data Directory Location	36
Database Choices	37
SQLite	37
MYSQl/MariaDB	38
PostgreSQL	38
Oracle 11g	38
Database Setup By ownCloud	38
Post-Installation Steps	39
Installing with Docker	43
Installation on a Local Machine	43
Logging In	44
Stopping the Containers	44
Upgrading ownCloud on Docker	45
Command Line Installation	45
Configuration Notes & Tips	46
SELinux	46
php.ini	46
php.ini - Used by the Web server	46
php.ini - used by the php-cli and so by ownCloud CRON jobs	46
session.auto_start && enable_post_data_reading	46
session.save_path	47
suhosin.session.encryptkey	47
post_max_size	47
realpath_cache_size	47
How to get a working value	47
PHP-FPM	48
System Environment Variables	48
Maximum Upload Size	49
.htaccess Notes for Apache	49
No basic authentication headers were found	49
Other Web Servers	49
Troubleshooting	49

Database Configuration Issues	49
Changing Your ownCloud URL	49
Installing and Managing Apps	50
Supported Apps	50
Viewing Enabled Apps	50
Managing Apps	50
Adding Apps	51
Using Custom App Directories	51
Manually Installing Apps	52
Supported Apps in ownCloud	52
AGPL Apps	52
Enterprise-Only Apps	53
SELinux Configuration	53
Enable updates via the web interface	54
Disallow write access to the whole web directory	54
Allow access to a remote database	54
Allow access to LDAP server	54
Allow access to remote network	54
Allow access to network memcache	54
Allow access to SMTP/sendmail	54
Allow access to CIFS/SMB	54
Allow access to FuseFS	54
Allow access to GPG for Rainloop	54
Troubleshooting	55
General Troubleshooting	55
Redis on RHEL 7 & Derivatives	55
NGINX Configuration	55
Example Configurations	55
Note 1	56
Note 2	56
ownCloud in the web root of NGINX	56
ownCloud in a subdirectory of NGINX	59
Troubleshooting	61
JavaScript (.js) or CSS (.css) files not served properly	61
Not all of my contacts are synchronized	62
Windows: Error 0x80070043 “The network name cannot be found.” while adding a network drive	62
Log Optimisation	62
Suppressing <code>htaccesstest.txt</code> and <code>.ocdata</code> Log Messages	62
Prevent access log entries when accessing thumbnails	62
Performance Tuning	63
Configure NGINX to use caching for ownCloud internal images and thumbnails	63

Using Let's Encrypt SSL Certificates	65
Requirements & Dependencies	65
Install Let's Encrypt's Certbot client	66
Via GitHub	66
Via Apt	66
Updating Certbot	67
Register your email address	67
Create Let's Encrypt's config files	67
cli.ini	67
list.sh	68
renew.sh	68
renew-cron.sh	68
delete.sh	68
<your-domain-name>.sh	69
Create an SSL certificate	69
Listing Existing Certificates	70
Web Server setup	70
Test the setup	70
Renewing Certificates	71
Manual renewal	71
Automatic renewal via crontab	71
Add extra domains to the certificate	72
Deleting SSL Certificates	72
Upgrading	72
Upgrade PHP on RedHat 7 and Centos 7	72
Upgrade PHP to version 5.6	72
Upgrade PHP to version 7.0	73
Upgrade Marketplace Applications	74
Single-Server Environment	74
Clustered/Multi-Server Environment	74
Configuration	75
Database Configuration	75
Converting Database Type	75
Run the conversion	75
Unconvertible Tables	75
Database Configuration	76
Requirements	76
MySQL / MariaDB with Binary Logging Enabled	76
MySQL / MariaDB "READ COMMITTED" transaction isolation level	76
MySQL / MariaDB storage engine	77
Parameters	77
Configuring a MySQL or MariaDB Database	77

Configure MySQL for 4-byte Unicode Support	78
PostgreSQL Database	78
Troubleshooting	80
How to workaround General error: 2006 MySQL server has gone away	80
How can I find out if my MySQL/PostgreSQL server is reachable?	80
How can I find out if a created user can access a database?	80
Useful SQL commands	81
File Sharing and Management	81
File Sharing	81
Transferring Files to Another User	83
Creating Persistent File Shares	83
Configuring Federation Sharing	83
Sharing With ownCloud 8 and Older	83
Creating a new Federation Share (9.0+ only)	83
Configuring Trusted ownCloud Servers	85
Creating Federation Shares via Public Link Share	85
Configuration Tips	87
Uploading big files > 512MB	87
System Configuration	87
Configuring Your Web server	88
Apache	88
Apache with mod_fcgid	88
NGINX	88
Configuring PHP	89
Configuring ownCloud	89
Configuring upload limits within the GUI	89
General upload issues	90
Providing Default Files	90
Additional Configuration	91
Configuring External Storage (GUI)	92
Enabling External Storage Support	92
Storage Configuration	92
User and Group Permissions	93
Mount Options	93
Using Self-Signed Certificates	94
Available storage backends	94
Amazon S3	94
Dropbox	95
Step One - Install the “External Storage: Dropbox” app from the ownCloud Marketplace	95
Step Two - Create a Dropbox app	95
Step Three - Create a Dropbox Share	96

Other Options	96
Step Four - Using the Dropbox Share	96
FTP/FTPS	97
Google Drive	98
Local	104
OpenStack Object Storage	105
ownCloud	106
SFTP	107
SMB/CIFS	107
WebDAV	108
Allow Users to Mount External Storage	109
Setting Up Google Drive and Dropbox Connections	109
Detecting Files Added to External Storages	109
Configuring External Storage (Configuration File)	110
External Storage Authentication mechanisms	110
Special Mechanisms	110
Password-based Mechanisms	110
Public-key Mechanisms	110
OAuth	110
Encryption Configuration	111
Encryption Types	112
Before Enabling Encryption	112
How To Enable Encryption	113
Enabling Encryption From the Command-line	114
Enabling Master Key Based Encryption	114
Sharing Encrypted Files	115
Encrypting External Mountpoints	115
How To Enable Users File Recovery Keys	115
Changing The Recovery Key Password	117
Disabling Encryption	117
Not All Files Are Encrypted	117
LDAP and Other External User Back-ends	118
occ Encryption Commands	118
View Current Encryption Status	118
List Available Encryption Modules	118
Encrypt and Decrypt Data Files For All Users	118
Move Key Location	119
Create a New Master Key	119
Recreating an Existing Master Key	119
Disabling Encryption	119
Files Not Encrypted	120
LDAP and Other External User Back-ends	120

Encryption migration to ownCloud 8.0	120
Encryption migration to ownCloud 8.1	120
Transactional File Locking	121
Previews Configuration	121
Parameters	122
Disabling previews	122
Maximum preview size	123
Maximum scale factor	123
Controlling File Versions and Aging	123
Enterprise File Retention	124
Managing the Trashbin	124
How To Install and Configure an LDAP Proxy-Cache Server	125
Background	125
How To Setup the Server	125
1. Install OpenLDAP	125
2. Configure the Server	126
3. Enable the configuration file	132
4. Open the log	132
5. Perform a test search	132
6. Configure ownCloud LDAP App	132
Cache Multiple Active Directory Servers	133
Mimetypes Management	137
Mimetype Aliases	137
Changing Existing Icons and Using Custom Icons	137
Example - Changing the JSON File Icon	138
Mimetype Mapping	139
Icon retrieval	139
Server Configuration	140
Warnings on Admin Page	140
Cache Warnings	140
Transactional file locking is disabled	140
You are accessing this site via HTTP	140
The test with getenv("PATH") only returns an empty response	140
The "Strict-Transport-Security" HTTP header is not configured	140
/dev/urandom is not readable by PHP	141
Your Web server is not yet set up properly to allow file synchronization	141
Outdated NSS / OpenSSL version	141
Your Web server is not set up properly to resolve /.well-known/caldav/ or .well-known/carddav/	141
Some files have not passed the integrity check	141
Your database does not run with "READ COMMITTED" transaction isolation level	141
Importing System-wide and Personal SSL Certificates	141

Importing Personal SSL Certificates	141
Site-wide SSL Import	142
Using OCC to Import and Manage SSL Certificates	142
Using the occ Command	142
occ Command Directory	142
Run occ As Your HTTP User	143
Apps Commands	145
Background Jobs Selector	146
Config Commands	146
Getting a Single Configuration Value	147
Setting a Single Configuration Value	147
Setting an Array of Configuration Values	147
Deleting a Single Configuration Value	148
Dav Commands	148
Database Conversion	149
Encryption	149
Federation Sync	150
File Operations	151
The files:cleanup command	151
The files:scan command	151
The --path Option	152
The --repair Option	152
The files:transfer-ownership command	152
Files External	153
Group Commands	153
Creating Groups	153
Listing Groups	153
Listing Group Members	154
Adding Members to Groups	154
Removing Members from Groups	154
Deleting a Group	155
Integrity Check	155
I10n, Create Javascript Translation Files for Apps	155
A Working Example	156
Create Translations in Multiple Languages	156
LDAP Commands	156
Logging Commands	159
Maintenance Commands	160
Market	161
Install an Application	161
Install Apps From The Marketplace	161
Install Apps From a File Archive	161

Usage Example	161
Reports	162
Security	162
Ransomware Protection	163
Sharing	163
Shibboleth Modes (Enterprise Edition only)	164
Trashbin	164
User Commands	165
Creating Users	165
Setting a User's Password	166
Deleting A User	166
Listing Users	166
Listing Group Membership of a User	166
Finding The User's Last Login	167
User Application Settings	167
Retrieving User Settings	167
Setting a Setting	168
Deleting a Setting	168
Modify user details	168
Generating a User Count Report	168
Syncing User Accounts	169
LDAP	169
Samba	169
Shibboleth	169
Syncing via cron job	169
Versions	169
Command Line Installation	170
Command Line Upgrade	172
Two-factor Authentication	173
Disable Users	173
Finding Inactive Users	173
Configuring the Activity App	174
Enabling the Activity App	174
Configuring your ownCloud for the Activity App	174
Virus Scanner Support	174
Overview	174
How ClamAV Works With ownCloud	174
Things To Note	175
Configuring the ClamAV Antivirus Scanner	175
Installing ClamAV	175
Debian, Ubuntu, Linux Mint	175
Red Hat 7 and CentOS 7	175

Configuring and Running ClamAV	176
Automating ClamAV Virus Database Updates	176
Install the Anti-Virus App	177
Configuring ClamAV within ownCloud	177
Configuration Warnings	177
Mode Configuration	178
Daemon (Socket)	178
Daemon	178
Executable	179
Rule Configuration	179
Default Ruleset	180
Update An Existing Rule	180
Add A New Rule	181
Delete An Existing Rule	181
Memory Caching	181
Supported Caching Backends	181
Cache Directory Location	182
Cache Types	182
APCu	182
Installing APCu	182
Redis	182
Installing Redis on Debian-based Distributions	183
Installing Redis on RedHat, CentOS, and Fedora	183
Additional notes for Redis vs. APCu on Memory Caching	183
Installing Redis on other distributions	183
On Debian/Mint/Ubuntu	183
On CentOS and Fedora	184
Clearing the Redis Cache	184
Further Reading	184
Memcached	184
Installing Memcached	184
On Debian/Ubuntu/Mint	184
On RedHat/CentOS/Fedora	185
Configuration File Paths	185
Clearing the Memcached Cache	185
Configuring Memory Caching	185
APCu Configuration	186
Redis Configuration	186
Memcached Configuration	186
Configuration Recommendations Based on Type of Deployment	186
Small/Private Home Server	186
Small Organization, Single-server Setup	186

Large Organization, Clustered Setup	187
Configuring Transactional File Locking	187
Caching Exceptions	187
Background Jobs	188
Cron Jobs	188
Cron	188
Webscron	189
AJAX	189
Parallel Task Execution	189
Available Background Jobs	189
CleanupChunks	190
ExpireTrash	190
ExpireVersions	190
SyncJob (CardDAV)	190
SyncJob (Federation)	190
Config.php Parameters	190
Default Parameters	191
Default config.php Examples	192
User Experience	193
Mail Parameters	194
Proxy Configurations	195
Deleted Items (trash bin)	196
File versions	197
ownCloud Verifications	197
Logging	198
Alternate Code Locations	199
Previews	199
Comments	201
Maintenance	201
SSL	202
Memory caching backend configuration	202
Sharing	204
All other configuration options	204
App config options	207
Overriding Existing Parameter Values Using Environment Variables	208
Apache Web Server	208
Nginx Web Server (php-fpm)	208
Command Line	208
Email Configuration	208
Configuring an SMTP Server	208
The Graphical Email Configuration Wizard	209
Configuring PHP and Sendmail	209

Setting Mail Server Parameters in config.php	210
SMTP	210
SSL/TLS	211
STARTTLS	211
Authentication	211
PHP Mail	211
Sendmail	212
Qmail	212
Send a Test Email	212
Using Self-Signed Certificates	212
Troubleshooting	212
Why is my web domain different from my mail domain?	213
How can I find out if an SMTP server is reachable?	213
How can I find out if the SMTP server is listening on a specific TCP port?	213
How can I determine if the SMTP server supports SMTPS?	213
How can I determine what authorization and encryption protocols the mail server supports?	213
Enabling Debug Mode	214
Using Email Templates	214
Excluding Directories and Blacklisting Files	215
Definitions of terms	215
Impact on System Performance	215
Blacklisted Files	215
Excluded Directories	216
Linking External Sites	217
Custom Client Download Repositories	218
Knowledge Base Configuration	219
Parameters	219
Language Configuration	219
Parameters	219
Logging Configuration	219
Parameters	219
ownCloud	220
syslog	220
Conditional Logging Level Increase	220
Hardening and Security Guidance	221
Limit on Password Length	221
Operating system	221
Give PHP read access to /dev/urandom	221
Enable hardening modules such as SELinux	221
Deployment	222
Place data directory outside of the web root	222

Disable preview image generation	222
Use HTTPS	222
Redirect all unencrypted traffic to HTTPS	222
Enable HTTP Strict Transport Security	222
Proper SSL configuration	222
Use a dedicated domain for ownCloud	223
Ensure that your ownCloud instance is installed in a DMZ	223
Serve security related Headers by the Web server	223
Use Fail2ban	223
Using Fail2ban to secure an ownCloud login	224
Password Policy	226
Password Policy	226
Reverse Proxy Configuration	227
Defining Trusted Proxies	227
Overwrite Parameters	228
Example	228
Multiple Domains Reverse SSL Proxy	228
Using Third Party PHP Components	228
Managing Third Party Parameters	228
Automatic Configuration Setup	229
Parameters	229
Automatic Configurations Examples	229
Data Directory	229
SQLite Database	229
MySQL Database	229
PostgreSQL Database	230
All Parameters	230
ownCloud Server Tuning	230
Using Cron to Perform Background Jobs	231
Enable JavaScript and CSS Asset Management	231
Enable Memory Caching	231
Use Redis-based Transactional File Locking	231
Redis Tuning	231
TCP-Backlog	231
Transparent Huge Pages (THP)	231
Redis Latency Problems	232
Database Tuning	232
Using MariaDB/MySQL Instead of SQLite	232
Tune MariaDB/MySQL	232
Tune PostgreSQL	232
SSL / Encryption App	232
Webserver Tuning	232

Tune Apache	233
Enable HTTP/2 Support	233
Apache Processes	233
Use KeepAlive	233
Hostname Lookups	233
Log files	233
Enable index.php-less URLs	233
Prerequisites	233
Configuration steps	233
Troubleshooting	234
User Management	234
User Management	234
Creating a New User	235
Reset a User's Password	236
Renaming a User	236
Granting Administrator Privileges to a User	236
Managing Groups	237
Setting Storage Quotas	237
Deleting users	237
Enabling Custom Groups	237
Resetting a Lost Admin Password	238
Resetting a User Password	239
User Authentication with IMAP, SMB, and FTP	239
IMAP	239
Example	240
SMB	240
Example	240
FTP	240
Example	241
User Authentication with LDAP	241
Configuration	242
Server Tab	242
User Filter	243
Login Filter	244
Group Filter	245
Advanced Settings	246
Connection Settings	246
Directory Settings	247
Special Attributes	250
Expert Settings	251
Testing the configuration	253
Syncing Users	253

How Often Should the Job Run?	253
ownCloud Avatar integration	253
Troubleshooting, Tips and Tricks	254
SSL Certificate Verification (LDAPS, TLS)	254
Microsoft Active Directory	254
memberOf / Read MemberOf permissions	254
Duplicating Server Configurations	254
Performance tips	255
Caching	255
LDAP indexing	255
Use precise base DNs	255
Use precise filters	255
ownCloud LDAP Internals	255
User and Group Mapping	255
Handling with Backup Server	256
User Provisioning API	256
Instruction Set For Users	256
users / adduser	256
Example	256
XML Output	256
users / getusers	256
Example	257
XML Output	257
users / getuser	257
Example	257
XML Output	257
users / edituser	258
Examples	258
XML Output	258
users / deleteuser	258
Example	259
XML Output	259
users / getgroups	259
Example	259
XML Output	259
users / addtogroup	259
Example	260
XML Output	260
users / removefromgroup	260
Example	260
XML Output	260
users / createsubadmin	261

Example	261
XML Output	261
users / removesubadmin	261
Example	262
XML Output	262
users / getsubadmingroups	262
Example	262
XML Output	262
Instruction Set For Groups	262
groups / getgroups	262
Example	263
XML Output	263
groups / addgroup	263
Example	263
XML Output	263
groups / getgroup	264
Example	264
XML Output	264
groups / getsubadmins	264
Example	264
XML Output	264
groups / deletegroup	265
Example	265
XML Output	265
Instruction Set For Apps	265
apps / getapps	265
Example	265
XML Output	266
apps / getappinfo	266
Example	266
XML Output	266
apps / enable	267
Example	267
XML Output	267
apps / disable	267
Example	267
XML Output	267
ownCloud Roles	268
Anonymous	268
Guest	268
Standard User	269
Federated User	269

ownCloud Group Administrator	269
ownCloud Administrator	269
System Administrator	269
Auditor	270
Maintenance	270
Maintenance Mode Configuration	270
Backing up ownCloud	270
Backing Up the config/ and data/ Directories	271
Backup Database	271
MySQL/MariaDB	271
SQLite	271
PostgreSQL	271
Restoring Files From Backup When Encryption Is Enabled	271
How to Upgrade Your ownCloud Server	272
Prerequisites	272
Upgrade Options	272
Upgrade ownCloud From Packages	272
Upgrade Quickstart	273
Upgrade Tips	273
Setting Strong Directory Permissions	274
Upgrading Across Skipped Releases	274
Upgrading ownCloud with the Updater App	274
Setting Permissions for Updating	276
Command Line Options	277
updater.secret value in config.php	277
Manual ownCloud Upgrade	277
Backup Your Existing Installation	278
Review Third-Party Apps	278
Check ownCloud's Mandatory Requirements	278
Enable Maintenance Mode	278
Stop the Webserver	278
Download the Latest Installation	278
Setup the New Installation	279
The Standard Upgrade	279
The Power User Upgrade	279
Disable Core Apps	279
Market and Marketplace App Upgrades	280
Start the Upgrade	280
Copy Old Apps	280
Disable Maintenance Mode	280
Restart the Webserver	280
Finalize the Installation	280

Test the Upgrade	281
Reverse Upgrade	281
Troubleshooting	281
Restoring ownCloud	281
Restore Directories	282
Restore Database	282
MySQL	282
SQLite	282
PostgreSQL	282
Migrating to a Different Server	282
How to Migrate	283
Managing Trusted Domains	283
Example Migration	284
Preparation	284
Migration	284
Enable Maintenance Mode	284
Transfer the Database	284
Transfer Data and Configure the New Server	285
Finish the Migration	285
Reverse the Changes to ssh-config	286
Update DNS and Trusted Domains	286
How To Manually Move a Data Directory	286
Fix Hardcoded Database Path Variables	286
Update the oc_storages table	286
Update the oc_accounts table	287
Update the oc_jobs table	287
Fix Application Settings	287
Issues and Troubleshooting	287
General Troubleshooting	287
Bugs	288
General Troubleshooting	288
Disable 3rdparty / non-shipped apps	288
ownCloud Logfiles	288
PHP Version and Information	288
Debugging Sync Issues	289
Common problems / error messages	289
OAuth2	290
ownCloud clients cannot connect to the ownCloud server	290
Missing Data Directory	290
Troubleshooting Web server and PHP problems	290
Logfiles	290
Web Server and PHP Modules	290

Troubleshooting WebDAV	291
General troubleshooting	291
Error 0x80070043 “The network name cannot be found.” while adding a network drive	291
Troubleshooting Contacts & Calendar	292
Service discovery	292
Unable to update Contacts or Events	292
Client Sync Stalls	292
Other issues	293
Code Signing	293
FAQ	293
Why Did ownCloud Add Code Signing?	293
Do We Lock Down ownCloud?	293
Not Open Source Anymore?	293
Is Code Signing Mandatory For Apps?	293
Fixing Invalid Code Integrity Messages	293
Rescans	296
Errors	296
Impersonating Users	297
Impersonating a User	297
Ending an Impersonation	297
Restrict Impersonation to Groups & Group administrators	297
Enterprise Features	298
Installation	298
Installing & Upgrading ownCloud Enterprise Edition	298
Manual Installation	298
SELinux	298
License Keys	298
Introduction	298
Configuration	299
Supported ownCloud Enterprise Edition Apps	299
Oracle Database Setup	299
Outline of Steps	299
Configuring Oracle	299
Setting up the User Space for ownCloud	299
Add OCI8 Client Packages	299
Ubuntu	300
RedHat / Centos / Fedora	300
Install the OCI8 PHP Extension	300
Configuration File Paths	300
Debian & Ubuntu	300
RedHat, Centos, & Fedora	300

Validating the Extension	301
Configure ownCloud	301
Configuration Wizard	301
Database user	302
Database password	302
Database Name	302
Database Table Space	303
Configuration File	303
Useful SQL Commands	303
Firewall Configuration	303
File Firewall	304
How the File Firewall Works	304
Using the File Firewall	304
Available Conditions	305
User Group	305
User Agent	305
User Device	305
Request Time	305
Request URL	305
Request Type	306
Request IP Range (IPv4) and IP Range (IPv6)	306
File Size Upload	306
File Mimetype Upload	306
System File Tag	306
Regular Expression	306
Controlling Access to Folders	306
Custom Configuration for Branded Clients	307
Ransomware Protection	308
About Ransomware Protection	308
Prevention: Blocking Common Ransomware File Extensions	308
File Extension Blacklist	308
File Blocking	309
Account Locking	309
Protection: Data Retention and Rollback	309
Other Elements of Ransomware Protection	310
Requirements	310
Mandatory	310
Optional	310
Limitations	311
File Management	311
Advanced File Tagging With the Workflow App	311
Tag Manager	311

Automatic Tagging	312
Retention	313
Retention Engines	314
External Storage	314
Enterprise-Only Authentication Options	314
LDAP Home Connector	315
Mount Home Directory	315
Configure the LDAP Home Connector	315
Configure the LDAP Server	316
Configuring SharePoint Integration	317
Creating a SharePoint Mount	317
Enabling Users	318
Note	318
Troubleshooting	318
Unsharing	318
Logging	319
Mount Points	319
Installing and Configuring the External Storage: Windows Network Drives App	319
Installation	320
Example	320
Creating a New Share	320
Personal WND Mounts	322
libsmbclient Issues	322
Windows Network Drive Listener	323
Setting Up the WND Listener	323
wnd:listen	323
wnd:process-queue	324
Basic Setup for One ownCloud Server	324
Password Options	325
3rd Party Software Examples	325
Password Option Precedence	326
Optimizing wnd:process-queue	326
The File Serializer	327
Usage Recommendations	327
Number of Serializers	327
File Clean Up	327
Interaction Between Listener, Serializer, and Windows Password Lockout	327
Multiple Server Setup	328
Basic Command Execution Examples	328
Configuring S3 and OpenStack Swift Objects as Primary Storage	328
Implications	329
Configuration	329

Amazon S3	329
Ceph S3	329
S3 Multibucket Configuration	330
OpenStack Swift	330
How to Create and Configure Microsoft OneDrive	331
Create an Application Configuration	331
Application Password	331
Redirect URLs	331
Application Permissions	332
Configure a Mount Point in ownCloud	332
User Management	333
Shibboleth Integration	333
Introduction	333
The Apache Shibboleth module	333
Apache Configuration	333
The ownCloud Shibboleth App	334
Choosing the App Mode	335
Mapping ownCloud User IDs	337
Shibboleth with Desktop and Mobile Clients	337
WebDAV Support	337
Known Limitations	337
Encryption	337
Other Login Mechanisms	337
Session Timeout	337
UID Considerations and Windows Network Drive compatibility	337
Creating Branded ownCloud Clients	337
Creating Branded Client Apps	338
Overview	338
Building a Branded Desktop Sync Client	338
Building a Branded iOS App	338
Building an Android App	338
Custom Client Download Repositories	338
Logging Apps	338
Enterprise Logging Apps	338
Server Branding	339
Custom Theming ownCloud	339
Overview	339
The ownCloud X Appliance	341
What is the Appliance?	341
How to Install the Appliance	341
Download the Appliance	341
Install the Appliance	342

Start the Appliance	343
Activate the Appliance	344
Administer the Appliance	344
ownCloud Appliance Login Information	347
How to Update ownCloud	348
Use the Univention Management Console	348
In-place Upgrade (for 10.0 users)	348
Uninstall the Existing Version and Install the New Version (for 9.1 users)	349
Use the Command Line	352
Upgrading From Version 10.0.1 to 10.0.3	352
Upgrading From Versions Prior to 10.0	353
Managing UCS	353
Adding Users and Groups in UCS for ownCloud	353
Login to the Univention Management Console	353
Create the User	353
Create the Group	356
Install Antivirus Software in the ownCloud Appliance	359
Install ClamAV and Related Components	359
Configure ownCloud to Use ClamAV	360
How to add certificates	360
Active Directory Integration	361
Backup	361
FAQ	361
How do I transfer files from one user to another?	361
How do I deal with problems caused by using self-signed SSL certificates?	361
I'm the admin and I lost my password! What do I do now!	361
What is a Federated System?	361

Table of Contents

Introduction

Welcome to the ownCloud Server Administration Guide. This guide describes administration tasks for ownCloud, the flexible open source file synchronization and sharing solution. ownCloud includes the ownCloud server, which runs on Linux, client applications for Microsoft Windows, Mac OS X and Linux, and mobile clients for the Android and Apple iOS operating systems.

Current editions of ownCloud manuals are always available online at doc.owncloud.org and doc.owncloud.com.

ownCloud server is available in three editions:

- The free community-supported server. This is the core server for all editions.
- The Standard Subscription for customers who want paid support for the core Server, without Enterprise applications.
- The Enterprise Subscription provides paid support for the Enterprise Edition. This includes the core Server and Enterprise apps.

See What's New in ownCloud 10.0.7 for more information on the different ownCloud editions.

ownCloud Videos and Blogs

See the [official ownCloud channel](#) and [ownClouders community channel](#) on YouTube for tutorials, overviews, and conference videos.

Visit [ownCloud Planet](#) for news and developer blogs.

Target Audience

This guide is for users who want to install, administer, and optimize their ownCloud servers. To learn more about the ownCloud Web user interface, and desktop and mobile clients, please refer to their respective manuals:

- [ownCloud User Manual](#)
- [ownCloud Desktop Client](#)
- [ownCloud Android App](#)
- [ownCloud iOS App](#)

Release Notes

- Changes in 10.0.7
- Changes in 10.0.6
- Changes in 10.0.5
- Changes in 10.0.4
- Changes in 10.0.3
- Changes in 10.0.1
- Changes in 10.0.0
- Changes in 9.1
- Changes in 9.0
- Changes in 8.2
- Changes in 8.1
- Changes in 8.0

- Changes in 7.0

Changes in 10.0.7

ownCloud Server 10.0.7 is a hotfix follow-up release that takes care of an [issue](#) regarding OAuth authentication.

Please consider the ownCloud Server 10.0.5 release notes.

Known issues

- When using application passwords, [log entries related to “Login Failed” will appear](#) and can be ignored. For people using fail2ban or other account locking tools based on log parsing, please apply [this patch](#) with `patch -p1 < 50c78a4bf4c2ab4194f40111b8a34b7e9cc17a14.patch` (original pull request [here](#)).

Changes in 10.0.6

ownCloud Server 10.0.6 is a hotfix follow-up release that takes care of an issue during the build process (<https://github.com/owncloud/core/pull/30265>). Please consider the ownCloud Server 10.0.5 release notes.

Changes in 10.0.5

Dear ownCloud administrator, please find below the changes and known issues in ownCloud Server 10.0.5 that need your attention. You can also read the [full ownCloud Server changelog](#) for further details on what has changed.

Technology preview for PHP 7.2 support

ownCloud catches up with new web technologies. This has mainly been introduced for the open-source community to test and give feedback. PHP 7.2 is not yet supported nor recommended for production scenarios. ownCloud is going to fully support PHP 7.2 with the next major release.

php-intl now is a hard requirement

Please make sure to have the PHP extension installed before upgrading.

Changed: Only allow a single active theme app

The theming behavior has been changed so that only a single theme can be active concurrently. This change ensures that themes can not interfere in any way (e.g., override default theming in an arbitrary order). Please make sure to have the desired theme enabled after upgrading.

Removed old Dropbox external storage backend (Dropbox API v1)

Please switch to the new [External Storage: Dropbox app](#) (https://marketplace.owncloud.com/apps/files_external_dropbox) with Dropbox API v2 support to continue providing Dropbox external storages to your users.

Fixed: Only set CORS headers on WebDAV endpoint when Origin header is specified

ownCloud Server 10.0.4 known issue is resolved.

Fixes and improvements for the Mail Template Editor

- Known issues are resolved: Mail Template Editor works again, got support for app themes and additional templates were added for customization.
- Mail Template Editor is still bundled with ownCloud Server but will soon be released as a separate app to ownCloud Marketplace.
- Changelog: <https://github.com/owncloud/templateeditor/blob/release/0.2.0/CHANGELOG.md>

Known issues

- When using application passwords, [log entries related to “Login Failed” will appear](#), please upgrade to 10.0.7 and check the fix mentioned in its release notes.

Changes in 10.0.4

Dear ownCloud administrator, please find below the changes and known issues in ownCloud Server 10.0.4 that need your attention. You can also read [the full ownCloud Server 10.0.4 changelog](#) for further details on what has changed.

More granular sharing restrictions

The “*Restrict users to only share with users in their groups*” option, in the Sharing settings, restricts users to only share with groups which they are a member of, while simultaneously prohibiting sharing with single users that do not belong to any of the users’ groups.

To make this more granular, we split this option into two parts and added “*Restrict users to only share with groups they are member of*”, which differentiates between users and groups. Doing so makes it possible to restrict users from sharing with all users of an installation, limiting them to only being able to share with groups which they are a member of, and vice versa.

Configurable solution for indistinguishable user display names

The ownCloud sharing dialog displays users according to their display name. As users can choose their display name in self-service (which can be disabled in `config.php`) and display names are not unique, it is possible that a user can’t distinguish sharing results. To cover this case the displayed user identifiers are now configurable. In the Sharing settings administrators can now configure the display of either mail addresses or user ids.

Added “occ files:scan” repair mode to repair filecache inconsistencies

We recommend to use this command when directed to do so in the upgrade process. Please refer to [the occ command’s files:scan –repair documentation](#) for more information.

Detailed mode for “occ security:routes”

Administrators can use the output of this command when using a network firewall, to check the appropriateness of configured rules or to get assistance when setting up.

Added mode of operations to differentiate between single-instance or clustered setup

As ownCloud needs to behave differently when operating in a clustered setup versus a single instance setup, the new `config.php` option `operation.mode` has been added. It can take one of two values: `single-instance` and `clustered-instance`. For example: `'operation.mode' => 'clustered-instance'`.

Currently the Market App (ownCloud Marketplace integration) does not support clustered setups and can do harm when used for installing or updating apps. The new config setting prevents this and other actions that are undesired in cluster mode.

When operating in a clustered setup, it is mandatory to set this option. Please check [the config_sample_php_parameters documentation](#) for more information.

Added occ dav:cleanup-chunks command to clean up expired uploads

When file uploads are interrupted for any reason, already uploaded file parts (chunks) remain in the underlying storage so that the file upload can resume in a future upload attempt. However, resuming an upload is only possible until the partial upload is expired and deleted, respectively.

To clean up chunks (expire and delete) originating from unfinished uploads, administrators can use this newly introduced command. The default expiry time is two days, but it can be specified as a parameter to the command. **It is recommended to configure CRON to execute this background job regularly.**

Table of Contents

It is not included in the regular ownCloud background jobs so that the administrators have more flexibility in scheduling it. Please check [the background jobs configuration documentation](#) for more information.

Administrators can now exclude files from integrity check in config.php

When administrators did intentional changes to the ownCloud code they now have the ability to exclude certain files from the integrity checker. Please check "config.sample.php" for the usage of 'integrity.excluded.files'.

Modification time value of files is now 64 bits long

When upgrading to 10.0.4 migrations may increase update duration dependent on number of files.

Updated minimum supported browser versions

Users with outdated browsers might get warnings. See [the list of supported browser versions](#).

Known issues

- When using application passwords, [log entries related to "Login Failed" will appear](#), please upgrade to 10.0.7 and check the fix mentioned in its release notes.

10.0.3 resolved known issues

- SFTP external storages with key pair mode work again
- Added support for MariaDB 10.2.7+
- Encryption panel in admin settings fixed to properly detect current mode after upgrade to ownCloud 10
- Removed double quotes from boolean values in status.php output

Known issues

- Impersonate app 0.1.1 does not work with ownCloud Server 10.0.4. Please update to [Impersonate 0.1.2](#) to be able to use the feature with ownCloud 10.0.4.
- Mounting ownCloud storage via davfs does not work

Changes in 10.0.3

Dear ownCloud administrator, please find below the changes and known issues of ownCloud Server 10.0.3 that need your attention:

The full ownCloud Server 10.0.3 changelog can be found here:
<https://github.com/owncloud/core/blob/stable10/CHANGELOG.md>

- It is now possible to directly upgrade from 8.2.11 to 10.0.3 in a single upgrade process.
- Added occ command to list routes which can help administrators setting up network firewall rules.
- 'occ upgrade' is now verbose by default. Administrators may need to adjust scripts for automated setup/upgrade procedures that rely on 'occ upgrade' outputs.
- **Reenabled medial search by default**
 - Enables partial search in sharing dialog autocomplete (e.g. a user wants to share with the user "Peter": Entering "pe" will find the user, entering "ter" will only find the user if the option is enabled)
 - New default is set to enabled as there is no performance impact anymore due to the introduction of the user account table in ownCloud Server 10.0.1.
 - Please check the setting. You need to disable it explicitly if the functionality is undesired.
- All database columns that use the fileid have been changed to bigint (64-bits). For large instances it is therefore highly recommended to upgrade in order to avoid reaching limits.

- **Upgrade and Market app information**

- Removed “appstoreenabled” setting from config.php. If you want to disable the app store / Marketplace integration, please disable the Market app.
- Added setting ‘upgrade.automatic-app-update’ to config.php to disable automatic app updates with ‘occ upgrade’ when Market app is enabled
- On upgrade from OC < 10 the Market app won’t be enabled if “appstoreenabled” was false in config.php.
- Clustering: Better support of read only config file and apps folder
- Default minimum desktop client version in config.php is now 2.2.4.

Known issues

- Added quotes in boolean result values of yourdomain/status.php output
- Setting up SFTP external storages with keypairs does not work. <https://github.com/owncloud/core/issues/28669>
- If you have storage encryption enabled, the web UI for encryption will ask again what mode you want to operate with even if you already had a mode selected before. The administrator must select the mode they had selected before. <https://github.com/owncloud/core/issues/28985>
- Uploading a folder in Chrome in a way that would overwrite an existing folder can randomly fail (race conditions). <https://github.com/owncloud/core/issues/28844>
- Federated shares can not be accepted in WebUI for SAML/Shibboleth users
- For **MariaDB users**: Currently, Doctrine has no support for the breaking changes introduced in MariaDB 10.2.7, and above. If you are on MariaDB 10.2.7 or above, and have encountered the message “1067 Invalid default value for ‘lastmodified’”, [please apply this patch](#) to Doctrine. We expect this bug to be fixed in ownCloud 10.0.4. For more information on the bug, [check out the related issue](#).
- When updating from ownCloud < 9.0 the CLI output may hang for some time (potentially up to 20 minutes for big instances) whilst sharing is updated. This can happen in a variety of places during the upgrade and is to be expected. Please be patient as the update is performed and the output will continue as normal.

Changes in 10.0.1

Hello ownCloud administrator, please read carefully to be prepared for updates and operations of your ownCloud setup.

- **A new update path:** ownCloud 10.0.1 contains migration logic to allow upgrading directly from 9.0 to 10.0.1.
- **Marketplace:** Please create an account for [the new marketplace](#). Access to optional ownCloud extensions and enterprise apps will be provided by the marketplace from now on. Currently some apps are still shipped with the tarballs / packages and will be moved to the marketplace in the near future.
- **Apps:** *LDAP, gallery, activity, PDF viewer, and text editor* were moved to the marketplace.
- **Updates with marketplace:** During the upgrade, enabled apps are also updated by fetching new versions directly from the marketplace. If during an update, sources for some apps are missing, and the ownCloud instance has no access to the marketplace, the administrator needs to disable these apps or manually download and provide the apps before updating.
- **App updates:** Third party apps are not disabled anymore when upgrading.
- **Upgrade migration test:** The upgrade migration test has been removed; see [Test the Upgrade](#). (Option `--skip-migration-tests` removed from update command).

Note

The template editor app is not included in the 10.0.1 release due to technical reasons, but will be distributed via the marketplace. However, you can still edit template files manually.

Settings

- **Settings design:** Admin, personal pages, and app management are now merged together into a single “Settings” entry.
- **Disable users:** The ability to disable users in the user management panel has been added.
- **Password Policy:** Rules now apply not only to link passwords but also to user passwords.

Infrastructure

- **Client:** You need to update to [the latest desktop client version](#).
- **Cron jobs:** The user account table has been reworked. As a result the Cron job for [syncing user backends](#), e.g., LDAP, needs to be configured.
- **Logfiles:** App logs, e.g., auditing and owncloud.log, can now be split, see: https://doc.owncloud.org/server/latest/admin_manual/configuration/server/config_sample_php_parameters.html#logging.

Known Issues

Converting the Database Type doesn't work

Converting a Database from e.g. SQLite to MySQL or PostgreSQL with the occ db:convert-type currently doesn't work. See <https://github.com/owncloud/core/issues/27075> for more info.

Installing the LDAP user backend will trigger the installation twice

This causes an SQL error such as the following:

```
sudo -u www-data ./occ market:install user_ldap
user_ldap: Installing new app ...
user_ldap: An exception occurred while executing 'CREATE TABLE `ldap_user_mapping` (`ldap_dn
SQLSTATE[42S01]: Base table or view already exists: 1050 Table 'ldap_user_mapping' already e
```

This can be safely ignored. And the app can be used after enabling it. Please be aware that when upgrading an existing ownCloud installation that already has user_ldap this error will not occur. It was fixed by <https://github.com/owncloud/core/pull/27982>. However, this could happen for other apps as well that use database.xml. If it does please use the same workaround.

SAML authentication only works for users synced with occ user:sync

We will re-enable SSO for LDAP users with an update of the app in the market after completing internal testing.

The web UI prevents uninstalling apps marked as shipped, e.g., user_ldap

To uninstall, disable the app with occ and rm the app directory.

Moving files around in external storages outside of ownCloud will invalidate the metadata

All shares, comments, and tags on the moved files will be lost.

Existing LDAP users only show up in the user management page and the share dialog after being synced

The account table introduced in ownCloud 10.0.0 significantly reduces LDAP communication overhead. Password checks are yet to be accounted for. LDAP user metadata in the account table will be updated when users log in or when the administrator runs occ user:sync "OCA\User_LDAP\User_Proxy". We recommend setting up a nightly Cron job to keep metadata of users not actively logging in up to date.

Error pages will not use the configured theme but will instead fall back to the community default

Changes in 10.0.0

- PHP 7.1 support added (supported PHP versions are 5.6 and 7.0+)
- The upgrade migration test has been removed; see Test the Upgrade. (Option "--skip-migration-tests" removed from update command)
- Requires to use the latest desktop client version 2.3
- Third party apps are not disabled anymore when upgrading
- User account table has been reworked. CRON job for syncing with e.g. LDAP needs to be configured (see https://doc.owncloud.com/server/latest/admin_manual/configuration/server/occ_command.html#syncing-user-accounts)
 - LDAP app is not released with ownCloud 10.0.0 and will be released on the marketplace after some more QA
 - files_drop app is not shipped anymore as it's integrated with core now. Since migrations are not possible you will have to reconfigure your drop folders (in the 'Public Link' section of the sharing dialog of the respective folders).
 - SAML/Shibboleth with device-specific app passwords: No migration possible; Users need to regenerate device-specific app passwords in the WebUI and enter those in their clients.
 - For security reasons status.php can now be configured in config.php to not return server version information anymore ('version.hide'; default 'false'). As clients still depend on version information this is not yet recommended. The default will change to 'true' with 10.0.2 once clients are ready.
 - Order of owncloud.log entries changed a bit, please review any application (e.g. fail2ban rules) relying on this file

• External storages

- FTP external storage moved to a separate app (https://marketplace.owncloud.com/apps/files_external_ftp)
- "Local" storage type can now be disabled by sysadmin in config.php (to prevent users mounting the local file system)

Full changelog: <https://github.com/owncloud/core/wiki/ownCloud-10.0-Features>

Changes in 9.1

General

- Background jobs (cron) can now run in parallel
- Update notifications in client via API - You can now be notified in your desktop client about available updates for core and apps. The notifications are made available via the notifications API.
- Multi-bucket support for primary objectstore integration
- Support for Internet Explorer below version 11 was dropped
- Symlinks pointing outside of the data directory are disallowed. Please use the Configuring External Storage (GUI) with the Local storage backend instead.
- Removed dav:migrate-calendars and dav:migrate-addressbooks commands for occ. Users planning to upgrade from ownCloud 9.0 or below to ownCloud 9.1 needs to make sure that their calendars and address books are correctly migrated **before** continuing to upgrade to 9.1.

Authentication

- Pluggable authentication: plugin system that supports different authentication schemes
- Token-based authentication
- Ability to invalidate sessions

Table of Contents

- List connected browsers/devices in the personal settings page. Allows the user to disconnect browsers/devices.
- Device-specific passwords/tokens, can be generated in the personal page and revoked
- Disable users and automatically revoke their sessions
- Detect disabled LDAP users or password changes and revoke their sessions
- Log in with email address
- Configuration option to enforce token-based login outside the web UI
- Two Factor authentication plug-in system
- OCC command added to (temporarily) disable/enable two-factor authentication for single users

Note

The current desktop and mobile client versions do not support two-factor yet, this will be added later. It is already possible to generate a device specific password and enter that in the current client versions.

Files app

- Ability to toggle displaying hidden files
- Remember sort order
- Permalinks for internal shares
- Visual cue when dragging in files app
- Autoscroll file list when dragging files
- Upload progress estimate

Federated sharing

- Ability to create federated shares with CRUDS permissions
- Resharing a federated share does not create a chain of shares any more but connects the share owner's server to the reshare recipient

External storage

- UTF-8 NFD encoding compatibility support for NFD file names stored directly on external storages (new mount option in external storage admin page)
- Direct links to the configuration pages for setting up a GDrive or Dropbox application for use with ownCloud
- Some performance and memory usage improvements for GDrive, stream download and chunk upload
- Performance and memory usage improvements for Dropbox with stream download
- GDrive library update provides exponential backoff which will reduce rate limit errors

Shibboleth

- The WebDAV endpoint was changed from /remote.php/webdav to /remote.php/dav. You need to check your Apache configuration if you have exceptions or rules for WebDAV configured.

Minor additions

- Support for print style sheets
- Command line based update will now be suggested if the instance is bigger to avoid potential timeouts
- Web updater will be disabled if LDAP or shibboleth are installed
- DB/application update process now shows better progress information
- Added occ files:scan --unscanned to only scan folders that haven't yet been explored on external storages

Table of Contents

- Chunk cache TTL can now be configured
- Added warning for wrongly configured database transactions, helps prevent “database is locked” issues
- Use a capped memory cache to reduce memory usage especially in background jobs and the file scanner
- Allow login by email
- Respect CLASS property in calendar events
- Allow addressbook export using VCFExportPlugin
- Birthdays are also generated based on shared addressbooks

For developers

- New DAV endpoint with a new chunking protocol aiming to solve many issues like timeouts (not used by clients yet)
- New webdav property for share permissions
- Background repair steps can be specified info.xml
- Background jobs (cron) can now be declared in info.xml
- Apps can now define repair steps to run at install/uninstall time
- Export contact images via Sabre DAV plugin
- Sabre DAV’s browser plugin is available in debug mode to allow easier development around webdav

Technical debt

- PSR-4 autoloading forced for OC\ and OCP\, optional for OCA\ docs at https://doc.owncloud.org/server/latest/developer_manual/app/classloader.html
- More cleanup of the sharing code (ongoing)

Changes in 9.0

9.0 requires .ico files for favicons. This will change in 9.1, which will use .svg files. See [Changing favicon](#) in the Developer Manual.

Home folder rule is enforced in the user_ldap application in new ownCloud installations; see User Authentication with LDAP. This affects ownCloud 8.0.10, 8.1.5 and 8.2.0 and up.

The Calendar and Contacts apps have been rewritten and the CalDAV and CardDAV backends of these apps were merged into ownCloud core. During the upgrade existing Calendars and Addressbooks are automatically migrated (except when using the IMAP user backend). As a fallback for failed upgrades, when using the IMAP user backend or as an option to test a migration `dav:migrate-calendars` and/or `dav:migrate-addressbooks` scripts are available (**only in ownCloud 9.0**) via the `occ` command. See Using the `occ` Command.

Warning

After upgrading to ownCloud 9.0 and **before** continuing to upgrade to 9.1 make sure that all of your and your users Calendars and Addressbooks are migrated correctly. Especially when using the IMAP user backend (other user backends might be also affected) you need to manually run the mentioned `occ` migration commands described above.

Updates on systems with large datasets will take longer, due to the addition of checksums to the ownCloud database. See <https://github.com/owncloud/core/issues/22747>.

Linux packages are available from our [official download repository](#). New in 9.0: split packages. `owncloud` installs ownCloud plus dependencies, including Apache and PHP. `owncloud-files` installs only ownCloud. This is useful for custom LAMP stacks, and allows you to install your own LAMP apps and versions without packaging conflicts with ownCloud. See Linux Package Manager Installation.

New option for the ownCloud admin to enable or disable sharing on individual external mountpoints (see Mount Options). Sharing on such mountpoints is disabled by default.

Enterprise 9.0

owncloud-enterprise packages are no longer available for CentOS 6, RHEL6, Debian 7, or any version of Fedora. A new package, owncloud-enterprise-files, is available for all supported platforms, including the above. This new package comes without dependencies, and is installable on a larger number of platforms. System administrators must install their own LAMP stacks and databases. See <https://owncloud.org/blog/time-to-upgrade-to-owncloud-9-0/>

Changes in 8.2

New location for Linux package repositories; ownCloud admins must manually change to the new repos. See How to Upgrade Your ownCloud Server

PHP 5.6.11+ breaks the LDAP wizard with a 'Could not connect to LDAP' error. See <https://github.com/owncloud/core/issues/20020>.

`filesystem_check_changes` in `config.php` is set to 0 by default. This prevents unnecessary update checks and improves performance. If you are using external storage mounts such as NFS on a remote storage server, set this to 1 so that ownCloud will detect remote file changes.

XSendFile support has been removed, so there is no longer support for serving static files from your ownCloud server.

LDAP issue: 8.2 uses the `memberof` attribute by default. If this is not activated on your LDAP server your user groups will not be detected, and you will see this message in your ownCloud log: Error PHP Array to string conversion at /var/www/html/owncloud/lib/private/template/functions.php#36. Fix this by disabling the `memberof` attribute on your ownCloud server with the `occ` command, like this example on Ubuntu Linux:

```
sudo -u www-data php occ ldap:set-config "s01" useMemberOfToDetectMembership 0
```

Run `sudo -u www-data php occ ldap:show-config` to find the correct `sNN` value; if there is not one then use empty quotes, " ". (See Using the `occ` Command.)

Users of the Linux Package need to update their repository setup as described in this [blogpost](#).

Changes in 8.1

Use APCu only if available in version 4.0.6 and higher. If you install an older version, you will see a APCu below version 4.0.6 is installed, for stability and performance reasons we recommend to update to a newer APCu version warning on your ownCloud admin page.

SMB external storage now based on `php5-libsmbclient`, which must be downloaded from the ownCloud software repositories ([installation instructions](#)).

"Download from link" feature has been removed.

The `.htaccess` and `index.html` files in the `data/` directory are now updated after every update. If you make any modifications to these files they will be lost after updates.

The SabreDAV browser at `/remote.php/webdav` has been removed.

Using ownCloud without a `trusted_domain` configuration will not work anymore.

The logging format for failed logins has changed and considers now the proxy configuration in `config.php`.

A default set of security and privacy HTTP headers have been added to the ownCloud `.htaccess` file, and ownCloud administrators may now customize which headers are sent.

More strict SSL certificate checking improves security but can result in "cURL error 60: SSL certificate problem: unable to get local issuer certificate" errors with certain broken PHP versions. Please verify your SSL setup, update your PHP or contact your vendor if you receive these errors.

The persistent file-based cache (e.g. used by LDAP integration) has been dropped and replaced with a memory-only cache, which must be explicitly configured. See User Authentication with LDAP. Memory cache configuration for the

Table of Contents

ownCloud server is no longer automatic, requiring installation of your desired cache backend and configuration in `config.php` (see Memory Caching.)

The `OC_User_HTTP` backend has been removed. Administrators are encouraged to use the `user_webdavauth` application instead.

ownCloud ships now with its own root certificate bundle derived from Mozilla's root certificates file. The system root certificate bundle will not be used anymore for most requests.

When you upgrade from ownCloud 8.0, with encryption enabled, to 8.1, you must enable the new encryption backend and migrate your encryption keys. See Encryption migration to ownCloud 8.0.

Encryption can no longer be disabled in ownCloud 8.1. It is planned to re-add this feature to the command line client for a future release.

It is not recommended to upgrade encryption-enabled systems from ownCloud Server 8.0 to version 8.1.0 as there is a chance the migration will break. We recommend migrating to the first bugfix release, ownCloud Server 8.1.1.

Due to various technical issues, by default desktop sync clients older than 1.7 are not allowed to connect and sync with the ownCloud server. This is configurable via the `minimum_supported_desktop_version` switch in `config.php`.

Previews are now generated at a maximum size of 2048 x 2048 pixels. This is configurable via the `preview_max_x` and `preview_max_y` switches in `config.php`.

The ownCloud 8 server is not supported on any version of Windows.

The 8.1.0 release has a minor bug which makes application updates fail at first try. Reload the apps page and try again, and the update will succeed.

The `forceSSL` option within the `config.php` and the `Enforce SSL` option within the Admin-Backend was removed. This now needs to be configured like described in Use HTTPS.

WebDAV file locking was removed in ownCloud 8.1 which causes Finder on Mac OS X to mount WebDAV read-only.

Enterprise 8.1

The SharePoint Drive application does not verify the SSL certificate of the SharePoint server or the ownCloud server, as it is expected that both devices are in the same trusted environment.

Changes in 8.0

Manual LDAP Port Configuration

When you are configuring the LDAP user and group backend application, ownCloud may not auto-detect the LDAP server's port number, so you will need to enter it manually.

No Preview Icon on Text Files

There is no preview icon displayed for text files when the file contains fewer than six characters.

Remote Federated Cloud Share Cannot be Reshared With Local Users

When you mount a Federated Cloud share from a remote ownCloud server, you cannot re-share it with your local ownCloud users. (See Configuring Federation Sharing to learn more about federated cloud sharing)

Manually Migrate Encryption Keys after Upgrade

If you are using the Encryption application and upgrading from older versions of ownCloud to ownCloud 8.0, you must manually migrate your encryption keys. See Encryption migration to ownCloud 8.0.

Windows Server Not Supported

Windows Server is not supported in ownCloud 8.

PHP 5.3 Support Dropped

PHP 5.3 is not supported in ownCloud 8, and PHP 5.4 or better is required.

Disable Apache Multiviews

If Multiviews are enabled in your Apache configuration, this may cause problems with content negotiation, so disable Multiviews by removing it from your Apache configuration. Look for lines like this:

```
<Directory /var/www/owncloud>
Options Indexes FollowSymLinks Multiviews
```

Delete Multiviews and restart Apache.

ownCloud Does Not Follow Symlinks

ownCloud's file scanner does not follow symlinks, which could lead to infinite loops. To avoid this do not use soft or hard links in your ownCloud data directory.

No Commas in Group Names

Creating an ownCloud group with a comma in the group name causes ownCloud to treat the group as two groups.

Hebrew File Names Too Large on Windows

On Windows servers Hebrew file names grow to five times their original size after being translated to Unicode.

Google Drive Large Files Fail with 500 Error

Google Drive tries to download the entire file into memory, then write it to a temp file, and then stream it to the client, so very large file downloads from Google Drive may fail with a 500 internal server error.

Encrypting Large Numbers of Files

When you activate the Encryption application on a running server that has large numbers of files, it is possible that you will experience timeouts. It is best to activate encryption at installation, before accumulating large numbers of files on your ownCloud server.

Enterprise 8.0

Sharepoint Drive SSL Not Verified

The SharePoint Drive application does not verify the SSL certificate of the SharePoint server or the ownCloud server, as it is expected that both devices are in the same trusted environment.

No Federated Cloud Sharing with Shibboleth

Federated Cloud Sharing (formerly Server-to-Server file sharing) does not work with Shibboleth .

Direct Uploads to SWIFT do not Appear in ownCloud

When files are uploaded directly to a SWIFT share mounted as external storage in ownCloud, the files do not appear in ownCloud. However, files uploaded to the SWIFT mount through ownCloud are listed correctly in both locations.

SWIFT Objectstore Incompatible with Encryption App

The current SWIFT implementation is incompatible with any application that uses direct file I/O and circumvents the ownCloud virtual filesystem. Using the Encryption application on a SWIFT object store incurs twice as many HTTP requests and increases latency significantly.

application Store is Back

Table of Contents

The ownCloud application Store has been re-enabled in ownCloud 8. Note that third-party apps are not supported.

Changes in 7.0

Manual LDAP Port Configuration

When you are configuring the LDAP user and group backend application, ownCloud may not auto-detect the LDAP server's port number, so you will need to enter it manually.

LDAP Search Performance Improved

Prior to 7.0.4, LDAP searches were substring-based and would match search attributes if the substring occurred anywhere in the attribute value. Rather, searches are performed on beginning attributes. With 7.0.4, searches will match at the beginning of the attribute value only. This provides better performance and a better user experience.

Substring searches can still be performed by prepending the search term with “*”. For example, a search for te will find Terri, but not Nate:

```
occ ldap:search "te"
```

If you want to broaden the search to include Nate, then search for *te:

```
occ ldap:search "*te"
```

Refine searches by adjusting the User Search Attributes field of the Advanced tab in your LDAP configuration on the Admin page. For example, if your search attributes are givenName and sn you can find users by first name + last name very quickly. For example, you'll find Terri Hanson by searching for te ha. Trailing whitespaces are ignored.

Protecting ownCloud on IIS from Data Loss

Under certain circumstances, running your ownCloud server on IIS could be at risk of data loss. To prevent this, follow these steps.

- In your ownCloud server configuration file, `owncloud\config\config.php`, set `config_is_read_only` to true.
- Set the `config.php` file to read-only.
- When you make server updates `config.php` must be made writeable. When your updates are completed re-set it to read-only.

Antivirus Application Modes

The Antivirus application offers three modes for running the ClamAV anti-virus scanner: as a daemon on the ownCloud server, a daemon on a remote server, or an executable mode that calls `clamscan` on the local server. We recommend using one of the daemon modes, as they are the most reliable.

“Enable Only for Specific Groups” Fails

Some ownCloud applications have the option to be enabled only for certain groups. However, when you select specific groups they do not get access to the app.

Changes to File Previews

For security and performance reasons, file previews are available only for image files, covers of MP3 files, and text files, and have been disabled for all other filetypes. Files without previews are represented by generic icons according to their file types.

4GB Limit on SFTP Transfers

Because of limitations in `phpseclib`, you cannot upload files larger than 4GB over SFTP.

"Not Enough Space Available" on File Upload

Setting user quotas to `unlimited` on an ownCloud installation that has unreliable free disk space reporting—for example, on a shared hosting provider—may cause file uploads to fail with a “Not Enough Space Available” error. A workaround is to set file quotas for all users instead of `unlimited`.

No More Expiration Date On Local Shares

In older versions of ownCloud, you could set an expiration date on both local and public shares. Now you can set an expiration date only on public shares, and local shares do not expire when public shares expire.

Zero Quota Not Read-Only

Setting a user’s storage quota should be the equivalent of read-only, however, users can still create empty files.

Enterprise 7.0

No Federated Cloud Sharing with Shibboleth

Federated Cloud Sharing (formerly Server-to-Server file sharing) does not work with Shibboleth .

Windows Network Drive

Windows Network Drive runs only on Linux servers because it requires the Samba client, which is included in all Linux distributions.

`php5-libsmbclient` is also required, and there may be issues with older versions of `libsmbclient`; see Using External Storage > Installing and Configuring the Windows Network Drive application in the Enterprise Admin manual for more information.

By default CentOS has activated SELinux, and the `httpd` process can not make outgoing network connections. This will cause problems with curl, LDAP and samba libraries. Again, see Using External Storage > Installing and Configuring the Windows Network Drive application in the Enterprise Admin manual for instructions.

Sharepoint Drive SSL

The SharePoint Drive application does not verify the SSL certificate of the SharePoint server or the ownCloud server, as it is expected that both devices are in the same trusted environment.

Shibboleth and WebDAV Incompatible

Shibboleth and standard WebDAV are incompatible, and cannot be used together in ownCloud. If Shibboleth is enabled, the ownCloud client uses an extended WebDAV protocol

No SQLite

SQLite is no longer an installation option for ownCloud Enterprise Edition, as it not suitable for multiple-user installations or managing large numbers of files.

No Application Store

The application Store is disabled for the Enterprise Edition.

LDAP Home Connector Linux Only

The LDAP Home Connector application requires Linux (with MySQL, MariaDB, or PostgreSQL) to operate correctly.

What's New in ownCloud 10.0.7

See the [ownCloud 10.0 Features page](#) on Github for a comprehensive list of new features and updates.

Installation

System Requirements

Officially Recommended & Supported Options

For best performance, stability, support, and full functionality we officially recommend and support:

Server

Platform	Options
Operating System	Ubuntu 16.04, 17.04 and 17.10; Debian 7, 8 and 9; SUSE Linux Enterprise Server 12 with SP1, SP2 and SP3; Red Hat Enterprise Linux/Centos 6.9, 7.3 and 7.4; Fedora 26, 27 and 28; openSuse Tumbleweed and Leap 42.1, 42.2, 42.3
Database	MySQL or MariaDB 5.5+, Oracle 11g, PostgreSQL, & SQLite
Web server	Apache 2.4 with prefork Multi-Processing Module (MPM) and mod_php
PHP Runtime	PHP (5.6+, 7.0, & 7.1)

Important

For the future release of ownCloud 10.1, a minimum PHP version of 7.1 is needed. If you use Ubuntu 16.04:

- PHP 7.1 is only available via PPA. To add a PPA to your system, use this command:
`sudo add-apt-repository ppa:user/ppa-name.`
- PHP 7.2 standard installable, but you have to install some mandatory modules yourself, like [intl](#).

Note

- Red Hat Enterprise Linux & Centos 7 are 64-bit only.
- Oracle 11g is only supported for the Enterprise edition.
- SQLite is not encouraged for production use.

Mobile

- iOS 9.0+
- Android 4.0+

Web Browser

- Edge (current version on Windows 10)
- IE11+ (except Compatibility Mode)
- Firefox 57+ or 52 ESR
- Chrome 64+
- Safari 10+

Hypervisors

Installation

- Hyper-V
- VMware ESX
- Xen
- KVM

Desktop

- Windows 7+
- Mac OS X 10.7+ (**64-bit only**)
- CentOS 6 & 7 (64-bit only)
- Debian 7.0 & 8.0 & 9.0
- Fedora 26 & 27
- Ubuntu 16.04 & 17.04 & 17.10
- openSUSE Leap 42.2 & 42.3

Note

For Linux distributions, we support, if technically feasible, the latest 2 versions per platform and the previous [LTS](#).

Alternative (But Unsupported) Options

If you are not able to use one or more of the above tools, the following options are also available.

Web Server

- NGINX with PHP-FPM

Memory Requirements

Memory requirements for running an ownCloud server are greatly variable, depending on the numbers of users and files, and volume of server activity. ownCloud officially requires a minimum of 128MB RAM. But, we recommend a minimum of 512MB.

Note

Consideration for low memory environments

Scanning of files is committed internally in 10k files chunks. Based on tests, server memory usage for scanning greater than 10k files uses about 75MB of additional memory.

Database Requirements

The following are currently required if you're running ownCloud together with a MySQL or MariaDB database:

- Disabled or `BINLOG_FORMAT = MIXED` or `BINLOG_FORMAT = ROW` configured Binary Logging (See: MySQL / MariaDB with Binary Logging Enabled)
- InnoDB storage engine (The MyISAM storage engine is not supported, see: MySQL / MariaDB storage engine)
- “READ COMMITTED” transaction isolation level (See: MySQL / MariaDB “READ COMMITTED” transaction isolation level)

Deployment Recommendations

What is the best way to install and maintain ownCloud? The answer to that is, as always: “*it depends*”.

This is because every ownCloud customer has their own particular needs and IT infrastructure. However, both ownCloud and the LAMP stack are highly-configurable. Given that, in this document we present a set of general recommendations, followed by three typical scenarios, and finish up with making best-practice recommendations for both software and hardware.

Note

The recommendations presented here are based on a standard ownCloud installation, one without any particular apps, *themes*, or *code changes*. But, server load is dependent upon the number of *clients*, *files*, and *user activity*, as well as other usage patterns. Given that, these recommendations are only a rule of thumb based on our experience, as well as that of one of our customers.

General Recommendations

- Operating system: Linux.
- Web server: Apache 2.4.
- Database: MySQL/MariaDB with InnoDB storage engine (MyISAM is not supported, see: MySQL / MariaDB storage engine)
- PHP 5.6+.
- Consider setting up a scale-out deployment, or using [Federated Cloud Sharing](#) to keep individual ownCloud instances to a manageable size.

Note

Whatever the size of your organization, always keep one thing in mind: *The amount of data stored in ownCloud will only grow. So plan ahead.*

ownCloud Administrators Must Have Command Line or Cron Access

We only recommend using hosts that provide command-line or Cron access (ideally both) to *ownCloud administrators*, for three key reasons:

1. Without command-line access, OCC commands, required for administrative tasks such as repairs and upgrades, are not available.
2. Without crontab access, you cannot run background jobs reliably. ajax/cron.php is available, but it is not reliable enough, because it only runs when people are using the web UI. Additionally, ownCloud relies heavily on background jobs especially for long-running operations, which will likely cause PHP timeouts.
3. PHP timeout values are often low. Having low timeout settings can break long-running operations, such as moving a huge folder.

Scenario 1: Small Workgroups and Departments

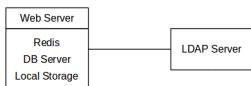
This recommendation applies if you meet the following criteria:

Option	Value
Number of users	Up to 150 users
Storage size	100 GB to 10TB

High availability level	Zero-downtime backups via Btrfs snapshots, component failure leads to interruption of service. Alternate backup scheme on other filesystems: nightly backups — with service interruption.
-------------------------	---

Recommended System Requirements

One machine running the application, web, and database server, as well as local storage. Authentication via an existing LDAP or Active Directory server.



Components

One server with at least 2 CPU cores, 16GB RAM, and local storage as needed.

Operating system

Enterprise-grade Linux distribution with full support from an operating system vendor. We recommend both RedHat Enterprise Linux and SUSE Linux Enterprise Server 12.

SSL Configuration

The SSL termination is done in Apache. A standard SSL certificate is required to be installed according to [the official Apache documentation](#).

Load Balancer

None.

Database

MySQL, MariaDB, or PostgreSQL. We currently recommend MySQL / MariaDB, as our customers have had good experiences when moving to a Galera cluster to scale the DB. If using either MySQL or MariaDB, you must use the InnoDB storage engine as MyISAM is not supported, see: MySQL / MariaDB storage engine

Warning

If you are using MaxScale/Galera, then you need to use at least version 1.3.0. In earlier versions, there is a bug where the value of `last_insert_id` is not routed to the master node. This bug can cause loops within ownCloud and corrupt database rows. You can find out more information [in the issue documentation](#).

Backup

Install ownCloud, the ownCloud data directory, and database on a [Btrfs filesystem](#). Make regular snapshots at desired intervals for zero downtime backups. Mount DB partitions with the “nodatacow” option to prevent fragmentation.

Alternatively, you can make nightly backups — with service interruption — as follows:

1. Shut down Apache.
2. Create database dump.
3. Push data directory to backup.
4. Push database dump to backup.
5. Start Apache.

Installation

After these steps have been completed, then, optionally, rsync the backup to either an external backup storage or tape backup. See the [Maintenance](#) section of the Administration manual for tips on backups and restores.

Authentication

User authentication via one or several LDAP or Active Directory (AD) servers. See [User Authentication with LDAP](#) for information on configuring ownCloud to use LDAP and AD.

Session Management

Local session management on the application server. PHP sessions are stored in a temporary filesystem, mounted at the operating system-specific session storage location. You can find out where that is by running `grep -R 'session.save_path' /etc/php5` and then add it to the `/etc/fstab` file, for example:

```
echo "tmpfs /var/lib/php5/pool-www tmpfs defaults,noatime,mode=1777 0 0" >> /etc/fstab``.
```

Memory Caching

A memory cache speeds up server performance, and ownCloud supports four of them. Refer to [Configuring Memory Caching](#) for information on selecting and configuring a memory cache.

Storage

Local storage.

ownCloud Edition

Standard Edition. See [ownCloud Server](#) or [Enterprise Edition](#) for comparisons of the ownCloud editions.

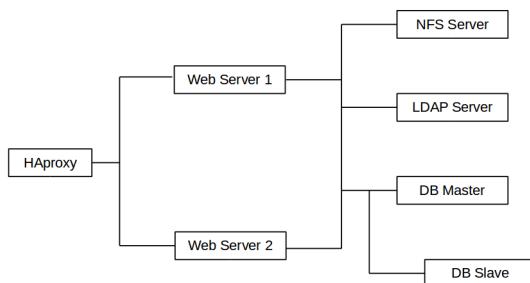
Scenario 2: Mid-Sized Enterprises

These recommendations apply if you meet the following criteria:

Option	Value
Number of users	150 to 1,000 users.
Storage size	Up to 200TB.
High availability level	Every component is fully redundant and can fail without service interruption. Backups without service interruption

Recommended System Requirements

- 2 to 4 application servers.
- A cluster of two database servers.
- Storage on an NFS server.
- Authentication via an existing LDAP or Active Directory server.



Components

Installation

- 2 to 4 application servers with four sockets and 32GB RAM.
- 2 DB servers with four sockets and 64GB RAM.
- 1 [HAProxy load balancer](#) with two sockets and 16GB RAM.
- NFS storage server as needed.

Operating System

Enterprise grade Linux distribution with full support from an operating system vendor. We recommend both RedHat Enterprise Linux and SUSE Linux Enterprise Server 12.

SSL Configuration

The SSL termination is done in the [HAProxy load balancer](#). A standard SSL certificate is needed, installed according to the [HAProxy documentation](#).

Load Balancer

HAProxy running on a dedicated server in front of the application servers. Sticky session needs to be used because of local session management on the application servers.

Database

MySQL/MariaDB Galera cluster with [master-master replication](#). InnoDB storage engine, MyISAM is not supported, see: MySQL / MariaDB storage engine.

Backup

Minimum daily backup without downtime. All MySQL/MariaDB statements should be replicated to a backup MySQL/MariaDB slave instance.

- Create a snapshot on the NFS storage server.
- At the same time stop the MySQL replication.
- Create a MySQL dump of the backup slave.
- Push the NFS snapshot to the backup.
- Push the MySQL dump to the backup.
- Delete the NFS snapshot.
- Restart MySQL replication.

Authentication

User authentication via one or several LDAP or Active Directory servers. See [User Authentication with LDAP](#) for information on configuring ownCloud to use LDAP and AD.

Session Management

Session management on the application server. PHP sessions are stored in a temporary filesystem, mounted at the operating system-specific session storage location. You can find out where that is by running `grep -R 'session.save_path' /etc/php5` and then add it to the `/etc/fstab` file, for example:

```
echo "tmpfs /var/lib/php5/pool-www tmpfs defaults,noatime,mode=1777 0 0" >> /etc/fstab
```

Memory Caching

A memory cache speeds up server performance, and ownCloud supports four memory cache types. Refer to [Configuring Memory Caching](#) for information on selecting and configuring a memory cache.

Storage

Installation

Use an off-the-shelf NFS solution, such as [IBM Elastic Storage](#) or [RedHat Ceph](#).

ownCloud Edition

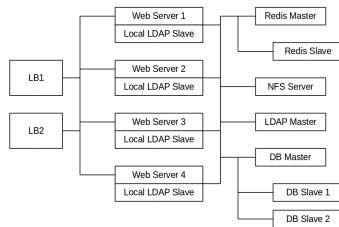
Enterprise Edition. See [ownCloud Server](#) or [Enterprise Edition](#) for comparisons of the ownCloud editions.

Scenario 3: Large Enterprises and Service Providers

Option	Value
Number of users	5,000 to >100,000 users.
Storage size	Up to 1 petabyte.
High availability level	Every component is fully redundant and can fail without service interruption. Backups without service interruption.

Recommended System Requirements

- 4 to 20 application/Web servers.
- A cluster of two or more database servers.
- Storage is an NFS server or an object store that is S3 compatible.
- Cloud federation for a distributed setup over several data centers.
- Authentication via an existing LDAP or Active Directory server, or SAML.



Components

- 4 to 20 application servers with four sockets and 64GB RAM.
- 4 DB servers with four sockets and 128GB RAM.
- 2 Hardware load balancer, for example, [BIG IP](#) from F5.
- NFS storage server as needed.

Operating system

RHEL 7 with latest service packs.

SSL Configuration

The SSL termination is done in the load balancer. A standard SSL certificate is needed, installed according to the load balancer documentation.

Load Balancer

A redundant hardware load-balancer with heartbeat, for example, F5 Big-IP. This runs two load balancers in front of the application servers.

Database

Installation

MySQL/MariaDB Galera Cluster with 4x master-master replication. InnoDB storage engine, MyISAM is not supported, see: MySQL / MariaDB storage engine.

Backup

Minimum daily backup without downtime. All MySQL/MariaDB statements should be replicated to a backup MySQL/MariaDB slave instance. To do this, follow these steps:

1. Create a snapshot on the NFS storage server.
2. At the same time stop the MySQL replication.
3. Create a MySQL dump of the backup slave.
4. Push the NFS snapshot to the backup.
5. Push the MySQL dump to the backup.
6. Delete the NFS snapshot.
7. Restart MySQL replication.

Authentication

User authentication via one or several LDAP or Active Directory servers, or SAML/Shibboleth. See [User Authentication with LDAP](#) and [Shibboleth Integration](#).

LDAP

Read-only slaves should be deployed on every application server for optimal scalability.

Session Management

Redis should be used for the session management storage.

Caching

Redis for distributed in-memory caching, see [Configuring Memory Caching](#).

Storage

An off-the-shelf NFS solution should be used. Some examples are [IBM Elastic Storage](#) or [RedHat Ceph](#). Optionally, an S3 compatible object store can also be used.

ownCloud Edition

Enterprise Edition. See [ownCloud Server](#) or [Enterprise Edition](#) for comparisons of the ownCloud editions.

Redis Configuration

Redis in a master-slave configuration is a [hot failover setup](#), and is usually sufficient. A slave can be omitted if high availability is provided via other means. And when it is, in the event of a failure, restarting Redis typically occurs quickly enough. Regarding Redis cluster, we don't, usually, recommend it, as it requires a greater level of both maintenance and management in the case of failure. A single Redis server, however, just needs to be rebooted, in the event of failure.

Known Issues

Deadlocks When Using MariaDB Galera Cluster

If you're using [MariaDB Galera Cluster](#) with your ownCloud installation, you may encounter deadlocks when you attempt to sync a large number of files. You may also encounter database errors, such as this one:

```
SQLSTATE[40001]: Serialization failure: 1213 Deadlock found when trying to get lock; try res
```

The issue, identified by Michael Roth, is caused when MariaDB Galera cluster sends write requests to all servers in the cluster; [here is a detailed explanation](#). The solution is to send all write requests to a single server, instead of all of them.

Set `wsrep_sync_wait` to 1 on all Galera Cluster nodes

What the parameter does

When enabled, the node triggers causality checks in response to certain types of queries. During the check, the node blocks new queries while the database server catches up with all updates made in the cluster to the point where the check began. Once it reaches this point, the node executes the original query.

Why enable it

A Galera Cluster write operation is sent to the master while reads are retrieved from the slaves. Since Galera Cluster replication is, by default, not strictly synchronous it could happen that items are requested before the replication has actually taken place.

Note

This setting is disabled by default.

Note

See [the Galera Cluster WSREP documentation](#) for more details.

References

- [Database High Availability](#)
- [Performance enhancements for Apache and PHP](#)
- [How to Set Up a Redis Server as a Session Handler for PHP on Ubuntu 14.04](#)

Deployment Considerations

Hardware

- Solid-state drives (SSDs) for I/O.
- Separate hard disks for storage and database, SSDs for databases.
- Multiple network interfaces to distribute server synchronisation and backend traffic across multiple subnets.

Single Machine / Scale-Up Deployment

The single-machine deployment is widely used in the community.

Pros:

- Easy setup: no session storage daemon, use tmpfs and memory caching to enhance performance, local storage.
- No network latency to consider.
- To scale buy a bigger CPU, more memory, larger hard drive, or additional hard drives.

Installation

Cons:

- Fewer high availability options.
- The amount of data in ownCloud tends to continually grow. Eventually a single machine will not scale; I/O performance decreases and becomes a bottleneck with multiple up- and downloads, even with solid-state drives.

Scale-Out Deployment

Provider setup:

- DNS round robin to HAProxy servers (2-n, SSL offloading, cache static resources)
- Least load to Apache servers (2-n)
- Memcached/Redis for shared session storage (2-n)
- Database cluster with single Master, multiple slaves and proxy to split requests accordingly (2-n)
- GPFS or Ceph via phprados (2-n, 3 to be safe, Ceph 10+ nodes to see speed benefits under load)

Pros:

- Components can be scaled as needed.
- High availability.
- Test migrations easier.

Cons:

- More complicated to setup.
- Network becomes the bottleneck (10GB Ethernet recommended).
- Currently DB filecache table will grow rapidly, making migrations painful in case the table is altered.

What About NGINX / PHP-FPM?

Could be used instead of HAProxy as the load balancer. But on uploads stores the whole file on disk before handing it over to PHP-FPM.

A Single Master DB is Single Point of Failure, Does Not Scale

When master fails another slave can become master. However, the increased complexity carries some risks: Multi-master has the risk of split brain, and deadlocks. ownCloud tries to solve the problem of deadlocks with high-level file locking.

Software

Operating System

We are dependent on distributions that offer an easy way to install the various components in up-to-date versions. ownCloud has a partnership with RedHat and SUSE for customers who need commercial support. Canonical, the parent company of Ubuntu Linux, also offers enterprise service and support. Debian and Ubuntu are free of cost, and include newer software packages. CentOS is the community-supported free-of-cost Red Hat Enterprise Linux clone. openSUSE is community-supported, and includes many of the same system administration tools as SUSE Linux Enterprise Server.

Web server

Taking Apache and NGINX as the contenders, Apache with mod_php is currently the best option, as NGINX does not support all features necessary for enterprise deployments. Mod_php is recommended instead of PHP_FPM, because in scale-out deployments separate PHP pools are simply not necessary.

Relational Database

Installation

More often than not the customer already has an opinion on what database to use. In general, the recommendation is to use what their database administrator is most familiar with. Taking into account what we are seeing at customer deployments, we recommend MySQL/MariaDB in a master-slave deployment with a MySQL proxy in front of them to send updates to master, and selects to the slave(s).

The second best option is PostgreSQL (alter table does not lock table, which makes migration less painful) although we have yet to find a customer who uses a master-slave setup.

What about the other DBMS?

- SQLite is adequate for simple testing, and for low-load single-user deployments. It is not adequate for production systems.
- Microsoft SQL Server is not a supported option.
- Oracle DB is the de facto standard at large enterprises and is fully supported with ownCloud Enterprise Edition only.

File Storage

While many customers are starting with NFS, sooner or later that requires scale-out storage. Currently the options are GPFS or GlusterFS, or an object store protocol like S3 (supported in Enterprise Edition only) or Swift. S3 also allows access to Ceph Storage.

Session Storage

- Redis is required for transactional file locking, provides session persistence, and graphical inspection tools available.
- If you need to scale out Shibboleth you must use Memcached, as Shibboleth does not provide an interface to Redis. Memcached can also be used to scale-out shibd session storage (see [Memcache StorageService](#)).

Manual Installation on Linux

Installing ownCloud on Linux from our Open Build Service packages is the preferred method (see [Linux Package Manager Installation](#)). These are maintained by ownCloud engineers, and you can use your package manager to keep your ownCloud server up-to-date.

Note

Enterprise customers should refer to [Installing & Upgrading ownCloud Enterprise Edition](#)

If there are no packages for your Linux distribution, or you prefer installing from the source tarball, you can setup ownCloud from scratch using a classic LAMP stack (Linux, Apache, MySQL/MariaDB, PHP). This document provides a complete walk-through for installing ownCloud on Ubuntu 14.04 LTS Server with Apache and MariaDB, using [the ownCloud .tar archive](#).

- Prerequisites
- Install the Required Packages
- Configure Apache Web Server
- Enable SSL
- Run the Installation Wizard
- Set Strong Directory Permissions
- SELinux
- php.ini
- PHP-FPM
- Other Web Servers

Note

Admins of SELinux-enabled distributions such as CentOS, Fedora, and Red Hat Enterprise Linux may need to set new rules to enable installing ownCloud. See SELinux for a suggested configuration.

Prerequisites

The ownCloud tar archive contains all of the required third-party PHP libraries. As a result, no extra ones are, strictly, necessary. However, ownCloud does require that PHP has a set of extensions installed, enabled, and configured.

This section lists both the required and optional PHP extensions. If you need further information about a particular extension, please consult the relevant section of [the extensions section of the PHP manual](#).

If you are using a Linux distribution, it should have packages for all the required extensions. You can check the presence of a module by typing `php -m | grep -i <module_name>`. If you get a result, the module is present.

Required

PHP Version

PHP >= 5.6 (ideally 7.0 or above)

PHP Extensions

Name	Description
Ctype	For character type checking
cURL	Used for aspects of HTTP user authentication
DOM	For operating on XML documents through the DOM API
GD	For creating and manipulating image files in a variety of different image formats, including GIF, PNG, JPEG, WBMP, and XPM.
HASH Message Digest Framework	For working with message digests (hash).
iconv	For working with the iconv character set conversion facility.
intl	Increases language translation performance and fixes sorting of non-ASCII characters
JSON	For working with the JSON data-interchange format.
libxml	This is required for the _DOM_, _libxml_, _SimpleXML_, and _XMLWriter_ extensions to work. It requires that libxml2, version 2.7.0 or higher, is installed.
Multibyte String	For working with multibyte character encoding schemes.
OpenSSL	For symmetric and asymmetric encryption and decryption, PBKDF2, PKCS7, PKCS12, X509 and other crypto operations.
PDO	This is required for the pdo_msql function to work.
Phar	For working with PHP Archives (.phar files).
POSIX	For working with UNIX POSIX functionality.
SimpleXML	For working with XML files as objects.
XMLWriter	For generating streams or files of XML data.
Zip	For reading and writing ZIP compressed archives and the files inside them.
Zlib	For reading and writing gzip (.gz) compressed files.

Tip

The *Phar*, *OpenSSL*, and *cUrl* extensions are mandatory if you want to use [Make to setup your ownCloud environment](#), prior to running either the web installation wizard, or the command line installer.

Database Extensions

Name	Description
<code>pdo_mysql</code>	For working with MySQL & MariaDB.
<code>pgsql</code>	For working with PostgreSQL. It requires PostgreSQL 9.0 or above.
<code>sqlite</code>	For working with SQLite. It requires SQLite 3 or above. This is, usually, not recommended, for performance reasons.

Required For Specific Apps

Name	Description
<code>ftp</code>	For working with FTP storage
<code>sftp</code>	For working with SFTP storage
<code>imap</code>	For IMAP integration
<code>ldap</code>	For LDAP integration
<code>smbclient</code>	For SMB/CIFS integration

Note

SMB/Windows Network Drive mounts require the PHP module `smbclient` version 0.8.0+; see [SMB/CIFS](#).

Optional

Extension	Reason
<code>Bzip2</code>	Required for extraction of applications
<code>Fileinfo</code>	Highly recommended, as it enhances file analysis performance
<code>Mcrypt</code>	Increases file encryption performance
<code>OpenSSL</code>	Required for accessing HTTPS resources
<code>imagick</code>	Required for creating and modifying images and preview thumbnails

Recommended**For Specific Apps**

Extension	Reason
<code>Exif</code>	For image rotation in the pictures app
<code>GMP</code>	For working with arbitrary-length integers

For Server Performance

Installation

For enhanced server performance consider installing one of the following cache extensions:

- [apcu](#)
- [memcached](#)
- [redis \(>= 2.2.6+, required for transactional file locking\)](#)

See [Memory Caching](#) to learn how to select and configure a memcache.

For Preview Generation

- [avconv or ffmpeg](#)
- [OpenOffice or LibreOffice](#)

For Command Line Processing

Extension	Reason
PCNTL	Enables command interruption by pressing <code>ctrl-c</code>

Note

You don't need the WebDAV module for your Web server (i.e. Apache's `mod_webdav`), as ownCloud has a built-in WebDAV server of its own, [SabreDAV](#). If `mod_webdav` is enabled you must disable it for ownCloud. (See [Configure Apache Web Server](#) for an example configuration.)

For MySQL/MariaDB

The InnoDB storage engine is required, and MyISAM is not supported, see: [MySQL / MariaDB storage engine](#).

Install the Required Packages

Note

When Are Stable Channel Packages Updated?

Packages in the supported distributions' stable channels are not immediately updated following a release. This is because we need to make sure that the release is sufficiently stable, as many people use automatic updates. By waiting a number of business days after a tarball has been released, we are able to make this assessment, based on a number of criteria which include the submitted bug reports from systems administrators.

On Ubuntu 16.04 LTS Server

On a machine running a pristine Ubuntu 16.04 LTS server, install the required and recommended modules for a typical ownCloud installation, using Apache and MariaDB, by issuing the following commands in a terminal:

```
apt-get install -y apache2 mariadb-server libapache2-mod-php7.0 \
    openssl php-imagine php7.0-common php7.0-curl php7.0-gd \
    php7.0-imap php7.0-intl php7.0-json php7.0-ldap php7.0-mbstring \
    php7.0-mcrypt php7.0-mysql php7.0-pgsql php-smbclient php-ssh2 \
    php7.0-sqlite3 php7.0-xml php7.0-zip
```

Please note:

- `php7.0-common` provides: `ftp`, `Phar`, `posix`, `iconv`, `ctype`
- The Hash extension is available from PHP 5.1.2 by default

Installation

- `php7.0-xml` provides DOM, SimpleXML, XML, & XMLWriter
- `php7.0-zip` provides zlib

The remaining steps are analogous to the installation on Ubuntu 14.04 as shown below.

On Ubuntu 14.04 LTS Server

On a machine running a pristine Ubuntu 14.04 LTS server, install the required and recommended modules for a typical ownCloud installation, using Apache and MariaDB, by issuing the following commands in a terminal:

```
apt-get install -y wget expect apache2 mariadb-server libapache2-mod-php5 \
    libsmbclient-dev libssh2-1-dev openssl php5-imagick \
    php5-common php5-curl php5-dev php5-gd \
    php5-imap php5-intl php5-json php5-ldap \
    php5-mcrypt php5-mysql php5-pgsql php5-sqlite
```

Please note:

`libapache2-mod-php5` provides the following PHP extensions.

- ctype
- dom
- ftp
- hash
- iconv
- libxml
- mbstring
- openssl
- Phar
- posix
- SimpleXML
- xml
- xmlreader
- xmlwriter
- zip
- zlib

So if you don't see an applicable package in the list above, that's why.

Installing smbclient

To install smbclient, you can use the following script. It first installs PEAR, which at the time of writing only installs version 1.9.4. However, smbclient requires version 1.9.5. So the final two commands upgrade PEAR to version 1.9.5 and then install smbclient using Pecl.

```
#!/usr/bin/expect
spawn wget -O /tmp/go-pear.phar http://pear.php.net/go-pear.phar
expect eof

spawn php /tmp/go-pear.phar

expect "1-11, 'all' or Enter to continue:"
send "\r"
expect eof

spawn rm /tmp/go-pear.phar
```

Installation

```
pear install PEAR-1.9.5  
pecl install smbclient
```

Installing ssh2

To install ssh2, which provides sftp, you can use the following command:

```
spawn pecl install ssh2
```

Running Additional Apps?

If you are planning on running additional apps, keep in mind that you might require additional packages. See Prerequisites for details.

Note

During the installation of the MySQL/MariaDB server, you will be prompted to create a root password. Be sure to remember your password as you will need it during ownCloud database setup.

Additional Extensions

```
apt-get install -y php-apcu php-redis redis-server php7.0-ldap
```

RHEL (RedHat Enterprise Linux) 7.2

Required Extensions

```
# Enable the RHEL Server 7 repository  
subscription-manager repos --enable rhel-server-rhscl-7-eus-rpms  
  
# Install the required packages  
yum install httpd mariadb-server php55 php55-php \  
php55-php-gd php55-php-mbstring php55-php-mysqlnd
```

Optional Extensions

```
yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm \  
php-pecl-apcu redis php-pecl-redis php55-php-ldap
```

SLES (SUSE Linux Enterprise Server) 12

Required Extensions

```
zypper install apache2 apache2-mod_php5 php5-gd php5-json php5-curl \  
php5-intl php5-mcrypt php5-zip php5-zlib
```

Optional Extensions

```
zypper install php5-ldap
```

APCu

We are not aware of any officially supported APCu package for SLES 12. However, if you want or need to install it, then we suggest the following steps:

Installation

```
wget http://download.opensuse.org/repositories/server:/php:/extensions/SLE_12_SP1/ server:ph  
zypper refresh  
zypper install php5-APCu
```

Redis

The latest versions of Redis servers have shown to be incompatible with SLES 12. Therefore it is currently recommended to download and install version 2.2.7 or a previous release from: <https://pecl.php.net/package/redis>. Keep in mind that version 2.2.5 is the minimum version which ownCloud supports.

Install ownCloud

Now download the archive of the latest ownCloud version:

- Go to the [ownCloud Download Page](#).
- Go to **Download ownCloud Server > Download > Archive file for server owners** and download either the tar.bz2 or .zip archive.
- This downloads a file named owncloud-x.y.z.tar.bz2 or owncloud-x.y.z.zip (where x.y.z is the version number).
- Download its corresponding checksum file, e.g. owncloud-x.y.z.tar.bz2.md5, or owncloud-x.y.z.tar.bz2.sha256.
- Verify the MD5 or SHA256 sum:

```
md5sum -c owncloud-x.y.z.tar.bz2.md5 < owncloud-x.y.z.tar.bz2  
sha256sum -c owncloud-x.y.z.tar.bz2.sha256 < owncloud-x.y.z.tar.bz2  
md5sum -c owncloud-x.y.z.zip.md5 < owncloud-x.y.z.zip  
sha256sum -c owncloud-x.y.z.zip.sha256 < owncloud-x.y.z.zip
```

- You may also verify the PGP signature:

```
wget https://download.owncloud.org/community/owncloud-x.y.z.tar.bz2.asc  
wget https://owncloud.org/owncloud.asc  
gpg --import owncloud.asc  
gpg --verify owncloud-x.y.z.tar.bz2.asc owncloud-x.y.z.tar.bz2
```

- Now you can extract the archive contents. Run the appropriate unpacking command for your archive type:

```
tar -xjf owncloud-x.y.z.tar.bz2  
unzip owncloud-x.y.z.zip
```

- This unpacks to a single owncloud directory. Copy the ownCloud directory to its final destination. When you are running the Apache HTTP server, you may safely install ownCloud in your Apache document root:

```
cp -r owncloud /path/to/webserver/document-root
```

where /path/to/webserver/document-root is replaced by the document root of your Web server:

```
cp -r owncloud /var/www
```

On other HTTP servers, it is recommended to install ownCloud outside of the document root.

Configure Apache Web Server

On Debian, Ubuntu, and their derivatives, Apache installs with a useful configuration, so all you have to do is create a /etc/apache2/sites-available/owncloud.conf file with these lines in it, replacing the **Directory** and other file paths with your own file paths:

```
Alias /owncloud "/var/www/owncloud/"  
  
<Directory /var/www/owncloud/>  
    Options +FollowSymlinks  
    AllowOverride All  
  
<IfModule mod_dav.c>  
    Dav off
```

Installation

```
</IfModule>

SetEnv HOME /var/www/owncloud
SetEnv HTTP_HOME /var/www/owncloud

</Directory>
```

Then create a symlink to /etc/apache2/sites-enabled:

```
ln -s /etc/apache2/sites-available/owncloud.conf /etc/apache2/sites-enabled/owncloud.conf
```

Additional Apache Configurations

- For ownCloud to work correctly, we need the module `mod_rewrite`. Enable it by running:

```
a2enmod rewrite
```

Additional recommended modules are `mod_headers`, `mod_env`, `mod_dir` and `mod_mime`:

```
a2enmod headers
a2enmod env
a2enmod dir
a2enmod mime
```

- You must disable any server-configured authentication for ownCloud, as it uses Basic authentication internally for DAV services. If you have turned on authentication on a parent folder (via, e.g., an `AuthType Basic` directive), you can disable the authentication specifically for the ownCloud entry. Following the above example configuration file, add the following line in the `<Directory` section

```
Satisfy Any
```

- When using SSL, take special note of the `ServerName`. You should specify one in the server configuration, as well as in the `CommonName` field of the certificate. If you want your ownCloud to be reachable via the internet, then set both of these to the domain you want to reach your ownCloud server.

- Now restart Apache

```
service apache2 restart
```

- If you're running ownCloud in a sub-directory and want to use CalDAV or CardDAV clients make sure you have configured the correct Service discovery URLs.

Multi-Processing Module (MPM)

Apache `prefork` has to be used. Don't use a threaded MPM like `event` or `worker` with `mod_php`, because PHP is currently not thread safe.

Enable SSL

Note

You can use ownCloud over plain HTTP, but we strongly encourage you to use SSL/TLS to encrypt all of your server traffic, and to protect user's logins and data in transit.

Apache installed under Ubuntu comes already set-up with a simple self-signed certificate. All you have to do is to enable the `ssl` module and the default site. Open a terminal and run:

```
a2enmod ssl
a2ensite default-ssl
service apache2 reload
```

Note

Self-signed certificates have their drawbacks - especially when you plan to make your ownCloud server publicly accessible. You might want to consider getting a certificate signed by a commercial signing authority. Check with your domain name registrar or hosting service for good deals on commercial certificates.

Run the Installation Wizard

After restarting Apache, you must complete your installation by running either the Graphical Installation Wizard or on the command line with the `occ` command. To enable this, temporarily change the ownership on your ownCloud directories to your HTTP user (see Set Strong Directory Permissions to learn how to find your HTTP user):

```
chown -R www-data:www-data /var/www/owncloud/
```

Note

Admins of SELinux-enabled distributions may need to write new SELinux rules to complete their ownCloud installation; see SELinux.

To use `occ` see Command Line Installation. To use the graphical Installation Wizard see The Installation Wizard.

Warning

Please know that ownCloud's data directory **must be exclusive to ownCloud** and not be modified manually by any other process or user.

Set Strong Directory Permissions

After completing the installation, you must immediately set the directory permissions in your ownCloud installation as strictly as possible for stronger security. After you do so, your ownCloud server will be ready to use.

Managing Trusted Domains

All URLs used to access your ownCloud server must be whitelisted in your `config.php` file, under the `trusted_domains` setting. Users are allowed to log into ownCloud only when they point their browsers to a URL that is listed in the `trusted_domains` setting.

Note

This setting is important when changing or moving to a new domain name. You may use IP addresses and domain names.

A typical configuration looks like this:

```
'trusted_domains' => [
    0 => 'localhost',
    1 => 'server1.example.com',
    2 => '192.168.1.50',
],
```

The loopback address, `127.0.0.1`, is automatically whitelisted, so as long as you have access to the physical server you can always log in. In the event that a load-balancer is in place, there will be no issues as long as it sends the correct `X-Forwarded-Host` header.

Note

For further information on improving the quality of your ownCloud installation, please see the Configuration Notes & Tips guide.

Linux Package Manager Installation

Note

Package managers should only be used for single-server setups. For production environments, we recommend installing from [the tar archive](#).

Available Packages

The recommended package to use is `owncloud-files`. It only installs ownCloud, and does not install Apache, a database, or any of the required PHP dependencies.

Installing ownCloud Community Edition

First, install your own LAMP stack, as doing so allows you to create your own custom LAMP stack without dependency conflicts with the ownCloud package. Then, [update package manager's configuration](#).

Configurations are available for the following Linux distributions:

- Ubuntu 14.04 & 16.04
- Debian 7 & 8
- RHEL 6 & 7
- CentOS 7.2 & 7.3
- SLES 11SP4 & 12SP2
- openSUSE Leap 42.2 & 42.3

Note

Repositories for Fedora, openSUSE Tumbleweed, and Ubuntu 15.04 have been dropped. If you use Fedora, use [the tar archive](#) with your own LAMP stack. openSUSE users can rely on LEAP packages for Tumbleweed.

Once your package manager has been updated, follow the rest of the instructions on the download page to install ownCloud. Once ownCloud's installed, run the Installation Wizard to complete your installation.

Note

See the System Requirements for the recommended ownCloud setup and supported platforms.

Warning

Do not move the folders provided by these packages after the installation, as this will break updates.

What is the Correct Version?

Package versions are composed of a major, a minor, and a patch number, such as 9.0, 9.1, 10.0, 10.0.1, and 10.0.2. The second number represents a major release, and the third number represents a minor release.

Major Releases

If you want to follow either of the most recent major releases, then substitute `version` with either 9.0 or 10.0.

Minor Releases

If you want to follow any of the four most recent patch releases, then substitute `version` with one of 10.0.1, 10.0.2, 10.0.3, or 10.0.4. Following a minor release avoids you accidentally upgrading to the next major release before you're ready.

The Latest Stable Version

Alternatively you can use `stable` for the latest stable version. If you do, you never have to change it as it always tracks the current stable ownCloud version through all major releases.

Installing ownCloud Enterprise Edition

See [Installing & Upgrading ownCloud Enterprise Edition](#) for instructions on installing ownCloud Enterprise edition.

Downgrading

Downgrading is not supported and risks corrupting your data! If you want to revert to an older ownCloud version, install it from scratch and then restore your data from backup. Before doing this, file a support ticket ([if you have paid support](#)) or ask for help in the ownCloud forums to see if your issue can be resolved without downgrading.

Additional Guides and Notes

See [The Installation Wizard](#) for important steps, such as choosing the best database and setting correct directory permissions. See [SELinux Configuration](#) for a suggested configuration for SELinux-enabled distributions such as Fedora and CentOS.

If your distribution is not listed, your Linux distribution may maintain its own ownCloud packages or you may prefer to install from source.

Archlinux

The current `stable` version is in the official community repository, and more packages are in the [Arch User Repository](#).

Mageia

The [Mageia Wiki](#) has a good page on installing ownCloud from the Mageia software repository.

Note for MySQL/MariaDB environments

Please refer to [MySQL / MariaDB with Binary Logging Enabled](#) on how to correctly configure your environment if you have binary logging enabled.

Running ownCloud in a sub-directory

If you're running ownCloud in a sub-directory and want to use CalDAV or CardDAV clients, make sure you have configured the correct service discovery URLs.

The Installation Wizard

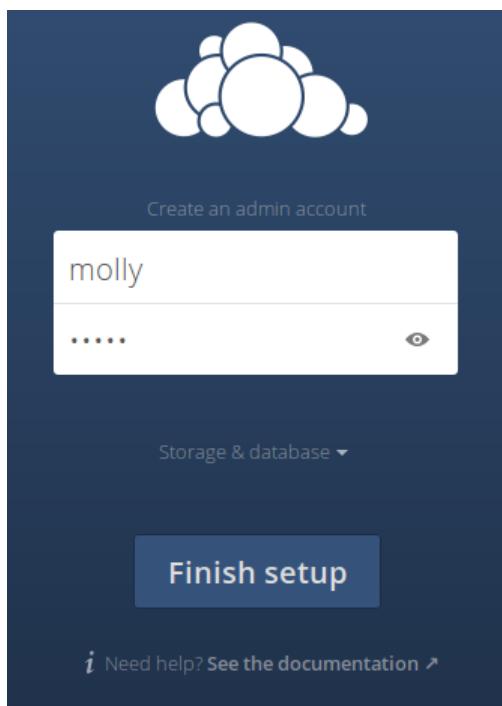
Warning

If you are planning to use the installation wizard, we **strongly** encourage you to protect it, through some form of [password authentication](#), or [access control](#). If the installer is left unprotected when exposed to the public internet, there is the possibility that a malicious actor could finish the installation and block you out — or worse. So please ensure that only you — or someone from your organization — can access the web installer.

Quick Start

When the ownCloud prerequisites are fulfilled and all ownCloud files are installed, the last step to completing the installation is running the Installation Wizard. This involves just three steps:

1. Point your web browser to `http://localhost/owncloud`
2. Enter your desired administrator's username and password.
3. Click "Finish Setup".



You're now finished and can start using your new ownCloud server. Of course, there is much more that you *can* do to set up your ownCloud server for best performance and security. In the following sections we will cover important installation and post-installation steps. Note that you must follow the instructions in [Setting Strong Permissions](#) in order to use the occ Command.

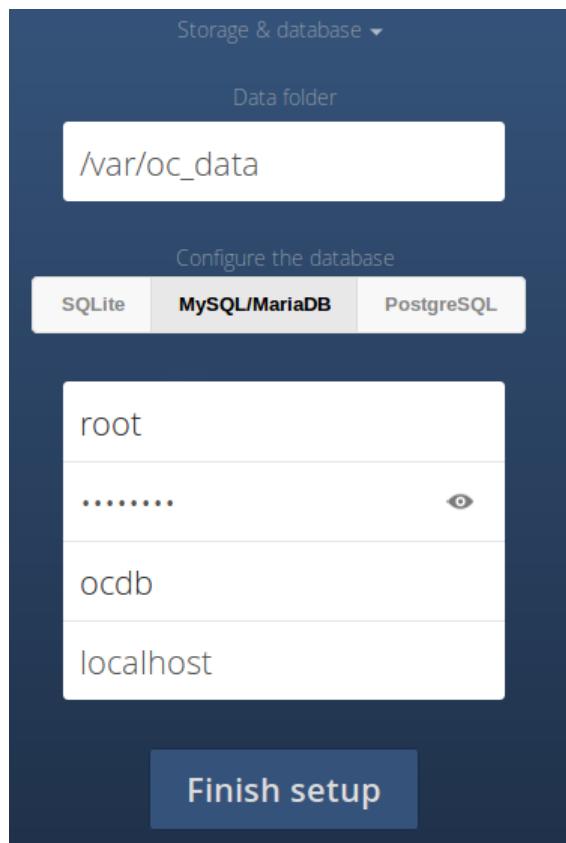
In-Depth Guide

This section provides a more detailed guide to the installation wizard. Specifically, it is broken down into three steps:

1. Data Directory Location
2. Database Choices
3. Post-Installation Steps

Data Directory Location

Click "Storage and Database" to expose additional installation configuration options for your ownCloud data directory and database.



You should locate your ownCloud data directory outside of your Web root if you are using an HTTP server other than Apache, or you may wish to store your ownCloud data in a different location for other reasons (e.g. on a storage server).

Warning

Please know that ownCloud's data directory **must be exclusive to ownCloud** and not be modified manually by any other process or user.

It is best to configure your data directory location at installation, as it is difficult to move after installation. You may put it anywhere; in this example it located in `/var/oc_data`. This directory must already exist, and must be owned by your HTTP user (see Set Strong Directory Permissions).

Database Choices

When installing ownCloud Server & ownCloud Enterprise editions the administrator may choose one of 4 supported database products. These are:

- *SQLite*
- *MySQL/MariaDB*
- *PostgreSQL*
- *Oracle 11g* (Enterprise-edition only)

SQLite

SQLite is the default database for ownCloud Server — but is not supported by the ownCloud Enterprise edition.

Note

SQLite is only good for testing and lightweight single user setups. It has no client synchronization support, so other devices will not be able to synchronize with the data stored in an ownCloud SQLite database.

SQLite will be installed by the ownCloud package and all the necessary dependencies will be satisfied. If you used the package manager to install ownCloud, you may “Finish Setup” with no additional steps to configure ownCloud using the SQLite database for limited use.

MySQL/MariaDB

MariaDB is the ownCloud recommended database. It may be used with either ownCloud Server or ownCloud Enterprise editions. To install the recommended MySQL/MariaDB database, use the following command:

```
sudo apt-get install mariadb-server
```

If you have an administrator login that has permissions to create and modify databases, you may choose “Storage & Database”. Then, enter your database administrator username and password, and the name you want for your ownCloud database. Alternatively, you can use these steps to create a temporary database administrator account.

```
sudo mysql --user=root mysql
CREATE USER 'dbadmin'@'localhost' IDENTIFIED BY 'Apassword';
GRANT ALL PRIVILEGES ON *.* TO 'dbadmin'@'localhost' WITH GRANT OPTION;
FLUSH PRIVILEGES;
exit
```

For more detailed information, see MySQL/MariaDB.

PostgreSQL

PostgreSQL is also supported by ownCloud. To install it, use the following command (or that of your preferred package manager):

```
sudo apt-get install postgresql
```

In order to allow ownCloud access to the database, create a known password for the default user, `postgres`, which was added when the database was installed.

```
sudo -i -u postgres psql
postgres=# \password
Enter new password:
Enter it again:
postgres=# \q
exit
```

Oracle 11g

Oracle 11g is only supported for the ownCloud Enterprise edition.

Database Setup By ownCloud

Your database and PHP connectors must be installed before you run the Installation Wizard by clicking the “Finish setup” button. After you enter your temporary or root administrator login for your database, the installer creates a special database user with privileges limited to the ownCloud database.

Following this, ownCloud needs only this special ownCloud database user and drops the temporary or root database login. This new user is named from your ownCloud admin user, with an `oc_` prefix, and given a random password. The ownCloud database user and password are written into `config.php`:

For MySQL/MariaDB:

```
'dbuser' => 'oc_dbadmin',
'dbpassword' => 'pX65Ty5DrHQkYPE5HRsDvyFHlZZHcm',
```

For PostgreSQL:

```
'dbuser' => 'oc_postgres',
'dbpassword' => 'pX65Ty5DrHQkYPE5HRsDvyFHlZZHcm',
```

Click Finish Setup, and you're ready to start using your new ownCloud server.

Post-Installation Steps

Now we will look at some important post-installation steps. For hardened security we recommend setting the permissions on your ownCloud directories as strictly as possible, and for proper server operations. This should be done immediately after the initial installation and before running the setup.

Your HTTP user must own the config/, data/, apps/ respectively the apps-external/ directories so that you can configure ownCloud, create, modify and delete your data files, and install apps via the ownCloud Web interface.

You can find your HTTP user in your HTTP server configuration files, or you can use PHP Version and Information (Look for the **User/Group** line).

- The HTTP user and group in Debian/Ubuntu is www-data.
- The HTTP user and group in Fedora/CentOS is apache.
- The HTTP user and group in Arch Linux is http.
- The HTTP user in openSUSE is wwwrun, and the HTTP group is www.

Note

When using an NFS mount for the data directory, do not change its ownership from the default. The simple act of mounting the drive will set proper permissions for ownCloud to write to the directory. Changing ownership as above could result in some issues if the NFS mount is lost.

The easy way to set the correct permissions is to copy and run the script, below. The script sets proper permissions and ownership including the handling of necessary directories. The script also prepares for an apps-external directory, for details see config.sample.php:

- Replace the ocpath variable with the path to your ownCloud directory.
- Replace the ocdata variable with the path to your ownCloud data directory.
- Replace the apps_external variable with the path to your ownCloud apps-external directory.

In case use want to use links for the data and apps-external directory:

- Replace the linkdata variable with the path to your ownCloud linked data directory.
- Replace the linkapps-external variable with the path to your ownCloud linked apps-external directory.

Set the correct HTTP user and group according your needs:

- Replace the htuser and htgroup variables with your HTTP user and group.

In case of upgrading using tar:

- Replace the oldocpath variable with the path to your old ownCloud directory.

```
#!/bin/bash

ocpath='/var/www/owncloud'
ocdata='/var/www/owncloud/data'
ocapps_external='/var/www/owncloud/apps-external'
oldocpath='/var/www/owncloud.old'
linkdata="/var/mylinks/data"
linkapps-external="/var/mylinks/apps-external"
htuser='www-data'
htgroup='www-data'
rootuser='root'
```

```

# Because the data directory can be huge or on external storage, an automatic chmod/chown can
# Therefore this directory can be treated differently.
# If you have already created an external data and apps-external directory which you want to
# set the paths above accordingly. This script can link and set the proper rights and permissions
# depending what you enter when running the script.
# You have to run this script twice, one time to prepare installation and one time post install

# Example input
# New install using mkdir:      n/n/n (create missing directories, setup permissions and ownership)
# Upgrade using mkdir:          n/n/n (you move/replace data, apps-external and config.php maintained)
# New install using links:      y/y/n (link existing directories, setup permissions and ownership)
# Upgrade using links:          y/n/y (link existing directories, copy config.php, permission)
# Post installation/upgrade:   either n/n/n or y/y/n
# Reset all perm & own:        either n/n/n or y/y/n

echo
read -p "Do you want to use ln instead of mkdir for creating directories (y/N)? " -r -e answer
if echo "$answer" | grep -iq "^[Yy]"; then
    uselinks="y"
else
    uselinks="n"
fi

read -p "Do you also want to chmod/chown these links (y/N)? " -r -e answer
if echo "$answer" | grep -iq "^[Yy]"; then
    chmdir="y"
else
    chmdir="n"
fi

read -p "If you upgrade, do you want to copy an existing config.php file (y/N)? " -r -e answer
if echo "$answer" | grep -iq "^[Yy]"; then
    upgrdcfg="y"
else
    upgrdcfg="n"
fi

printf "\nCreating or linking possible missing directories \n"
mkdir -p $ocpath/updater
# check if directory creation is possible and create if ok
if [ "$uselinks" = "n" ]; then
    if [ -L ${ocdata} ]; then
        echo "Symlink for $ocdata found but mkdir requested. Exiting."
        echo
        exit
    else
        echo "mkdir $ocdata"
        echo
        mkdir -p $ocdata
    fi
    if [ -L ${ocapps_external} ]; then
        echo "Symlink for $ocapps_external found but mkdir requested. Exiting."
        echo
        exit
    else
        printf "mkdir $ocapps_external \n"
        mkdir -p $ocapps_external
    fi
else

```

Installation

```
if [ -d ${ocdata} ]; then
    echo "Directory for $ocdata found but link requested. Exiting."
    echo
    exit
else
    printf "ln $ocdata \n"
    ln -sfn $linkdata $ocdata
fi
if [ -d ${ocapps_external} ]; then
    echo "Directory for $ocapps_external found but link requested. Exiting."
    echo
    exit
else
    printf "ln $ocapps_external \n"
    ln -sfn $linkapps-external $ocapps_external
fi
fi

# Copy if requested an existing config.php
if [ "$upgrdcfg" = "y" ]; then
    if [ -f ${oldocpath}/config/config.php ]; then
        printf "\nCopy existing config.php file \n"
        cp ${oldocpath}/config/config.php ${ocpath}/config/config.php
    else
        printf "Skipping copy config.php, not found: ${oldocpath}/config/config.php \n"
    fi
fi

printf "\nchmod files and directories excluding data and apps-external directory \n"
find -L ${ocpath} -path ${ocdata} -prune -o -path ${ocapps_external} -prune -o -type f -print
find -L ${ocpath} -path ${ocdata} -prune -o -path ${ocapps_external} -prune -o -type d -print

# no error messages on empty directories
if [ "$chmdir" = "n" ] && [ "$uselinks" = "n" ]; then
    printf "chmod data and apps-external directory (mkdir) \n"
    if [ -n "$(ls -A $ocdata)" ]; then
        find ${ocdata}/ -type f -print0 | xargs -0 chmod 0640
    fi
    find ${ocdata}/ -type d -print0 | xargs -0 chmod 0750
    if [ -n "$(ls -A $ocapps_external)" ]; then
        find ${ocapps_external}/ -type f -print0 | xargs -0 chmod 0640
    fi
    find ${ocapps_external}/ -type d -print0 | xargs -0 chmod 0750
fi

if [ "$chmdir" = "y" ] && [ "$uselinks" = "y" ]; then
    printf "chmod data and apps-external directory (linked) \n"
    if [ -n "$(ls -A $ocdata)" ]; then
        find -L ${ocdata}/ -type f -print0 | xargs -0 chmod 0640
    fi
    find -L ${ocdata}/ -type d -print0 | xargs -0 chmod 0750
    if [ -n "$(ls -A $ocapps_external)" ]; then
        find -L ${ocapps_external}/ -type f -print0 | xargs -0 chmod 0640
    fi
    find -L ${ocapps_external}/ -type d -print0 | xargs -0 chmod 0750
fi

printf "\nchown files and directories excluding data and apps-external directory \n"
find -L ${ocpath} -path ${ocdata} -prune -o -path ${ocapps_external} -prune -o -type d -print
find -L ${ocpath} -path ${ocdata} -prune -o -path ${ocapps_external} -prune -o -type f -print
```

```

# do only if the directories are present
if [ -d ${ocpath}/apps/ ]; then
    printf "chown apps directory \n"
    chown -R ${htuser}:${htgroup} ${ocpath}/apps/
fi
if [ -d ${ocpath}/config/ ]; then
    printf "chown config directory \n"
    chown -R ${htuser}:${htgroup} ${ocpath}/config/
fi
if [ -d ${ocpath}/updater/ ]; then
    printf "chown updater directory \n"
    chown -R ${htuser}:${htgroup} ${ocpath}/updater
fi

if [ "$chmdir" = "n" ] && [ "$uselinks" = "n" ]; then
    printf "chown data and apps-external directories (mkdir) \n"
    chown -R ${htuser}:${htgroup} ${ocapps_external}/
    chown -R ${htuser}:${htgroup} ${ocdata}/
fi
if [ "$chmdir" = "y" ] && [ "$uselinks" = "y" ]; then
    printf "chown data and apps-external directories (linked) \n"
    chown -R ${htuser}:${htgroup} ${ocapps_external}/
    chown -R ${htuser}:${htgroup} ${ocdata}/
fi

printf "\nchmod occ command to make it executable \n"
if [ -f ${ocpath}/occ ]; then
    chmod +x ${ocpath}/occ
fi

printf "chmod/chown .htaccess \n"
if [ -f ${ocpath}/.htaccess ]; then
    chmod 0644 ${ocpath}/.htaccess
    chown ${rootuser}:${htgroup} ${ocpath}/.htaccess
fi
if [ -f ${ocdata}/.htaccess ];then
    chmod 0644 ${ocdata}/.htaccess
    chown ${rootuser}:${htgroup} ${ocdata}/.htaccess
fi
echo

```

If you have customized your ownCloud installation and your file paths are different than the standard installation, modify this script accordingly.

This summary lists the recommended modes and ownership for your ownCloud directories and files:

- All files should be read-write for the file owner, read-only for the group owner, and zero for the world
- All directories should be executable (because directories always need the executable bit set), read-write for the directory owner, and read-only for the group owner
- The apps/ directory should be owned by [HTTP user]:[HTTP group]
- The apps-external/ directory should be owned by [HTTP user]:[HTTP group]
- The config/ directory should be owned by [HTTP user]:[HTTP group]
- The data/ directory should be owned by [HTTP user]:[HTTP group]
- The updater/ directory should be owned by [HTTP user]:[HTTP group]
- The [ocpath]/.htaccess file should be owned by root:[HTTP group]
- The data/.htaccess file should be owned by root:[HTTP group]

Installation

- Both .htaccess files are read-write file owner, read-only group and world

These strong permissions prevent upgrading your ownCloud server; see [Setting Permissions for Updating](#) for a script to quickly change permissions to allow upgrading.

Installing with Docker

ownCloud can be installed using Docker, using [the official ownCloud Docker image](#). This official image is designed to work with a data volume in the host filesystem and with separate *MariaDB* and *Redis* containers. The configuration:

- exposes ports 80 and 443, allowing for HTTP and HTTPS connections.
- mounts the data and MySQL data directories on the host for persistent storage.

Installation on a Local Machine

To use it, first create a new project directory and download `docker-compose.yml` from [the ownCloud Docker GitHub repository](#) into that new directory. Next, create a `.env` configuration file, which contains the required configuration settings. Only a few settings are required, these are:

Setting Name	Description	Example
VERSION	The ownCloud version	latest
DOMAIN	The ownCloud domain	localhost
ADMIN_USERNAME	The admin username	admin
ADMIN_PASSWORD	The admin user's password	admin
HTTP_PORT	The HTTP port to bind to	80
HTTPS_PORT	The HTTP port to bind to	443

Then, you can start the container, using your preferred Docker command-line tool. The example below shows how to use [Docker Compose](#).

Note

You can find instructions for using plain docker [in the GitHub repository](#).

```
# Create a new project directory
mkdir owncloud-docker-server

cd owncloud-docker-server

# Copy docker-compose.yml from the GitHub repository
wget https://raw.githubusercontent.com/owncloud/docker/master/docker-compose.yml

# Create the environment configuration file
cat << EOF > .env
VERSION=10.0.4
DOMAIN=localhost
ADMIN_USERNAME=admin
ADMIN_PASSWORD=admin
HTTP_PORT=80
HTTPS_PORT=443
EOF

# Build and start the container
docker-compose up -d
```

When the process completes, then check that all the containers have successfully started, by running `docker-compose ps`. If they are all working correctly, you should expect to see output similar to that below:

Installation

Name	Command	State	Ports
server_db_1	/usr/bin/entrypoint /bin/s ...	Up	3306/tcp
server_owncloud_1	/usr/local/bin/entrypoint ...	Up	0.0.0.0:443->443/tcp, 0.0.0.0:8080->8080/tcp
server_redis_1	/bin/s6-svscan /etc/s6	Up	6379/tcp

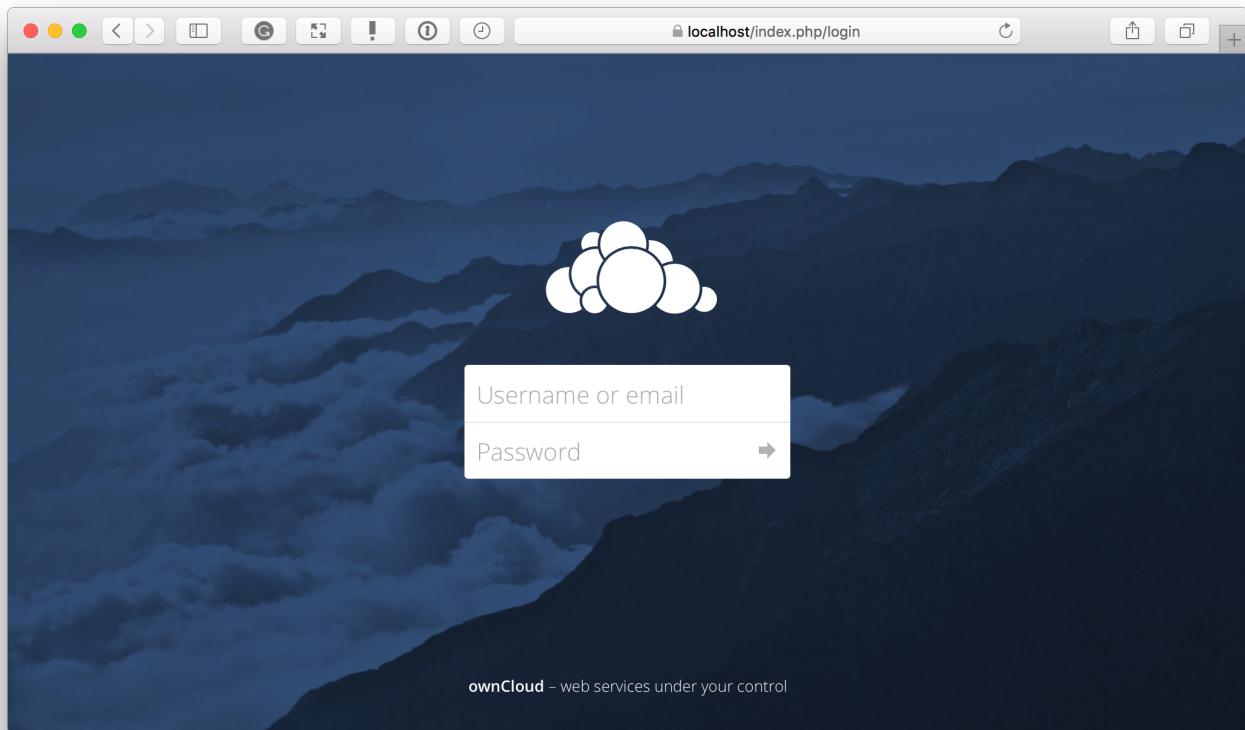
In it, you can see that the database, ownCloud, and Redis containers are running, and that ownCloud is accessible via ports 443 and 8080 on the host machine.

Note

Just because all the containers are running, it takes a few minutes for ownCloud to be fully functional. If you run `docker-compose logs --follow owncloud` and see a significant amount of information logging to the console, then please wait until it slows down to attempt to access the web UI.

Logging In

To log in to the ownCloud UI, open `https://localhost` in your browser of choice, where you see the standard ownCloud login screen, as in the image below.



The username and password are the admin username and password which you stored in `.env` earlier.

Note

The first time that you access the login page via HTTPS, a browser warning appears, as the SSL certificate in the Docker setup is self-signed. However, the self-signed certificate can be overwritten with a valid cert, within the host volume.

Stopping the Containers

Installation

Assuming you used docker-compose, as in the previous example, to stop the containers use docker-compose stop. Alternatively, use docker-compose down to stop and remove containers, along with the related networks, images, and volumes.

Upgrading ownCloud on Docker

When a new version of ownCloud gets released, you should update your instance. To do so, follow these simple steps.

First, go to your docker directory where your .yaml or .env file exists. Second, put ownCloud into maintenance mode; you can do so using the following command:

```
docker-compose exec server occ maintenance:mode --on
```

Third, create a backup in case something goes wrong during the upgrade process, using the following command:

```
docker-compose exec db backup
```

Note

This assumes that you are using [the default database container from Webhippie](#).

Fifth, shutdown the containers.

```
docker-compose down
```

Sixth, update the version number of ownCloud in your .env file or the YAML file. You can use sed for it, as in the following example.

```
# Make sure that you adjust the example to match your installation.  
sed -i 's/^owncloud_version=.*/$/owncloud_version=<newVersion>/' /compose/*/.env
```

Seventh, view the file to ensure the changes has been implemented.

```
cat .env
```

Eighth, start your docker instance again.

```
docker-compose up -d
```

Now you should have the current ownCloud running with docker-compose.

Command Line Installation

ownCloud can be installed entirely from the command line. This is convenient for scripted operations and for systems administrators who prefer using the command line over a GUI. It involves five steps:

1. Ensure your server meets the ownCloud prerequisites
2. Download and unpack the source
3. Install using the occ command
4. Set the correct owner and permissions
5. Optional post-installation considerations

Let's begin. To install ownCloud, first [download the source](#) (whether community or enterprise) directly from ownCloud, and then unpack (decompress) the tarball into the appropriate directory.

With that done, you next need to set your webserver user to be the owner of your unpacked owncloud directory, as in the example below.

```
$ sudo chown -R www-data:www-data /var/www/owncloud/
```

With those steps completed, next use the occ command, from the root directory of the ownCloud source, to perform the installation. This removes the need to run the [Graphical Installation Wizard](#). Here's an example of how to do it

```
# Assuming you've unpacked the source to /var/www/owncloud/
$ cd /var/www/owncloud/
$ sudo -u www-data php occ maintenance:install \
--database "mysql" --database-name "owncloud" \
--database-user "root" --database-pass "password" \
--admin-user "admin" --admin-pass "password"
```

Note

You must run `occ` as your HTTP user. See Run `occ` As Your HTTP User

If you want to use a directory other than the default (which is `data` inside the root ownCloud directory), you can also supply the `--data-dir` switch. For example, if you were using the command above and you wanted the data directory to be `/opt/owncloud/data`, then add `--data-dir /opt/owncloud/data` to the command.

When the command completes, apply the correct permissions to your ownCloud files and directories (see Set Strong Directory Permissions). This is extremely important, as it helps protect your ownCloud installation and ensure that it will operate correctly. See Command Line Installation for more information.

Configuration Notes & Tips

SELinux

See SELinux Configuration for a suggested configuration for SELinux-enabled distributions such as Fedora and CentOS.

php.ini

Several core PHP settings must be configured correctly, otherwise ownCloud may not work properly. Known settings causing issues are listed here. Please note that, there might be other settings which cause unwanted behavior. In general, however, it is recommended to keep the `php.ini` settings at their defaults, except when you know exactly why the change is required, and its implications.

Note

Keep in mind that, changes to `php.ini` may have to be configured in more than one ini file. This can be the case, for example, for the `date.timezone` setting.

php.ini - Used by the Web server

For PHP version 7.0 onward, replace `php_version` with the version number installed, e.g., `7.0` in the following examples.

```
/etc/php/[php_version]/apache2/php.ini  
or  
/etc/php/[php_version]/fpm/php.ini  
or ...
```

php.ini - used by the php-cli and so by ownCloud CRON jobs

```
/etc/php/[php_version]/cli/php.ini
```

session.auto_start && enable_post_data_reading

Ensure that `session.auto_start` is set to 0 or off and `enable_post_data_reading` to 1 or on in your configuration. If not, you may have issues logging in to ownCloud via the WebUI, where you see the error: “Access denied. CSRF check failed”.

`session.save_path`

In addition to setting `session.auto_start` and `enable_post_data_reading` correctly, ensure that, if `session.save_handler` is set to `files`, that `session.save_path` is set to a path on the filesystem which **only** the web server process (or process which PHP is running as) can read from and write to.

This is especially important if your ownCloud installation is using a shared-hosting arrangement. In these situations, `session poisoning` can occur if all of the session files are stored in the same location. Session poisoning is where one web application can manipulate data in the `$_SESSION` superglobal array of another.

When this happens, the original application has no way of knowing that this corruption has occurred and may not treat the data with any sense of suspicion. You can [read through a thorough discussion of local session poisoning](#) if you’d like to know more.

`suhosin.session.encryptkey`

When `suhosin.session.encryptkey` is enabled, session data will be transparently encrypted. If enabled, there is less of a concern in storing application session files in the same location, as discussed in `session.save_path`. Ideally, however, session files for each application should always be stored in a location specific to that application, and never stored collectively with any other.

Note

This is only relevant if you’re using PHP 5.x.

`post_max_size`

Please ensure that you have `post_max_size` configured with *at least* the minimum amount of memory for use with ownCloud, which is 512 MB.

Important

Please be careful when you set this value if you use the byte value shortcut as it is very specific. Use `K` for kilobyte, `M` for megabyte and `G` for gigabyte. `KB`, `MB`, and `GB` **do not work!**

`realpath_cache_size`

This determines the size of the realpath cache used by PHP. This value should be increased on systems where PHP opens many files, to reflect the number of file operations performed. For a detailed description see `realpath-cache-size`. This setting has been available since PHP 5.1.0. Prior to PHP 7.0.16 and 7.1.2, the default was 16 KB.

To see your current value, query your `phpinfo()` output for this key. It is recommended to set the value if it is currently set to the default of 16 KB. A good reading about the background can be found at [tideways.io](#).

How to get a working value

With the assumption of 112 bytes per file path needed, this would allow the cache to hold around 37.000 items with a cache size of 4096K (4M), but only about a hundred entries for a cache size of 16 KB.

Note

It's a good rule of thumb to always have a realpath cache that can hold entries for all your files paths in memory. If you use symlink deployment, then set it to double or triple the amount of files.

The easiest way to get the quantity of PHP files is to use cloc, which can be installed by running sudo apt-get install cloc. The cloc package is available for nearly all distributions.

```
sudo cloc /var/www/owncloud --exclude-dir=data --follow-links
 12179 text files.
 11367 unique files.
 73126 files ignored.

http://cloc.sourceforge.net v 1.60 T=1308.98 s (6.4 files/s, 1283.5 lines/s)
-----
Language           files      blank   comment    code
-----
PHP               4896      96509   285384   558135
...
```

Taking the math from above and assuming a symlinked instance, using factor 3. For example: $4896 * 3 * 112 = 1.6\text{MB}$ This result shows that you can run with the PHP setting of 4M two instances of ownCloud.

Having the default of 16 KB means that only 1/100 of the existing PHP file paths can be cached and need continuous cache refresh slowing down performance. If you run more web services using PHP, you have to calculate accordingly.

PHP-FPM

System Environment Variables

When you are using php-fpm, system environment variables like PATH, TMP or others are not automatically populated in the same way as when using php-cli. A PHP call like `getenv('PATH');` can therefore return an empty result. So you may need to manually configure environment variables in the appropriate php-fpm ini/config file.

Here are some example root paths for these ini/config files:

Ubuntu/Mint	CentOS/Red Hat/Fedora
/etc/php/[php_version]/fpm/	/etc/php-fpm.d/

In both examples, the ini/config file is called `www.conf`, and depending on the distribution or customizations which you have made, it may be in a sub-directory.

Usually, you will find some or all of the environment variables already in the file, but commented out like this:

```
;env[HOSTNAME] = $HOSTNAME
;env[PATH] = /usr/local/bin:/usr/bin:/bin
;env[TMP] = /tmp
;env[TMPDIR] = /tmp
;env[TEMP] = /tmp
```

Uncomment the appropriate existing entries. Then run `printenv PATH` to confirm your paths, for example:

```
$ printenv PATH
/home/user/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/
```

If any of your system environment variables are not present in the file then you must add them.

Installation

When you are using shared hosting or a control panel to manage your ownCloud virtual machine or server, the configuration files are almost certain to be located somewhere else, for security and flexibility reasons, so check your documentation for the correct locations.

Please keep in mind that it is possible to create different settings for `php-cli` and `php-fpm`, and for different domains and Web sites. The best way to check your settings is with PHP Version and Information.

Maximum Upload Size

If you want to increase the maximum upload size, you will also have to modify your `php-fpm` configuration and increase the `upload_max_filesize` and `post_max_size` values. You will need to restart `php5-fpm` and your HTTP server in order for these changes to be applied.

.htaccess Notes for Apache

ownCloud comes with its own `owncloud/.htaccess` file. Because `php-fpm` can't read PHP settings in `.htaccess` these settings and permissions must be set in the `owncloud/.user.ini` file.

No basic authentication headers were found

This error is shown in your `data/owncloud.log` file. Some Apache modules like `mod_fastcgi`, `mod_fcgid` or `mod_proxy_fcgi` are not passing the needed authentication headers to PHP and so the login to ownCloud via WebDAV, CalDAV and CardDAV clients is failing. Information on how to correctly configure your environment can be found [in the forums](#) but we generally recommend against the use of these modules and recommend `mod_php` instead.

Other Web Servers

- Other HTTP servers
- Univention Corporate Server installation

Troubleshooting

Database Configuration Issues

If your ownCloud installation fails and you see the following error in your ownCloud log please refer to MySQL / MariaDB with Binary Logging Enabled for how to resolve it.

An unhandled exception has been thrown: exception 'PDOException' with message 'SQLSTATE[HY000]: General error: 1665 Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine limited to row-based logging. InnoDB is limited to row-logging when transaction isolation level is READ COMMITTED or READ UNCOMMITTED.'

Changing Your ownCloud URL

This admin manual assumes that the ownCloud server is already accessible under the route `/owncloud` (which is the default, e.g. `https://example.com/owncloud`). If you like, you can change this in your web server configuration, for example by changing it from `https://example.com/owncloud/` to `https://example.com/`.

To do so on Debian/Ubuntu Linux, you need to edit these files:

- `/etc/apache2/sites-enabled/owncloud.conf`
- `/var/www/owncloud/config/config.php`

Edit the `Alias` directive in `/etc/apache2/sites-enabled/owncloud.conf` to alias your ownCloud directory to the Web server root:

```
Alias / "/var/www/owncloud/"
```

Installation

Edit the `overwrite.cli.url` parameter in `/var/www/owncloud/config/config.php`:

```
'overwrite.cli.url' => 'http://localhost/' ,
```

When the changes have been made and the file saved, restart Apache. Now you can access ownCloud from either `https://example.com/` or `https://localhost/`.

Note

Note that you will not be able to run any other virtual hosts, as ownCloud is aliased to your web root. On CentOS/Fedora/Red Hat, edit `/etc/httpd/conf.d/owncloud.conf` and `/var/www/html/owncloud/config/config.php`, then restart Apache.

Installing and Managing Apps

After installing ownCloud, you may provide added functionality by installing applications.

Supported Apps

See Supported Apps in ownCloud for a list of supported Enterprise edition apps.

Viewing Enabled Apps

During the ownCloud installation, some apps are installed and enabled by default, and some are able to be installed and enabled later on. To see the status of your installation's applications, go to your Apps page.

The screenshot shows the ownCloud Apps interface. At the top, there is a navigation bar with a cloud icon, the word "Apps", a search icon, and a user profile icon. Below the navigation bar, there is a sidebar with filters: "Enabled", "Not enabled", "Multimedia", "Productivity", "Game", "Tool", and "Developer documentation". The main content area displays two apps: "Collaborative tags 0.2" and "Comments 0.2". Each app card includes the app icon, name, developer information ("by Vincent Petry (AGPL-licensed)" or "by Arthur Shiwon, Vincent Petry (AGPL-licensed)"), an "Official" badge with a checkmark, a "Show description ..." link, and a "Disable" button.

There, you will see which apps are currently: *enabled*, *not enabled*, and *recommended*. You'll also see additional filters, such as Multimedia, Productivity, and Tool for finding more apps quickly.

Managing Apps

In the Apps page, you can enable or disable applications. Some apps have configurable options on the Apps page, such as **Enable only for specific groups**, but mainly they are enabled or disabled here and are configured on your ownCloud *Admin page*, *Personal page*, or in `config.php`.

Adding Apps

Click the app name to view a description of the app and any of the app settings in the Application View field. Clicking the **Install** button installs the app. If the app is not part of your ownCloud installation, it will be downloaded from the ownCloud Marketplace, installed, and enabled.

Sometimes the installation of a third-party app fails silently, possibly because 'appcodechecker' => true, is enabled in config.php. When appcodechecker is enabled it checks if third-party apps are using the private API, rather than the public API. If they are, then they will not be installed.

Note

If you would like to create or add your own ownCloud app, please refer to the [developer manual](#).

Using Custom App Directories

There are several reasons for using custom app directories instead of ownCloud's default. These are:

1. It separates ownCloud's core apps from user or admin downloaded apps. Doing so distinguishes which apps are core and which aren't, simplifying upgrades.
2. It eases manual upgrades. Downloaded apps must be manually copied. Having them in a separate directory makes it simpler to manage.
3. ownCloud may gain new core apps in newer versions. Doing so orphans deprecated apps, but doesn't remove them.

If you want to store apps in a custom directory, instead of ownCloud's default (/app), you need to modify the apps_paths element in config/config.php. There, you need to add a new associative array that contains three elements. These are:

- path: The absolute file system path to the custom app folder.
- url: The request path to that folder relative to the ownCloud web root, prefixed with /.
- writable: Whether users can install apps in that folder. After the configuration is added, new apps will only install in a directory where writable is set to true.

The configuration example below shows how to add a second directory, called apps-external.

```
<?php
$CONFIG = [
    'apps_paths' => [
        [
            'path' => '/var/www/owncloud/apps',
            'url' => '/apps',
            'writable' => false,
        ],
        [
            'path' => '/var/www/owncloud/apps-external',
            'url' => '/apps-external',
            'writable' => true,
        ],
    ],
    // remainder of the configuration
];

```

After you add a new directory configuration, you can then move apps from the original app directory to the new one. To do so, follow these steps:

1. Enable maintenance mode.
2. Disable the apps that you want to move.

3. Create a new apps directory and assign it the same user and group, and ownership permissions as the core apps directory.
4. Move the apps from the old apps directory to the new apps directory.
5. Add a new app directory in config/config.php.
6. If you're using a cache, such as [Redis](#) or [Memcached](#), ensure that you clear the cache.
7. Re-enable the apps.
8. Disable maintenance mode.

Manually Installing Apps

To install an app manually instead of by using the [Marketplace](#), copy the app either into ownCloud's default app folder (</path/to/owncloud>/apps) or a custom app folder.

Be aware that the name of the app and its folder name **must be identical!** You can find these details in *the application's metadata file*, located in <app directory>/appinfo/info.xml.

Using the example below, both the app's name and directory name would be yourappname.

```
<?xml version="1.0"?>
<info>
  <id>yourappname</id>
  <name>Your App</name>
  <version>1.0</version>
</info>
```

Supported Apps in ownCloud

AGPL Apps

- [Activity](#)
- [Anti-Virus](#)
- Collaborative Tags
- Comments
- Encryption
- External Sites
- External Storage
- ownCloud WebDAV Endpoint (handles old and new webdav endpoints)
- Federated File Sharing (allows file sharing across ownCloud instances)
- Federation (allows username auto-complete across ownCloud instances)
- Files (cannot be disabled)
- Files PDF Viewer
- Files Sharing
- Files TextEditor
- Files Trashbin
- Files Versions
- Files VideoPlayer
- First Run Wizard
- [Gallery](#)
- Notifications

Installation

- Object Storage (Swift)
- Provisioning API
- Template Editor (for notification emails)
- Update Notifications
- User External
- User LDAP

Enterprise-Only Apps

- Auditing
- Collaborative Tags Management
- Enterprise License Key
- File Firewall
- LDAP Home Connector
- Object Storage Support
- Password Policy
- External Storage: SharePoint
- SAML/Shibboleth User Backend
- Windows Network Drives (requires External Storage)
- Workflows
- ownCloud X Enterprise Theme

SELinux Configuration

When you have SELinux enabled on your Linux distribution, you may run into permissions problems after a new ownCloud installation, and see permission denied errors in your ownCloud logs.

The following settings should work for most SELinux systems that use the default distro profiles. Run these commands as root, and remember to adjust the filepaths in these examples for your installation

```
semanage fcontext -a -t httpd_sys_rw_content_t '/var/www/html/owncloud/data(/.*)?'
semanage fcontext -a -t httpd_sys_rw_content_t '/var/www/html/owncloud/config(/.*)?'
semanage fcontext -a -t httpd_sys_rw_content_t '/var/www/html/owncloud/apps(/.*)?'
semanage fcontext -a -t httpd_sys_rw_content_t '/var/www/html/owncloud/.htaccess'
semanage fcontext -a -t httpd_sys_rw_content_t '/var/www/html/owncloud/.user.ini'

restorecon -Rv '/var/www/html/owncloud/'
```

If you uninstall ownCloud you need to remove the ownCloud directory labels. To do this execute the following commands as root after uninstalling ownCloud

```
semanage fcontext -d '/var/www/html/owncloud/data(/.*)?'
semanage fcontext -d '/var/www/html/owncloud/config(/.*)?'
semanage fcontext -d '/var/www/html/owncloud/apps(/.*)?'
semanage fcontext -d '/var/www/html/owncloud/.htaccess'
semanage fcontext -d '/var/www/html/owncloud/.user.ini'

restorecon -Rv '/var/www/html/owncloud/'
```

If you have customized SELinux policies and these examples do not work, you must give the HTTP server write access to these directories:

```
/var/www/html/owncloud/data  
/var/www/html/owncloud/config  
/var/www/html/owncloud/apps
```

Enable updates via the web interface

To enable updates via the ownCloud web interface, you may need this to enable writing to the ownCloud directories:

```
setsebool httpd_unified on
```

When the update is completed, disable write access:

```
setsebool -P httpd_unified off
```

Disallow write access to the whole web directory

For security reasons it's suggested to disable write access to all folders in /var/www/ (default):

```
setsebool -P httpd_unified off
```

Allow access to a remote database

An additional setting is needed if your installation is connecting to a remote database:

```
setsebool -P httpd_can_network_connect_db on
```

Allow access to LDAP server

Use this setting to allow LDAP connections:

```
setsebool -P httpd_can_connect_ldap on
```

Allow access to remote network

ownCloud requires access to remote networks for functions such as Server-to-Server sharing, external storages or the ownCloud Marketplace. To allow this access use the following setting:

```
setsebool -P httpd_can_network_connect on
```

Allow access to network memcache

This setting is not required if httpd_can_network_connect is already on:

```
setsebool -P httpd_can_network_memcache on
```

Allow access to SMTP/sendmail

If you want to allow ownCloud to send out e-mail notifications via sendmail you need to use the following setting:

```
setsebool -P httpd_can_sendmail on
```

Allow access to CIFS/SMB

If you have placed your datadir on a CIFS/SMB share use the following setting:

```
setsebool -P httpd_use_cifs on
```

Allow access to FuseFS

If your owncloud data folder resides on a Fuse Filesystem (e.g. EncFS etc), this setting is required as well:

```
setsebool -P httpd_use_fusefs on
```

Allow access to GPG for Rainloop

If you use the rainloop webmail client app which supports GPG/PGP, you might need this:

```
setsebool -P httpd_use_gpg on
```

Troubleshooting

General Troubleshooting

For general Troubleshooting of SELinux and its profiles try to install the package `setroubleshoot` and run:

```
sealert -a /var/log/audit/audit.log > /path/to/mylogfile.txt
```

to get a report which helps you configuring your SELinux profiles.

Another tool for troubleshooting is to enable a single ruleset for your ownCloud directory:

```
semanage fcontext -a -t httpd_sys_rw_content_t '/var/www/html/owncloud(/.*)?'
restorecon -RF /var/www/html/owncloud
```

It is much stronger security to have a more fine-grained ruleset as in the examples at the beginning, so use this only for testing and troubleshooting. It has a similar effect to disabling SELinux, so don't use it on production systems.

See this [discussion on GitHub](#) to learn more about configuring SELinux correctly for ownCloud.

Redis on RHEL 7 & Derivatives

On RHEL 7 and its derivatives, if you are using Redis for both local server cache and file locking and Redis is configured to listen on a Unix socket instead of a TCP/IP port (*which is recommended if Redis is running on the same system as ownCloud*) you must instruct SELinux to allow daemons to enable cluster mode. You can do this using the following command:

```
setsebool -P daemons_enable_cluster_mode 1
```

NGINX Configuration

This page covers example NGINX configurations to use with running an ownCloud server. Note that NGINX is *not officially supported*, and this page is *community-maintained*. Thank you, contributors!

- Depending on your setup, you need to insert the code examples into your NGINX configuration file.
- Adjust `server_name`, `root`, `ssl_certificate`, `ssl_certificate_key` etc. to suit your needs.
- Make sure your SSL certificates are readable by the server (see [NGINX HTTP SSL Module documentation](#)).
- `add_header` statements are only valid in the current `location` block and are not derived or cascaded from or to a different `location` block. All necessary `add_header` statements **must** be defined in each `location` block needed.
- For better readability it is possible to move *common* `add_header` directives into a separate file and include that file wherever necessary. However, each `add_header` directive must be written in a single line to prevent connection problems with sync clients.
- The same is true for `map` directives which also can be collected into a single file and then be included.

Example Configurations

Note

Be careful about line breaks if you copy the examples, as long lines may be broken for page formatting.

You can use ownCloud over plain HTTP. However, we *strongly encourage you* to use SSL/TLS to encrypt all of your server traffic **and** to protect users' logins, and their data while it is in transit. To use plain HTTP:

1. Remove the server block containing the redirect
2. Change **listen 443 ssl http2** to **listen 80;**
3. Remove all **ssl_** entries.
4. Remove **fastcgi_params HTTPS on;**

Note 1

```
fastcgi_buffers 8 4K;
```

- Do not set the number of buffers to greater than 63. In our example, it is set to 8.
- If you exceed this maximum, big file downloads may consume a lot of system memory over time. This is especially problematic on low-memory systems.

Note 2

```
fastcgi_ignore_headers  
X-Accel-Buffering
```

- From ownCloud version 10.0.4 on, a header will be sent to NGINX not to use buffers to avoid problems with problematic `fastcgi_buffers` values. See note above.
- If the values of `fastcgi_buffers` are properly set and no problems are expected, you can use this directive to reenable buffering **overriding** the sent header.
- In case you use an earlier version of ownCloud or can't change the buffers, or can't remove a existing ignore header directive, you can explicitly enable following directive in the location block `fastcgi_buffering off;`

Note

The directives `fastcgi_ignore_headers X-Accel-Buffering;` and `fastcgi_buffering off;` can be used separately but not together.

ownCloud in the web root of NGINX

The following config should be used when ownCloud is placed in the web root of your NGINX installation.

The configuration assumes that ownCloud is installed in

```
/var/www/owncloud and is accessed via http(s)://cloud.example.com.

upstream php-handler {
    server 127.0.0.1:9000;
    # Depending on your used PHP version
    #server unix:/var/run/php5-fpm.sock;
    #server unix:/var/run/php7-fpm.sock;
}

server {
    listen 80;
    server_name cloud.example.com;

    # For Lets Encrypt, this needs to be served via HTTP
    location /.well-known/acme-challenge/ {
        root /var/www/owncloud; # Specify here where the challenge file is placed
    }

    # enforce https
```

```

location / {
    return 301 https://$server_name$request_uri;
}
}

server {
    listen 443 ssl http2;
    server_name cloud.example.com;

    ssl_certificate /etc/ssl/nginx/cloud.example.com.crt;
    ssl_certificate_key /etc/ssl/nginx/cloud.example.com.key;

    # Example SSL/TLS configuration. Please read into the manual of NGINX before applying
    ssl_session_timeout 5m;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers "-ALL:EECDH+AES256:EDH+AES256:AES256-SHA:EECDH+AES:EDH+AES:!ADH:!NULL:!aNULL";
    ssl_dhparam /etc/nginx/dh4096.pem;
    ssl_prefer_server_ciphers on;
    keepalive_timeout    70;
    ssl_stapling on;
    ssl_stapling_verify on;

    # Add headers to serve security related headers
    # Before enabling Strict-Transport-Security headers please read into this topic first.
    #add_header Strict-Transport-Security "max-age=15552000; includeSubDomains";
    add_header X-Content-Type-Options nosniff;
    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Robots-Tag none;
    add_header X-Download-Options noopen;
    add_header X-Permitted-Cross-Domain-Policies none;

    # Path to the root of your installation
    root /var/www/owncloud/;

    location = /robots.txt {
        allow all;
        log_not_found off;
        access_log off;
    }

    # The following 2 rules are only needed for the user_webfinger app.
    # Uncomment it if you're planning to use this app.
    #rewrite ^/.well-known/host-meta /public.php?service=host-meta last;
    #rewrite ^/.well-known/host-meta.json /public.php?service=host-meta-json last;

    location = /.well-known/carddav {
        return 301 $scheme://$host/remote.php/dav;
    }
    location = /.well-known/caldav {
        return 301 $scheme://$host/remote.php/dav;
    }

    # set max upload size
    client_max_body_size 512M;
    fastcgi_buffers 8 4K;                      # Please see note 1
    fastcgi_ignore_headers X-Accel-Buffering; # Please see note 2

    # Disable gzip to avoid the removal of the ETag header
    # Enabling gzip would also make your server vulnerable to BREACH
}

```

```

# if no additional measures are done. See https://bugs.debian.org/cgi-bin/bugreport.cgi
gzip off;

# Uncomment if your server is build with the ngx_pagespeed module
# This module is currently not supported.
#pagespeed off;

error_page 403 /core/templates/403.php;
error_page 404 /core/templates/404.php;

location / {
    rewrite ^ /index.php$uri;
}

location ~ ^/(?:build|tests|config|lib|3rdparty|templates|data)/ {
    return 404;
}
location ~ ^/(?:\.|autotest|occ|issue|indie|db_|console) {
    return 404;
}

location ~ ^/(?:index|remote|public|cron|core/ajax/update|status|ocs/v[12]|updater/.+|
fastcgi_split_path_info ^(.+\.\php)(/.*)$;
include fastcgi_params;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
fastcgi_param SCRIPT_NAME $fastcgi_script_name; # necessary for owncloud to detect
fastcgi_param PATH_INFO $fastcgi_path_info;
fastcgi_param HTTPS on;
fastcgi_param modHeadersAvailable true; #Avoid sending the security headers twice
fastcgi_param front_controller_active true;
fastcgi_read_timeout 180; # increase default timeout e.g. for long running carddav
fastcgi_pass php-handler;
fastcgi_intercept_errors on;
fastcgi_request_buffering off; #Available since NGINX 1.7.11
}

location ~ ^/(?:updater|ocs-provider)(?:$|/) {
    try_files $uri $uri/ =404;
    index index.php;
}

# Adding the cache control header for js and css files
# Make sure it is BELOW the PHP block
location ~ \.(?:css|js)$ {
    try_files $uri /index.php$uri$is_args$args;
    add_header Cache-Control "max-age=15778463";
    # Add headers to serve security related headers (It is intended to have those dupl
    # Before enabling Strict-Transport-Security headers please read into this topic fi
    #add_header Strict-Transport-Security "max-age=15552000; includeSubDomains";
    add_header X-Content-Type-Options nosniff;
    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Robots-Tag none;
    add_header X-Download-Options noopen;
    add_header X-Permitted-Cross-Domain-Policies none;
    # Optional: Don't log access to assets
    access_log off;
}

location ~ \.(?:svg|gif|png|html|ttf|woff|ico|jpg|jpeg|map)$ {

```

```

        add_header Cache-Control "public, max-age=7200";
        try_files $uri /index.php$uri$args;
        # Optional: Don't log access to other assets
        access_log off;
    }
}

```

ownCloud in a subdirectory of NGINX

The following config should be used when ownCloud is placed under a different context root of your NGINX installation such as /owncloud or /cloud.

The configuration assumes that ownCloud is installed in

```
/var/www/owncloud is accessed via http(s)://example.com/owncloud
and that you have 'overwriteweb root' => '/owncloud', set in your config/config.php.
```

```

upstream php-handler {
    server 127.0.0.1:9000;
    # Depending on your used PHP version
    #server unix:/var/run/php5-fpm.sock;
    #server unix:/var/run/php7-fpm.sock;
}

server {
    listen 80;
    server_name cloud.example.com;

    # For Lets Encrypt, this needs to be served via HTTP
    location /.well-known/acme-challenge/ {
        root /var/www/owncloud; # Specify here where the challenge file is placed
    }

    # enforce https
    location / {
        return 301 https://$server_name$request_uri;
    }
}

server {
    listen 443 ssl http2;
    server_name cloud.example.com;

    ssl_certificate /etc/ssl/nginx/cloud.example.com.crt;
    ssl_certificate_key /etc/ssl/nginx/cloud.example.com.key;

    # Example SSL/TLS configuration. Please read into the manual of NGINX before applying
    ssl_session_timeout 5m;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers "-ALL:EECDH+AES256:EDH+AES256:AES256-SHA:EECDH+AES:EDH+AES:!ADH:!NULL:!aNULL";
    ssl_dhparam /etc/nginx/dh4096.pem;
    ssl_prefer_server_ciphers on;
    keepalive_timeout    70;
    ssl_stapling on;
    ssl_stapling_verify on;

    # Add headers to serve security related headers
    # Before enabling Strict-Transport-Security headers please read into this topic first.
    #add_header Strict-Transport-Security "max-age=15552000; includeSubDomains";
    add_header X-Content-Type-Options nosniff;
    add_header X-Frame-Options "SAMEORIGIN";
}

```



```

fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
fastcgi_param SCRIPT_NAME $fastcgi_script_name; # necessary for owncloud to de
fastcgi_param PATH_INFO $fastcgi_path_info;
fastcgi_param HTTPS on;
fastcgi_param modHeadersAvailable true; #Avoid sending the security headers tw
# EXPERIMENTAL: active the following if you need to get rid of the 'index.php'
#fastcgi_param front_controller_active true;
fastcgi_read_timeout 180; # increase default timeout e.g. for long running car
fastcgi_pass php-handler;
fastcgi_intercept_errors on;
fastcgi_request_buffering off; #Available since NGINX 1.7.11
}

location ~ ^/owncloud/(?:updater|ocs-provider)(?:$|/) {
    try_files $uri $uri/ =404;
    index index.php;
}

# Adding the cache control header for js and css files
# Make sure it is BELOW the PHP block
location ~ /owncloud/.*\.(?:css|js) {
    try_files $uri /owncloud/index.php$uri$is_args$args;
    add_header Cache-Control "max-age=15778463";
    # Add headers to serve security related headers (It is intended to have those
    # Before enabling Strict-Transport-Security headers please read into this topic
    #add_header Strict-Transport-Security "max-age=15552000; includeSubDomains";
    add_header X-Content-Type-Options nosniff;
    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Robots-Tag none;
    add_header X-Download-Options noopen;
    add_header X-Permitted-Cross-Domain-Policies none;
    # Optional: Don't log access to assets
    access_log off;
}

location ~ /owncloud/.*\.(?:svg|gif|png|html|ttf|woff|ico|jpg|jpeg|map) {
    try_files $uri /owncloud/index.php$uri$is_args$args;
    add_header Cache-Control "public, max-age=7200";
    # Optional: Don't log access to other assets
    access_log off;
}
}

```

Troubleshooting

JavaScript (.js) or CSS (.css) files not served properly

A standard issue with custom NGINX configurations is, that JavaScript (.js) or CSS (.css) files are not served properly, leading to a 404 (File Not Found) error on those files and a broken web interface.

- This could be caused by an improper sequence of location blocks.

The following sequence is correct:

```
location ~ \.php(?:$|/) {  
    ...  
}  
  
location ~ \.(css|js)$ {
```

```
...}
```

Other custom configurations like caching JavaScript (.js) or CSS (.css) files via gzip could also cause such issues.

Not all of my contacts are synchronized

Check for server timeouts! It turns out that CardDAV sync often fails silently if the request runs into timeouts. With PHP-FPM you might see a “CoreDAVHTTPStatusErrorDomain error 504” which is an “HTTP 504 Gateway timeout” error. To solve this, first check the `default_socket_timeout` setting in `/etc/php/7.0/fpm/php.ini` and increase the above `fastcgi_read_timeout` accordingly. Depending on your server’s performance a timeout of 180s should be sufficient to sync an address book of ~1000 contacts.

Windows: Error 0x80070043 “The network name cannot be found.” while adding a network drive

The windows native WebDAV client might fail with the following error message:

```
Error 0x80070043 "The network name cannot be found." while adding a network drive
```

A known workaround for this issue is to update your web server configuration.

Because NGINX does not allow nested `if` directives, you need to use the `map` directive.

The position of the `location` directive is important for success.

1 Create a map directive outside your server block

```
# Fixes Windows WebDav client error 0x80070043 "The network name cannot be found."
map "$http_user_agent:$request_method" $WinWebDav {
    default          0;
    "DavClnt:OPTIONS" 1;
}
```

2 Inside your server block on top of your location directives

```
location = / {
    if ($WinWebDav) { return 401; }
}
```

Log Optimisation

Suppressing htaccesstest.txt and .ocdata Log Messages

If you are seeing meaningless messages in your logfile, for example `client denied by server configuration: /var/www/data/htaccesstest.txt`, or access to `.ocdata`, add this section to your NGINX configuration to suppress them:

```
location = /data/htaccesstest.txt {
    allow all;
    log_not_found off;
    access_log off;
}
```

```
location = /data/\.ocdata {
    access_log off;
}
```

Prevent access log entries when accessing thumbnails

When using eg. the Gallery App, any access to a thumbnail of a picture will be logged. This can cause a massive log quantity making log reading challenging. With this approach, you can prevent access logging for those thumbnails.

1 Create a map directive outside your server block like

(Adopt the path queried according your needs.)

```
# do not access log to gallery thumbnails, flooding access logs only, error will be logged as info
map $request_uri $loggable {
    default                      1;
    ~*/apps/gallery/thumbnails   0;
}
```

2 Inside your server block where you define your logs

```
access_log /path-to-your-log-file combined if=$loggable;
```

If you want or need to log thumbnails access, you can easily add another logfile which only logs this access. You can easily enable / disable this kind of logging if you uncomment / comment the line starting with 0 in the following map directive.

Below the above map statement

```
# invert the $logabble variable
map $logabble $invertlogabble {
    default                      0;
    0                           1;
}
```

Below the above access_log statement

```
access_log /var/log/nginx/<your-log-file-inverted> combined if=$invertlogabble;
```

Performance Tuning

1 HTTP/2

To increase the performance of your NGINX installation, we recommend using either the SPDY or HTTP_V2 modules, depending on your installed NGINX version.

- nginx (<1.9.5) [ngx_http_spdy_module](#)
- nginx (+1.9.5) [ngx_http_v2_module](#)

To use HTTP_V2 for NGINX you have to check two things:

1. Be aware that this module may not built in by default, due to a dependency to the OpenSSL version used on your system. It will be enabled with the `--with-http_v2_module` configuration parameter during compilation. The dependencies should be checked automatically. You can check the presence of `ngx_http_v2_module` by using the command: `nginx -V 2>&1 | grep http_v2 -o`. A description of how to compile NGINX to include modules can be found in [Compiling Modules](#).
2. When changing from [SPDY](#) to [HTTP v2](#), the NGINX config has to be changed from `listen 443 ssl spdy;` to `listen 443 ssl http2;`

2 Caching Metadata

The `open_file_cache` directive can help you to cache file metadata information. This can increase performance on high loads respectively when using eg NFS as backend. That cache can store:

- Open file descriptors, their sizes and modification times;
- Information on existence of directories;
- File lookup errors, such as “file not found”, “no read permission”, and so on.

To configure metadata caching, add following directives either in your `http`, `server` or `location` block:

```
open_file_cache           max=10000 inactive=5m;
open_file_cache_valid     1m;
open_file_cache_min_uses  1;
open_file_cache_errors    on;
```

Configure NGINX to use caching for ownCloud internal images and thumbnails

Installation

This mechanism speeds up presentation as it shifts requests to NGINX and minimizes PHP invocations, which otherwise would take place for every thumbnail or internal image presented every time.

1 Preparation

- Create a directory where NGINX will save the cached thumbnails or internal images. Use any path that fits to your environment. Replace `/opt/cachezone` in this example with your path created:

```
sudo mkdir -p /opt/cachezone
sudo chown www-data:www-data /opt/cachezone
```

2 Configuration

a. Define when to skip the cache:

- **Option 1: map**

This is the preferred method. In the `http{ }` block, but *outside* the `server{ }` block:

```
# skip_cache, default skip
map $request_uri $skip_cache {
    default      1;
    ~*/thumbnail.php      0;
    ~*/apps/gallery/*      0;
    ~*/core/img/*      0;
}
```

- **Option 2: if**

In the `server{ }` block, above the location block mentioned below:

```
set $skip_cache 1;
if ($request_uri ~* "thumbnail.php")      { set $skip_cache 0; }
if ($request_uri ~* "/apps/gallery/")      { set $skip_cache 0; }
if ($request_uri ~* "/core/img/")          { set $skip_cache 0; }
```

b. General Config:

In case you want to have multiple cache paths with different cache keys, follow the NGINX documentation where to place the directives. For the sake of simplicity, we both add them to the `http{ }` block.

- Add *inside* the `http{ }` block:

```
fastcgi_cache_path /opt/cache levels=1:2 keys_zone=cachezone:100m
                   max_size=500m inactive=60m use_temp_path=off;
fastcgi_cache_key $http_cookie$request_method$host$request_uri;
```

- Add *inside* the `server{ }` block the following FastCGI caching directives, as an example of a configuration:

```
location ~ \.php(?:$/|.*\?.*) {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;

    include fastcgi_params;
    # ...

    ## Begin - FastCGI caching
    fastcgi_ignore_headers "Cache-Control"
                           "Expires"
                           "Set-Cookie";
    fastcgi_cache_use_stale error
                           timeout
                           updating
                           http_429
                           http_500
                           http_503;
    fastcgi_cache_background_update on;
    fastcgi_no_cache $skip_cache;
```

```
fastcgi_cache_bypass $skip_cache;
fastcgi_cache cachezone;
fastcgi_cache_valid 60m;
fastcgi_cache_methods GET HEAD;
## End - FastCGI caching
}
```

3 Test the configuration

```
sudo nginx -t
sudo service nginx reload
```

- Open your browser and clear your cache.
- Logon to your ownCloud instance, open the gallery app, move thru your folders and watch while the thumbnails are generated for the first time.
- You may also watch with eg. `htop` your system load while the thumbnails are processed.
- Go to another app or logout and relogin.
- Open the gallery app again and browse to the folders you accessed before. Your thumbnails should appear more or less immediately.
- `htop` will not show up additional load while processing, compared to the high load before.

Using Let's Encrypt SSL Certificates

This page covers how to configure your web server to use [Let's Encrypt](#) as the certificate authority for your ownCloud server. Note that Let's Encrypt is *not officially supported*, and this page is *community-maintained*. Thank you, contributors!

- For ease of handling, SSL-specific directives have been moved into a separately included file. This can help for first-time certificate issuance as well as for reusing configurations.
- The examples shown are based on Ubuntu 17.10.
- Read the [Certbot user guide](#) for details of the commands.
- Let's Encrypt CA issues short-lived certificates valid for 90 days. Make sure you renew the certificates at least once in this period, because expired certificates need reissuing. A certificate is due for renewal earliest 30 days before expiring. Certbot can be forced to renew via options at any time as long the certificate is valid.

Excellent introductions to strong SSL security measures can be found here: [Apache](#) and [NGINX](#).

1. Requirements & Dependencies
2. [Install Let's Encrypt's Certbot client](#)
3. Register your email address
4. [Create Let's Encrypt's config files](#)
5. [Create an SSL certificate](#)
6. [Web Server setup](#)
7. [Test the setup](#)
8. [`Certificate renewal`](#)

Requirements & Dependencies

- You require a domain name with a valid [A record](#) pointing back to your servers IP address. In case your server is behind a firewall, take the necessary measures to ensure that your server is accessible, worldwide, from the internet, by adding the required firewall and port forward rules.

Install Let's Encrypt's Certbot client

The latest Certbot client can be installed in two ways:

1. From source.
2. With the Ubuntu ppa repository.

Via GitHub

```
sudo apt-get update  
sudo apt-get install -y git  
sudo git clone https://github.com/certbot/certbot /opt/letsencrypt
```

To run Certbot use the following command:

```
sudo /opt/letsencrypt/certbot-auto
```

Note

For the sake of simplicity, the path chosen for the installation is /opt/letsencrypt. You can use any path that fits your needs.

Note

Unless explicitly denied, Certbot will auto-update on each run.

As part of the first run, certbot-auto will install any missing dependencies.

Via Apt

To install Certbot via the PPA repository, run the following commands. These will add the repository, update Apt's cache, and install Certbot.

```
sudo apt-get install certbot
```

Note

If you're using a version of Ubuntu prior to 17.10, you may need to run the following commands before you can install Certbot:

```
sudo apt-get update  
sudo apt-get install software-properties-common  
sudo add-apt-repository ppa:certbot/certbot
```

To run Certbot use the following command:

```
sudo /usr/bin/certbot  
  
# Alternatively, you could run the following instead  
sudo certbot
```

Note

Depending on how you installed Let's Encrypt, certbot may also be named letsencrypt or certbot-auto. However, this guide will refer to it as certbot. Please bear that in mind, and update the ..examples and scripts used in this guide to reflect your Certbot installation.

Updating Certbot

If you need to update Certbot at a later date, run `sudo apt-get install --only-upgrade certbot`.

Register your email address

Now that Certbot is installed, register your email address for urgent renewal and security notifications. This command also prepares Certbot's environment if it's not already installed. To do this, run the following command:

```
sudo certbot register --agree-tos --email <your-email-address>
```

When it executes, you'll see the following question, which you can answer "Yes" or "No" to:

Saving debug log to /var/log/letsencrypt/letsencrypt.log

```
-----  
Would you be willing to share your email address with the Electronic Frontier Foundation, a founding partner of the Let's Encrypt project and the non-profit organization that develops Certbot? We'd like to send you email about EFF and our work to encrypt the web, protect its users and defend digital rights.  
-----
```

(Y)es/(N)o:

When that completes, you'll see a message similar to the following:

IMPORTANT NOTES:

1. Your account credentials have been saved in your Certbot configuration directory at /etc/letsencrypt. You should make a secure backup of this folder now. This configuration directory will also contain certificates and private keys obtained by Certbot so making regular backups of this folder is ideal.

Please, **strongly**, consider following its recommendation.

Create Let's Encrypt's config files

- Create following files in the Let's Encrypt directory. They will help to maintain your certificates.
- Replace the path to Certbot and the Certbot script name based on your installation. You can find it by running `which certbot`.
- Rename `<your-domain-name>.sh` with the name of the domain(s) you want to issue a certificate for. As an example, the script could be renamed to `your-domain-name.com.sh`.
- Make all files executable except `cli.ini` by running `sudo chmod +x <script-name>`.

Note

All scripts have to be executed with sudo.

```
cd /etc/letsencrypt  
touch cli.ini list.sh renew.sh renew-cron.sh delete.sh <your-domain-name>.sh
```

cli.ini

Installation

This file defines some settings used by Certbot. Use the email address you registered with. Comment / un-comment the post-hook parameter according which web server you use.

```
rsa-key-size = 4096
email = <your-email-address>
agree-tos = True
authenticator = webroot
webroot-path = /var/www/letsencrypt/
post-hook = service nginx reload
# post-hook = service apache2 reload
```

list.sh

This script lists all your issued certificates.

```
#!/bin/bash

LE_PATH="/usr/bin"
LE_CB="certbot"

"$LE_PATH/$LE_CB" certificates
```

renew.sh

This script:

1. Renews all your issued certificates.
2. Updates Certbot, when using Git as the installation source.
3. Reloads the web server configuration automatically if a certificate has been renewed.

```
#!/bin/bash

LE_PATH="/usr/bin"
LE_CB="certbot"

"$LE_PATH/$LE_CB" renew
```

renew-cron.sh

This script:

- Renews all your issued certificates but does not upgrade Certbot.
- Reloads the web server configuration automatically if a certificate has been renewed.

Note

It is intended for use via Cron.

```
#!/bin/bash

LE_PATH="/usr/bin"
LE_CB="certbot"

"$LE_PATH/$LE_CB" renew --no-self-upgrade --noninteractive
```

delete.sh

This script deletes an issued certificate. Use the list.sh script to list issued certificates.

Installation

```
#!/bin/bash

LE_PATH="/usr/bin"
LE_CB="certbot"

##
## Retrieve and print a list of the installed Let's Encrypt SSL certificates.
##
function get_certificate_names()
{
    "$LE_PATH/$LE_CB" certificates | grep -iE "certificate name" | awk -F: '{gsub(/\s+/, "", $2)}'
}

echo "Available Certificates:"

get_certificate_names
echo

read -p "Which certificate do you want to delete: " -r -e answer
if [ -n "$answer" ]; then
    "$LE_PATH/$LE_CB" delete --cert-name "$answer"
fi
```

<your-domain-name>.sh

As an example, this script creates a certificate for following domain / sub-domains. You can add or remove sub-domains as necessary. Use your domain / sub-domain names. The first (sub)domain name used in the script is taken for naming the directories created by Certbot.

Note: You can create different certificates for different sub-domains, such as mydom.tld, www.mydom.tld, and sub.mydom.tld, by creating different scripts. You can see an example script here below:

```
#!/bin/bash
# export makes the variable available for all subprocesses

LE_PATH="/usr/bin"
LE_CB="certbot"

# Assumes that mydom.tld www.mydom.tld and sub.mydom.tld are the domains that you want a cert for
export DOMAINS="-d mydom.tld -d www.mydom.tld -d sub.mydom.tld"

"$LE_PATH/$LE_CB" certonly --config /etc/letsencrypt/cli.ini "$DOMAINS" # --dry-run
```

Note

You can enable the --dry-run option which does a test run of the client only.

Create an SSL certificate

With all the scripts created, to create an SSL certificate, run the following command:

```
sudo /etc/letsencrypt/<your-domain-name>.sh
```

After you run the script, you will see output similar to the following:

```
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Obtaining a new certificate
Performing the following challenges:
http-01 challenge for your-domain-name.com
```

Installation

```
Using the webroot path /var/www/html for all unmatched domains.  
Waiting for verification...  
Cleaning up challenges  
Running post-hook command: service apache2 reload  
  
IMPORTANT NOTES:  
1. Congratulations! Your certificate and chain have been saved at:  
/etc/letsencrypt/live/your-domain-name.com/fullchain.pem  
Your key file has been saved at:  
/etc/letsencrypt/live/your-domain-name.com/privkey.pem  
Your cert will expire on 2018-06-18. To obtain a new or tweaked  
version of this certificate in the future, simply run certbot  
again. To non-interactively renew *all* of your certificates, run  
"certbot renew"  
2. If you like Certbot, please consider supporting our work by:  
  
Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate  
Donating to EFF: https://eff.org/donate-le
```

You can see that the SSL certificate's been successfully created, and that it will expire on 2018-06-18.

Listing Existing Certificates

If you want to list (view) the existing SSL certificates, use `list.sh`, which can be run as follows:

```
sudo /etc/letsencrypt/list.sh
```

Depending on the number of certificates, you can expect to see output similar to the following:

```
-----  
Found the following certs:  
  Certificate Name: your-domain-name.com  
    Domains: your-domain-name.com  
    Expiry Date: 2018-06-18 10:57:18+00:00 (VALID: 82 days)  
    Certificate Path: /etc/letsencrypt/live/your-domain-name.com/fullchain.pem  
    Private Key Path: /etc/letsencrypt/live/your-domain-name.com/privkey.pem  
-----
```

Web Server setup

Follow the links to set up your web server and issue a certificate.

- [apache](#)
- [nginx](#)

Test the setup

After you have setup and configured the web server and installed the SSL certificate using Certbot, you should now test the security of your new configuration. To do so, you can use the free service of [SSL Labs](#). See an example screenshot of a test run below.



You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > [REDACTED]

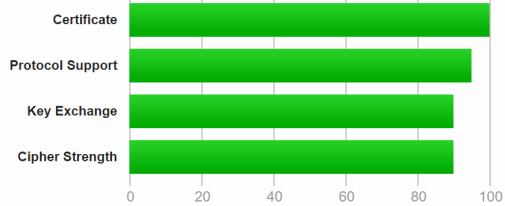
SSL Report: [REDACTED]

Assessed on: Sat, 17 Mar 2018 14:27:01 UTC | HIDDEN | [Clear cache](#)

[Scan Another »](#)

Summary

Overall Rating



Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. [MORE INFO »](#)

Renewing Certificates

As Let's Encrypt certificates expire every 90 days, you should ensure you renew them before that time. There are two ways to do so: **manually** and automatically.

Manual renewal

If you have provided your email address, you will receive reminder notifications.

```
sudo /etc/letsencrypt/renew.sh
```

If the certificate is not yet due for renewal, you can expect to see output similar to that below:

```
-----  
Processing /etc/letsencrypt/renewal/your-domain-name.com.conf  
-----
```

```
Cert not yet due for renewal
```

```
The following certs are not due for renewal yet:
```

```
  /etc/letsencrypt/live/your-domain-name.com/fullchain.pem (skipped)
```

```
No renewals were attempted.
```

```
No hooks were run.
```

Automatic renewal via crontab

Certificates are only renewed if they are due, so you can schedule Cron jobs to renew your SSL certificates on a more frequent basis. However, a weekly check is sufficient.

To add a new Cron job to auto-renew your certificates, firstly run the following command to edit the job list.

```
sudo crontab -e
```

Note

It is essential to use sudo to derive proper permissions.

Upgrading

Then, add the following at the end of the existing configuration:

```
30 03 * * 6 /etc/letsencrypt/renew-cron.sh
```

After you save and exit the file, the new job will have been added to the Cron job scheduler.

Note

If you want to use own values, you can check them at [crontab.guru](#) or modify the script for other options.

Add extra domains to the certificate

If you want to add an extra domain, like `test.mydom.tld`, to your certificate, add the domain in the domain shell script above, re-run it and reload the web server config. This can be useful when migrating from a sub-directory to sub-domain access.

Note

This also implies that you need to comment the `include` directive (please refer to the relevant web server setup) and follow the steps afterward.

Deleting SSL Certificates

If you want to delete an SSL certificate, use the `delete.sh` script, running it as follows:

```
sudo /etc/letsencrypt/delete.sh
```

It will start off, as below, by displaying a list of the currently available SSL certificate domain names, and then prompt you to supply the certificate that you want to delete.

Available Certificates:

1. `your-domain-name.com`

Which certificate do you want to delete:

Provide the SSL certificate name that you want to delete and click enter, and the certificate and all of its related files will be deleted. After that you should expect to see a confirmation, as in the example output below.

```
-----  
Deleted all files relating to certificate your-domain-name.com.  
-----
```

Upgrading

Upgrade PHP on RedHat 7 and Centos 7

You should, almost, always upgrade to the latest version of PHP, if and where possible. And if you're on a version of PHP older than 5.6 you need to upgrade. This guide steps you through upgrading your installation of PHP to version 5.6 or 7.0 if you're on RedHat or Centos 7.

- Upgrade PHP to version 5.6
- Upgrade PHP to version 7.0

Upgrade PHP to version 5.6

Note

You should really be upgrading to PHP 7, as version 5.6 is [no longer actively supported](#), and security support ends on 31 Dec, 2018.

You will first need to subscribe to the Red Hat Software Collections channel repository to be able to download and install the PHP 5.6 package in RHEL 7. To do that, run the following command:

```
subscription-manager repos --enable rhel-server-rhscl-7-rpms
```

Note

To know more about registering and subscribing a system to the Red Hat Customer Portal using the Red Hat Subscription-Manager, please refer to [the official documentation](#).

When that's completed, then proceed by installing PHP 5.6, along with *the other required PHP packages*.

```
yum install rh-php56 rh-php56-php rh-php56-php-gd rh-php56-php-mbstring rh-php56-php-mysqlnd
```

Once they're all installed, you next need to enable PHP 5.6 system-wide. To do this, run the following command:

```
cp /opt/rh/rh-php56/enable /etc/profile.d/rh-php56.sh source /opt/rh/rh-php56/enable
```

With PHP 5.6 enabled system-wide, you next need to disable the loading the previous version of PHP 5.4. For this example, we'll assume that you're upgrading from PHP 5.4. Here, you disable it from loading by renaming it's Apache configuration files.

```
mv /etc/httpd/conf.d/php.conf /etc/httpd/conf.d/php54.conf  
mv /etc/httpd/conf.modules.d/10-php.conf /etc/httpd/conf.modules.d/10-php54.conf
```

Note

You could also delete the files if you prefer.

Next, you need to enable loading of the PHP 5.6 Apache shared-object file. This you do by copying the shared object along with its two Apache configuration files, as in the command below.

```
cp /opt/rh/httpd24/root/etc/httpd/conf.d/rh-php56-php.conf /etc/httpd/conf.d/  
cp /opt/rh/httpd24/root/etc/httpd/conf.modules.d/10-rh-php56-php.conf /etc/httpd/conf.modules.d/  
cp /opt/rh/httpd24/root/etc/httpd/modules/librh-php56-php5.so /etc/httpd/modules/
```

With all that done, you lastly need to restart Apache.

```
service httpd restart
```

Upgrade PHP to version 7.0

As with upgrading to PHP 5.6, to upgrade to PHP 7 you will first need to subscribe to the Red Hat Software Collections channel repository to download and install the PHP 7 package in RHEL 7 (if you've not done this already). This uses the same command as you will find there.

Note

This section assumes that you're upgrading from PHP 5.6.

Then, proceed by installing the required PHP 7 modules. You can use the command below to save you time.

```
yum install rh-php70 rh-php70-php rh-php70-php-gd rh-php70-php-mbstring rh-php70-php-mysqlnd
```

Next, you need to enable PHP 7 and disable PHP 5.6 system-wide. To enable PHP 7 system-wide, run the following command:

```
cp /opt/rh/rh-php70/enable /etc/profile.d/rh-php70.sh source /opt/rh/rh-php70/enable
```

Then, you need to disable loading of the PHP 5.6 Apache modules. You can do this either by changing their names, as in the example below, or deleting the files.

```
mv /etc/httpd/conf.d/php.conf /etc/httpd/conf.d/php56.off  
mv /etc/httpd/conf.modules.d/10-php.conf /etc/httpd/conf.modules.d/10-php56.off
```

With that done, you next need to copy the PHP 7 Apache modules into place; that being the two Apache configuration files and the shared object file.

```
cp /opt/rh/httpd24/root/etc/httpd/conf.d/rh-php70-php.conf /etc/httpd/conf.d/  
cp /opt/rh/httpd24/root/etc/httpd/conf.modules.d/15-rh-php70-php.conf /etc/httpd/conf.modules.d/  
cp /opt/rh/httpd24/root/etc/httpd/modules/librh-php70-php7.so /etc/httpd/modules/
```

Finally, you need to restart Apache to make the changes permanent, as in the command below.

```
service httpd restart
```

Upgrade Marketplace Applications

To upgrade Marketplace applications, please refer to the documentation below, as applicable for your ownCloud setup.

Single-Server Environment

To upgrade Marketplace applications when running ownCloud in a single server environment, you can use the Market app, specifically by running `market:upgrade`. This will install new versions of your installed apps if updates are available in the marketplace.

Note

The user running the update command, which will likely be your webserver user, needs write permission for the `/apps` folder. If they don't have write permission, the command may report that the update was successful, however it may silently fail.

Clustered/Multi-Server Environment

The Market app, both the UI and command line, are not, *currently*, designed to operate on clustered installations. Given that, you will have to update the applications on each server in the cluster individually. There are several ways to do this. But here is a concise approach:

1. Download the latest server release (whether [the tarball](#) or [the zip archive](#)).
2. Download your installed apps from [the ownCloud marketplace](#).

3. Combine them together into one installation source, such as a *Docker* or *VM image*, or an *Ansible script*, etc.
4. Apply the combined upgrade across all the cluster nodes in your ownCloud setup.

Configuration

Database Configuration

Converting Database Type

SQLite is good for testing ownCloud, as well as small, single-user, ownCloud servers. But, **it does not scale** for large, multi-user sites. If you have an existing ownCloud installation which uses SQLite, and you want to convert to a better performing database, such as *MySQL*, *MariaDB* or *PostgreSQL*, you can use the ownCloud command line tool: occ.

Note

ownCloud Enterprise edition does not support SQLite.

Run the conversion

After you have setup the new database, in **the ownCloud root folder** run the following command to convert the database to the new format:

```
php occ db:convert-type [options] type username hostname database
```

Note

The converter searches for apps in your configured app folders and uses the schema definitions in the apps to create the new table. As a result, tables of removed apps will not be converted — even with option `--all-apps`

For example

```
php occ db:convert-type --all-apps mysql oc_mysql_user 127.0.0.1 new_db_name
```

To successfully proceed with the conversion, you must type `yes` when prompted with the question `Continue with the conversion?` On success the converter will automatically configure the new database in your ownCloud config `config.php`.

Unconvertible Tables

If you updated your ownCloud installation then the old tables, which are not used anymore, might still exist. The converter will tell you which ones.

```
The following tables will not be converted:  
oc_permissions  
...
```

You can ignore these tables. Here is a list of known old tables:

- `oc_calendar_calendars`
- `oc_calendar_objects`
- `oc_calendar_share_calendar`
- `oc_calendar_share_event`
- `oc_fscache`

- oc_log
- oc_media_albums
- oc_media_artists
- oc_media_sessions
- oc_media_songs
- oc_media_users
- oc_permissions
- oc_queuedtasks
- oc_sharing

Database Configuration

ownCloud requires a database in which administrative data is stored. The following databases are currently supported:

- MySQL / MariaDB
- PostgreSQL
- Oracle (ownCloud Enterprise edition only)

The MySQL or MariaDB databases are the recommended database engines.

Requirements

Choosing to use MySQL / MariaDB, PostgreSQL, or Oracle (ownCloud Enterprise edition only) as your database requires that you install and set up the server software first. (Oracle users, see [Oracle Database Setup](#).)

Note

The steps for configuring a third party database are beyond the scope of this document. Please refer to the documentation for your specific database choice for instructions.

MySQL / MariaDB with Binary Logging Enabled

ownCloud is currently using a TRANSACTION_READ_COMMITTED transaction isolation to avoid data loss under high load scenarios (e.g., by using the sync client with many clients/users and many parallel operations). This requires a disabled or correctly configured binary logging when using MySQL or MariaDB. Your system is affected if you see the following in your log file during the installation or update of ownCloud:

An unhandled exception has been thrown: exception ‘PDOException’ with message ‘SQLSTATE[HY000]: General error: 1665 Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine limited to row-based logging. InnoDB is limited to row-logging when transaction isolation level is READ COMMITTED or READ UNCOMMITTED.’

There are two solutions. One is to disable binary logging. Binary logging records all changes to your database, and how long each change took. The purpose of binary logging is to enable replication and to support backup operations.

The other is to change the BINLOG_FORMAT = STATEMENT in your database configuration file, or possibly in your database startup script, to BINLOG_FORMAT = MIXED or BINLOG_FORMAT = ROW. See [Overview of the Binary Log](#) and [The Binary Log](#) for detailed information.

MySQL / MariaDB “READ COMMITTED” transaction isolation level

As discussed above ownCloud is using the TRANSACTION_READ_COMMITTED transaction isolation level. Some database configurations are enforcing other transaction isolation levels. To avoid data loss under high load scenarios

Configuration

(e.g., by using the sync client with many clients/users and many parallel operations) you need to configure the transaction isolation level accordingly. Please refer to the [MySQL manual](#) for detailed information.

MySQL / MariaDB storage engine

Since ownCloud 7 only InnoDB is supported as a storage engine. There are some shared hosts who do not support InnoDB and only MyISAM. Running ownCloud on such an environment is not supported.

Parameters

For setting up ownCloud to use any database, use the instructions in The Installation Wizard. You should not have to edit the respective values in the config/config.php. However, in special cases (for example, if you want to connect your ownCloud instance to a database created by a previous installation of ownCloud), some modification might be required.

Configuring a MySQL or MariaDB Database

If you decide to use a MySQL or MariaDB database, ensure the following:

- That you have installed and enabled the `pdo_mysql` extension in PHP
- That the `mysql.default_socket` points to the correct socket (if the database runs on the same server as ownCloud).

Note

MariaDB is backwards compatible with MySQL. All instructions work for both, so you will not need to replace or revise any, existing, MySQL client commands.

The PHP configuration in `/etc/php5/conf.d/mysql.ini` could look like this:

```
# configuration for PHP MySQL module
extension=pdo_mysql.so

[mysql]
mysql.allow_local_infile=On
mysql.allow_persistent=On
mysql.cache_size=2000
mysql.max_persistent=-1
mysql.max_links=-1
mysql.default_port=
mysql.default_socket=/var/lib/mysql/mysql.sock # Debian squeeze: /var/run/mysqld/mysqld.sock
mysql.default_host=
mysql.default_user=
mysql.default_password=
mysql.connect_timeout=60
mysql.trace_mode=Off
```

Now you need to create a database user and the database itself by using the MySQL command line interface. The database tables will be created by ownCloud when you login for the first time.

To start the MySQL command line mode use:

```
mysql -uroot -p
```

Then a `mysql>` or `MariaDB [root]>` prompt will appear. Now enter the following lines and confirm them with the enter key:

```
CREATE DATABASE IF NOT EXISTS owncloud;
GRANT ALL PRIVILEGES ON owncloud.* TO 'username'@'localhost' IDENTIFIED BY 'password';
```

You can quit the prompt by entering:

```
quit
```

An ownCloud instance configured with MySQL would contain the hostname on which the database is running, a valid username and password to access it, and the name of the database. The config/config.php as created by the The Installation Wizard would therefore contain entries like this:

```
<?php  
  
"dbtype"      => "mysql" ,  
"dbname"      => "owncloud" ,  
"dbuser"       => "username" ,  
"dbpassword"   => "password" ,  
"dbhost"       => "localhost" ,  
"dbtableprefix" => "oc_ " ,
```

Configure MySQL for 4-byte Unicode Support

For supporting such features as emoji, you have to enable 4-byte Unicode support in MySQL (instead of the default 3) and in ownCloud. If you have a new installation, you don't need to do anything, as mb4 support is checked during setup, and used if available. If it's available, ownCloud is configured to use it.

However, if you have an existing installation that you need to convert to use 4-byte Unicode support, then you need to do two things:

1. In your MySQL configuration, add the configuration settings below. If you already have them configured, update them to reflect the values specified:

```
[mysqld]  
innodb_large_prefix=ON  
innodb_file_format=Barracuda  
innodb_file_per_table=ON
```

Then, run the following command:

```
./occ db:convert-mysql-charset
```

When this is done, tables will be created with a:

- utf8mb4 character set.
- utf8mb4_bin collation.
- row_format of compressed.

For more information, please either refer to lines 1126 to 1156 in config/config.sample.php, or have a read through the following links:

- https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html#sysvar_innodb_large_prefix
- https://mariadb.com/kb/en/mariadb/xtradbinnodb-server-system-variables/#innodb_large_prefix
- <http://www.tocker.ca/2013/10/31/benchmarking-innodb-page-compression-performance.html>
- <http://mechanics.flite.com/blog/2014/07/29/using-innodb-large-prefix-to-avoid-error-1071/>
- <http://dev.mysql.com/doc/refman/5.7/en/charset-unicode-utf8mb4.html>

Note

This is not required for new installations, only existing ones, as mb4 support is checked during setup, and used if available.

PostgreSQL Database

If you decide to use a PostgreSQL database make sure that you have installed and enabled the PostgreSQL extension in PHP. The PHP configuration in /etc/php5/conf.d/pgsql.ini could look like this:

Configuration

```
# configuration for PHP PostgreSQL module
extension=pdo_pgsql.so
extension=pgsql.so

[PostgreSQL]
pgsql.allow_persistent = On
pgsql.auto_reset_persistent = Off
pgsql.max_persistent = -1
pgsql.max_links = -1
pgsql.ignore_notice = 0
pgsql.log_notice = 0
```

The default configuration for PostgreSQL (at least in Ubuntu 14.04) is to use the peer authentication method. Check /etc/postgresql/9.3/main/pg_hba.conf to find out which authentication method is used in your setup. To start the postgres command line mode use:

```
sudo -u postgres psql -d template1
```

Then a **template1=#** prompt will appear. Now enter the following lines and confirm them with the enter key:

```
CREATE USER username CREATEDB;
CREATE DATABASE owncloud OWNER username;
```

You can quit the prompt by entering:

```
\q
```

An ownCloud instance configured with PostgreSQL would contain the path to the socket on which the database is running as the hostname, the system username the php process is using, and an empty password to access it, and the name of the database. The config/config.php as created by the The Installation Wizard would therefore contain entries like this:

```
<?php

"dbtype"      => "pgsql",
"dbname"      => "owncloud",
"dbuser"       => "username",
"dbpassword"   => " ",
"dbhost"       => "/var/run/postgresql",
"dbtableprefix" => "oc_",
```

Note

The host actually points to the socket that is used to connect to the database. Using localhost here will not work if PostgreSQL is configured to use peer authentication. Also note, that no password is specified, because this authentication method doesn't use a password.

If you use another authentication method (not peer), you'll need to use the following steps to get the database setup: Now you need to create a database user and the database itself by using the PostgreSQL command line interface. The database tables will be created by ownCloud when you login for the first time.

To start the PostgreSQL command line mode use:

```
psql -hlocalhost -Upostgres
```

Then a **postgres=#** prompt will appear. Now enter the following lines and confirm them with the enter key:

```
CREATE USER username WITH PASSWORD 'password';
CREATE DATABASE owncloud TEMPLATE template0 ENCODING 'UNICODE';
ALTER DATABASE owncloud OWNER TO username;
GRANT ALL PRIVILEGES ON DATABASE owncloud TO username;
```

You can quit the prompt by entering:

```
\q
```

An ownCloud instance configured with PostgreSQL would contain the hostname on which the database is running, a valid username and password to access it, and the name of the database. The config/config.php as created by the The Installation Wizard would therefore contain entries like this:

```
<?php  
  
"dbtype"      => "pgsql" ,  
"dbname"       => "owncloud" ,  
"dbuser"        => "username" ,  
"dbpassword"    => "password" ,  
"dbhost"        => "localhost" ,  
"dbtableprefix" => "oc_ " ,
```

Troubleshooting

How to workaround General error: 2006 MySQL server has gone away

The database request takes too long and therefore the MySQL server times out. Its also possible that the server is dropping a packet that is too large. Please refer to the manual of your database for how to raise the configuration options `wait_timeout` and/or `max_allowed_packet`.

Some shared hosts are not allowing the access to these config options. For such systems ownCloud is providing a `dbdriveroptions` configuration option within your config/config.php where you can pass such options to the database driver. Please refer to Config.php Parameters for an example.

How can I find out if my MySQL/PostgreSQL server is reachable?

To check the server's network availability, use the ping command on the server's host name (db.server.com in this example):

```
ping db.server.dom
```

```
PING db.server.dom (ip-address) 56(84) bytes of data.  
64 bytes from your-server.local.lan (192.168.1.10): icmp_req=1 ttl=64 time=3.64 ms  
64 bytes from your-server.local.lan (192.168.1.10): icmp_req=2 ttl=64 time=0.055 ms  
64 bytes from your-server.local.lan (192.168.1.10): icmp_req=3 ttl=64 time=0.062 ms
```

For a more detailed check whether the access to the database server software itself works correctly, see the next question.

How can I find out if a created user can access a database?

The easiest way to test if a database can be accessed is by starting the command line interface:

MySQL:

Assuming the database server is installed on the same system you're running the command from, use:

```
mysql -uUSERNAME -p
```

To access a MySQL installation on a different machine, add the `-h` option with the respective host name:

```
mysql -uUSERNAME -p -h HOSTNAME
```

```
mysql> SHOW VARIABLES LIKE "version";  
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| version      | 5.1.67 |  
+-----+-----+  
1 row in set (0.00 sec)  
mysql> quit
```

PostgreSQL:

Assuming the database server is installed on the same system you're running the command from, use:

```
psql -Username -downcloud
```

To access a MySQL installation on a different machine, add the -h option with the respective host name:

```
psql -Username -downcloud -h HOSTNAME
```

```
postgres=# SELECT version();
PostgreSQL 8.4.12 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 4.1.3 20080704 (prerelease)
(1 row)
postgres=# \q
```

Useful SQL commands

Show Database Users:

```
MySQL      : SELECT User,Host FROM mysql.user;
PostgreSQL: SELECT * FROM pg_user;
```

Show available Databases:

```
MySQL      : SHOW DATABASES;
PostgreSQL: \l
```

Show ownCloud Tables in Database:

```
MySQL      : USE owncloud; SHOW TABLES;
PostgreSQL: \c owncloud; \d
```

Quit Database:

```
MySQL      : quit
PostgreSQL: \q
```

File Sharing and Management

File Sharing

ownCloud users can :

- Share files with their ownCloud groups and other users on the same ownCloud server
- Share files with ownCloud users on other ownCloud servers
- Create public shares for people who are not ownCloud users.

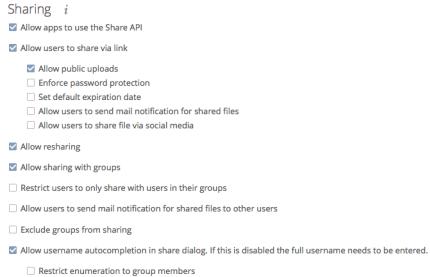
You have control of a number of user permissions on file shares:

- Allow users to share files
- Allow users to create public shares
- Require a password on public shares
- Allow public uploads to public shares
- Require an expiration date on public share links
- Allow resharing
- Restrict sharing to group members only
- Allow email notifications of new public shares
- Exclude groups from creating shares

Note

ownCloud Enterprise includes a Share Link Password Policy app; see [Password Policy](#).

Configure your sharing policy on your Admin page in the Sharing section.



- Check Allow apps to use the Share API to enable users to share files. If this is not checked, no users can create file shares.
- Check Allow users to share via link to enable creating public shares for people who are not ownCloud users via hyperlink.
- Check Enforce password protection to force users to set a password on all public share links. This does not apply to local user and group shares.
- Check Allow public uploads to allow anyone to upload files to public shares.
- Check Allow users to send mail notification for shared files to enable sending notifications from ownCloud. (Your ownCloud server must be configured to send mail)
- Check Allow users to share file via social media to enable displaying of a set of links that allow for quickly sharing files and share links via *Twitter*, *Facebook*, *Google+*, *Disaspora*, and email.



- Check Set default expiration date to set a default expiration date on public shares.
- Check Allow resharing to enable users to re-share files shared with them.
- Check Restrict users to only share with users in their groups to confine sharing within group memberships.

Note

This setting does not apply to the Federated Cloud sharing feature. If Federated Cloud Sharing is enabled, users can still share items with any users on any instances (including the one they are on) via a remote share.

- Check Allow users to send mail notification for shared files enables users to send an email notification to every ownCloud user that the file is shared with.
- Check Exclude groups from sharing to prevent members of specific groups from creating any file shares in those groups. When you check this, you'll get a dropdown list of all your groups to choose from. Members of excluded groups can still receive shares, but not create any
- Check Allow username autocompletion in share dialog to enable auto-completion of ownCloud usernames.
- Check Restrict enumeration to group members to restrict auto-completion of ownCloud usernames to only those users who are members of the same group(s) that the user is in.

Note

ownCloud does not preserve the mtime (modification time) of directories, though it does update the mtimes on files. See [Wrong folder date when syncing](#) for discussion of this.

Transferring Files to Another User

You may transfer files from one user to another with `occ`. The command transfers either all or a limited set of files from one user to another. It also transfers the shares and metadata info associated with those files (*shares*, *tags*, and *comments*, etc). This is useful when you have to transfer a user's files to another user before you delete them.

Important

Trashbin contents are not transferred.

Here is an example of how to transfer all files from one user to another.

```
occ files:transfer-ownership <source-user> <destination-user>
```

Here is an example of how to transfer a *limited group* a single folder from one user to another. In it, `folder/to/move`, and any file and folder inside it will be moved to `<destination-user>`.

```
sudo -u www-data php occ files:transfer-ownership --path="folder/to/move" <source-user> <des...
```

When using this command keep two things in mind:

1. The directory provided to the `--path` switch **must** exist inside `data/<source-user>/files`.
2. The directory (and its contents) won't be moved as is between the users. It'll be moved inside the destination user's `files` directory, and placed in a directory which follows the format: transferred from `<source-user>` on `<timestampe>`. Using the example above, it will be stored under: `data/<destination-user>/files/transferred from <source-user> on 20170426_124510/` (See Using the `occ` Command for a complete `occ` reference.)

Creating Persistent File Shares

When a user is deleted, their files are also deleted. As you can imagine, this is a problem if they created file shares that need to be preserved, because these disappear as well. In ownCloud files are tied to their owners, so whatever happens to the file owner also happens to the files.

One solution is to create persistent shares for your users. You can retain ownership of them, or you could create a special user for the purpose of establishing permanent file shares. Simply create a shared folder in the usual way, and share it with the users or groups who need to use it. Set the appropriate permissions on it, and then no matter which users come and go, the file shares will remain. Because all files added to the share, or edited in it, automatically become owned by the owner of the share regardless of who adds or edits them.

Configuring Federation Sharing

Federated Cloud Sharing is now managed by the Federation app (9.0+), and is now called Federation sharing. When you enable the Federation app you can easily and securely link file shares between ownCloud servers, in effect creating a cloud of ownClouds.

Sharing With ownCloud 8 and Older

Direct Federation shares (Creating a new Federation Share (9.0+ only)) are not supported in ownCloud 8 and older, so you must create Federation shares with public links (Creating Federation Shares via Public Link Share).

Creating a new Federation Share (9.0+ only)

Configuration

Follow these steps to create a new Federation share between two ownCloud 9.0+ servers. This requires no action by the user on the remote server; all it takes is a few steps on the originating server.

1. Enable the Federation app.
2. Go to your ownCloud Admin page and scroll to the Sharing section. Verify that **Allow users on this server to send shares to other servers** and **Allow users on this server to receive shares from other servers** are enabled.
3. Now go to the Federation section. By default, **Add server automatically once a federated share was created successfully** is checked. The Federation app supports creating a list of trusted ownCloud servers, which allows the trusted servers to exchange user directories and auto-complete the names of external users when you create shares. If you do not want this enabled, then un-check it.

Federation

ownCloud Federation allows you to connect with other trusted ownClouds to exchange the user directory. For example this will be used to auto-complete external users for federated sharing.

Add server automatically once a federated share was created successfully

Trusted ownCloud Servers

[+ Add ownCloud server](#)

4. Then, go to your Files page and select a folder to share. Click the share icon, and then enter the username and URL of the user on the remote ownCloud server. In this example, that is `freda@https://example.com/owncloud`. When ownCloud verifies the link, it displays it with the **(remote)** label. Click on this label to establish the link.

Name	Size
Documents	35 k
Dropbox	1.1 G
kitties	93 k
Photos	663 k

Sharing details for 'kitties':

- Sharing tab selected
- Sharing with `freda@https://example.com/owncloud (remote)`
- Info icon (*i*) next to the sharing entry

5. When the link is successfully completed, you have a single share option, and that is **can edit**.

Activities Comments **Sharing**

Share with users, groups or roles

F

i



freda@https://example.com/owncloud/

can edit

You may disconnect the share at any time by clicking the trash can icon.

Configuring Trusted ownCloud Servers

You may create a list of trusted ownCloud servers for Federation sharing. This allows your linked ownCloud servers to share user directories, and to auto-fill user names in share dialogs. If **Add server automatically once a federated share was created successfully** is enabled on your Admin page, servers will be automatically added to your trusted list when you create new Federation shares.

You may also enter ownCloud server URLs in the **Add ownCloud Server** field. The yellow light indicates a successful connection, with no user names exchanged. The green light indicates a successful connection with user names exchanged. A red light means the connection failed.

Federation

ownCloud Federation allows you to connect with other trusted ownClouds to exchange user data.

Add server automatically once a federated share was created successfully

Trusted ownCloud Servers

+ Add ownCloud server



http://localhost/federation/



https://server2



https://server3

Creating Federation Shares via Public Link Share

You'll need to use a Public Link Share to create Federation shares with ownCloud 8.x and older.

Check the **Share Link** checkbox to expose more sharing options (which are described more fully in File Sharing). You may create a Federation share by allowing ownCloud to create a public link for you, and then email it to the person you want to create the share with.

Share link
<http://ubuntu2/owncloud/index.php/s/ujh7VbEFuvjXIGu>

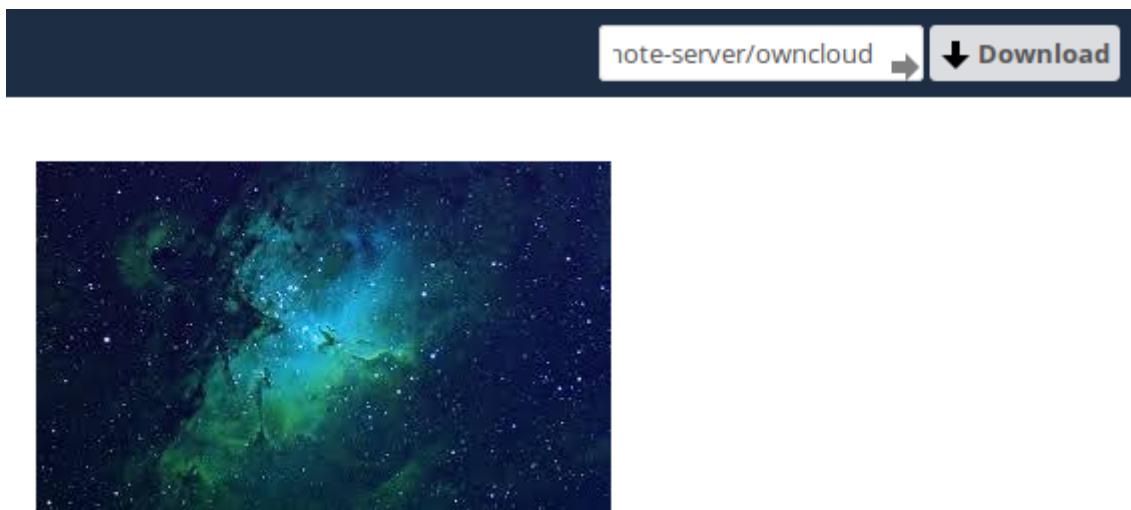
Password protect
friend@example.com

Set expiration date

You may optionally set a password and expiration date on it. When your recipient receives your email they must click the link, or copy it to a Web browser. They will see a page displaying a thumbnail of the file, with a button to **Add to your ownCloud**.

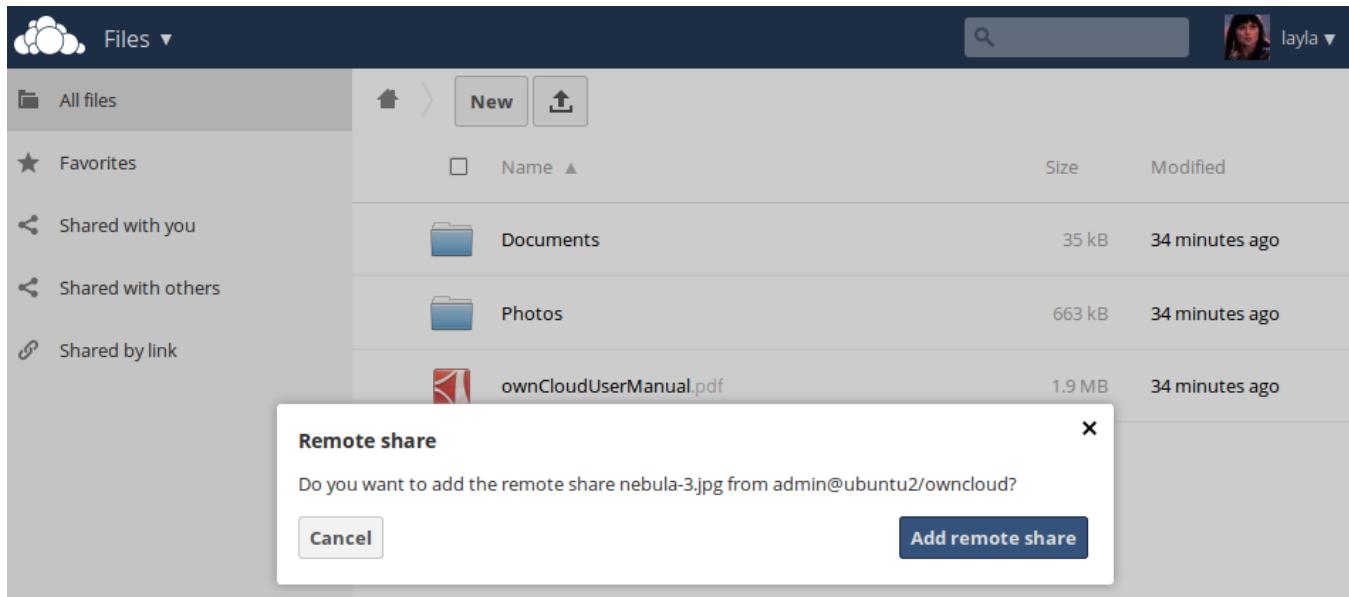


Your recipient should click the **Add to your ownCloud** button. On the next screen your recipient needs to enter the URL to their ownCloud server, and then press the return key.



Your recipient has to take one more step, and that is to confirm creating the federated cloud share link by clicking the **Add remote share** button.

Configuration



Un-check the `Share Link` checkbox to disable any federated cloud share created this way.

Configuration Tips

The Sharing section on your Admin page allows you to control how your users manage federated cloud shares:

- Check `Enforce password protection` to require passwords on link shares.
- Check `Set default expiration date` to require an expiration date on link shares.
- Check `Allow public uploads` to allow two-way file sharing.

Your Apache Web server must have `mod_rewrite` enabled, and you must have `trusted_domains` correctly configured in `config.php` to allow external connections (see The Installation Wizard). Consider also enabling SSL to encrypt all traffic between your servers .

Your ownCloud server creates the share link from the URL that you used to log into the server, so make sure that you log into your server using a URL that is accessible to your users. For example, if you log in via its LAN IP address, such as `http://192.168.10.50`, then your share URL will be something like `http://192.168.10.50/owncloud/index.php/s/jWfCfTVztGlWTJe`, which is not accessible outside of your LAN. This also applies to using the server name; for access outside of your LAN you need to use a fully-qualified domain name such as `http://myserver.example.com`, rather than `http://myserver`.

Uploading big files > 512MB

The default maximum file size for uploads, in ownCloud, is 512MB. You can increase this limit up to the maximum file size which your filesystem, operating system, or other software allows, for example:

- < 2GB on a 32Bit OS-architecture
- < 2GB with IE6 - IE8
- < 4GB with IE9 - IE11

64-bit filesystems have much higher limits. Please consult the documentation for your filesystem.

Note

The ownCloud sync client itself however is able to upload files of any size, as it uploads files by transmitting them in small chunks. But, it can never exceed the maximum file size limits of the remote host.

System Configuration

- Make sure that the latest version of PHP (at least 5.6) is installed
- Disable user quotas, which makes them unlimited
- Your temp file or partition has to be big enough to hold multiple parallel uploads from multiple users; e.g. if the max upload size is 10GB and the average number of users uploading at the same time is 100: temp space has to hold at least 10x100 GB

Configuring Your Web server

Note

ownCloud comes with its own `owncloud/.htaccess` file. Because `php-fpm` can't read PHP settings in `.htaccess` these settings must be set in the `owncloud/.user.ini` file.

Set the following two parameters inside the corresponding `php.ini` file (see the **Loaded Configuration File** section of PHP Version and Information to find your relevant `php.ini` files)

```
php_value upload_max_filesize = 16G  
php_value post_max_size = 16G
```

Adjust these values for your needs. If you see PHP timeouts in your logfiles, increase the timeout values, which are in seconds:

```
php_value max_input_time 3600  
php_value max_execution_time 3600
```

The `mod_reqtimeout` Apache module could also stop large uploads from completing. If you're using this module and getting failed uploads of large files either disable it in your Apache config or raise the configured `RequestReadTimeout` timeouts.

There are also several other configuration options in your Web server config which could prevent the upload of larger files. Please see the manual of your Web server for how to configure those values correctly:

Apache

- `LimitRequestBody`
- `SSLRenegBufferSize`

Apache with mod_fcgid

- `FcgidMaxRequestInMem`
- `FcgidMaxRequestLen`

Note

If you are using Apache/2.4 with `mod_fcgid`, as of February/March 2016, `FcgidMaxRequestInMem` still needs to be significantly increased from its default value to avoid the occurrence of segmentation faults when uploading big files. This is not a regular setting but serves as a workaround for [Apache with mod_fcgid bug #51747](#).

Setting `FcgidMaxRequestInMem` significantly higher than normal may no longer be necessary, once bug #51747 is fixed.

NGINX

- `client_max_body_size`
- `fastcgi_read_timeout`

- [client_body_temp_path](#)

Since NGINX 1.7.11 a new config option `fastcgi_request_buffering` is available. Setting this option to `fastcgi_request_buffering off;` in your NGINX config might help with timeouts during the upload. Furthermore it helps if you're running out of disc space on the `/tmp` partition of your system.

For more info how to configure NGINX to raise the upload limits see also [this](#) wiki entry.

Note

Make sure that `client_body_temp_path` points to a partition with adequate space for your upload file size, and on the same partition as the `upload_tmp_dir` or `tempdirectory` (see below). For optimal performance, place these on a separate hard drive that is dedicated to swap and temp storage.

If your site is behind a NGINX frontend (for example a loadbalancer):

By default, downloads will be limited to 1GB due to `proxy_buffering` and `proxy_max_temp_file_size` on the frontend.

- If you can access the frontend's configuration, disable `proxy_buffering` or increase `proxy_max_temp_file_size` from the default 1GB.
- If you do not have access to the frontend, set the `X-Accel-Buffering` header to `add_header X-Accel-Buffering no;` on your backend server.

Configuring PHP

If you don't want to use the ownCloud `.htaccess` or `.user.ini` file, you may configure PHP instead. Make sure to comment out any lines `.htaccess` pertaining to upload size, if you entered any.

If you are running ownCloud on a 32-bit system, any `open_basedir` directive in your `php.ini` file needs to be commented out.

Set the following two parameters inside `php.ini`, using your own desired file size values:

```
upload_max_filesize = 16G  
post_max_size = 16G
```

Tell PHP which temp file you want it to use:

```
upload_tmp_dir = /var/big_temp_file/
```

Output Buffering must be turned off in `.htaccess` or `.user.ini` or `php.ini`, or PHP will return memory-related errors:

- `output_buffering = 0`

Configuring ownCloud

As an alternative to the `upload_tmp_dir` of PHP (e.g., if you don't have access to your `php.ini`) you can also configure a temporary location for uploaded files by using the `tempdirectory` setting in your `config.php` (See Config.php Parameters).

If you have configured the `session_lifetime` setting in your `config.php` (See Config.php Parameters) file then make sure it is not too low. This setting needs to be configured to at least the time (in seconds) that the longest upload will take. If unsure remove this completely from your configuration to reset it to the default shown in the `config.sample.php`.

Configuring upload limits within the GUI

If all prerequisites described in this documentation are in place an admin can change the upload limits on demand by using the File handling input box within the administrative backend of ownCloud.

File handling

Maximum upload size **1 GB**

With PHP-FPM this value may take up to 5 minutes to take effect after saving.

Save

Depending on your environment you might get an insufficient permissions message shown for this input box.

File handling

Maximum upload size **1 GB**

Can not be edited from here due to insufficient permissions.

To be able to use this input box you need to make sure that:

- Your Web server is able to use the `.htaccess` file shipped by ownCloud (Apache only)
- The user your Web server is running as has write permissions to the files `.htaccess` and `.user.ini`

Set Strong Directory Permissions might prevent write access to these files. As an admin you need to decide between the ability to use the input box and a more secure ownCloud installation where you need to manually modify the upload limits in the `.htaccess` and `.user.ini` files described above.

General upload issues

Various environmental factors could cause a restriction of the upload size. Examples are:

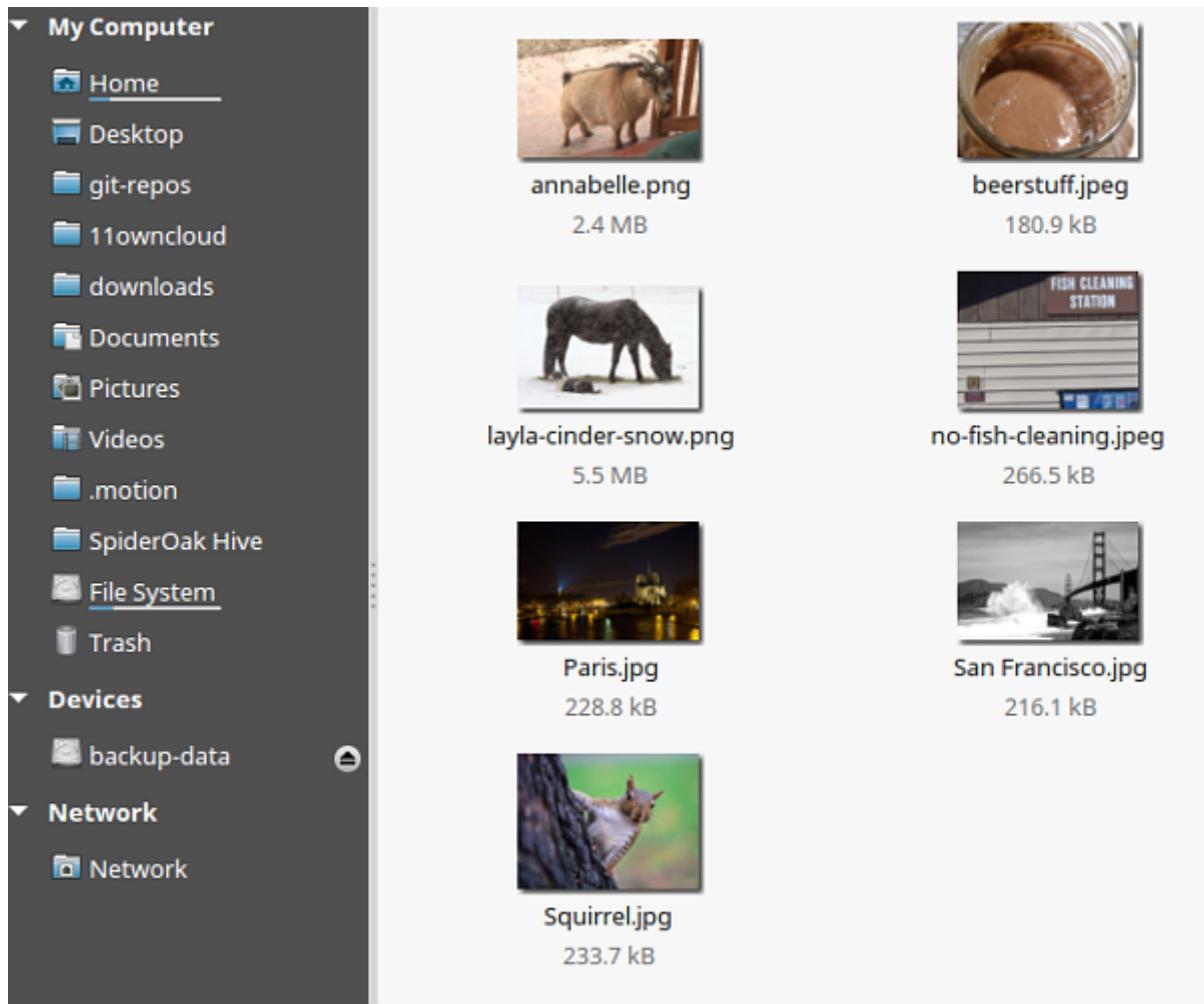
- The LVE Manager of CloudLinux which sets a I/O limit
- Some services like Cloudflare are also known to cause uploading issues
- Upload limits enforced by proxies used by your clients
- Other webserver modules like described in General Troubleshooting

Providing Default Files

You may distribute a set of default files and folders to all users by placing them in the `owncloud/core/skeleton` directory on your ownCloud server. These files appear only to new users after their initial login, and existing users will not see files that are added to this directory after their first login. The files in the skeleton directory are copied into the users' data directories, so they may change and delete the files without affecting the originals.

This screenshot shows a set of photos in the skeleton directory.

Configuration



They appear on the user's ownCloud Files page just like any other files.

The screenshot shows the ownCloud Files interface. On the left, there is a sidebar with a tree view navigation:

- All files
- Favorites
- Shared with you
- Shared with others
- Shared by link
- Deleted files

The main area displays a list of files:

Name	Size	Modified
annabelle.png	2.3 MB	6 minutes ago
beerstuff.jpeg	177 kB	6 minutes ago
layla-cinder-snow.png	5.3 MB	6 minutes ago
no-fish-cleaning.jpeg	260 kB	6 minutes ago
Paris.jpg	223 kB	6 minutes ago
San Francisco.jpg	211 kB	6 minutes ago
Squirrel.jpg	228 kB	6 minutes ago

Additional Configuration

Configuration

The configuration option `skeletondirectory` available in your `config.php` (See `Config.php` Parameters) allows you to configure the directory where the skeleton files are located. These files will be copied to the data directory of new users. Leave empty to not copy any skeleton files.

Configuring External Storage (GUI)

The External Storage Support application enables you to mount external storage services and devices as secondary ownCloud storage devices. You may also allow users to mount their own external storage services.

ownCloud 9.0 introduces a new set of occ commands for managing external storage.

Also new in 9.0 is an option for the ownCloud admin to enable or disable sharing on individual external mountpoints (see Mount Options). Sharing on such mountpoints is disabled by default.

Enabling External Storage Support

Warning

Enabling this app will disable the **Stay logged in** checkbox on the login page.

The External storage support application is enabled on your Apps page.



External storage support 0.5.2

by Robin Appelman, Michael Gapczynski, Vincent Petry (AGPL-licensed)

✓ Official

Show description ...

Disable

Storage Configuration

To create a new external storage mount, select an available backend from the dropdown **Add storage**. Each backend has different required options, which are configured in the configuration fields.

External Storage

Folder name

External storage

Configuration

Available for

Folder name

Add storage

- Amazon S3 and compliant
- Dropbox
- FTP
- Google Drive
- Local
- OpenStack Object Storage
- ownCloud
- SFTP
- SMB / CIFS
- SMB / CIFS using OC login
- WebDAV

Configuration

Each backend may also accept multiple authentication methods. These are selected with the dropdown under **Authentication**. Different backends support different authentication mechanisms; some specific to the backend, others are more generic. See External Storage Authentication mechanisms for more detailed information.

When you select an authentication mechanism, the configuration fields change as appropriate for the mechanism. The SFTP backend, for one example, supports **username and password**, **Log-in credentials, save in session**, and **RSA public key**.

External Storage

Folder name	External storage	Authentication	Configuration
SFTP	SFTP	Username and password	Host Root Username Password

The screenshot shows the 'External storage' configuration page. In the 'Authentication' column, a dropdown menu is open, showing four options: 'Username and password' (selected), 'Log-in credentials, save in session', and 'RSA public key'. To the right of the dropdown, there are four input fields with red borders: 'Host', 'Root', 'Username', and 'Password'. The 'Host' field contains 'SFTP' and the 'Root' field contains 'SFTP'.

Required fields are marked with a red border. When all required fields are filled, the storage is automatically saved. A green dot next to the storage row indicates the storage is ready for use. A red or yellow icon indicates that ownCloud could not connect to the external storage, so you need to re-check your configuration and network availability.

If there is an error on the storage, it will be marked as unavailable for ten minutes. To re-check it, click the colored icon or reload your Admin page.

User and Group Permissions

A storage configured in a user's Personal settings is available only to the user that created it. A storage configured in the Admin settings is available to all users by default, and it can be restricted to specific users and groups in the **Available for** field.



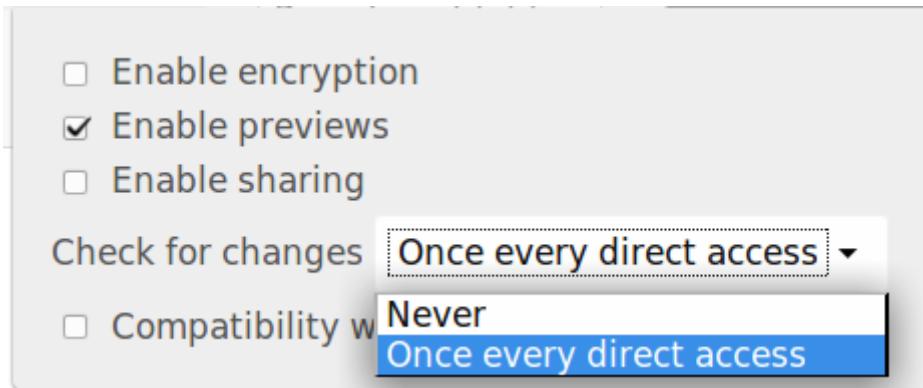
Mount Options

Hover your cursor to the right of any storage configuration to expose the settings button and trashcan. Click the trashcan to delete the mountpoint. The settings button allows you to configure each storage mount individually with the following options:

- Encryption
- Previews
- Enable Sharing
- Filesystem check frequency (Never, Once per direct access)

The **Encryption** checkbox is visible only when the Encryption app is enabled.

Enable Sharing allows the ownCloud admin to enable or disable sharing on individual mountpoints. When sharing is disabled the shares are retained internally, so that you can re-enable sharing and the previous shares become available again. Sharing is disabled by default.



Using Self-Signed Certificates

When using self-signed certificates for external storage mounts the certificate must be imported into ownCloud. Please refer to Importing System-wide and Personal SSL Certificates for more information.

Available storage backends

The following backends are provided by the external storages app. Other apps may provide their own backends, which are not listed here.

Amazon S3

To connect your Amazon S3 buckets to ownCloud, you will need:

- S3 access key
- S3 secret key
- Bucket name

In the **Folder name** field enter a local folder name for your S3 mountpoint. If this does not exist it will be created.

In the **Available for** field enter the users or groups who have permission to access your S3 mount.

The `Enable SSL` checkbox enables HTTPS connections; using HTTPS is always highly-recommended.

External Storage

Folder name	External storage	Configuration	Available for
 AmazonS3	Amazon S3 and compliant	AKIAIOSHDCA77WFI oc-files-wc Hostname (optional) Port (optional) Region (optional) <input checked="" type="checkbox"/> Enable SSL <input checked="" type="checkbox"/> Enable Path Style	All Users 

Optionally, you can override the hostname, port and region of your S3 server, which is required for non-Amazon servers such as Ceph Object Gateway.

Configuration

Enable path style is usually not required (and is, in fact, incompatible with newer Amazon datacenters), but can be used with non-Amazon servers where the DNS infrastructure cannot be controlled. Ordinarily, requests will be made with `http://bucket.hostname.domain/`, but with path style enabled, requests are made with `http://hostname.domain/bucket` instead.

See Configuring External Storage (GUI) for additional mount options and information.

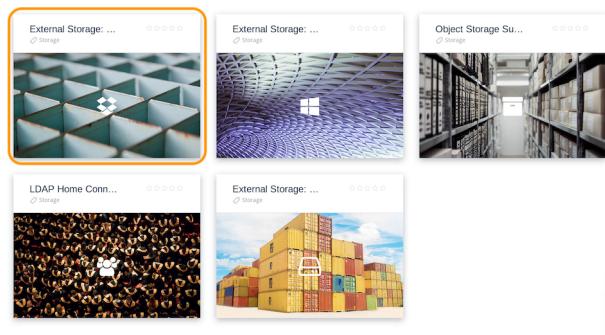
See External Storage Authentication mechanisms for more information on authentication schemes.

Dropbox

To connect Dropbox to your ownCloud installation requires four steps to be completed.

1. Install the “External Storage: Dropbox” app from the ownCloud Marketplace
2. Create a Dropbox app
3. Create a Dropbox storage share
4. Use the Dropbox share

Step One - Install the “External Storage: Dropbox” app from the ownCloud Marketplace



1. Click **Market** in the ownCloud web UI dropdown menu on the left side
2. Go to the **Storage** category
3. select **External Storage: Dropbox** App
4. Click **INSTALL**

Step Two - Create a Dropbox app

Next, you need to create a Dropbox app. To do that, open the new app creation form, where you see three questions:

1. Choose an API → “Dropbox API”
2. Choose the type of access → “App folder”
3. Name your app

With all of the required details filled out, click the blue “Create app” button, in the bottom, right-hand corner. After you do that, the settings page for the application loads.

App folder name	owncloud_x_share	Change
App key	https://(http allowed for localhost)	
App secret	Show	
OAuth 2	Redirect URIs	
	https:// (http allowed for localhost)	
	Add	

Important

Redirect URI: Here you must enter the exact URL of the page where you configure the storage.

Examples:

When configuring as admin:
http(s)://<<Server_Address>>/index.php/settings/admin?sectionid=storage

When configuring as user:
http(s)://<<Server_Address>>/index.php/settings/personal?sectionid=storage

Step Three - Create a Dropbox Share

To create a Dropbox share, under “admin -> Settings -> Admin -> Storage”, check the “Enable external storage” checkbox, if it’s not already checked. Then, in the dropdown list under “External storage”, click the first “Dropbox” option.

Note

There are two Dropbox options in the dropdown list, as Dropbox functionality is currently part of ownCloud's core. However, the internal Dropbox functionality should be removed in ownCloud 10.0.4.

Then, you need to provide a name for the folder in the “Folder name” field, and a “client key” and “client secret”, located in the “Configuration” column. The client key and client secret values are the “App key” and “App secret” fields which you saw earlier in your Dropbox app’s configuration settings page.

After you have added these three settings, click "Grant access". ownCloud then interacts with Dropbox's API to set up the new shared folder. If the process is successful, a green circle icon appears, at the far left-hand side of the row, next to the folder's name.



Other Options

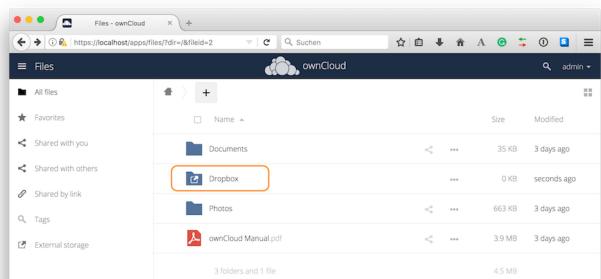
If you want to restrict access to the share to a select list of users and groups, you can add them to the field in the “Available for” column.

Step Four - Using the Dropbox Share

After a Dropbox-backed share is created, a new folder is available under "All Files". It has the name that you gave it when you created the share, and it is represented by an external share folder icon, as in the image below.

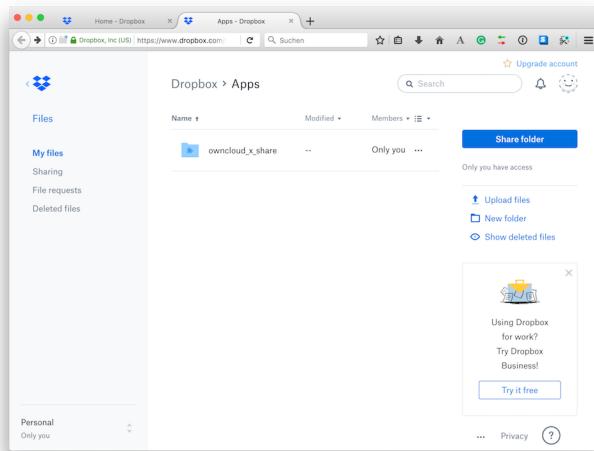


This links to a new folder in your Dropbox account, under “Dropbox > Apps”, with the name of the Dropbox app that you created.



Configuration

Now, if you add files and folders in either the new Dropbox folder or the new ownCloud folder, after being synced, they will be visible inside the other.



FTP/FTPS

To connect to an FTP server, you will need:

- A folder name for your local mountpoint; the folder will be created if it does not exist
- The URL of the FTP server
- Port number (default: 21)
- FTP server username and password
- Remote Subfolder, the FTP directory to mount in ownCloud. ownCloud defaults to the root directory. If you specify a subfolder you must leave off the leading slash. For example, `public_html/images`

Your new mountpoint is available to all users by default, and you may restrict access by entering specific users or groups in the **Available for** field.

Optionally, ownCloud can use **FTPS** (FTP over SSL) by checking **Secure ftps://**. This requires additional configuration with your root certificate if the FTP server uses a self-signed certificate (See Importing System-wide and Personal SSL Certificates).

External Storage

Folder name	External storage	Configuration	Available for
<input checked="" type="checkbox"/> FTP	FTP	<input type="text" value="ftp.example.com:22"/> <input type="text" value="username"/> <input type="password" value=""/> <input type="text" value="public.html/"/> <input checked="" type="checkbox"/> Secure ftps://	<input checked="" type="checkbox"/> support(group)

Note

The external storage **FTP/FTPS** needs the `allow_url_fopen` PHP setting to be set to 1. When having connection problems make sure that it is not set to 0 in your `php.ini`. See [PHP Version and Information](#) to learn how to find the right `php.ini` file to edit.

See [Configuring External Storage \(GUI\)](#) for additional mount options and information.

Configuration

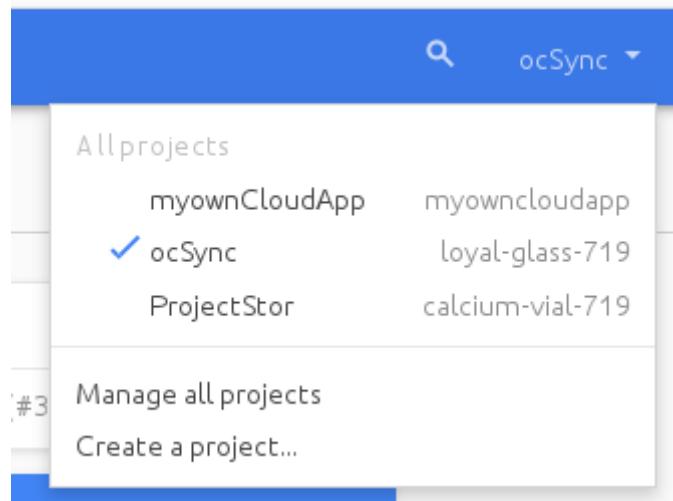
FTP uses the password authentication scheme; see External Storage Authentication mechanisms for more information on authentication schemes.

Google Drive

ownCloud uses OAuth 2.0 to connect to Google Drive. This requires configuration through Google to get an app ID and app secret, as ownCloud registers itself as an app.

All applications that access a Google API must be registered through the [Google Cloud Console](#). Follow along carefully because the Google interface is a bit of a maze and it's easy to get lost.

If you already have a Google account, such as Groups, Drive, or Mail, you can use your existing login to log into the Google Cloud Console. After logging in click the **Create Project** button.



Give your project a name, and either accept the default **Project ID** or create your own, then click the **Create** button.

New Project

Project name [?](#)

Your project ID will be gdrive-owncloud-1146 [?](#) [Edit](#)

[Show advanced options...](#)

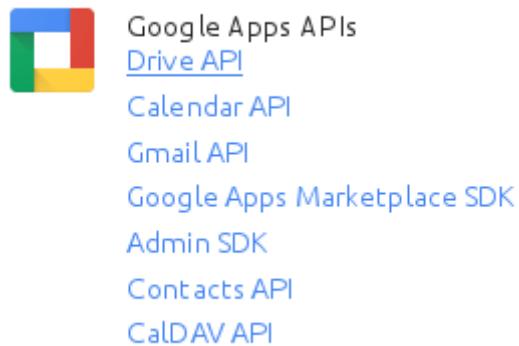
[Create](#) [Cancel](#)

A screenshot of a 'New Project' dialog box. It starts with the text 'New Project'. Below that is a 'Project name' field with a question mark icon, containing the text 'gdrive-owncloud'. Underneath the field, it says 'Your project ID will be gdrive-owncloud-1146' followed by a question mark and an 'Edit' link. There is a link 'Show advanced options...' below the project ID. At the bottom are two buttons: a blue 'Create' button and a white 'Cancel' button.

You'll be returned to your dashboard.

The screenshot shows the Google Developers Console dashboard. At the top, it displays the project name "gdrive-owncloud". Below this, there are two main sections: "Use Google APIs" (highlighted in blue) and "API" (highlighted in blue). The "Use Google APIs" section contains the text "Enable APIs, create credentials, and track your usage". The "API" section contains the text "Enable and manage APIs". On the left sidebar, the "Dashboard" tab is selected.

Google helpfully highlights your next step in blue, the **Use Google APIs** box. Make sure that your new project is selected, click on **Use Google APIs**, and it takes you to Google's APIs screen. There are many Google APIs; look for the **Google Apps APIs** and click **Drive API**.



Drive API takes you to the API Manager overview. Click the blue **Enable API** button.

The screenshot shows the API Manager overview for the Drive API. The left sidebar has tabs for "Overview" and "Credentials", with "Overview" selected. The main content area shows the "Drive API" with a brief description: "The Drive API allows clients to access resources from Google Drive.", a "Learn more" link, and a "Try this API in APIs Explorer" button.

Now you must create your credentials, so click on **Go to credentials**.

Overview



[Disable API](#)

Drive API

This API is enabled, but you can't use it in your project until you create credentials. Click "Go to Credentials" to do this now (strongly recommended).

[Go to Credentials](#)

For some reason Google warns us again that we need to create credentials. We will use OAuth 2.0.

Credentials

[Credentials](#) [OAuth consent screen](#) [Domain verification](#)

APIs
Credentials

You need credentials to access APIs. [Enable the APIs you plan to use](#) and then create the credentials they require. Depending on the API, you need an API key, a service account, or an OAuth 2.0 client ID. [Refer to the API documentation](#) for details.

Add credentials ▾

API key
Identifies your project using a simple API key to check quota and access.
For APIs like Google Translate.

OAuth 2.0 client ID
Requests user consent so your app can access the user's data.
For APIs like Google Calendar.

Service account
Enables server-to-server, app-level authentication using robot accounts.
For use with Google Cloud APIs.

Now we have to create a consent screen. This is the information in the screen Google shows you when you connect your new Google app to ownCloud the first time. Click **Configure consent screen**. Then fill in the required form fields. Your logo must be hosted, as you cannot upload it, so enter its URL. When you're finished click **Save**.

Credentials

Credentials OAuth consent screen Domain verification

Email address ?

dev@gmail.com

Product name shown to users

MyGoogleDriveApp

Homepage URL (Optional)

<https://example.com>

Product logo URL (Optional) ?

<https://owncloud.org/imggs>



This is how your logo will look to end users

Max size: 120x120 px

Privacy policy URL (Optional)

<https://example.com/privacy>

Terms of service URL (Optional)

<https://example.com/tos>

Save

Cancel

The next screen that opens is **Create Client ID**. Check **Web Application**, then enter your app name. **Authorized JavaScript Origins** is your root domain, for example <https://example.com>, without a trailing slash. You need two **Authorized Redirect URIs**, and they must be in this form:

<https://example.com/owncloud/index.php/settings/personal>
<https://example.com/owncloud/index.php/settings/admin>

Replace <https://example.com/owncloud/> with your own ownCloud server URL, then click **Create**.

Credentials



Create client ID

Application type

- Web application
- Android [Learn more](#)
- Chrome App [Learn more](#)
- iOS [Learn more](#)
- PlayStation 4
- Other

Name

Authorized JavaScript origins

Enter JavaScript origins here or redirect URIs below (or both) [?](#)

Cannot contain a wildcard (`http://*.example.com`) or a path (`http://example.com/subdir`).

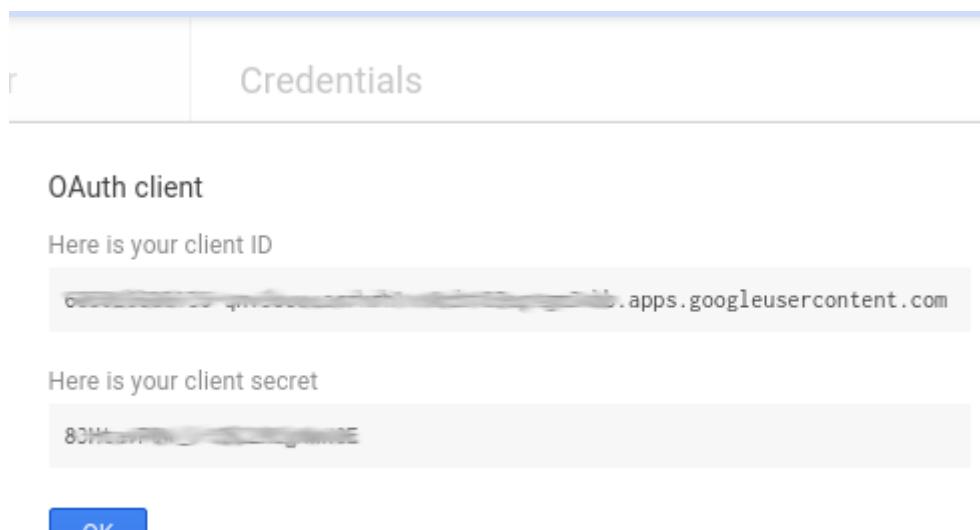
 x

Authorized redirect URIs

Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

 x x

Now Google reveals to you your **Client ID** and **Client Secret**. Click **OK**.



You can see these anytime in your Google console; just click on your app name to see complete information.

The screenshot shows the 'Credentials' section of the Google Developers Console. The left sidebar has 'API Manager' selected under 'API'. The main area shows a table of OAuth 2.0 client IDs. One entry is visible:

Name	Creation date	Type	Client ID
MyGoogleDriveApp	Dec 1, 2015	Web application	600200000120-qw9000caclh5v0ch00-pj...2bb.apps.googleusercontent.com

Now you have everything you need to mount your Google Drive in ownCloud.

Go to the External Storage section of your Admin page, create your new folder name, enter the Client ID and Client Secret. If you wish limit access to a single folder, simply enter the path to the desired folder, separated by '/'. Finally, click **Grant Access**. Your consent page appears when ownCloud makes a successful connection. Click **Allow**.



▼ MyGoogleDriveApp would like to:

 View and manage the files in your Google Drive i

By clicking Allow, you allow this app and Google to use your information in accordance with their respective [terms of service](#) and [privacy policies](#). You can change this and other [Account Permissions](#) at any time.

Deny

Allow

When you see the green light confirming a successful connection you're finished.



See Configuring External Storage (GUI) for additional mount options and information.

See External Storage Authentication mechanisms for more information on authentication schemes.
603026686136-qnv90ooocacrkh1vs0cht83eprgm2sbb.apps.googleusercontent.com

Local

Local storage provides the ability to mount any directory on your ownCloud server that is:

- Outside of your ownCloud data/ directory
- Both readable and writable by your HTTP server user

Since this is a significant security risk, Local storage is only configurable via the ownCloud admin settings. Non-admin users cannot create Local storage mounts.

Note

See Set Strong Directory Permissions for information on correct file permissions, and find your HTTP user PHP Version and Information.

To manage Local storage, navigate to `admin`, and then to `Storage`. You can see an example in the screenshot below.

External Storage

Folder name	External storage	Configuration	Available for
Local	Local	/shared/projects	All Users 

In the **Folder name** field enter the folder name that you want to appear on your ownCloud Files page. In the **Configuration** field enter the full file path of the directory you want to mount. In the **Available for** field enter the users or groups who have permission to access the mount; by default all users have access.

In addition to these steps, you have to ensure that Local storage is enabled in your ownCloud installation's config/config.php file. It should have the following configuration:

```
'files_external_allow_create_new_local' => 'true',
```

Note

See Configuring External Storage (GUI) for additional mount options and information, and External Storage Authentication mechanisms for more information on authentication schemes.

OpenStack Object Storage

OpenStack Object Storage is used to connect to an OpenStack Swift server, or to Rackspace. Two authentication mechanisms are available: one is the generic OpenStack mechanism, and the other is used exclusively for Rackspace, a provider of object storage that uses the OpenStack Swift protocol.

The OpenStack authentication mechanism uses the OpenStack Keystone v2 protocol. Your ownCloud configuration needs:

- **Bucket.** This is user-defined; think of it as a subdirectory of your total storage. The bucket will be created if it does not exist.
- **Username** of your account.
- **Password** of your account.
- **Tenant name** of your account. (A tenant is similar to a user group.)
- **Identity Endpoint URL**, the URL to log in to your OpenStack account.

Configuration

Folder name	External storage	Authentication	Configuration	A
OpenStackObjectSt	OpenStack Object Storage	OpenStack ▾	Service name Region myfiles Request timeout (se molly foobar http://devstack:5001	

The Rackspace authentication mechanism requires:

- **Bucket**
- **Username**
- **API key.**

You must also enter the term **cloudFiles** in the **Service name** field.

Folder name	External storage	Authentication	Configuration	A
OpenStackObjectSt	OpenStack Object Storage	Rackspace ▾	cloudFiles Region myfiles Request timeout (se molly	

It may be necessary to specify a **Region**. Your region should be named in your account information, and you can read about Rackspace regions at [About Regions](#).

The timeout of HTTP requests is set in the **Request timeout** field, in seconds.

See [Configuring External Storage \(GUI\)](#) for additional mount options and information.

See [External Storage Authentication mechanisms](#) for more information on authentication schemes.

ownCloud

An ownCloud storage is a specialized WebDAV storage, with optimizations for ownCloud-ownCloud communication. See the WebDAV documentation to learn how to configure an ownCloud external storage.

Configuration

When filling in the **URL** field, use the path to the root of the ownCloud installation, rather than the path to the WebDAV endpoint. So, for a server at `https://example.com/owncloud`, use `https://example.com/owncloud` and not `https://example.com/owncloud/remote.php/dav`.

See Configuring External Storage (GUI) for additional mount options and information.

See External Storage Authentication mechanisms for more information on authentication schemes.

SFTP

ownCloud's SFTP (FTP over an SSH tunnel) backend supports both password and public key authentication.

The **Host** field is required; a port can be specified as part of the **Host** field in the following format: `hostname.domain:port`. The default port is 22 (SSH).

For public key authentication, you can generate a public/private key pair from your **SFTP with secret key login** configuration.

External Storage

Folder name	External storage	Authentication	Configuration
SFTP	SFTP	<input type="button" value="Username and password"/> <input type="button" value="Username and password"/> <input type="button" value="Log-in credentials, save in session"/> <input type="button" value="RSA public key"/>	<input type="text" value="Host"/> <input type="text" value="Root"/> <input type="text" value="Username"/> <input type="text" value="Password"/>

After generating your keys, you need to copy your new public key to the destination server to `.ssh/authorized_keys`. ownCloud will then use its private key to authenticate to the SFTP server.

The default **Remote Subfolder** is the root directory (`/`) of the remote SFTP server, and you may enter any directory you wish.

See Configuring External Storage (GUI) for additional mount options and information.

See External Storage Authentication mechanisms for more information on authentication schemes.

SMB/CIFS

ownCloud can connect to Windows file servers or other SMB-compatible servers with the SMB/CIFS backend.

Note

ownCloud's SMB/CIFS backend requires either [the libsmbclient-php module](#) (version 0.8.0+) or [the smbclient command](#) (and its dependencies) to be installed on the ownCloud server. We highly recommend `libsmbclient-php`, but it isn't required. If installed, however, `smbclient` won't be needed. Most Linux distributions provide `libsmbclient-php` and, typically, name it `php-smbclient`.

You also need the Samba client installed on your Linux system. This is included in all Linux distributions; on Debian, Ubuntu, and other Debian derivatives this is `smbclient`. On SUSE, Red Hat, CentOS, and other Red Hat derivatives it is `samba-client`. You also need `which` and `stdbuf`, which should be included in most Linux distributions.

You need the following information:

- Folder name for your local mountpoint.
- Host: The URL of the Samba server.
- Username: The username or domain/username used to login to the Samba server.

Configuration

- Password: the password to login to the Samba server.
- Share: The share on the Samba server to mount.
- Remote Subfolder: The remote subfolder inside the Samba share to mount (optional, defaults to /). To assign the ownCloud logon username automatically to the subfolder, use \$user instead of a particular subfolder name.
- And finally, the ownCloud users and groups who get access to the share.

Optionally, you can specify a Domain. This is useful in cases where the SMB server requires a domain and a username, and an advanced authentication mechanism like session credentials is used so that the username cannot be modified. This is concatenated with the username, so the backend gets domain\username

The screenshot shows the 'SMB / CIFS' configuration screen. On the left, there's a radio button for 'smbcifs' which is selected. Next to it is a dropdown menu labeled 'Session credentials' with two options: 'Username and password' and 'Session credentials', where 'Session credentials' is currently selected. To the right of the dropdown are four input fields: 'smbserver' (containing 'users'), 'users' (containing '/shared'), 'All users. Type to select' (empty), and 'Domain' (empty).

See Configuring External Storage (GUI) for additional mount options and information.

See External Storage Authentication mechanisms for more information on authentication schemes.

WebDAV

Use this backend to mount a directory from any WebDAV server, or another ownCloud server.

You need the following information:

- Folder name: The name of your local mountpoint.
- The URL of the WebDAV or ownCloud server.
- Username and password for the remote server
- Secure <https://>: We always recommend <https://> for security, though you can leave this unchecked for <http://>.

Optionally, a Remote Subfolder can be specified to change the destination directory. The default is to use the whole root.

The screenshot shows the 'ownCloud' configuration screen. On the left, there's a radio button for 'oc-remote' which is selected. Next to it is a dropdown menu labeled 'ownCloud'. To the right are five input fields: 'https://remoteserver' (containing 'admin'), 'admin' (containing '.....'), 'All Users' (with a delete icon), an empty field for 'Remote Subfolder', and a checkbox for 'Secure https://' which is checked.

Note

CPanel users should install [Web Disk](#) to enable WebDAV functionality.

See Configuring External Storage (GUI) for additional mount options and information.

See External Storage Authentication mechanisms for more information on authentication schemes.

Note

A non-blocking or correctly configured SELinux setup is needed for these backends to work. Please refer to the SELinux Configuration.

Allow Users to Mount External Storage

Check “Allow users to mount external storage” to allow your users to mount storages on external services. Then enable the backends you want to allow.



Warning

Be careful with the choices that you enable, as it allows a user to make potentially arbitrary connections to other services on your network!

Setting Up Google Drive and Dropbox Connections

When an external storage is created which uses either Google Drive or Dropbox, a link to the respective configuration page is available, next to the service name.

External Storage	Folder name	External storage	Authentication	Configuration	Available for
	GoogleDrive	Google Drive	OAuth2 ▾	Client ID	Client secret Grant access All users. Type to select user or group.
	Dropbox	Dropbox	OAuth1 ▾	App key	App secret Grant access All users. Type to select user or group.
Folder name					Add storage ▾

In the screenshot above, you can see that two external storage connections have been created, but not configured. One goes to Google Drive, the other to Dropbox. If you click the cog icon next to the name of either, the respective app configuration page will open in a new tab, or a new window. From there, you can manage the configuration and obtain the respective credentials needed for configuring the connection.

Detecting Files Added to External Storages

We recommend configuring the background job **Webcron** or **Cron** (see Background Jobs) to enable ownCloud to automatically detect files added to your external storages.

Note

You cannot scan/detect changed files on external storage mounts when you select the **Log-in credentials, save in session** authentication mechanism. However, there is a workaround, and that is to use Ajax cron mode. See Password-based Mechanisms for more information.

ownCloud may not always be able to find out what has been changed remotely (files changed without going through ownCloud), especially when it's very deep in the folder hierarchy of the external storage.

You might need to setup a cron job that runs `sudo -u www-data php occ files:scan --all` (or replace “--all” with the user name, see also Using the occ Command) to trigger a rescan of the user's files periodically (for example every 15 minutes), which includes the mounted external storage.

Configuring External Storage (Configuration File)

Starting with ownCloud 9.0, the data/mount.json file for configuring external storages has been removed, and replaced with a set of occ commands.

External Storage Authentication mechanisms

ownCloud storage backends accept one or more authentication schemes such as passwords, OAuth, or token-based, to name a few examples. Each authentication scheme may be implemented by multiple authentication mechanisms. Different mechanisms require different configuration parameters, depending on their behaviour.

Special Mechanisms

The **None** authentication mechanism requires no configuration parameters, and is used when a backend requires no authentication.

The **Built-in** authentication mechanism itself requires no configuration parameters, but is used as a placeholder for legacy storages that have not been migrated to the new system and do not take advantage of generic authentication mechanisms. The authentication parameters are provided directly by the backend.

Password-based Mechanisms

The **Username and password** mechanism requires a manually-defined username and password. These get passed directly to the backend.

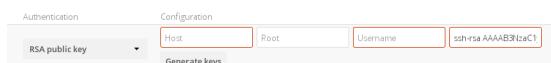
The **Log-in credentials, save in session** mechanism uses the ownCloud login credentials of the user to connect to the storage. These are not stored anywhere on the server, but rather in the user session, giving increased security. The drawbacks are that sharing is disabled when this mechanism is in use, as ownCloud has no access to the storage credentials, and background file scanning does not work.

Note

There is a workaround that allows background file scanning when using **Log-in credentials, save in session**, and that is using Ajax cron mode. (See Background Jobs.) Be aware that the Ajax cron mode is triggered by browsing the ownCloud Web GUI.

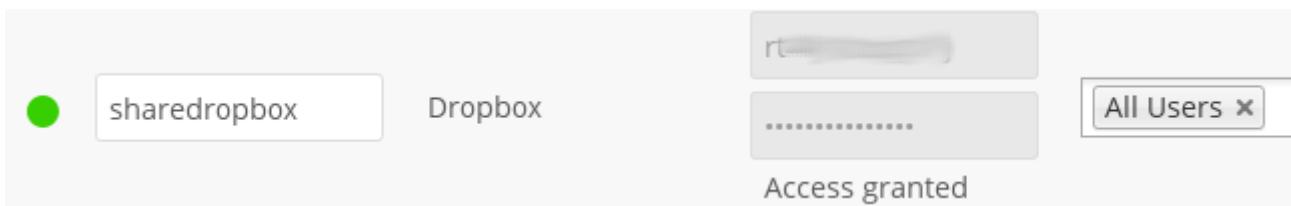
Public-key Mechanisms

Currently only the RSA mechanism is implemented, where a public/private keypair is generated by ownCloud and the public half shown in the GUI. The keys are generated in the SSH format, and are currently 1024 bits in length. Keys can be regenerated with a button in the GUI.



OAuth

OAuth 1.0 and OAuth 2.0 are both implemented, but currently limited to the Dropbox and Google Drive backends respectively. These mechanisms require additional configuration at the service provider, where an app ID and app secret are provided and then entered into ownCloud. Then ownCloud can perform an authentication request, establishing the storage connection.



If ownCloud client's are unable to connect to your ownCloud server, check that the bearer authorization header is not being stripped out.

Encryption Configuration

The primary purpose of the ownCloud server-side encryption is to protect users' files when they're located on remote storages, such as Dropbox and Google Drive, and to do it smoothly and seamlessly from within ownCloud.

From ownCloud 9.0, server-side encryption for local and remote storages can operate independently of each other. By doing so, you can encrypt a remote storage *without* also having to encrypt your home storage on your ownCloud server.

Note

Starting with ownCloud 9.0 we support Authenticated Encryption for all newly encrypted files. See <https://hackerone.com/reports/108082> for more technical information about the impact.

For maximum security make sure to configure external storage with “*Check for changes: Never.*” This will let ownCloud ignore new files not added via ownCloud. By doing so, a malicious external storage administrator cannot add new files to the storage without your knowledge. However, this is not wise *if* your external storage is subject to legitimate external changes.

ownCloud's server-side encryption encrypts files stored on the ownCloud server and files on remote storages that are connected to your ownCloud server. Encryption and decryption are performed on the ownCloud server. All files sent to remote storage will be encrypted by the ownCloud server and decrypted before serving them to you or anyone whom you have shared them with.

Note

Encrypting files increases their size by roughly 35%. Remember to take this into account when you are both provisioning storage and setting storage quotas. Secondly, user quotas are based on the *unencrypted* file size — **not** the encrypted size.

When files on an external storage are encrypted in ownCloud, you cannot share them directly from the external storage services, only through ownCloud sharing. This is because the key to decrypt the data **never** leaves the ownCloud server.

ownCloud's server-side encryption generates a strong encryption key, which is unlocked by users' passwords. As a result, your users don't need to track an extra password. All they need to do is log in as they normally would. ownCloud, transparently, encrypts only the contents of files, and not filenames and directory structures.

Important

You should regularly backup all encryption keys to prevent permanent data loss.

The encryption keys are stored in the following directories:

Directory	Description
data/<user>/files_encryption	Users' private keys and all other keys necessary to decrypt the users' files.
data/files_encryption	Private keys and all other keys necessary to decrypt the files stored on a system wide external storage.

Note

You can move the keys to a different location. To do so, refer to the [Move Key Location](#) section of the documentation.

When encryption is enabled, all files are encrypted and decrypted by the ownCloud application, and stored encrypted on your remote storage. This protects your data on externally hosted storage. The ownCloud admin and the storage admin will see only encrypted files when browsing backend storage.

Warning

Encryption keys are stored only on the ownCloud server, eliminating exposure of your data to third-party storage providers. The encryption application does **not** protect your data if your ownCloud server is compromised, and it does not prevent ownCloud administrators from reading users' files. This would require client-side encryption, which this application does not provide. If your ownCloud server is not connected to any external storage services, it is better to use other encryption tools, such as file-level or whole-disk encryption.

Important

SSL terminates at or before Apache on the ownCloud server. Consequently, all files are in an unencrypted state between the SSL connection termination and the ownCloud code that encrypts and decrypts them. This is, potentially, exploitable by anyone with administrator access to your server. For more information, read: [How ownCloud uses encryption to protect your data](#).

Encryption Types

ownCloud provides two encryption types:

- **Basic encryption:** every user has their own private/public key pairs, and the private key is protected by the user's password.
- **Master Key:** there is only one key (or key pair) and all files are encrypted using that key pair.

Warning

These encryption types are **not compatible**.

Before Enabling Encryption

Plan very carefully before enabling encryption, because it is not reversible via the ownCloud Web interface. If you lose your encryption keys, your files are **not** recoverable. Always have backups of your encryption keys stored in a safe location, and consider enabling all recovery options.

You have more options via the `occ` command (see [Enabling Master Key Based Encryption](#))

Warning

You can't manage encryption without access to the command line. If your ownCloud installation is on a hosted environment and you don't have access to the command line, you won't be able to run `occ` commands. In this case, **don't enable encryption!**

How To Enable Encryption

The base encryption system is enabled and disabled on your Admin page. First, you must enable this, and then select an encryption module to load. Go to the **Server-side encryption** section of your Admin page and check **Enable encryption**.

Server-side encryption *i*

Enable server-side encryption

Please read carefully before activating server-side encryption:

- Once encryption is enabled, all files uploaded to the server from that point forward will be encrypted at rest on the server. It will only be possible to disable encryption at a later date if the active encryption module supports that function, and all pre-conditions (e.g. setting a recover key) are met.
- Encryption alone does not guarantee security of the system. Please see ownCloud documentation for more information about how the encryption app works, and the supported use cases.
- Be aware that encryption always increases the file size.
- It is always good to create regular backups of your data, in case of encryption make sure to backup the encryption keys along with your data.

This is the final warning: Do you really want to enable encryption? [Enable encryption](#)

After clicking **Enable encryption**, you will see the message “*No encryption module loaded, please load an encryption module in the app menu*”. Currently, the only available encryption module is the ownCloud Default Encryption module. So, go to your Apps page to enable the ownCloud Default Encryption module.



Default encryption module 1.1.0

by Bjoern Schiessle, Clark Tomlinson (AGPL-licensed)

Official

Show description ...

[Disable](#)

Then, return to your Admin page to see that the ownCloud Default Encryption module has been added to the module selector and automatically selected. Now you **must** log out and then log back in to initialize your encryption keys.

Server-side encryption *i*

Enable server-side encryption

Select default encryption module:

Default encryption module

Encryption App is enabled but your keys are not initialized, please log-out and log-in again

When you log back in, a checkbox for enabling encryption on your home storage, will now be available — checked by default. Uncheck it to avoid encrypting your home storage.

Server-side encryption *i*

Enable server-side encryption

Select default encryption module:

Default encryption module

Encrypt the home storage

Enabling this option encrypts all files stored on the main storage otherwise only files on external storage will be encrypted

Enabling Encryption From the Command-line

To enable encryption via the command-line, involves two commands. These are:

```
# Enables the default encryption module app  
php occ app:enable encryption  
  
# Enables encryption  
php occ encryption:enable
```

Note

Please note, the commands have to be run in this order.

Warning

If you already have Basic encryption enabled - you **must not** enable Master Key Based encryption.

Enabling Master Key Based Encryption

Configuration

To enable master key based encryption:

1. Enable the default encryption module app using the following command

```
php occ app:enable encryption
```

2. Then enable encryption using the following command

```
php occ encryption:enable
```

3. Then enable the master key using the following command

```
php occ encryption:select-encryption-type masterkey
```

Note

The master key mode has to be set up in a newly created instance.

4. Encrypt all data

```
php occ encryption:encrypt-all
```

Note

This is not typically required, as the master key is often enabled at install time. As a result, when enabling it, there should be no data to encrypt. But, in case it's being enabled after install, and the installation does have files which are unencrypted, encrypt-all can be used to encrypt them.

Sharing Encrypted Files

After encryption is enabled, your users must also log out and log back in to generate their personal encryption keys. They will see a yellow warning banner that says “*Encryption App is enabled, but your keys are not initialized. Please log-out and log-in again.*”

Also, share owners may need to re-share files after encryption is enabled. Users who are trying to access the share will see a message advising them to ask the share owner to re-share the file with them.

For individual shares, un-share and re-share the file. For group shares, share with any individuals who can't access the share. This updates the encryption, and then the share owner can remove the individual shares.

Can not decrypt this file, probably
this is a shared file. Please ask the file
owner to reshare the file with you.

Encrypting External Mountpoints

You and your users can encrypt individual external mount points. You must have external storage enabled on your Admin page, and enabled for your users. Encryption settings can be configured in the mount options for an external storage mount; see Mount Options (Configuring External Storage (GUI))

How To Enable Users File Recovery Keys

Configuration

Once a user has encrypted their files, if they lose their ownCloud password, then they lose access to their encrypted files, as their files will be unrecoverable. It is not possible, when user files are encrypted, to reset a user's password using the standard reset process.

If so, you'll see a yellow banner warning:

Please provide an admin recovery password; otherwise, all user data will be lost.

To avoid all this, create a Recovery Key. To do so, go to the Encryption section of your Admin page and set a recovery key password.

Server-side encryption *i*

Enable server-side encryption

Select default encryption module:

Default encryption module

Enable recovery key

The recovery key is an extra encryption key that is used to encrypt files. It allows recovery of a user's files if the user forgets his or her password.

●●●●●●●●

●●●●●●●●

Disable recovery key

You then need to ask your users to opt-in to the Recovery Key. For the users to do this, they need to go to the “Personal” page and enable the recovery key. This signals that they are OK that the admin might have a way to decrypt their data for recovery reasons. If they do *not* do this, then the Recovery Key won't work for them.

Encryption

Enable password recovery:

Enabling this option will allow you to reobtain access to your encrypted files in case of password loss

Enabled

Disabled

File recovery settings updated

For users who have enabled password recovery, give them a new password and recover access to their encrypted files, by supplying the Recovery Key on the Users page.

Admin Recovery Password		Enter the recovery password in order to recover the users files during password change	
Group Admin	Quota	Storage Location	
Group Admin ▾	Default ▾	/var/www/owncloud.	

You may change your recovery key password.

Change recovery key password:

Old Recovery key password
New Recovery key password
Repeat New Recovery key password

Note

Sharing a recovery key with a user group is **not** supported. This is only supported with the master key.

Changing The Recovery Key Password

If you have misplaced your recovery key password and need to replace it, here's what you need to do:

1. Delete the recovery key from both `data/owncloud_private_keys` and `data/public-keys`
2. Edit your database table `oc_appconfig` and remove the rows with the config keys `recoveryKeyId` and `recoveryAdminEnabled` for the appid `files_encryption`
3. Login as admin and activate the recovery key again with a new password. This will generate a new key pair
4. All users who used the original recovery key will need to disable it and enable it again. This deletes the old recovery share keys from their files and encrypts their files with the new recovery key

Note

You can only change the recovery key password if you know the original. This is by design, as only admins who know the recovery key password should be able to change it. If not, admins could hijack the recovery key from each other

Warning

Replacing the recovery key will mean that all users will lose the possibility to recover their files until they have applied the new recovery key

Disabling Encryption

To disable encryption, put your ownCloud server into single-user mode, and then disable your encryption module with these commands:

```
occ maintenance:singleuser --on  
occ encryption:disable
```

Take it out of single-user mode when you are finished, by using the following command:

```
occ maintenance:singleuser --off
```

Important

You may only disable encryption with by using the [occ Encryption Commands](#). Make sure you have backups of all encryption keys, including those for all your users.

Not All Files Are Encrypted

Configuration

Only the data in the files in `data/user/files` are encrypted, not the filenames or folder structures.

In addition, these files are never encrypted:

- Existing files in the trash bin & Versions. Only new and changed files after encryption is enabled are encrypted.
- Image thumbnails from the Gallery app
- Previews from the Files app
- The search index from the full-text search app
- Third-party app data

There may be other files that are not encrypted. Only files that are exposed to third-party storage providers are guaranteed to be encrypted.

LDAP and Other External User Back-ends

If you use an external user back-end, such as an LDAP or Samba server, and you change a user's password on that back-end, the user will be prompted to change their ownCloud login to match on their next ownCloud login. The user will need both their old and new passwords to do this. If you have enabled the recovery key then you can change a user's password in the ownCloud Users panel to match their back-end password and then — of course — notify the user and give them their new password.

occ Encryption Commands

If you have shell access, you may use the `occ` command to perform encryption operations. You also have additional options such as decryption and creating a single master encryption key. See [Encryption](#) for detailed instructions on using `occ`.

View Current Encryption Status

Get the current encryption status and the loaded encryption module:

```
occ encryption:status
- enabled: false
- defaultModule: OC_DEFAULT_MODULE
```

This is equivalent to checking [Enable server-side encryption](#) on your Admin page:

```
occ encryption:enable
Encryption enabled

Default module: OC_DEFAULT_MODULE
```

List Available Encryption Modules

To list the available encryption modules:

```
occ encryption:list-modules
- OC_DEFAULT_MODULE: Default encryption module [default*]
```

Select a different default Encryption module (currently the only available module is `OC_DEFAULT_MODULE`):

```
occ encryption:set-default-module [Module ID].
```

The [module ID] is taken from the `encryption:list-modules` command.

Encrypt and Decrypt Data Files For All Users

For performance reasons, when you enable encryption on an ownCloud server only new and changed files are encrypted. This command gives you the option to encrypt all files. You must first put your ownCloud server into single-user mode to prevent any user activity until encryption is completed:

```
occ maintenance:singleuser --on
Single user mode is currently enabled
```

Configuration

Then run occ:

```
occ encryption:encrypt-all
```

You are about to start encrypting all files stored in your ownCloud.
It will depend on the encryption module you use which files get encrypted.
Depending on the number and size of your files this can take some time.
Please make sure that no users access their files during this process!

```
Do you really want to continue? (y/n)
```

When you type `y` it creates a key pair for each of your users, and then encrypts their files, displaying progress until all user files are encrypted.

Decrypt all user data files, or optionally a single user:

```
occ encryption:decrypt-all [username]
```

View current location of keys:

```
occ encryption:show-key-storage-root  
Current key storage root: default storage location (data/)
```

Move Key Location

Move keys to a different root folder, either locally or on a different server. The folder must already exist, be owned by root and your HTTP group, and be restricted to root and your HTTP group. This example is for Ubuntu Linux. Note that the new folder is relative to your `occ` directory:

```
mkdir /etc/keys  
chown -R root:www-data /etc/keys  
chmod -R 0770 /etc/keys  
occ encryption:change-key-storage-root ../../etc/keys  
Start to move keys:  
 4 [=====]  
Key storage root successfully changed to ../../etc/keys
```

Create a New Master Key

Use this when you have:

- A single-sign-on infrastructure
- A fresh installation with no existing data
- Systems where encryption has not already been enabled

```
occ encryption:enable-master-key
```

Important

It is not possible to disable it.

Recreating an Existing Master Key

If the master key needs replacing, for example, because it has been compromised, an `occ` command is available. The command is `encryption:recreate-master-key`. It replaces existing master key with new one and encrypts the files with the new key.

Disabling Encryption

You may disable encryption only with `occ`. Make sure you have backups of all the encryption keys, including those for all users. When you do, put your ownCloud server into single-user mode, and then disable your encryption module with this command:

```
occ maintenance:singleuser --on  
occ encryption:disable
```

Warning

Encryption cannot be disabled without the user's password or file recovery key. If you don't have access to at least one of these then there is no way to decrypt all files.

Then, take it out of single-user mode when you are finished with this command:

```
occ maintenance:singleuser --off
```

It is possible to disable encryption with the file recovery key, *if* every user uses them. If so, "decrypt all" will use it to decrypt all files.

Note

It is **not** planned to move this to the next user login or a background job. If that was done, then login passwords would need to be stored in the database, which could be a security issue.

Files Not Encrypted

Only the data in the files in `data/user/files` are encrypted, and not the filenames or folder structures. These files are never encrypted:

- Existing files in the trash bin & Versions. Only new and changed files after encryption is enabled are encrypted.
- Existing files in Versions
- Image thumbnails from the Gallery app
- Previews from the Files app
- The search index from the full-text search app
- Third-party app data

There may be other files that are not encrypted; only files that are exposed to third-party storage providers are guaranteed to be encrypted.

LDAP and Other External User Back-ends

If you use an external user back-end, such as an LDAP or Samba server, and you change a user's password on the back-end, the user will be prompted to change their ownCloud login to match on their next ownCloud login. The user will need both their old and new passwords to do this. If you have enabled the Recovery Key, then you can change a user's password in the ownCloud Users panel to match their back-end password, and then, of course, notify the user and give them their new password.

Encryption migration to ownCloud 8.0

When you upgrade from older versions of ownCloud to ownCloud 8.0, you must manually migrate your encryption keys with the `occ` command after the upgrade is complete, like this example for CentOS:
`sudo -u apache php occ encryption:migrate-keys` You must run `occ` as your HTTP user. See Using the `occ` Command to learn more about `occ`.

Encryption migration to ownCloud 8.1

Configuration

The encryption backend has changed again in ownCloud 8.1, so you must take some additional steps to migrate encryption correctly. If you do not follow these steps you may not be able to access your files.

Before you start your upgrade, put your ownCloud server into `maintenance:singleuser` mode (See Maintenance Mode Configuration.) You must do this to prevent users and sync clients from accessing files before you have completed your encryption migration.

After your upgrade is complete, follow the steps in How To Enable Encryption to enable the new encryption system. Then click the **Start Migration** button on your Admin page to migrate your encryption keys, or use the `occ` command. We strongly recommend using the `occ` command; the **Start Migration** button is for admins who do not have access to the console, for example, installations on shared hosting. This example is for Debian/Ubuntu Linux:

```
$ sudo -u www-data php occ encryption:migrate
```

This example is for Red Hat/CentOS/Fedora Linux:

```
$ sudo -u apache php occ encryption:migrate
```

You must run `occ` as your HTTP user; see Using the `occ` Command.

When you are finished, take your ownCloud server out of `maintenance:singleuser` mode.

Transactional File Locking

ownCloud's Transactional File Locking mechanism locks files to avoid file corruption during normal operation. It performs these functions:

- Operates at a higher level than the filesystem, so you don't need to use a filesystem that supports locking
- Locks parent directories so they cannot be renamed during any activity on files inside the directories
- Releases locks after file transactions are interrupted, for example when a sync client loses the connection during an upload
- Manages locking and releasing locks correctly on shared files during changes from multiple users
- Manages locks correctly on external storage mounts
- Manages encrypted files correctly

Transactional File locking will not prevent multiple users from editing the same document, nor give notice that other users are working on the same document. Multiple users can open and edit a file at the same time and Transactional File locking does not prevent this. Rather, it prevents simultaneous file saving.

Note

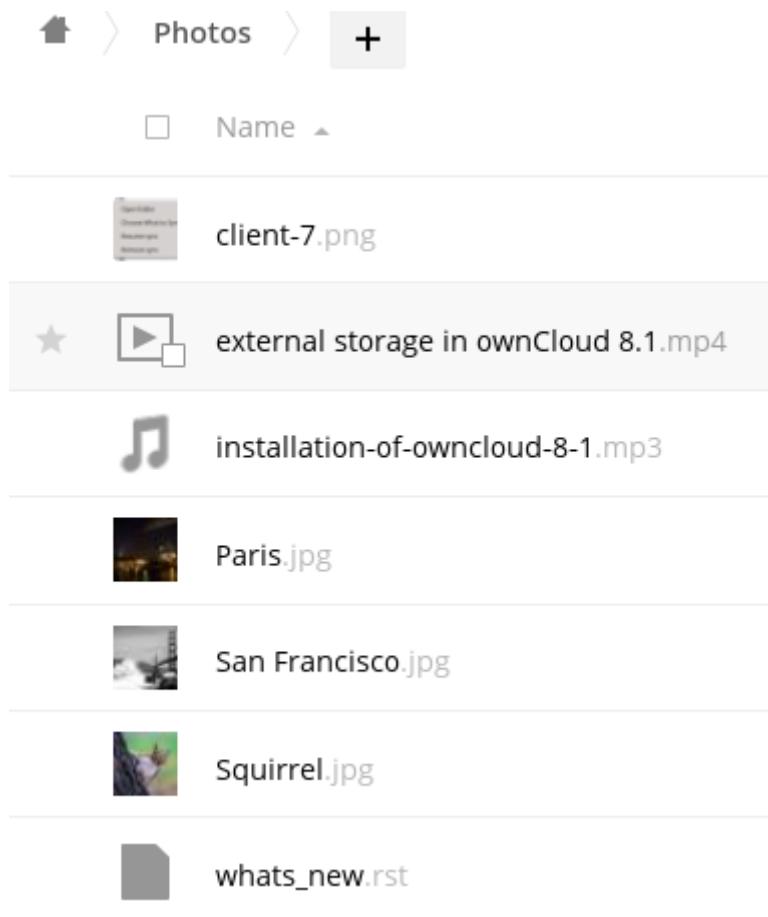
Transactional file locking is in ownCloud core, and replaces the old File Locking app. The File Locking app was removed from ownCloud in version 8.2.1. If your ownCloud server still has the File Locking app, you **must** visit your Apps page to verify that it is disabled; the File Locking app and Transactional File Locking cannot both operate at the same time.

File locking is enabled by default, using the database locking backend. This places a significant load on your database. Using `memcache.locking` relieves the database load and improves performance. Admins of ownCloud servers with heavy workloads should install a memory cache.

Previews Configuration

The ownCloud thumbnail system generates previews of files for all ownCloud apps that display files, such as Files and Gallery.

The following image shows some examples of previews of various file types.



By default, ownCloud can generate previews for the following filetypes:

- Images files
- Cover of MP3 files
- Text documents

Note

Older versions of ownCloud also supported the preview generation of other file types such as PDF, SVG or various office documents. Due to security concerns those providers have been disabled by default and are considered unsupported. While those providers are still available, we discourage enabling them, and they are not documented.

Parameters

Please notice that the ownCloud preview system comes already with sensible defaults, and therefore it is usually unnecessary to adjust those configuration values.

Disabling previews

Under certain circumstances, for example if the server has limited resources, you might want to consider disabling the generation of previews. Note that if you do this all previews in all apps are disabled, including the Gallery app, and will display generic icons instead of thumbnails.

Set the configuration option `enable_previews` in `config.php` to `false`:

```
<?php
'enable_previews' => false,
```

Maximum preview size

There are two configuration options for setting the maximum size (in pixels) of a preview. These are `preview_max_x` which represents the x-axis and `preview_max_y` which represents the y-axis. In `config/config.sample.php`, which you can see below, both options are set to a default of 2048.

```
<?php  
    'preview_max_x' => 2048,  
    'preview_max_y' => 2048,
```

The following example would limit previews to a maximum size of 100 px × 100 px:

```
<?php  
    'preview_max_x' => 100,  
    'preview_max_y' => 100,
```

Note

If you want no limit applied for one or both of these values then set them to null.

Maximum scale factor

If a lot of small pictures are stored on the ownCloud instance and the preview system generates blurry previews, you might want to consider setting a maximum scale factor. By default, pictures are upscaled to 10 times the original size:

```
<?php  
    'preview_max_scale_factor' => 10,
```

If you want to disable scaling at all, you can set the config value to '1':

```
<?php  
    'preview_max_scale_factor' => 1,
```

If you want to disable the maximum scaling factor, you can set the config value to 'null':

```
<?php  
    'preview_max_scale_factor' => null,
```

Controlling File Versions and Aging

The Versions app (`files_versions`) expires old file versions automatically to ensure that users don't exceed their storage quotas. This is the default pattern used to delete old versions:

- For the first second we keep one version
- For the first 10 seconds ownCloud keeps one version every 2 seconds
- For the first minute ownCloud keeps one version every 10 seconds
- For the first hour ownCloud keeps one version every minute
- For the first 24 hours ownCloud keeps one version every hour
- For the first 30 days ownCloud keeps one version every day
- After the first 30 days ownCloud keeps one version every week

The versions are adjusted along this pattern every time a new version is created.

The Versions app never uses more than 50% of the user's storage quota. If the stored versions exceed this limit, ownCloud deletes the oldest file versions until it meets the disk space limit again.

You may alter the default pattern in `config.php`. The default setting is `auto`, which sets the default pattern:

```
'versions_retention_obligation' => 'auto',
```

Configuration

Additional options are:

- D, auto
Keep versions at least for D days, apply expiration rules to all versions that are older than D days
- auto, D
Delete all versions that are older than D days automatically, delete other versions according to expiration rules
- D1, D2
Keep versions for at least D1 days and delete when they exceed D2 days.
- disabled
Disable Versions; no files will be deleted.

Enterprise File Retention

Enterprise customers have additional tools for managing file retention policies; see Advanced File Tagging With the Workflow App.

Managing the Trashbin

The ownCloud Trashbin (`files_trashbin`) permanently deletes files according to users' storage quotas and file ages. When a user deletes a file it is not immediately removed from your ownCloud server, but goes into the Trashbin. Then the user has the options to un-delete the file, or to delete it permanently.

Name	Restore	Deleted
eyeballs.jpg	Restore	6 minutes ago
WaterTender009.jpg	Restore	6 minutes ago
mallet-chisel-480.jpg	Delete	6 minutes ago

As the ownCloud server administrator, you have two `occ` commands for permanently deleting files from the Trashbin manually, without waiting for the normal aging-out process:

```
trashbin
trashbin:cleanup    Remove deleted files
trashbin:expire     Expires the users trashbin
```

The `trashbin:cleanup` command removes the deleted files of all users, or you may specify certain users in a space-delimited list. This example removes all the deleted files of all users:

```
sudo -u www-data php occ trashbin:cleanup
Remove all deleted files
Remove deleted files for users on backend Database
user1
user2
user3
user4
```

This example removes the deleted files of user2 and user4:

```
sudo -u www-data php occ trashbin:cleanup user2 user4
Remove deleted files of user2
Remove deleted files of user4
```

trashbin:expire deletes only expired files according to the trashbin_retention_obligation setting in config.php. The default setting is auto, which keeps files in the Trashbin for 30 days, then deletes the oldest files as space is needed to keep users within their storage quotas. Files may not be deleted if the space is not needed.

The default is to delete expired files for all users, or you may list users in a space-delimited list:

```
sudo -u www-data php occ trashbin:cleanup user1 user2
Remove deleted files of user1
Remove deleted files of user2
```

See the **Deleted Files** section in Config.php Parameters, and the Trashbin section of Using the occ Command.

How To Install and Configure an LDAP Proxy-Cache Server

Background

To reduce network traffic overhead and avoid problems either logging in or performing user searches while sharing, it's an excellent idea to implement an LDAP proxy cache.

An LDAP proxy cache server, similar to other kinds of caching servers, is a special type of LDAP replica. It can cache a range of LDAP records, often resulting in improved LDAP server performance.

Specifically, the records which need to be cached, for improved ownCloud performance, are:

- Users that are allowed to log in
- Groups (limited to the allowed users)
- Search fields (e.g., sAMAccountName, cn, sn, givenName, and displayName)

How To Setup the Server

There's not that much to it, just the following five steps:

1. Install OpenLDAP
2. Configure the server
3. Edit the default configuration directory
4. Perform a test search
5. Check the logs
6. Configure ownCloud LDAP app

Let's begin.

1. Install OpenLDAP

There are a number of [LDAP server implementations](#) available. The one used in this guide is [OpenLDAP](#).

Note

While OpenLDAP does work on any operating system, for the purposes of this guide, we'll be using a Debian-based Linux distribution.

Firstly, log in as root, and update your system, to ensure that you're using the latest packages.:

```
apt-get update && apt-get upgrade -y
```

Next, install OpenLDAP and its associated packages.:

```
apt-get install slapd ldap-utils -y
```

2. Configure the Server

With OpenLDAP installed and running, you now need to configure the server. One way of doing so, is to create a configuration file. So, create `/etc/ldap/slapd.conf` with your text editor of choice, and add the following configuration to it.

```
# This an example of a config file:

# See slapd.conf(5)

# Global Directives:

# Schema and objectClass definitions

include          /etc/ldap/schema/core.schema
include          /etc/ldap/schema/cosine.schema
include          /etc/ldap/schema/nis.schema
include          /etc/ldap/schema/inetorgperson.schema

# Where the pid file is put. The init.d script
# will not stop the server if you change this.

pidfile         /var/run/slapd/slapd.pid

# List of arguments that were passed to the server

argsfile        /var/run/slapd/slapd.args

# Read slapd.conf(5) for possible values
# Change loglevel to "any" if you want to see everything.

loglevel        none

# Where the dynamically loaded modules are stored

modulepath      /usr/lib/ldap

# Here are the recommended modules:

# module for the target ldap-server

moduleload      back_ldap.la

# module for your local database

moduleload      back_hdb.la

# module for rewriting attributes

moduleload      rwm

# caching module

moduleload      pcache.la

# module to enable memberof in LDAP

moduleload      memberof.la
```

Configuration

```
# The maximum number of entries that is returned for a search operation
sizelimit 500

# The tool-threads parameter sets the actual amount of cpu's that is used
# for indexing.

tool-threads 1

# Type of backend, for example "ldap"

backend          ldap

# Type of database

database         ldap

# If you only have read access, set this to "yes"

readonly        yes

# Set which protocol to use, we suggest "3"

protocol-version 3

# remember bind credentials

rebind-as-user

# If you want to save time and don't want to list all the refferals, set to "yes"

norefs   yes

# Same as above

chase-referrals no

# Specify the URL of your ldap server and the port.
# For unencrypted access use the port 389, for encrypted 636
# If you have to use 636, you will also probably have to import
# the certificate of your target server. restart your webserver after you do.

uri "ldap://192.168.178.2:389"

# The base of your directory in database, for example "dc=ldap01,dc=com"
suffix          "dc=ldap01,dc=com"

# rootdn directive for specifying a superuser on the database.
# If you don't have access to the admin user, use the one you have.

rootdn          "cn=admin,dc=ldap01,dc=com"

# Now we start initialising the modules
# First the rewrite module

overlay        rwm
```

Configuration

```
# Now we rewrite the attributes

rwm-map          attribute uid sAMAccountName
rwm-map          attribute dn distinguishedName

# Next one is optional, if you want memberof, for the groups,
# you have to load it.

overlay          memberof

# Now we load the caching module

overlay pcache

# The directive enables proxy caching
# See slapo-pcache

# pcache <database> <max_entries> <numattrsets> <entry_limit> <cc_period>
# Parameters:
#
# <database> for cached entries.
# <max_entries> when reached - cache replacement is invoked
# <numattrsets> = pcacheAttrset
# <entry_limit> limit to the number of entries returned
# <cc_period> Consistency check time to wait

pcache hdb 100000 3 1000 100

# pcachePersist { TRUE | FALSE }
# Write cached results into the database
# Results remain in database after restart

pcachePersist TRUE

# Where the database file are physically stored for database #1

directory        "/var/lib/ldap"

# Caching templates for general search

# pcacheAttrset <index> <attrs...>
# First set the index number
# Then set the attribute to cache

pcacheAttrset    0 1.1

# pcacheTemplate <template_string> <attrset_index> <ttl>
# First define the query string to cache
# Then reference the Attrset
# Last set the time-to-live

pcacheTemplate  (&(|(objectClass=))) 0 3600

pcacheTemplate (objectClass=*) 0 3600

# User Name Field (Advanced Tab)

pcacheAttrset    1 displayname
pcacheTemplate (objectClass=*) 1 3600
```

Configuration

```
# Group Field

pcacheAttrset    2 memberOf
pcacheTemplate (objectClass=*) 2 3600

# This an example of a config file:

# See slapd.conf(5)

# Global Directives:

# Schema and objectClass definitions

include          /etc/ldap/schema/core.schema
include          /etc/ldap/schema/cosine.schema
include          /etc/ldap/schema/nis.schema
include          /etc/ldap/schema/inetorgperson.schema

# Where the pid file is put. The init.d script
# will not stop the server if you change this.

pidfile         /var/run/slapd/slapd.pid

# List of arguments that were passed to the server

argsfile        /var/run/slapd/slapd.args

# Read slapd.conf(5) for possible values
# Change loglevel to "any" if you want to see everything.

loglevel        none

# Where the dynamically loaded modules are stored

modulepath      /usr/lib/ldap

# Here are the recommended modules:

# module for the target ldap-server

moduleload      back_ldap.la

# module for your local database

moduleload      back_hdb.la

# module for rewriting attributes

moduleload      rwm

# caching module

moduleload      pcache.la

# module to enable memberof in LDAP

moduleload      memberof.la
```

Configuration

```
# The maximum number of entries that is returned for a search operation
sizelimit 500

# The tool-threads parameter sets the actual amount of cpu's that is used
# for indexing.

tool-threads 1

# Type of backend, for example "ldap"

backend      ldap

# If you only have read access, set this to "yes"

readonly     yes

# Set which protocol to use, we suggest "3"

protocol-version 3

# remember bind credentials

rebind-as-user

# If you want to save time and don't want to list all the refferals, set to "yes"

norefs   yes

# Same as above

chase-referrals no

# Specify the URL of your ldap server and the port.
# For unencrypted access use the port 389, for encrypted 636
# If you have to use 636, you will also probably have to import
# the certificate of your target server.
# Restart your webserver after you do.

uri "ldap://192.168.178.2:389"

# The base of your directory in database, for example "dc=ldap01,dc=com"
suffix      "dc=ldap01,dc=com"

# rootdn directive for specifying a superuser on the database.
# If you don't have access to the admin user, use the one you have.

rootdn      "cn=admin,dc=ldap01,dc=com"

# Now we start initialising the modules
# First the rewrite module

overlay      rwm

# Now we rewrite the attributes

rwm-map      attribute uid sAMAccountName
rwm-map      attribute dn distinguishedName
```

Configuration

```
# Next one is optional, if you want memberof, for the groups,
# you have to load it.

overlay      memberof

# Now we load the caching module

overlay pcache

# The directive enables proxy caching
# See slapd-pcache

# pcache <database> <max_entries> <numattrsets> <entry_limit> <cc_period>
# Parameters:
#
# <database> for cached entries.
# <max_entries> when reached - cache replacement is invoked
# <numattrsets> = pcacheAttrset
# <entry_limit> limit to the number of entries returned
# <cc_period> Consistency check time to wait

pcache hdb 100000 3 1000 100

# pcachePersist { TRUE | FALSE }
# Write cached results into the database
# Results remain in database after restart

pcachePersist TRUE

# Where the database file are physically stored for database #1

directory      "/var/lib/ldap"

# Caching templates for general search

# pcacheAttrset <index> <attrs...>
# First set the index number
# Then set the attribute to cache

pcacheAttrset 0 1.1

# pcacheTemplate <template_string> <attrset_index> <ttl>
# First define the query string to cache
# Then reference the Attrset
# Last set the time-to-live

pcacheTemplate (&(|(objectClass=))) 0 3600

pcacheTemplate (objectClass=*) 0 3600

# User Name Field (Advanced Tab)

pcacheAttrset 1 displayname
pcacheTemplate (objectClass=*) 1 3600

# Group Field

pcacheAttrset 2 memberOf
pcacheTemplate (objectClass=*) 2 3600
```

Note

This configuration only caches queries from a single Active Directory server. To cache queries from multiple Active Directory servers, a configuration is available below.

After you've done that, save the file, test that there are no errors in the configuration by running:

```
slaptest -f /etc/ldap/slapd.conf
```

Note

If you see warnings in the console output, they are not crucial.

3. Enable the configuration file

Next, we need to tell OpenLDAP to use our configuration. To do so, open `/etc/default/slapd` and add the following line to it:

```
SLAPD_CONF=/etc/ldap/slapd.conf
```

With that done, restart OpenLDAP by running the following command:

```
service slapd restart
```

4. Open the log

Open another terminal and see the systemlog with the following command.:

```
tail -f /var/log/syslog | grep QUERY
```

If there is no such file, you need to install rsyslog with.

```
apt install rsyslog
```

5. Perform a test search

Now that the server's installed, configured, and running, we next need to perform a search. This will check that records are being correctly cached. To do so, update the command below with values from your Active Directory server configuration, and then run it.:

```
ldapsearch -h localhost -x -LLL -D "cn=admin,cn=users,dc=example,dc=com" -b "cn=users,dc=example,dc=com"
```

or

```
ldapsearch -H ldaps://localhost:636 -x -LLL -D "cn=admin,cn=users,dc=example,dc=com" -b "cn=users,dc=example,dc=com" -W "(cn=Administrator)" name
```

-h = host address (Example: localhost or 192.168.1.1) -H = host address (Example: [ldap://](#) or ldaps:// hostname or ip and port :389 or :636) -x = simple authentication -b = Search Base, (Example: "cn=Users,dc=example,dc=com") -D = User with permissions (Example: "cn=Admin,dc=example,dc=com") -LLL = Show only results, no extra information -w = Password ("Password") -W = Password, will ask for password and hide your input "(cn=Administrator)" = Filter the search name = Show only this attributes

If you see results including: "Query cachable" and "Query answered (x) times", then the setup works.

6. Configure ownCloud LDAP App

Login as ownCloud admin in your ownCloud server.

Click on the dropdown menu in the top-left corner next to "Files", then on Apps.

Configuration

Click on “Not enabled”, and enable the “LDAP user and group backend” App.

Go to the admin dropdown menu in the top-right corner, select “Admin”.

In the administration section, click on the left side on “LDAP”.

Configure Server-Tab

First enter the server address, either IP or DN.

You can click on the button to detect your servers port or enter it manually.

Next, enter the dn of the user, you want to log in with and in the next line enter the password.

Then you can click on “detect base dn” or enter it manually. then click on “test base dn”.

If you fulfill all the requirements you should get a green light and “configuration ok” message.

Configure Users

Select the objectclass for the users, for example “user”.

Verify your settings; where you will see the number of users being found.

Configure Login Attributes

A configuration appears by default, adjusted it to your users configuration.

If required, adjust the login parameters additional login attributes.

You can check users with any of the allowed login options. You can adjust them or leave them the way they are.

Configure Groups

Select all the objectclasses for your groups, for example “group”. Verify your settings.

Configure Advanced

Configuration Active should be selected.

Adjust the Cache TTL (time to live) value as required.

ownCloud usually autoselects the best settings for each AD configuration.

Check if the Group-Member association is “Member (AD)”. That’s important for the users being shown in their respective groups.

Select “Nested groups”, if you have them.

Configure Expert

“Internal Username Attribute”

Here we need to set “cn” for the users being shown with their unique name. If you leave that field empty, each user will get a unique uid as a string of numbers and letters.

Clear the Username and Groupname Mapping and test your configuration by clicking on the buttons below.

Navigate to Admin -> Users and check if all your users are listed properly, and shown in the right groups.

Go to the homepage of your ownCloud server and try to share something with one of your users

If everything is set up correctly, you now have an LDAP proxy server to your active directory that will reduce the network traffic by caching the searches you perform.

Cache Multiple Active Directory Servers

If you have more than one that you want to cache, in /etc/ldap/slapd.conf add the following configuration instead, adjusting as necessary. The ownCloud LDAP app settings are the same as in section 6.

```
# This is an example of a config file:
```

```
# See slapd.conf(5)
```

Configuration

```
# Global Directives:

# Schema and objectClass definitions

include      /etc/ldap/schema/core.schema
include      /etc/ldap/schema/cosine.schema
include      /etc/ldap/schema/nis.schema
include      /etc/ldap/schema/inetorgperson.schema
include      /etc/ldap/schema/misc.schema

# Where the pid file is put. The init.d script
# will not stop the server if you change this.

pidfile      /var/run/slapd/slapd.pid

# List of arguments that were passed to the server

argsfile     /var/run/slapd/slapd.args

# Read slapd.conf(5) for possible values
# Change loglevel to "any" if you want to see everything.

loglevel     none

# Where the dynamically loaded modules are stored

modulepath   /usr/lib/ldap

# Here are the recommended modules:

# module for meta-database

moduleload back_meta.la

# module for the target ldap-server

moduleload   back_ldap.la

# module for your local database

moduleload   back_hdb.la

# module for rewriting attributes

moduleload   rwm

# caching module

moduleload   pcache.la

# module to enable memberof in ldap

moduleload   memberof.la

# The maximum number of entries that is returned for a search operation

sizelimit 500

# The tool-threads parameter sets the actual amount of cpu's that is used
```

Configuration

```
# for indexing.

tool-threads 1

# If you want to save time and don't want to list all the refferrals, set "yes"
noref yes

# Same as above

chase-referrals no

# See slapd-meta

# database type, for multiple ADS "meta" is required

database meta

# now we create a local ldap tree

# in our tree we put the multiple ADS on different branches

# we need a suffix, an admin, and a password

suffix "dc=owncloud,dc=com"

rootdn "cn=Administrator,cn=Users,dc=example,dc=com"

rootpw "Password"

# now we specify our ADS

# First-AD

# uri <protocol>://[<host>]/<naming context>

uri "ldap://first.ad.com:389/
cn=users,dc=first,dc=example,dc=com"

# here we need to set the virtual name to the real name

# the virtual name is a branch in our new created ldap tree

# suffixmassage <virtual naming context> <real naming context>

suffixmassage "cn=users,dc=first,dc=example,dc=com" "cn=users,dc=first,dc=ad,dc=com"

# authentication parameters

idassert-bind bindmethod=simple
binddn="cn=user01,cn=users,dc=first,dc=owncloud,dc=com"
credentials="Password01"

# Second-AD
uri "ldaps://second.ad.com:636/cn=users,dc=second,dc=example,dc=com"
suffixmassage "cn=users,dc=second,dc=example,dc=com" "cn=users,dc=second,dc=ad,dc=com"
idassert-bind bindmethod=simple
binddn="cn=user02,cn=users,dc=second,dc=owncloud,dc=com"
```

Configuration

```
        credentials="Password02"

# Now we start initialising the modules
# First the rewrite module

overlay          rwm

# Now we rewrite the attributes

rwm-map          attribute uid sAMAccountName
rwm-map          attribute dn distinguishedName

# Next one is optional, if you want memberof, for the groups,
# you have to load it.

overlay          memberof

# Now we load the caching module

overlay pcache

# The directive enables proxy caching
# See slapo-pcache

# pcache <database> <max_entries> <numattrsets> <entry_limit> <cc_period>
# Parameters:
#
# <database> for cached entries.
# <max_entries> when reached - cache replacement is invoked
# <numattrsets> = pcacheAttrset
# <entry_limit> limit to the number of entries returned
# <cc_period> Consistency check time to wait

pcache hdb 100000 3 1000 100

# pcachePersist { TRUE | FALSE }
# Write cached results into the database
# Results remain in database after restart

pcachePersist TRUE

# Where the database files are physically stored for database #1

directory        "/var/lib/ldap"

# Caching templates for general search

# pcacheAttrset <index> <attrs...>
# First set the index number
# Then set the attribute to cache

pcacheAttrset    0 1.1

# pcacheTemplate <template_string> <attrset_index> <ttl>
# First define the query string to cache
# Then reference the Attrset
# Last set the time-to-live

pcacheTemplate  (&(|(objectClass=))) 0 3600
```

```

pcacheTemplate (objectClass=*) 0 3600

# User Name Field (Advanced Tab)

pcacheAttrset 1 displayname
pcacheTemplate (objectClass=*) 1 3600

# Group Field

pcacheAttrset 2 memberOf
pcacheTemplate (objectClass=*) 2 3600

```

Mimetypes Management

ownCloud allows you to create aliases for mimetypes and map file extensions to a mimetype. These allow administrators the ability to change the existing icons that ownCloud uses to represent certain file types and folders, as well as to use custom icons for mimetypes and file extensions which ownCloud doesn't natively support. This is handy in a variety of situations, such as when you might want a custom audio icon for audio mimetypes, instead of the default file icon.

Mimetype Aliases

ownCloud's default mimetype configuration is defined in `owncloud/resources/config/mimetypealiases.dist.json`, which you can see a snippet of below. The mimetype's on the left, and the icon used to represent that mimetype is on the right.

```
{
  "application/coreldraw": "image",
  "application/font-sfnt": "image",
  "application/font-woff": "image",
  "application/illustrator": "image",
  "application/epub+zip": "text",
  "application/javascript": "text/code",
}
```

Stepping through that file, you can see that:

- the image icon is used to represent Corel Draw, SFNT and WOFF font files, and Adobe Illustrator files.
- ePUB files are represented by the text file icon.
- JavaScript files are represented by the text/code icon.

Changing Existing Icons and Using Custom Icons

If you want to change one or more of the existing icons which ownCloud uses, or if you want to expand the available list, here's how to do so.

First, create a copy of `resources/config/mimetypealiases.dist.json`, naming it `mimetypealiases.json` and storing it in `config/`. This is required for two reasons:

1. It will take precedence over the default file.
2. The original file will get replaced on each ownCloud upgrade.

Then, either override one or more existing definitions or add new, custom, aliases as required.

Note

Please refer to [the ownCloud theming documentation](#) for where to put the new image files.

Configuration

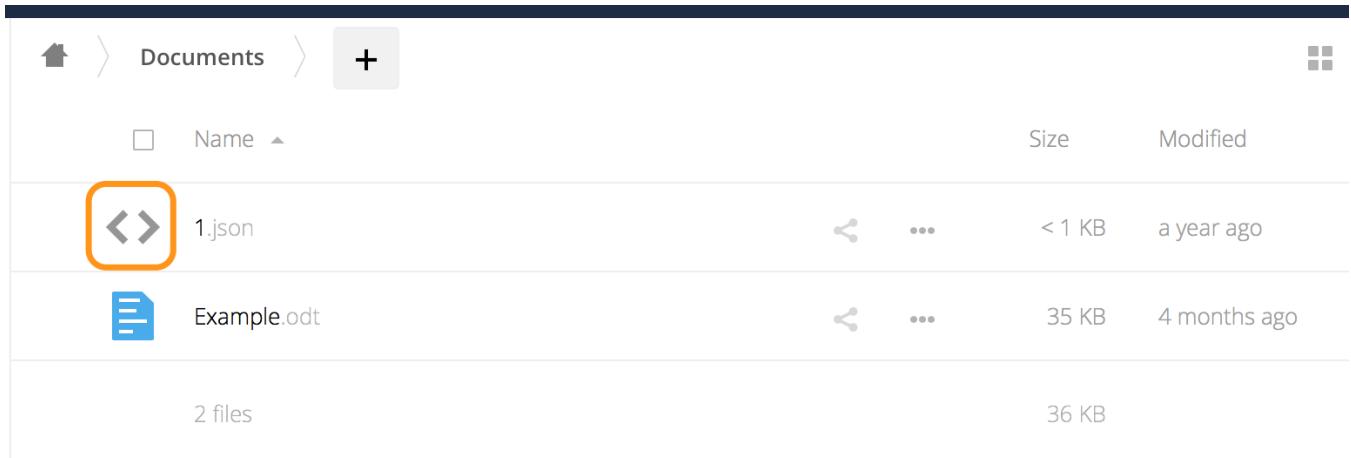
Some common mimetypes that may be useful in creating aliases are:

Mimetype	Description
image	Generic image
image/vector	Vector image
audio	Generic audio file
x-office/document	Word processed document
x-office/spreadsheet	Spreadsheet
x-office/presentation	Presentation
text	Generic text document
text/code	Source code

Once you have made changes to config/mimetypealiases.json, use the occ command to propagate the changes throughout your ownCloud installation. Here is an example for Ubuntu Linux:

```
$ sudo -u www-data php occ maintenance:mimetype:update-js
```

Example - Changing the JSON File Icon



Let's step through an example, from start to finish, of changing the icon that ownCloud uses to represent JSON files, which you can see above.

1. From the root directory of your ownCloud installation, copy resources/config/mimetypealiases.dist.json to /config/mimetypealiases.json.
2. Update the alias for application/json, which you should find on line 8, to match the following, and save the file:

```
"application/json": "text/json",
```

3. Copy a new SVG icon to represent JSON files to core/img/filetypes, calling it text-json.svg.

Note

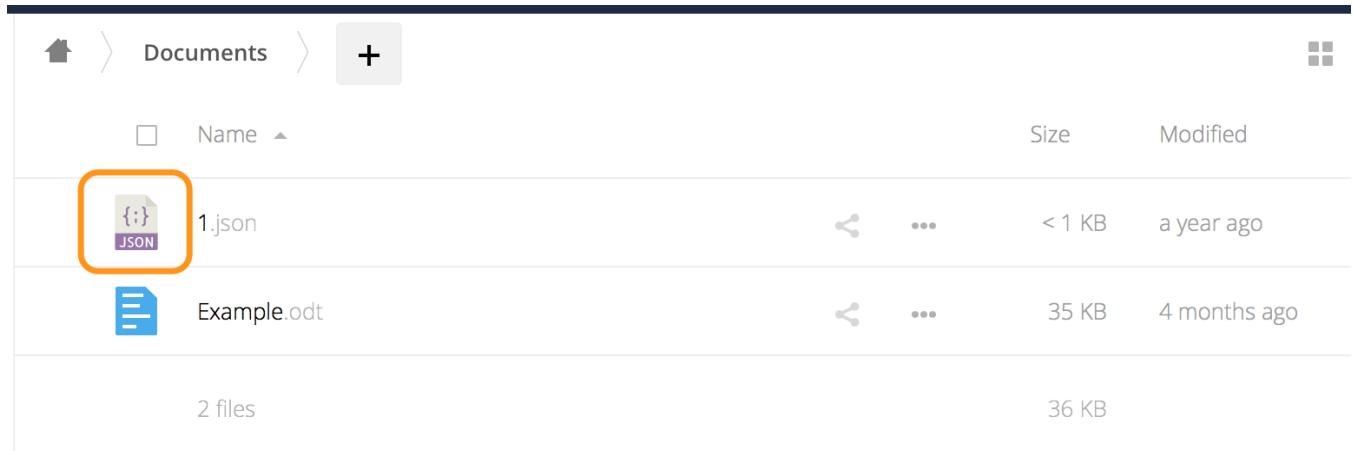
The name and location of the file are important. The location is because the core/img/filetypes directory stores the mimetype file icons. The name is important as it's a rough mapping between the alias name and the icon's file name, i.e., text/json becomes text-json.

4. Run the following command to update the mimetype alias database.

```
$ sudo -u www-data php occ maintenance:mimetype:update-js
```

Configuration

After doing so, whenever you view a folder that contains JSON files or upload one, your new icon file will be used to represent the file, as in the image below.



Mimetype Mapping

ownCloud allows administrators to map a file extension to a mimetype, e.g., such as mapping files ending in `.mp3` to `audio/mpeg`. Which then, in turn, allows ownCloud to show the audio icon.

The default file extension to mimetype mapping configuration is stored in `resources/config/mimetypesmapping.dist.json`. This is similar to `resources/config/mimetypealiases.dist.json`, and also returns a basic JSON array.

```
{  
  "3gp": [ "video/3gpp" ],  
  "7z": [ "application/x-7z-compressed" ],  
  "accdb": [ "application/msaccess" ],  
  "ai": [ "application/illustrator" ],  
  "apk": [ "application/vnd.android.package-archive" ],  
  "arw": [ "image/x-dcraw" ],  
  "avi": [ "video/x-msvideo" ],  
  "bash": [ "text/x-shellscript" ],  
  "json": [ "application/json", "text/plain" ],  
}
```

In the example above, you can see nine mimetypes mapped to file extensions. Each of them, except the last (`json`), maps a file extension to a mimetype. Now take a look at the JSON example.

In this case, ownCloud will first check if a mimetype alias is defined for `application/json`, in `mimetypealiases.json`. If it is, it will use that icon. If not, then ownCloud will fall back to using the icon for `text/plain`.

If you want to update or extend the existing mapping, as with updating the mimetype aliases, create a copy of `resources/config/mimetypesmapping.dist.json` and name it `mimetypesmapping.json` and storing it in `config/`. Then, in this new file, make any changes required.

Note

Please refer to [the ownCloud theming documentation](#) for where to put the new image files.

Icon retrieval

When an icon is retrieved for a mimetype, if the full mimetype cannot be found, the search will fallback to looking for the part before the slash. Given a file with the mimetype `image/my-custom-image`, if no icon exists for the full mimetype, the icon for `image` will be used instead. This allows specialized mimetypes to fallback to generic icons when the relevant icons are unavailable.

Server Configuration

Warnings on Admin Page

Your ownCloud server has a built-in configuration checker, and it reports its findings at the top of your Admin page. These are some of the warnings you might see, and what to do about them.

Security & setup warnings

- No memory cache has been configured. To enhance your performance please configure a memcache if available. Further information can be found in our [documentation](#).
- You are accessing this site via HTTP. We strongly suggest you configure your server to require using HTTPS instead.

Please double check the [installation guides](#), and check for any errors or warnings in the log.

Cache Warnings

"No memory cache has been configured. To enhance your performance please configure a memcache if available." ownCloud supports multiple php caching extenstions:

- APCu
- Memcached
- Redis (minimum required PHP extension version: 2.2.6)

You will see this warning if you have no caches installed and enabled, or if your cache does not have the required minimum version installed; older versions are disabled because of performance problems.

If you see "{Cache} below version {Version} is installed. for stability and performance reasons we recommend to update to a newer {Cache} version" then you need to upgrade, or, if you're not using it, remove it.

You are not required to use any caches, but caches improve server performance. See [Memory Caching](#).

Transactional file locking is disabled

"Transactional file locking is disabled, this might lead to issues with race conditions."

Please see [Transactional File Locking](#) for how to correctly configure your environment for transactional file locking.

You are accessing this site via HTTP

"You are accessing this site via HTTP. We strongly suggest you configure your server to require using HTTPS instead." Please take this warning seriously; using HTTPS is a fundamental security measure. You must configure your Web server to support it, and then there are some settings in the **Security** section of your ownCloud Admin page to enable. The following pages describe how to enable HTTPS on the Apache and Nginx Web servers.

[Enable SSL \(on Apache\)](#)

[Use HTTPS](#)

The test with `getenv("PATH")` only returns an empty response

Some environments are not passing a valid PATH variable to ownCloud. The PHP-FPM provides the information about how to configure your environment.

The "Strict-Transport-Security" HTTP header is not configured

"The "Strict-Transport-Security" HTTP header is not configured to least "15552000" seconds. For enhanced security we recommend enabling HSTS as described in our [security tips](#)."

The HSTS header needs to be configured within your Web server by following the Enable HTTP Strict Transport Security documentation

/dev/urandom is not readable by PHP

“/dev/urandom is not readable by PHP which is highly discouraged for security reasons. Further information can be found in our documentation.”

This message is another one which needs to be taken seriously. Please have a look at the Give PHP read access to /dev/urandom documentation.

Your Web server is not yet set up properly to allow file synchronization

“Your web server is not yet set up properly to allow file synchronization because the WebDAV interface seems to be broken.”

At the ownCloud community forums a larger [FAQ](#) is maintained containing various information and debugging hints.

Outdated NSS / OpenSSL version

“cURL is using an outdated OpenSSL version (OpenSSL/\$version). Please update your operating system or features such as installing and updating apps via the ownCloud Marketplace or Federated Cloud Sharing will not work reliably.”

“cURL is using an outdated NSS version (NSS/\$version). Please update your operating system or features such as installing and updating apps via the ownCloud Marketplace or Federated Cloud Sharing will not work reliably.”

There are known bugs in older OpenSSL and NSS versions leading to misbehaviour in combination with remote hosts using SNI. A technology used by most of the HTTPS websites. To ensure that ownCloud will work properly you need to update OpenSSL to at least 1.0.2b or 1.0.1d. For NSS the patch version depends on your distribution and an heuristic is running the test which actually reproduces the bug. There are distributions such as RHEL/CentOS which have this backport still [pending](#).

Your Web server is not set up properly to resolve ./well-known/caldav/ or ./well-known/carddav/

Both URLs need to be correctly redirected to the DAV endpoint of ownCloud. Please refer to Service discovery for more info.

Some files have not passed the integrity check

Please refer to the Fixing Invalid Code Integrity Messages documentation how to debug this issue.

Your database does not run with “READ COMMITTED” transaction isolation level

“Your database does not run with “READ COMMITTED” transaction isolation level. This can cause problems when multiple actions are executed in parallel.”

Please refer to MySQL / MariaDB “READ COMMITTED” transaction isolation level how to configure your database for this requirement.

Importing System-wide and Personal SSL Certificates

Modern Web browsers try to keep us safe, and so they blast us with scary warnings when sites have the smallest errors in their SSL certificates, or when they use self-signed SSL certificates. ownCloud admins encounter this when creating Federation shares, or setting up external storage mounts. There is no reason against using self-signed certificates on your own networks; they’re fast, free, and easy.

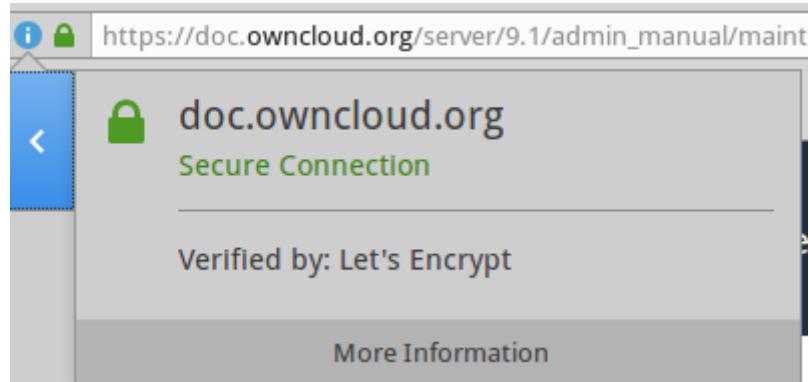
Importing Personal SSL Certificates

ownCloud has several methods for importing self-signed certificates so that you don’t have to hassle with Web browser warnings. When you allow your users to create their own external storage mounts or Federation shares, they can import SSL certificates for those shares on their Personal pages.

SSL Root Certificates

Import root certificate

Click the **Import root certificate** button to open a file picker. You can distribute copies of your SSL certificates to your users (via an ownCloud share!), or users can download them from their Web browsers. Click on the little padlock icon and click through until you see a **View Certificate** button, then keep going until you can download it. In Firefox and Chromium there is an **Export** button for downloading your own copy of a site's SSL certificate.



Click "More information" in Firefox to import SSL certificate

Site-wide SSL Import

The personal imports only work for individual users. You can enable site-wide SSL certificates for all of your users on your ownCloud admin page. To enable this, you must add this line to your `config.php` file:

```
'enable_certificate_management' => true,
```

Then you'll have a **Import root certificate** button on your admin page, just like the one on your personal page.

Using OCC to Import and Manage SSL Certificates

The `occ` command has options for listing and managing your SSL certificates:

```
security:certificates      list trusted certificates
security:certificates:import import trusted certificate
security:certificates:remove remove trusted certificate
```

See Using the `occ` Command to learn about how to use `occ`.

Using the `occ` Command

ownCloud's `occ` command (ownCloud console) is ownCloud's command-line interface. You can perform many common server operations with `occ`, such as installing and upgrading ownCloud, managing users and groups, encryption, passwords, LDAP setting, and more.

`occ` is in the `owncloud/` directory; for example `/var/www/owncloud` on Ubuntu Linux. `occ` is a PHP script. **You must run it as your HTTP user** to ensure that the correct permissions are maintained on your ownCloud files and directories.

occ Command Directory

- Run `occ` As Your HTTP User
- Apps Commands
- Background Jobs Selector

Configuration

- Config Commands
- Dav Commands
- Database Conversion
- Encryption
- Federation Sync
- File Operations
- Files External
- Group Commands
- Integrity Check
- I10n, Create Javascript Translation Files for Apps
- LDAP Commands
- Logging Commands
- Maintenance Commands
- Market
- Reports
- Security
- Ransomware Protection
- Shibboleth Modes (Enterprise Edition only)
- Trashbin
- User Commands
- Versions
- Command Line Installation
- Command Line Upgrade
- Two-factor Authentication
- Disable Users

Run occ As Your HTTP User

The HTTP user is different on the various Linux distributions. See Set Strong Directory Permissions to learn how to find your HTTP user.

- The HTTP user and group in Debian/Ubuntu is www-data.
- The HTTP user and group in Fedora/CentOS is apache.
- The HTTP user and group in Arch Linux is http.
- The HTTP user in openSUSE is wwwrun, and the HTTP group is www.

If your HTTP server is configured to use a different PHP version than the default (/usr/bin/php), occ should be run with the same version. For example, in CentOS 6.5 with SCL-PHP54 installed, the command looks like this:

```
sudo -u apache /opt/rh/php54/root/usr/bin/php /var/www/html/owncloud/occ
```

Running occ with no options lists all commands and options, like this example on Ubuntu:

```
sudo -u www-data php occ
ownCloud version 9.0.0
```

```
Usage:
  command [options] [arguments]
```

Configuration

```
Options:
-h, --help          Display this help message
-q, --quiet         Do not output any message
-V, --version        Display this application version
--ansi              Force ANSI output
--no-ansi            Disable ANSI output
-n, --no-interaction Do not ask any interactive question
--no-warnings        Skip global warnings, show command output only
-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output,
                       2 for more verbose output and 3 for debug

Available commands:
check               check dependencies of the server
environment
help                Displays help for a command
list                Lists commands
status              show some status information
upgrade             run upgrade routines after installation of
                     a new release. The release has to be
                     installed before.
```

This is the same as `sudo -u www-data php occ list`. Run it with the `-h` option for syntax help:

```
sudo -u www-data php occ -h
```

Display your ownCloud version:

```
sudo -u www-data php occ -V
ownCloud version 9.0.0
```

Query your ownCloud server status:

```
sudo -u www-data php occ status
- installed: true
- version: 9.0.0.19
- versionstring: 9.0.0
- edition:
```

`occ` has *options*, *commands*, and *arguments*. Commands are required. Options are optional. Arguments can be required or optional. The generic, syntax is:

```
occ [options] command [arguments]
```

Get detailed information on individual commands with the `help` command, like this example for the `maintenance:mode` command

```
sudo -u www-data php occ help maintenance:mode
Usage:
  maintenance:mode [options]

Options:
  --on           enable maintenance mode
  --off          disable maintenance mode
-h, --help        Display this help message
-q, --quiet       Do not output any message
-V, --version     Display this application version
--ansi          Force ANSI output
--no-ansi        Disable ANSI output
-n, --no-interaction Do not ask any interactive question
--no-warnings    Skip global warnings, show command output only
-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output,
                       2 for more verbose output and 3 for debug
```

Configuration

The `status` command from above has an option to define the output format. The default is plain text, but it can also be `json`:

```
sudo -u www-data php occ status --output=json
{"installed":true,"version":"9.0.0.19","versionstring":"9.0.0","edition":""}
```

or `json_pretty`:

```
sudo -u www-data php occ status --output=json_pretty
{
    "installed": true,
    "version": "9.0.0.19",
    "versionstring": "9.0.0",
    "edition": ""
}
```

This output option is available on all list and list-like commands, which include `status`, `check`, `app:list`, `config:list`, `encryption:status` and `encryption:list-modules`.

Apps Commands

The app commands `list`, `enable`, and `disable` apps

```
app
  app:check-code      check code to be compliant
  app:disable        disable an app
  app:enable         enable an app
  app:getpath        Get an absolute path to the app directory
  app:list           List all available apps
```

List all of your installed apps or optionally provide a search pattern to restrict the list of apps to those whose name matches the given regular expression. The output shows whether they are enabled or disabled

```
sudo -u www-data php occ app:list [<search-pattern>]
```

Enable an app, for example the Market app

```
sudo -u www-data php occ app:enable market
market enabled
```

Disable an app

```
sudo -u www-data php occ app:disable market
market disabled
```

`app:check-code` has multiple checks: it checks if an app uses ownCloud's public API (`OCP`) or private API (`OC_`), and it also checks for deprecated methods and the validity of the `info.xml` file. By default all checks are enabled. The Activity app is an example of a correctly-formatted app

```
sudo -u www-data php occ app:check-code notifications
App is compliant - awesome job!
```

If your app has issues, you'll see output like this

```
sudo -u www-data php occ app:check-code foo_app
Analysing /var/www/owncloud/apps/files/foo_app.php
4 errors
  line  45: OCP\Response - Static method of deprecated class must not be
  called
  line  46: OCP\Response - Static method of deprecated class must not be
  called
  line  47: OCP\Response - Static method of deprecated class must not be
  called
  line  49: OC_Util - Static method of private class must not be called
```

You can get the full file path to an app

Configuration

```
sudo -u www-data php occ app:getpath notifications  
/var/www/owncloud/apps/notifications
```

Background Jobs Selector

Use the `background` command to select which scheduler you want to use for controlling *background jobs*, *Ajax*, *Webcron*, or *Cron*. This is the same as using the **Cron** section on your ownCloud Admin page.

<code>background</code>	
<code>background:ajax</code>	Use ajax to run background jobs
<code>background:cron</code>	Use cron to run background jobs
<code>background:webcron</code>	Use webcron to run background jobs

This example selects Ajax:

```
sudo -u www-data php occ background:ajax  
Set mode for background jobs to 'ajax'
```

The other two commands are:

- `background:cron`
- `background:webcron`

See [Background Jobs](#) to learn more.

Config Commands

The `config` commands are used to configure the ownCloud server.

<code>config</code>	
<code>config:app:delete</code>	Delete an app config value
<code>config:app:get</code>	Get an app config value
<code>config:app:set</code>	Set an app config value
<code>config:import</code>	Import a list of configuration settings
<code>config:list</code>	List all configuration settings
<code>config:system:delete</code>	Delete a system config value
<code>config:system:get</code>	Get a system config value
<code>config:system:set</code>	Set a system config value

You can list all configuration values with one command:

```
sudo -u www-data php occ config:list
```

By default, passwords and other sensitive data are omitted from the report, so the output can be posted publicly (e.g., as part of a bug report). In order to generate a full backport of all configuration values the `--private` flag needs to be set:

```
sudo -u www-data php occ config:list --private
```

The exported content can also be imported again to allow the fast setup of similar instances. The import command will only add or update values. Values that exist in the current configuration, but not in the one that is being imported are left untouched.

```
sudo -u www-data php occ config:import filename.json
```

It is also possible to import remote files, by piping the input:

```
sudo -u www-data php occ config:import < local-backup.json
```

Note

While it is possible to update/set/delete the versions and installation statuses of apps and ownCloud itself, it is **not** recommended to do this directly. Use the `occ app:enable`, `occ app:disable` and `occ update` commands instead.

Getting a Single Configuration Value

These commands get the value of a single app or system configuration:

```
sudo -u www-data php occ config:system:get version  
9.0.0.19  
  
sudo -u www-data php occ config:app:get activity installed_version  
2.2.1
```

Setting a Single Configuration Value

These commands set the value of a single app or system configuration:

```
sudo -u www-data php occ config:system:set logtimezone  
--value="Europe/Berlin"  
System config value logtimezone set to Europe/Berlin  
  
sudo -u www-data php occ config:app:set files_sharing  
incoming_server2server_share_enabled --value="yes" --type=boolean  
Config value incoming_server2server_share_enabled for app files_sharing set to yes
```

The `config:system:set` command creates the value, if it does not already exist. To update an existing value, set `--update-only`:

```
sudo -u www-data php occ config:system:set doesnotexist --value="true"  
--type=boolean --update-only  
Value not updated, as it has not been set before.
```

Note that in order to write a Boolean, float, or integer value to the configuration file, you need to specify the type on your command. This applies only to the `config:system:set` command. The following values are known:

- boolean
- integer
- float
- string (default)

When you want to e.g., disable the maintenance mode run the following command:

```
sudo -u www-data php occ config:system:set maintenance --value=false  
--type=boolean  
ownCloud is in maintenance mode - no app have been loaded  
System config value maintenance set to boolean false
```

Setting an Array of Configuration Values

Some configurations (e.g., the trusted domain setting) are an array of data. In order to set (and also get) the value of one key, you can specify multiple config names separated by spaces:

```
sudo -u www-data php occ config:system:get trusted_domains  
localhost  
owncloud.local  
sample.tld
```

To replace `sample.tld` with `example.com` `trusted_domains => 2` needs to be set:

Configuration

```
sudo -u www-data php occ config:system:set trusted_domains 2  
--value=example.com  
System config value trusted_domains => 2 set to string example.com  
  
sudo -u www-data php occ config:system:get trusted_domains  
localhost  
owncloud.local  
example.com
```

Deleting a Single Configuration Value

These commands delete the configuration of an app or system configuration:

```
sudo -u www-data php occ config:system:delete maintenance:mode  
System config value maintenance:mode deleted  
  
sudo -u www-data php occ config:app:delete appname provisioning_api  
Config value provisioning_api of app appname deleted
```

The delete command will by default not complain if the configuration was not set before. If you want to be notified in that case, set the `--error-if-not-exists` flag.

```
sudo -u www-data php occ config:system:delete doesnotexist  
--error-if-not-exists  
Config provisioning_api of app appname could not be deleted because it did not  
exist
```

Dav Commands

A set of commands to create address books, calendars, and to migrate address books:

dav	
dav:cleanup-chunks	Cleanup outdated chunks
dav:create-addressbook	Create a dav address book
dav:create-calendar	Create a dav calendar
dav:sync-birthday-calendar	Synchronizes the birthday calendar
dav:sync-system-addressbook	Synchronizes users to the system address book

Note

These commands are not available in single-user (maintenance) mode.

`dav:cleanup-chunks` cleans up outdated chunks (uploaded files) more than a certain number of days old. By default, the command cleans up chunks more than 2 days old. However, by supplying the number of days to the command, the range can be increased. For example, in the example below, chunks older than 10 days will be removed.

```
sudo -u www-data php occ dav:cleanup-chunks 10  
  
# example output  
Cleaning chunks older than 10 days(2017-11-08T13:13:45+00:00)  
Cleaning chunks for admin  
0 [>-----]
```

The syntax for `dav:create-addressbook` and `dav:create-calendar` is `dav:create-addressbook [user] [name]`. This example creates the addressbook `mollybook` for the user `molly`:

```
sudo -u www-data php occ dav:create-addressbook molly mollybook
```

Configuration

This example creates a new calendar for molly:

```
sudo -u www-data php occ dav:create-calendar molly mollycal
```

Molly will immediately see these on her Calendar and Contacts pages. Your existing calendars and contacts should migrate automatically when you upgrade. If something goes wrong you can try a manual migration. First delete any partially-migrated calendars or address books. Then run this command to migrate user's contacts:

```
sudo -u www-data php occ dav:migrate-addressbooks [user]
```

Run this command to migrate calendars:

```
sudo -u www-data php occ dav:migrate-calendars [user]
```

dav:sync-birthday-calendar adds all birthdays to your calendar from address books shared with you. This example syncs to your calendar from user bernie:

```
sudo -u www-data php occ dav:sync-birthday-calendar bernie
```

dav:sync-system-addressbook synchronizes all users to the system addressbook.

```
sudo -u www-data php occ dav:sync-system-addressbook
```

Database Conversion

The SQLite database is good for testing, and for ownCloud servers with small single-user workloads that do not use sync clients, but production servers with multiple users should use MariaDB, MySQL, or PostgreSQL. You can use occ to convert from SQLite to one of these other databases.

db	
db:convert-type	Convert the ownCloud database to the newly configured one
db:generate-change-script	generates the change script from the current connected db to db_structure.xml

You need:

- Your desired database and its PHP connector installed.
- The login and password of a database admin user.
- The database port number, if it is a non-standard port.

This is example converts SQLite to MySQL/MariaDB:

```
sudo -u www-data php occ db:convert-type mysql oc_dbuser 127.0.0.1  
oc_database
```

For a more detailed explanation see Converting Database Type.

Encryption

occ includes a complete set of commands for managing encryption.

encryption	
encryption:change-key-storage-root	Change key storage root
encryption:decrypt-all	Disable server-side encryption and decrypt all files
encryption:disable	Disable encryption
encryption:enable	Enable encryption
encryption:encrypt-all	Encrypt all files for all users
encryption:list-modules	List all available encryption modules
encryption:migrate	initial migration to encryption 2.0
encryption:recreate-master-key	Replace existing master key with new one. Encrypt the f
encryption:select-encryption-type	newly created master key
encryption:set-default-module	Select the encryption type. The encryption types availa
	user-keys. There is also no way to disable it again.
	Set the encryption default module

encryption:show-key-storage-root	Show current key storage root
encryption:status	Lists the current status of encryption

encryption:status shows whether you have active encryption, and your default encryption module. To enable encryption you must first enable the Encryption app, and then run encryption:enable:

```
sudo -u www-data php occ app:enable encryption
sudo -u www-data php occ encryption:enable
sudo -u www-data php occ encryption:status
  - enabled: true
  - defaultModule: OC_DEFAULT_MODULE
```

encryption:change-key-storage-root is for moving your encryption keys to a different folder. It takes one argument, newRoot, which defines your new root folder. The folder must exist, and the path is relative to your root ownCloud directory.

```
sudo -u www-data php occ encryption:change-key-storage-root ../../etc/oc-keys
```

You can see the current location of your keys folder:

```
sudo -u www-data php occ encryption:show-key-storage-root
Current key storage root: default storage location (data/)
```

encryption:list-modules displays your available encryption modules. You will see a list of modules only if you have enabled the Encryption app. Use encryption:set-default-module [module name] to set your desired module.

encryption:encrypt-all encrypts all data files for all users. You must first put your ownCloud server into single-user mode to prevent any user activity until encryption is completed.

encryption:decrypt-all decrypts all user data files, or optionally a single user:

```
sudo -u www-data php occ encryption:decrypt freda
```

Users must have enabled recovery keys on their Personal pages. You must first put your ownCloud server into single-user mode to prevent any user activity until decryption is completed.

Use encryption:disable to disable your encryption module. You must first put your ownCloud server into single-user mode to prevent any user activity.

encryption:migrate migrates encryption keys after a major ownCloud version upgrade. You may optionally specify individual users in a space-delimited list. See Encryption Configuration to learn more.

encryption:recreate-master-key decrypts the ownCloud file system, replaces the existing master key with a new one, and encrypts the entire ownCloud file system with the new master key. Given the size of your ownCloud filesystem, this may take some time to complete. However, if your filesystem is quite small, then it will complete quite quickly. The -y switch can be supplied to automate acceptance of user input.

Federation Sync

Synchronize the address books of all federated ownCloud servers:

```
federation:sync-addressbooks Synchronizes address books of all
federated clouds
```

Servers connected with federation shares can share user address books, and auto-complete usernames in share dialogs. Use this command to synchronize federated servers:

```
sudo -u www-data php occ federation:sync-addressbooks
```

Note

This command is only available when the “Federation” app (federation) is enabled.

File Operations

occ has three commands for managing files in ownCloud:

`files`

`files:cleanup`

Deletes orphaned file cache entries.

`files:scan`

Rescans the filesystem.

`files:transfer-ownership`

All files and folders are moved to another user - outgoing shares are moved as well (incoming shares are no

Note

These commands are not available in single-user (maintenance) mode.

The `files:cleanup` command

`files:cleanup` tidies up the server's file cache by deleting all file entries that have no matching entries in the storage table.

The `files:scan` command

The `files:scan` command

- Scans for new files.
- Scans not fully scanned files.
- Repairs file cache holes.
- Updates the file cache.

File scans can be performed per-user, for a space-delimited list of users, and for all users.

```
sudo -u www-data php occ files:scan --help
```

Usage:

```
files:scan [options] [--] [<user_id>]...
```

Arguments:

<code>user_id</code>	will rescan all files of the given user(s)
----------------------	--

Options:

<code>--output[=OUTPUT]</code>	Output format (plain, json or json_pretty, default is plain) [defa
<code>-p, --path=PATH</code>	limit rescan to this path, eg. <code>--path="/alice/files/Music"</code> , the us
<code>-q, --quiet</code>	Do not output any message
<code>--all</code>	will rescan all files of all known users
<code>--repair</code>	will repair detached filecache entries (slow)
<code>--unscanned</code>	only scan files which are marked as not fully scanned
<code>-h, --help</code>	Display this help message
<code>-V, --version</code>	Display this application version
<code>--ansi</code>	Force ANSI output
<code>--no-ansi</code>	Disable ANSI output
<code>-n, --no-interaction</code>	Do not ask any interactive question
<code>--no-warnings</code>	Skip global warnings, show command output only
<code>-v vv vvv, --verbose</code>	Increase the verbosity of messages: 1 for normal output, 2 for mor

Note

If not using `--quiet`, statistics will be shown at the end of the scan.

The --path Option

When using the `--path` option, the path must be in one of the following formats:

```
"user_id/files/path"  
"user_id/files/mount_name"  
"user_id/files/mount_name/path"
```

For example:

```
--path="/alice/files/Music"
```

In the example above, the `user_id` `alice` is determined implicitly from the path component given.

Note

Mounts are only scannable at the point of origin. Scanning of shares including federated shares is not necessary on the receiver side and therefore not possible.

The `--path`, `--all` and `[user_id]` parameters are exclusive - only one must be specified.

The --repair Option

As noted above, repairs can be performed for individual users, groups of users, and for all users in an ownCloud installation. What's more, repair scans can be run even if no files are known to need repairing and if one or more files are known to be in need of repair. Two examples of when files need repairing are:

- If folders have the same entry twice in the web UI (known as a “*ghost folder*”), this can also lead to strange error messages in the desktop client.
- If entering a folder doesn't seem to lead into that folder.

The repair command needs to be run in single user mode. The following commands show how to enable single user mode, run a repair file scan, and then disable single user mode.

```
sudo -u www-data php occ maintenance:singleuser --on  
sudo -u www-data php occ files:scan --all --repair  
sudo -u www-data php occ maintenance:singleuser --off
```

Note

We strongly suggest that you backup the database before running this command.

The files:transfer-ownership command

You may transfer all files and shares from one user to another. This is useful before removing a user. For example, to move all files from `<source-user>` to `<destination-user>`, use the following command:

```
sudo -u www-data php occ files:transfer-ownership <source-user> <destination-user>
```

You can also move a limited set of files from `<source-user>` to `<destination-user>` by making use of the `--path` switch, as in the example below. In it, `folder/to/move`, and any file and folder inside it will be moved to `<destination-user>`.

```
sudo -u www-data php occ files:transfer-ownership --path="folder/to/move" <source-user> <des
```

When using this command, please keep in mind:

1. The directory provided to the `--path` switch **must** exist inside `data/<source-user>/files`.
2. The directory (and its contents) won't be moved as is between the users. It'll be moved inside the destination user's `files` directory, and placed in a directory which follows the format:

Configuration

transferred from <source-user> on <timestamp>. Using the example above, it will be stored under: data/<destination-user>/files/transferred from <source-user> on 20170426_124510/

3. Currently file versions can't be transferred. Only the latest version of moved files will appear in the destination user's account.

Files External

These commands replace the data/mount.json configuration file used in ownCloud releases before 9.0.

Commands for managing external storage:

files_external	
files_external:applicable	Manage applicable users and groups for a mount
files_external:backends	Show available authentication and storage backends
files_external:config	Manage backend configuration for a mount
files_external:create	Create a new mount configuration
files_external:delete	Delete an external mount
files_external:export	Export mount configurations
files_external:import	Import mount configurations
files_external:list	List configured mounts
files_external:option	Manage mount options for a mount
files_external:verify	Verify mount configuration

These commands replicate the functionality in the ownCloud Web GUI, plus two new features: files_external:export and files_external:import.

Use files_external:export to export all admin mounts to stdout, and files_external:export [user_id] to export the mounts of the specified ownCloud user.

Note

These commands are only available when the "External storage support" app (files_external) is enabled. It is not available in single-user (maintenance) mode.

Group Commands

The group commands provide a range of functionality for managing ownCloud groups. This includes creating and removing groups and managing group membership. Group names are case-sensitive, so "Finance" and "finance" are two different groups.

The full list of commands is:

group	
group:add	adds a group
group:add-member	add members to a group
group:delete	deletes the specified group
group:list	list groups
group:list-members	list group members
group:remove-member	remove member(s) from a group

Creating Groups

You can create a new group with the group:add command. The syntax is:

```
group:add groupname
```

This example adds a new group, called "Finance":

```
sudo -u www-data php occ group:add Finance  
Created group "Finance"
```

Listing Groups

Configuration

You can list the names of existing groups with the `group:list` command. The syntax is:

```
group:list [options] [<search-pattern>]
```

Groups containing the `<search-pattern>` string are listed. Matching is not case-sensitive. If you do not provide a `<search-pattern>` then all groups are listed.

This example lists groups containing the string `finance`:

```
sudo -u www-data php occ group:list finance
- All-Finance-Staff
- Finance
- Finance-Managers
```

The output can be formatted in JSON with the output option `json` or `json_pretty`:

```
sudo -u www-data php occ --output=json_pretty group:list finance
[
    "All-Finance-Staff",
    "Finance",
    "Finance-Managers"
]
```

Listing Group Members

You can list the user IDs of group members with the `group:list-members` command. The syntax is:

```
group:list-members [options] <group>
```

This example lists members of the `Finance` group:

```
sudo -u www-data php occ group:list-members Finance
- aaron: Aaron Smith
- julie: Julie Jones
```

The output can be formatted in JSON with the output option `json` or `json_pretty`:

```
sudo -u www-data php occ --output=json_pretty group:list-members Finance
{
    "aaron": "Aaron Smith",
    "julie": "Julie Jones"
}
```

Adding Members to Groups

You can add members to an existing group with the `group:add-member` command. Members must be existing users. The syntax is:

```
group:add-member [-m|--member [MEMBER]] <group>
```

This example adds members “`aaron`” and “`julie`” to group “`Finance`”:

```
sudo -u www-data php occ group:add-member --member aaron --member julie Finance
User "aaron" added to group "Finance"
User "julie" added to group "Finance"
```

You may attempt to add members that are already in the group, without error. This allows you to add members in a scripted way without needing to know if the user is already a member of the group. For example:

```
sudo -u www-data php occ group:add-member --member aaron --member julie --member fred Finance
User "aaron" is already a member of group "Finance"
User "julie" is already a member of group "Finance"
User "fred" added to group "Finance"
```

Removing Members from Groups

You can remove members from a group with the `group:remove-member` command. The syntax is:

Configuration

```
group:remove-member [ -m | --member [ MEMBER ] ] <group>
```

This example removes members “aaron” and “julie” from group “Finance”:

```
sudo -u www-data php occ group:remove-member --member aaron --member julie Finance
Member "aaron" removed from group "Finance"
Member "julie" removed from group "Finance"
```

You may attempt to remove members that have already been removed from the group, without error. This allows you to remove members in a scripted way without needing to know if the user is still a member of the group. For example:

```
sudo -u www-data php occ group:remove-member --member aaron --member fred Finance
Member "aaron" could not be found in group "Finance"
Member "fred" removed from group "Finance"
```

Deleting a Group

To delete a group, you use the `group:delete` command, as in the example below:

```
sudo -u www-data php occ group:delete Finance
```

Integrity Check

Apps which have an official tag MUST be code signed. Unsigned official apps won’t be installable anymore. Code signing is optional for all third-party applications.

<code>integrity</code>	
<code>integrity:check-app</code>	Check app integrity using a signature.
<code>integrity:check-core</code>	Check core integrity using a signature.
<code>integrity:sign-app</code>	Signs an app using a private key.
<code>integrity:sign-core</code>	Sign core using a private key

After creating your signing key, sign your app like this example:

```
sudo -u www-data php occ integrity:sign-app --privateKey=/Users/lukasreschke/contacts.key --
```

Verify your app:

```
sudo -u www-data php occ integrity:check-app --path=/path/to/app appname
```

When it returns nothing, your app is signed correctly. When it returns a message then there is an error. See [Code Signing](#) in the Developer manual for more detailed information.

`integrity:sign-core` is for ownCloud core developers only.

See [Code Signing](#) to learn more.

I10n, Create Javascript Translation Files for Apps

This command creates JavaScript and JSON translation files for ownCloud applications.

Note

The command does not update existing translations if the source translation file has been updated. It only creates translation files when none are present for a given language.

<code>l10n</code>	
<code>l10n:createjs</code>	Create Javascript translation files for a given app

The command takes two parameters; these are:

- `app`: the name of the application.
- `lang`: the output language of the translation files; more than one can be supplied.

To create the two translation files, the command reads translation data from a source PHP translation file.

A Working Example

In this example, we'll create Austrian German translations for the Gallery app.

Note

This example assumes that the ownCloud directory is `/var/www/owncloud` and that it uses ownCloud's standard apps directory, `app`.

First, create a source translation file in `/var/www/owncloud/apps/gallery/l10n`, called `de_AT.php`. In it, add the required translation strings, as in the following example. Refer to the developer documentation on [creating translation files](#), if you're not familiar with creating them.

```
<?php
// The source string is the key, the translated string is the value.
$TRANSLATIONS = [
    "Share" => "Freigeben"
];
$PLURAL_FORMS = "nplurals=2; plural=(n != 1);";
```

After that, run the following command to create the translation.

```
sudo -u www-data php occ l10n:createjs gallery de_AT
```

This will generate two translation files, `de_AT.js` and `de_AT.json`, in `/var/www/owncloud/apps/gallery/l10n`.

Create Translations in Multiple Languages

To create translations in multiple languages simultaneously, supply multiple languages to the command, as in the following example:

```
sudo -u www-data php occ l10n:createjs gallery de_AT de_DE hu_HU es fr
```

LDAP Commands

Note

These commands are only available when the “LDAP user and group backend” app (`user_ldap`) is enabled.

These LDAP commands appear only when you have enabled the LDAP app. Then you can run the following LDAP commands with `occ`:

<code>ldap</code>	
<code>ldap:check-user</code>	checks whether a user exists on LDAP.
<code>ldap:create-empty-config</code>	creates an empty LDAP configuration
<code>ldap:delete-config</code>	deletes an existing LDAP configuration
<code>ldap:search</code>	executes a user or group search
<code>ldap:set-config</code>	modifies an LDAP configuration
<code>ldap:show-config</code>	shows the LDAP configuration
<code>ldap:test-config</code>	tests an LDAP configuration
<code>ldap:update-group</code>	update the specified group membership information stored locally

Search for an LDAP user, using this syntax:

```
sudo -u www-data php occ ldap:search [--group] [--offset="..."] [--limit="..."] search
```

Configuration

Searches will match at the beginning of the attribute value only. This example searches for `givenNames` that start with "rob":

```
sudo -u www-data php occ ldap:search "rob"
```

This will find robbie, roberta, and robin. Broaden the search to find, for example, jeroboam with the asterisk wildcard:

```
sudo -u www-data php occ ldap:search "*rob"
```

User search attributes are set with `ldap:set-config` (below). For example, if your search attributes are `givenName` and `sn` you can find users by first name + last name very quickly. For example, you'll find Terri Hanson by searching for te ha. Trailing whitespace is ignored.

Check if an LDAP user exists. This works only if the ownCloud server is connected to an LDAP server.

```
sudo -u www-data php occ ldap:check-user robert
```

`ldap:check-user` will not run a check when it finds a disabled LDAP connection. This prevents users that exist on disabled LDAP connections from being marked as deleted. If you know for certain that the user you are searching for is not in one of the disabled connections, and exists on an active connection, use the `--force` option to force it to check all active LDAP connections.

```
sudo -u www-data php occ ldap:check-user --force robert
```

`ldap:create-empty-config` creates an empty LDAP configuration. The first one you create has no configID, like this example:

```
sudo -u www-data php occ ldap:create-empty-config
Created new configuration with configID ''
```

This is a holdover from the early days, when there was no option to create additional configurations. The second, and all subsequent, configurations that you create are automatically assigned IDs.

```
sudo -u www-data php occ ldap:create-empty-config
Created new configuration with configID 's01'
```

Then you can list and view your configurations:

```
sudo -u www-data php occ ldap:show-config
```

And view the configuration for a single configID:

```
sudo -u www-data php occ ldap:show-config s01
```

`ldap:delete-config [configID]` deletes an existing LDAP configuration.

```
sudo -u www-data php occ ldap:delete s01
Deleted configuration with configID 's01'
```

The `ldap:set-config` command is for manipulating configurations, like this example that sets search attributes:

```
sudo -u www-data php occ ldap:set-config s01 ldapAttributesForUserSearch
"cn;givenname;sn;displayname;mail"
```

The command takes the following format:

```
ldap:set-config <configID> <configKey> <configValue>
```

All of the available keys, along with default values for `configValue`, are listed in the table below.

Configuration	Setting
hasMemberOfFilterSupport	
hasPagedResultSupport	
homeFolderNamingRule	
lastJpegPhotoLookup	0

Configuration

ldapAgentName	<i>cn=admin,dc=owncloudqa,dc=com</i>
ldapAgentPassword	*
ldapAttributesForGroupSearch	
ldapAttributesForUserSearch	
ldapBackupHost	
ldapBackupPort	
ldapBase	<i>dc=owncloudqa,dc=com</i>
ldapBaseGroups	<i>dc=owncloudqa,dc=com</i>
ldapBaseUsers	<i>dc=owncloudqa,dc=com</i>
ldapCacheTTL	600
ldapConfigurationActive	1
ldapDynamicGroupMemberURL	
ldapEmailAttribute	
ldapExperiencedAdmin	0
ldapExpertUUIDGroupAttr	
ldapExpertUUIDUserAttr	
ldapExpertUsernameAttr	<i>ldapGroupDisplayName cn</i>
ldapGroupFilter	<i>ldapGroupFilterGroups</i>
ldapGroupFilterMode	0
ldapGroupFilterObjectclass	
ldapGroupMemberAssocAttr	<i>uniqueMember</i>
ldapHost	<i>ldap://host</i>
ldapIgnoreNamingRules	
ldapLoginFilter	<i>(&((objectclass=inetOrgPerson))(uid=%uid))</i>
ldapLoginFilterAttributes	
ldapLoginFilterEmail	0
ldapLoginFilterMode	0
ldapLoginFilterUsername	1
ldapNestedGroups	0
ldapOverrideMainServer	
ldapPagingSize	500
ldapPort	389
ldapQuotaAttribute	
ldapQuotaDefault	
ldapTLS	0
ldapUserDisplayName	<i>displayName</i>

ldapUserDisplayName2	
ldapUserFilter	((objectclass=inetOrgPerson))
ldapUserFilterGroups	
ldapUserFilterMode	0
ldapUserFilterObjectclas	inetOrgPerson
ldapUuidGroupAttribute	auto
ldapUuidUserAttribute	auto
turnOffCertCheck	0
useMemberOfToDetect Membership	1

ldap:test-config tests whether your configuration is correct and can bind to the server.

```
sudo -u www-data php occ ldap:test-config s01
The configuration is valid and the connection could be established!
```

ldap:update-group updates the specified group membership information stored locally.

The command takes the following format:

```
ldap:update-group <groupID> <groupID> <groupID> ...>
```

The command allows for running a manual group sync on one or more groups, instead of having to wait for group syncing to occur. If users have been added or removed from these groups in LDAP, ownCloud will update its details. If a group was deleted in LDAP, ownCloud will also delete the local mapping info about this group.

Note

New groups in LDAP won't be synced with this command. The LDAP TTL configuration (by default 10 minutes) still applies. This means that recently deleted groups from LDAP might be considered as "active" and might not be deleted in ownCloud immediately.

Configuring the LDAP Refresh Attribute Interval

You can configure the LDAP refresh attribute interval, but not with the ldap commands. Instead, you need to use the config:app:set command, as in the following example, which takes a number of seconds to the --value switch.

```
occ config:app:set user_ldap updateAttributesInterval --value=7200
```

In the example above, the interval is being set to 7200 seconds. Assuming the above example was used, the command would output the following:

```
Config value updateAttributesInterval for app user_ldap set to 7200
```

If you want to reset (or unset) the setting, then you can use the following command:

```
occ config:app:delete user_ldap updateAttributesInterval
```

Logging Commands

These commands view and configure your ownCloud logging preferences.

```
log
log:manage    manage logging configuration
log:owncloud  manipulate ownCloud logging backend
```

Run log:owncloud to see your current logging status:

```
sudo -u www-data php occ log:owncloud
Log backend ownCloud: enabled
Log file: /opt/owncloud/data/owncloud.log
Rotate at: disabled
```

Use the `--enable` option to turn on logging. Use `--file` to set a different log file path. Set your rotation by log file size in bytes with `--rotate-size`; 0 disables rotation. `log:manage` sets your logging backend, log level, and timezone. The defaults are `owncloud`, `Warning`, and `UTC`.

Log level can be adjusted by entering the number or the name:

```
sudo -u www-data php occ log:manage --level 4
sudo -u www-data php occ log:manage --level error
```

You can also choose `debug`, `info`, `warning`, `error` or `fatal`.

Note

Setting the log level to `debug` (0) can be used for finding the cause of an error, but should not be the standard as it increases the log file size.

Available options are:

- `--backend` [`owncloud`, `syslog`, `errorlog`]

Maintenance Commands

Use these commands when you upgrade ownCloud, manage encryption, perform backups and other tasks that require locking users out until you are finished:

<code>maintenance</code>	
<code>maintenance:data-fingerprint</code>	update the systems data-fingerprint after a backup is restored
<code>maintenance:mimetype:update-db</code>	Update database mimetypes and update filecache
<code>maintenance:mimetype:update-js</code>	Update mimetypelist.js
<code>maintenance:mode</code>	set maintenance mode
<code>maintenance:repair</code>	repair this installation
<code>maintenance:singleuser</code>	set single user mode
<code>maintenance:update:.htaccess</code>	Updates the .htaccess file

`maintenance:mode` locks the sessions of all logged-in users, including administrators, and displays a status screen warning that the server is in maintenance mode. Users who are not already logged in cannot log in until maintenance mode is turned off. When you take the server out of maintenance mode logged-in users must refresh their Web browsers to continue working.

```
sudo -u www-data php occ maintenance:mode --on
sudo -u www-data php occ maintenance:mode --off
```

Putting your ownCloud server into single-user mode allows admins to log in and work, but not ordinary users. This is useful for performing maintenance and troubleshooting on a running server.

```
sudo -u www-data php occ maintenance:singleuser --on
Single user mode enabled
```

Turn it off when you're finished:

```
sudo -u www-data php occ maintenance:singleuser --off
Single user mode disabled
```

Run `maintenance:data-fingerprint` to tell desktop and mobile clients that a server backup has been restored. Users will be prompted to resolve any conflicts between newer and older file versions.

Configuration

Run `maintenance:data-fingerprint` to tell desktop and mobile clients that a server backup has been restored. This command changes the ETag for all files in the communication with sync clients, informing them that one or more files were modified. After the command completes, users will be prompted to resolve any conflicts between newer and older file versions.

The `maintenance:repair` command runs automatically during upgrades to clean up the database, so while you can run it manually there usually isn't a need to.

```
sudo -u www-data php occ maintenance:repair
```

`maintenance:mimetype:update-db` updates the ownCloud database and file cache with changed mimetypes found in `config/mimetypemapping.json`. Run this command after modifying `config/mimetypemapping.json`. If you change a mimetype, run `maintenance:mimetype:update-db --repair-filecache` to apply the change to existing files.

Market

The market commands `install`, `list`, and `upgrade` applications from the ownCloud Marketplace.

market	
market:install	Install apps from the marketplace. If already installed and an update is available the update will be installed.
market:list	Lists apps as available on the marketplace.
market:upgrade	Installs new app versions if available on the marketplace

Note

The user running the update command, which will likely be your webserver user, needs write permission for the `/apps` folder. If they don't have write permission, the command may report that the update was successful, but it may silently fail.

Note

These commands are not available in single-user (maintenance) mode.

Install an Application

Applications can be installed both from the ownCloud Marketplace and from a local file archive.

Install Apps From The Marketplace

To install an application from the Marketplace, you need to supply the app's id, which can be found in the app's Marketplace URL. For example, the URL for Two factor backup codes is https://marketplace.owncloud.com/apps/twofactor_backup_codes. So its app id is `twofactor_backup_codes`.

Install Apps From a File Archive

To install an application from a local file archive, you need to supply the path to the archive, and that you pass the `-1` switch. Only `zip`, `gzip`, and `bzip2` archives are supported.

Usage Example

```
# Install an app from the marketplace.  
sudo -u www-data occ market:install twofactor_backup_codes
```

Configuration

```
# Install an app from a local archive.  
sudo -u www-data occ market:install -l /mnt/data/richdocuments-2.0.0.tar.gz
```

Reports

If you're working with ownCloud support and need to send them a configuration summary, you can generate it using the configreport:generate command. This command generates the same JSON-based report as the Admin Config Report, which you can access under admin -> Settings -> Admin -> Help & Tips -> Download ownCloud config report.

From the command-line in the root directory of your ownCloud installation, run it as your webserver user as follows, (assuming your webserver user is www-data):

```
sudo -u www-data occ configreport:generate
```

This will generate the report and send it to STDOUT. You can optionally pipe the output to a file and then attach it to an email to ownCloud support, by running the following command:

```
sudo -u www-data occ configreport:generate > generated-config-report.txt
```

Alternatively, you could generate the report and email it all in one command, by running:

```
sudo -u www-data occ configreport:generate | mail -s "configuration report" \  
-r <the email address to send from> \  
support@owncloud.com
```

Note

These commands are not available in single-user (maintenance) mode.

Security

Use these commands when you manage security related tasks

Routes displays all routes of ownCloud. You can use this information to grant strict access via firewalls, proxies or loadbalancers etc.

```
security:routes [options]
```

Options:

```
--output          Output format (plain, json or json-pretty, default is plain)  
--with-details   Adds more details to the output
```

Example 1:

```
sudo -uwww-data ./occ security:routes
```

Path	Methods
/apps/federation/auto-add-servers	POST
/apps/federation/trusted-servers	POST
/apps/federation/trusted-servers/{id}	DELETE
/apps/files/	GET
/apps/files/ajax/download.php	
...	

Example 2:

```
sudo -uwww-data ./occ security:routes --output=json-pretty
```

Configuration

```
[  
 {  
   "path": "\/apps\/federation\/auto-add-servers",  
   "methods": [  
     "POST"  
   ]  
 },  
 ...
```

Example 3:

```
sudo -u www-data ./occ security:routes --with-details
```

Path	Methods	Controller
/apps/files/api/v1/sorting	POST	OCA\Files\Controller\ApiController
/apps/files/api/v1/thumbnaill/{x}/{y}/{file}	GET	OCA\Files\Controller\ApiController
...		

The following commands manage server-wide SSL certificates. These are useful when you create federation shares with other ownCloud servers that use self-signed certificates.

```
security:certificates      list trusted certificates  
security:certificates:import import trusted certificate  
security:certificates:remove remove trusted certificate
```

This example lists your installed certificates:

```
sudo -u www-data php occ security:certificates
```

Import a new certificate:

```
sudo -u www-data php occ security:certificates:import /path/to/certificate
```

Remove a certificate:

```
sudo -u www-data php occ security:certificates:remove [certificate name]
```

Ransomware Protection

Use these commands to help users recover from a Ransomware attack. You can find more information about the application [in the documentation](#).

Note

Ransomware Protection (which is an Enterprise app) needs to be installed and enabled to be able to use these commands.

occ ransomguard:scan <timestamp> <user>	Report all changes in a user's account, starting timestamp.
occ ransomguard:restore <timestamp> <user>	Revert all operations in a user account after a p
occ ransomguard:lock <user>	Set a user account as read-only for ownCloud and clients when malicious activity is suspected.
occ ransomguard:unlock <user>	Unlock a user account after ransomware issues have resolved.

Sharing

Configuration

This is an occ command to cleanup orphaned remote storages. To explain why this is necessary, a little background is required. While shares are able to be deleted as a normal matter of course, remote storages with “shared::” are not included in this process.

This might not, normally, be a problem. However, if a user has re-shared a remote share which has been deleted it will. This is because when the original share is deleted, the remote re-share reference is not. Internally, the fileid will remain in the file cache and storage for that file will not be deleted.

As a result, any user(s) who the share was re-shared with will now get an error when trying to access that file or folder. That’s why the command is available.

So, to cleanup all orphaned remote storages, run it as follows:

```
sudo -u www-data php occ sharing:cleanup-remote-storages
```

You can also set it up to run as a background job

Note

These commands are not available in single-user (maintenance) mode.

Shibboleth Modes (Enterprise Edition only)

shibboleth:mode sets your Shibboleth mode to `notactive`, `autoprovision`, or `ssonly`:

```
shibboleth:mode [mode]
```

Note

These commands are only available when the “Shibboleth user backend” app (`user_shibboleth`) is enabled.

Trashbin

Note

These commands are only available when the “Deleted files” app (`files_trashbin`) is enabled. These commands are not available in single-user (maintenance) mode.

```
trashbin
trashbin:cleanup    Remove deleted files
trashbin:expire     Expires the users trash bin
```

The `trashbin:cleanup` command removes the deleted files of the specified users in a space-delimited list, or all users if none are specified. This example removes all the deleted files of all users:

```
sudo -u www-data php occ trashbin:cleanup
Remove all deleted files
Remove deleted files for users on backend Database
freda
molly
stash
rosa
edward
```

This example removes the deleted files of users “`molly`” and “`freda`”:

```
sudo -u www-data php occ trashbin:cleanup molly freda
Remove deleted files of    molly
Remove deleted files of    freda
```

trashbin:expire deletes only expired files according to the trashbin_retention_obligation setting in config.php (see the Deleted Files section in Config.php Parameters). The default is to delete expired files for all users, or you may list users in a space-delimited list.

User Commands

The user commands provide a range of functionality for managing ownCloud users. This includes: creating and removing users, resetting user passwords, displaying a report which shows how many users you have, and when a user was last logged in.

The full list of commands is:

user	
user:add	Adds a user
user:delete	Deletes the specified user
user:disable	Disables the specified user
user:enable	Enables the specified user
user:inactive	Reports users who are known to owncloud, but have not logged in for a certain number of days
user:lastseen	Shows when the user was logged in last time
user:list	List users
user:list-groups	List groups for a user
user:modify	Modify user details
user:report	Shows how many users have access
user:resetpassword	Resets the password of the named user
user:setting	Read and modify user application settings
user:sync	Sync local users with an external backend service

Creating Users

You can create a new user with the user:add command. This command lets you set the following attributes:

- **uid**: The uid is the user's username and their login name
- **display name**: This corresponds to the **Full Name** on the Users page in your ownCloud Web UI
- **email address**
- **group**
- **login name**
- **password**

The command's syntax is:

```
user:add [--password-from-env] [--display-name [DISPLAY-NAME]] [--email [EMAIL]] [-g --group
```

This example adds new user Layla Smith, and adds her to the **users** and **db-admins** groups. Any groups that do not exist are created.

```
sudo -u www-data php occ user:add --display-name="Layla Smith" \
--group="users" --group="db-admins" --email=layla.smith@example.com layla
Enter password:
Confirm password:
The user "layla" was created successfully
Display name set to "Layla Smith"
Email address set to "layla.smith@example.com"
User "layla" added to group "users"
User "layla" added to group "db-admins"
```

After the command completes, go to your Users page, and you will see your new user.

Setting a User's Password

`password-from-env` allows you to set the user's password from an environment variable. This prevents the password from being exposed to all users via the process list, and will only be visible in the history of the user (root) running the command. This also permits creating scripts for adding multiple new users.

To use `password-from-env` you must run as "real" root, rather than `sudo`, because `sudo` strips environment variables. This example adds new user Fred Jones:

```
export OC_PASS=newpassword
su -s /bin/sh www-data -c 'php occ user:add --password-from-env
--display-name="Fred Jones" --group="users" fred'
The user "fred" was created successfully
Display name set to "Fred Jones"
User "fred" added to group "users"
```

You can reset any user's password, including administrators (see Resetting a Lost Admin Password):

```
sudo -u www-data php occ user:resetpassword layla
Enter a new password:
Confirm the new password:
Successfully reset password for layla
```

You may also use `password-from-env` to reset passwords:

```
export OC_PASS=newpassword
su -s /bin/sh www-data -c 'php occ user:resetpassword --password-from-env
layla'
Successfully reset password for layla
```

Deleting A User

To delete a user, you use the `user:delete` command, as in the example below:

```
sudo -u www-data php occ user:delete fred
```

Listing Users

You can list existing users with the `user:list` command. The syntax is:

```
user:list [options] [<search-pattern>]
```

User IDs containing the `search-pattern` string are listed. Matching is not case-sensitive. If you do not provide a `search-pattern` then all users are listed.

This example lists user IDs containing the string ron:

```
sudo -u www-data php occ user:list ron
-aaron: Aaron Smith
```

The output can be formatted in JSON with the `output` option `json` or `json_pretty`:

```
sudo -u www-data php occ --output=json_pretty user:list
{
    "aaron": "Aaron Smith",
    "herbert": "Herbert Smith",
    "julie": "Julie Jones"
}
```

Listing Group Membership of a User

You can list the group membership of a user with the `user:list-groups` command. The syntax is:

```
user:list-groups [options] <uid>
```

This example lists group membership of user julie:

Configuration

```
sudo -u www-data php occ user:list-groups julie
- Executive
- Finance
```

The output can be formatted in JSON with the output option `json` or `json_pretty`:

```
sudo -u www-data php occ --output=json_pretty user:list-groups julie
[
    "Executive",
    "Finance"
]
```

Finding The User's Last Login

To view a user's most recent login, use the `user:lastseen` command, as in the example below:

```
sudo -u www-data php occ user:lastseen layla
layla's last login: 09.01.2015 18:46
```

User Application Settings

To manage application settings for a user, use the `user:setting` command. This command provides the ability to:

- Retrieve all settings for an application
- Retrieve a single setting
- Set a setting value
- Delete a setting

If you run the command and pass the help switch (`--help`), you will see the following output, in your terminal:

```
$ ./occ user:setting --help
Usage:
  user:setting [options] [--] <uid> [<app>] [<key>]

Arguments:
  uid                           User ID used to login
  app                            Restrict the settings to a given app [default: ""]
  key                           Setting key to set, get or delete [default: ""]
```

If you're new to the `user:setting` command, the descriptions for the `app` and `key` arguments may not be completely transparent. So, here's a lengthier description of both.

Argument	Description
app	When an value is supplied, <code>user:setting</code> limits the settings displayed, to those for that, specific, application — assuming that the application is installed, and that there are settings available for it. Some example applications are “core”, “files_trashbin”, and “user_ldap”. A complete list, unfortunately, cannot be supplied, as it is impossible to know the entire list of applications which a user could, potentially, install.
key	This value specifies the setting key to be manipulated (set, retrieved, or deleted) by the <code>user:setting</code> command.

Retrieving User Settings

To retrieve all settings for a user, you need to call the `user:setting` command and supply the user's username, as in the example below.

```
sudo -u www-data php occ user:setting layla
- core:
  - lang: en
- login:
  - lastLogin: 1465910968
```

Configuration

```
- settings:  
- email: layla@example.tld
```

Here, we see that the user has settings for the application core, when they last logged in, and what their email address is.

To retrieve the user's settings for a specific application, you have to supply the username and the application's name, which you want to retrieve the settings for; such as in the example below:

```
sudo -u www-data php occ user:setting layla core  
- core:  
- lang: en
```

In the output, you can see that one setting is in effect, lang, which is set to en. To retrieve the value of a single application for a user, use the user:setting command, as in the example below.

```
sudo -u www-data php occ user:setting layla core lang
```

This will display the value for that setting, such as en.

Setting a Setting

To set a setting, you need to supply four things; these are:

- the username
- the application (or setting category)
- the --value switch
- the, quoted, value for that setting

Here's an example of how you would set the email address of the user layla.

```
sudo -u www-data php occ user:setting layla settings email --value "new-layla@example.tld"
```

Deleting a Setting

Deleting a setting is quite similar to setting a setting. In this case, you supply the username, application (or setting category) and key as above. Then, in addition, you supply the --delete flag.

```
sudo -u www-data php occ user:setting layla settings email --delete
```

Modify user details

This command modifies either the users username or email address.

```
user:modify [options] [--] <uid> <key> <value>
```

Arguments:

uid	User ID used to login
key	Key to be changed. Valid keys are: displayname, email
value	The new value of the key

All three arguments are mandatory and can not be empty.

Example to set the email address:

```
sudo -u www-data php occ user:modify carla email foobar@foo.com
```

The email address of carla is updated to foobar@foo.com

Generating a User Count Report

Generate a simple report that counts all users, including users on external user authentication servers such as LDAP.

```
sudo -u www-data php occ user:report  
+-----+-----+
```

User Report		
Database	12	
LDAP	86	
total users	98	
user directories	2	

Syncing User Accounts

This command syncs users stored in external backend services, such as *LDAP*, *Shibboleth*, and *Samba*, with ownCloud's, internal, user database. However, it's not essential to run it regularly, unless you have a large number of users whose account properties have changed in a backend outside of ownCloud. When run, it will pick up changes from alternative user backends, such as LDAP where properties like `cn` or `display name` have changed, and sync them with ownCloud's user database. If accounts are found that no longer exist in the external backend, you are given the choice of either removing or disabling the accounts.

Note

It's also one of the commands that you should run on a regular basis to ensure that your ownCloud installation is running optimally.

Note

This command replaces the old `show-remnants` functionality, and brings the LDAP feature more in line with the rest of ownCloud's functionality.

Below are examples of how to use the command with an *LDAP*, *Samba*, and *Shibboleth* backend.

LDAP

```
sudo -u www-data ./occ user:sync "OCA\User_LDAP\User_Proxy"
```

Samba

```
sudo -u www-data ./occ user:sync "OCA\User\SMB" -vvv
```

Shibboleth

```
sudo -u www-data ./occ user:sync "OCA\User_Shibboleth\UserBackend"
```

Syncing via cron job

Here is an example for syncing with LDAP four times a day on Ubuntu:

```
crontab -e -u www-data
* * /6 * * * /usr/bin/php /var/www/owncloud/occ user:sync -vvv --missing-account-action="disa
```

Versions

```
versions
  versions:cleanup    Delete versions
  versions:expire     Expires the users file versions
```

versions:cleanup can delete all versioned files, as well as the files_versions folder, for either specific users, or for all users. The example below deletes all versioned files for all users:

```
sudo -u www-data php occ versions:cleanup
Delete all versions
Delete versions for users on backend Database
  freda
  molly
  stash
  rosa
  edward
```

You can delete versions for specific users in a space-delimited list:

```
sudo -u www-data php occ versions:cleanup freda molly
Delete versions of    freda
Delete versions of    molly
```

versions:expire deletes only expired files according to the versions_retention_obligation setting in config.php (see the File versions section in Config.php Parameters). The default is to delete expired files for all users, or you may list users in a space-delimited list.

Note

These commands are only available when the “Versions” app (files_versions) is enabled. These commands are not available in single-user (maintenance) mode.

Command Line Installation

ownCloud can be installed entirely from the command line. After downloading the tarball and copying ownCloud into the appropriate directories, or after installing ownCloud packages (See Linux Package Manager Installation and Manual Installation on Linux) you can use occ commands in place of running the graphical Installation Wizard.

Note

These instructions assume that you have a fully working and configured webserver. If not, please refer to the documentation on configuring the Apache web server for detailed instructions.

Apply correct permissions to your ownCloud directories; see Set Strong Directory Permissions. Then choose your occ options. This lists your available options:

```
sudo -u www-data php /var/www/owncloud/occ
ownCloud is not installed - only a limited number of commands are available
ownCloud version 9.0.0
```

Usage:
[options] command [arguments]

Options:
--help (-h) Display this help message
--quiet (-q) Do not output any message
--verbose (-v|vv|vvv) Increase the verbosity of messages: 1 for normal
output, 2 for more verbose output and 3 for debug
--version (-V) Display this application version

Configuration

```
--ansi           Force ANSI output
--no-ansi        Disable ANSI output
--no-interaction (-n) Do not ask any interactive question

Available commands:
check            check dependencies of the server environment
help             Displays help for a command
list              Lists commands
status            show some status information
app               app
app:check-code   check code to be compliant
l10n             l10n
l10n:createjs   Create javascript translation files for a given app
maintenance      maintenance
maintenance:install install ownCloud
```

Display your maintenance:install options:

```
sudo -u www-data php occ help maintenance:install
ownCloud is not installed - only a limited number of commands are available
Usage:
maintenance:install [--database="..."] [--database-name="..."]
[--database-host="..."] [--database-user="..."] [--database-pass[="..."]]
[--database-table-prefix[="..."]] [--admin-user="..."] [--admin-pass="..."]
[--data-dir="..."]

Options:
--database          Supported database type (default: "sqlite")
--database-name    Name of the database
--database-host    Hostname of the database (default: "localhost")
--database-user    User name to connect to the database
--database-pass    Password of the database user
--database-table-prefix Prefix for all tables (default: oc_)
--admin-user       User name of the admin account (default: "admin")
--admin-pass       Password of the admin account
--data-dir         Path to data directory (default:
                  "/var/www/owncloud/data")
--help (-h)         Display this help message
--quiet (-q)        Do not output any message
--verbose (-v|vv|vvv) Increase the verbosity of messages: 1 for normal
                      output, 2 for more verbose output and 3 for debug
--version (-V)      Display this application version
--ansi              Force ANSI output
--no-ansi           Disable ANSI output
--no-interaction (-n) Do not ask any interactive question
```

This example completes the installation:

```
cd /var/www/owncloud/
sudo -u www-data php occ maintenance:install --database
"mysql" --database-name "owncloud" --database-user "root" --database-pass
"password" --admin-user "admin" --admin-pass "password"
ownCloud is not installed - only a limited number of commands are available
ownCloud was successfully installed
```

Supported databases are:

- sqlite (SQLite3 - ownCloud Community edition only)
- mysql (MySQL/MariaDB)
- pgsql (PostgreSQL)
- oci (Oracle - ownCloud Enterprise edition only)

Command Line Upgrade

These commands are available only after you have downloaded upgraded packages or tar archives, and before you complete the upgrade. List all options, like this example on CentOS Linux:

```
sudo -u www-data php occ upgrade -h
Usage:
upgrade [options]

Options:
  --no-app-disable  skips the disable of third party apps
  -h, --help        Display this help message
  -q, --quiet       Do not output any message
  -V, --version     Display this application version
  --ansi           Force ANSI output
  --no-ansi         Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  --no-warnings    Skip global warnings, show command output only
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more v

Help:
run upgrade routines after installation of a new release. The release has to be installed b
```

When you are performing an update or upgrade on your ownCloud server (see the Maintenance section of this manual), it is better to use `occ` to perform the database upgrade step, rather than the Web GUI, in order to avoid timeouts. PHP scripts invoked from the Web interface are limited to 3600 seconds. In larger environments this may not be enough, leaving the system in an inconsistent state. After performing all the preliminary steps (see How to Upgrade Your ownCloud Server) use this command to upgrade your databases, like this example on CentOS Linux:

```
sudo -u www-data php occ upgrade
ownCloud or one of the apps require upgrade - only a limited number of
commands are available
Turned on maintenance mode
Checked database schema update
Checked database schema update for apps
Updated database
Updating <gallery> ...
Updated <gallery> to 0.6.1
Updating <activity> ...
Updated <activity> to 2.1.0
Update successful
Turned off maintenance mode
```

Note how it details the steps. Enabling verbosity displays timestamps:

```
sudo -u www-data php occ upgrade -v
ownCloud or one of the apps require upgrade - only a limited number of commands are available
2015-06-23T09:06:15+0000 Turned on maintenance mode
2015-06-23T09:06:15+0000 Checked database schema update
2015-06-23T09:06:15+0000 Checked database schema update for apps
2015-06-23T09:06:15+0000 Updated database
2015-06-23T09:06:15+0000 Updated <files_sharing> to 0.6.6
2015-06-23T09:06:15+0000 Update successful
2015-06-23T09:06:15+0000 Turned off maintenance mode
```

If there is an error it throws an exception, and the error is detailed in your ownCloud logfile, so you can use the log output to figure out what went wrong, or to use in a bug report.

```
Turned on maintenance mode
Checked database schema update
Checked database schema update for apps
Updated database
Updating <files_sharing> ...
Exception
```

```
ServerNotAvailableException: LDAP server is not available  
Update failed  
Turned off maintenance mode
```

Two-factor Authentication

If a two-factor provider app is enabled, it is enabled for all users by default (though the provider can decide whether or not the user has to pass the challenge). In the case of an user losing access to the second factor (e.g., a lost phone with two-factor SMS verification), the admin can temporarily disable the two-factor check for that user via the occ command:

```
sudo -u www-data php occ twofactor:disable <username>
```

To re-enable two-factor authentication again, use the following command:

```
sudo -u www-data php occ twofactor:enable <username>
```

Disable Users

Admins can disable users via the occ command too:

```
sudo -u www-data php occ user:disable <username>
```

Use the following command to enable the user again:

```
sudo -u www-data php occ user:enable <username>
```

Note

Once users are disabled, their connected browsers will be disconnected.

Finding Inactive Users

To view a list of users who've not logged in for a given number of days, use the `user:inactive` command. The example below searches for users inactive for five days, or more.

```
sudo -u www-data php occ user:inactive 5
```

By default, this will generate output in the following format:

```
- 0:  
- uid: admin  
- displayName: admin  
- inactiveSinceDays: 5
```

You can see the user's user id, display name, and the number of days they've been inactive. If you're passing or piping this information to another application for further processing, you can also use the `--output` switch to change its format. The switch supports three options, these are:

Setting	Description
plain	This is the default format.
json	This will render the output as a JSON-encoded, but not formatted, string.

```
[{"uid": "admin", "displayName": "admin", "inactiveSinceDays": 5}]
```

- `json_pretty`: This will render the output as a JSON-encoded string, formatted for ease of readability.

```
[  
  {  
    "uid": "admin",  
    "displayName": "admin",  
    "inactiveSinceDays": 5  
  }  
]
```

Configuring the Activity App

You can configure your ownCloud server to automatically send out e-mail notifications to your users for various events like:

- A file or folder has been shared
- A new file or folder has been created
- A file or folder has been changed
- A file or folder has been deleted

Users can see actions (delete, add, modify) that happen to files they have access to. Sharing actions are only visible to the sharer and recipient.

Enabling the Activity App

The Activity App is shipped and enabled by default. If it is not enabled simply go to your ownCloud Apps page to enable it.

Configuring your ownCloud for the Activity App

To configure your ownCloud to send out e-mail notifications a working Email Configuration is mandatory.

Furthermore it is recommended to configure the background job Webcron or Cron as described in Background Jobs.

There is also a configuration option `activity_expire_days` available in your `config.php` (See Config.php Parameters) which allows you to clean-up older activities from the database.

Virus Scanner Support

Overview

ClamAV is the only *officially* supported virus scanner available for use with ownCloud. It:

- Operates on all major operating systems, including *Windows*, *Linux*, and *Mac*
- Detects all forms of malware including *Trojan horses*, *viruses*, and *worms*
- Scans *compressed files*, *executables*, *image files*, *Flash*, *PDF*, as well as many others

What's more, ClamAV's *Freshclam* daemon automatically updates its malware signature database at scheduled intervals. However, other scanners can be used, so long as they:

1. Can receive data streams via pipe on the command-line and return an exit code
2. Return a parsable result on `stdout`

How ClamAV Works With ownCloud

Before you install and configure ClamAV, here is a bit of background which may be handy to know. ownCloud integrates with anti-virus tools by connecting to them via:

- A URL and port
- A socket
- Streaming the data from the command-line via a pipe with a configured executable

Configuration

In the case of ClamAV, ownCloud's Antivirus extension sends files as streams to a ClamAV service (which can be on the same ownCloud server or another server within the same network) which in turn scans them and returns a result to stdout.

Note

Individual chunks are **not** scanned. The whole file is scanned when it is moved to the final location.

The information is then parsed or an exit code is evaluated if no result is available to determine the response from the scan. Based on ownCloud's evaluation of the response (or exit code) an appropriate action is then taken, such as recording a log message or deleting the file.

Note

Scanner exit status rules are used to handle errors when ClamAV is run in CLI mode. Scanner output rules are used in daemon/socket mode.

Things To Note

1. Files are checked when they are uploaded or updated (whether because they were edited or saved) but *not* when they are downloaded.
2. ownCloud doesn't support a cache of previously scanned files.
3. If the app is either not configured or is misconfigured, then it rejects file uploads.
4. If ClamAV is unavailable, then the app rejects file uploads.
5. A file size limit applies both to background jobs and to file uploads.

Configuring the ClamAV Antivirus Scanner

You can configure your ownCloud server to automatically run a virus scan on newly-uploaded files using the [Antivirus App for Files](#).

Note

ClamAV must be installed before installing and configuring Antivirus App for Files.

Installing ClamAV

As always, Linux distributions install and configure ClamAV in different ways. Below you can find the instructions for installing it on Debian or Red Hat-based distributions.

Debian, Ubuntu, Linux Mint

On Debian, Ubuntu, and their many variants, install ClamAV with the following command:

```
sudo apt-get install clamav clamav-daemon
```

This automatically creates the default configuration files and launches the `clamd` and `freshclam` daemons. You shouldn't have to do anything else, though it is a good idea to review the ClamAV documentation, as well ClamAV's settings in `/etc/clamav/`.

Red Hat 7 and CentOS 7

Configuration

On Red Hat 7 and related systems, you must install the “Extra Packages for Enterprise Linux (EPEL)” repository, and then install ClamAV. To do so, run the following commands:

```
yum install epel-release  
yum install clamav clamav-scanner clamav-scanner-systemd clamav-server  
clamav-server-systemd clamav-update
```

Note

Regardless of your operating system, we recommend that you enable verbose logging in both `clamd.conf` and `freshclam.conf` until you get any kinks with your ClamAV installation worked out.

Configuring and Running ClamAV

After installing ClamAV and the related tools, you will now have two configuration files: `/etc/freshclam.conf` and `/etc/clamd.d/scan.conf`. You must edit both of these before you can run ClamAV. Both files are well commented. Running either `man clamd.conf` or `man freshclam.conf` will provide detailed information on all the available configuration options.

Note

Refer to `/etc/passwd` and `/etc/group` when you need to verify the ClamAV user and group.

When you’re finished editing the configuration files, you must enable the `clamd` service file and start `clamd`. You can do so using the following commands:

```
systemctl enable clamav-daemon.service  
systemctl start clamav-daemon.service
```

That should take care of everything.

Note

Enable verbose logging in `scan.conf` and `freshclam.conf` until it is running the way you want.

Automating ClamAV Virus Database Updates

To update your malware database and get the latest malware signatures, you need to run `freshclam` frequently. Do this by running `freshclam` or `sudo freshclam` on Debian-based distributions.

We recommend you do this, post-installation, to download your first set of malware signatures. If you want to adjust `freshclam`’s behavior, edit `/etc/clamav/freshclam.conf` and make any changes you believe are necessary.

After that, create a [cron job](#) to automate the process. For example, to run it every hour at 47 minutes past the hour, add the following in the applicable user’s crontab:

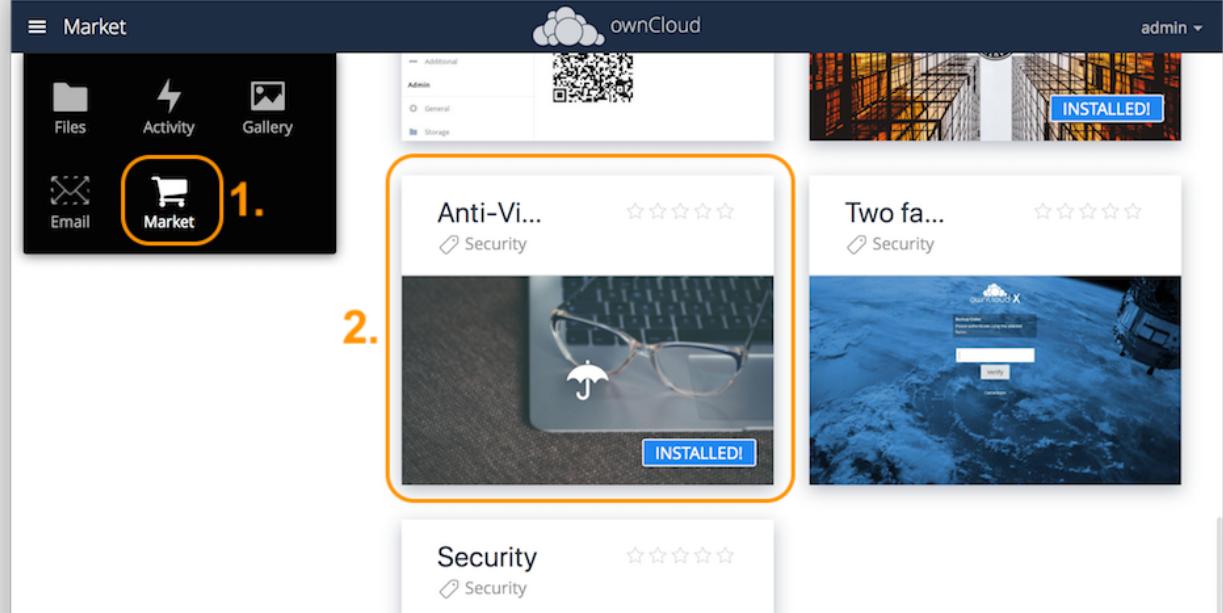
```
# m   h   dom mon dow   command  
47   *   *   *   *   /usr/bin/freshclam --quiet
```

Note

Please avoid any multiples of 10, because those are when the ClamAV servers are hit the hardest for updates.

Install the Anti-Virus App

The Anti-Virus app needs to be installed from the ownCloud Market, under “Security”. You can access the ownCloud Market via the App Menu (or App Switcher).



Configuring ClamAV within ownCloud

Once it is installed, go to your ownCloud Admin page and set your ownCloud logging level to Everything.



Now, navigate to Settings -> Admin -> Security, where you'll find the “Antivirus Configuration” panel. There, as below, you'll see the configuration options which ownCloud will pass to ClamAV.

Antivirus Configuration

Mode	Executable
Stream Length	26214400 bytes
Path to clamscan	/usr/bin/clamscan
Extra command line options (comma-separated)	
File size limit, -1 means no limit	-1 bytes
When infected files were found during a background scan	Only log

Save

Configuration Warnings

The Antivirus App for Files will show one of three warnings if it is either misconfigured, or ClamAV is not available. You can see an example of all three below.

Configuration

Antivirus app is misconfigured or antivirus inaccessible. Could not connect to host "localhost" on port 999 X

Antivirus app is misconfigured or antivirus inaccessible. The antivirus executable could not be found at path "/usr/bin/clamfscan" X

Antivirus app is misconfigured or antivirus inaccessible. Could not connect to socket "/var/run/clamav/cslamd-socket": No such file or directory (code 2) X

Mode Configuration

ClamAV runs in one of three modes: [Daemon \(Socket\)](#), [Daemon](#), and [Executable](#).

Daemon (Socket)

In this mode, ClamAV runs in the background on the same server as the ownCloud installation. When there is no activity clamd places a minimal load on your system. However, if your users upload large volumes of files, you will see high CPU usage. Please keep this in mind.

ownCloud should detect your clamd socket and fill in the `Socket` field. This is the `LocalSocket` option in `clamd.conf`. You can run `netstat` to verify:

```
netstat -a|grep clam  
unix 2 [ ACC ] STREAM LISTENING 15857 /var/run/clamav/clamd.ctl
```

Antivirus Configuration

Mode	Daemon (Socket) ▾
Socket	/var/run/clamav/clamd
Stream Length	26214400 bytes
File size limit, -1 means no limit	-1 bytes
When infected files were found during a background scan	
<input checked="" type="checkbox"/> Only log	
<input type="checkbox"/> Delete file	

Save

The Stream Length value sets the number of bytes to read in one pass. 10485760 bytes, or ten megabytes, is the default. This value should be no larger than the PHP `memory_limit` settings or physical memory if `memory_limit` is set to -1 (no limit).

Action for infected files found while scanning gives you the choice of logging any alerts without deleting the files or immediately deleting infected files.

Daemon

In this mode, ClamAV runs on a different server. This is a good option for ownCloud servers with high volumes of file uploads. For the Daemon option, you need the hostname or IP address of the remote server running ClamAV and the server's port number.

Antivirus Configuration

Mode: Daemon (Socket) ▾

Socket: /var/run/clamav/clamd

Stream Length: 26214400 bytes

File size limit, -1 means no limit: -1 bytes

When infected files were found during a background scan: Only log Delete file

Save

Executable

In this mode, ClamAV runs on the same server as the ownCloud installation, and the `clamscan` command only runs when a file is uploaded. `clamscan` is slow and not always reliable for on-demand usage; it is better to use one of the daemon modes.

This option requires the path to `clamscan`, which is the interactive ClamAV scanning command. ownCloud should find it automatically.

Antivirus Configuration

Mode: Executable ▾

Stream Length: 26214400 bytes

Path to clamscan: /usr/bin/clamscan

Extra command line options (comma-separated):

File size limit, -1 means no limit: -1 bytes

When infected files were found during a background scan: Only log ▾

Save

When you are satisfied with how ClamAV is operating, you might want to go back and change all of your logging to less verbose levels.

Rule Configuration

ownCloud provides the ability to customize how it reacts to the response given by an anti-virus scan. To do so, under *Admin -> Antivirus Configuration -> Advanced*, which you can see in the screenshot below, you can view and change the existing rules. You can also add new ones.

The exit status code returned by ClamAV to match against

Whether to match on the exit status or scanner output

Scanner exit status: 0, 1, 40, 50

A regular expression to match against the search response text returned by ClamAV

Scanner output: ^\+OK, ^\+FOUND, ^\+ERRNO

A human-readable description of what the status code means

Description: Unknown option passed, database initialization error.

The action to take when that status is returned

Mark as: Clean, Infected, Unchecked

Scanned successfully, FOUND virus, Error scanning file

Delete an existing rule

Confirm the new rule or existing rule changes

Configuration

Rules can match on either an exit status (e.g., 0, 1, or 40) or a pattern in the string returned from ClamAV (e.g., /.*: (.*) FOUND\$/).

Here are some points to bear in mind about rules:

- Scanner exit status rules are used to handle errors when ClamAV is run in CLI mode while
- scanner output rules are used in daemon/socket mode.
- Daemon output is parsed by regexp.
- In case there are no matching rules, the status is: Unknown, and a warning will be logged.

Default Ruleset

The default rule set for ClamAV is populated automatically with the following rules:

Exit Status or Signature	Description	Marks File As
0		Clean
1		Infected
40	Unknown option passed	Unchecked
50	Database initialization error	Unchecked
52	Not supported file type	Unchecked
53	Can't open directory	Unchecked
54	Can't open file	Unchecked
55	Error reading file	Unchecked
56	Can't stat input file	Unchecked
57	Can't get absolute path name of current working directory	Unchecked
58	I/O error	Unchecked
62	Can't initialize logger	Unchecked
63	Can't create temporary files/directories	Unchecked
64	Can't write to temporary directory	Unchecked
70	Can't allocate memory (calloc)	Unchecked
71	Can't allocate memory (malloc)	Unchecked
/.*: OK\$/		Clean
/.*: (.*) FOUND\$/		Infected
/.*: (.*) ERROR\$/		Unchecked

The rules are always checked in the following order:

1. Infected
2. Error
3. Clean

In case there are no matching rules, the status would be Unknown and a warning would be logged.

Update An Existing Rule

To match on an exit status, change the “Match by” dropdown list to “Scanner exit status” and in the “Scanner exit status or signature to search” field, add the status code to match on.

To match on the scanner’s output, change the “Match by” dropdown list to “Scanner output” and in the “Scanner exit status or signature to search” field, add the regular expression to match against the scanner’s output.

Then, while not mandatory, add a description of what the status or scan output means. After that, set what ownCloud should do when the exit status or regular expression you set matches the value returned by ClamAV. To do so change the value of the dropdown in the “**Mark as**” column.

The dropdown supports the following three options:

Option	Description
Clean	The file is clean, and contains no viruses
Infected	The file contains a virus
Unchecked	No action should be taken

With all these changes made, click the check mark on the lefthand side of the “**Match by**” column, to confirm the change to the rule.

Add A New Rule

To add a new rule, click the button marked “Add a rule” at the bottom left of the rules table. Then follow the process outlined in Update An Existing Rule.

Delete An Existing Rule

To delete an existing rule, click the rubbish bin icon on the far right-hand side of the rule that you want to delete.

Memory Caching

You can significantly improve ownCloud server performance by using memory caching. This is the process of storing frequently-requested objects in-memory for faster retrieval later. There are two types of memory caching available:

A PHP opcode Cache (OPcache): An opcode cache stores compiled PHP scripts so they don’t need to be re-compiled every time they are called. These compiled PHP scripts are stored in-memory, on the server on which they’re compiled.

A Data Cache: A data cache stores copies of *data*, *templates*, and other types of *information-based files*. Depending on the cache implementation, it can be either *local*, or specific, to one server, or *distributed* across multiple servers. This cache type is ideal when you have a scale-out installation.

Supported Caching Backends

The caching backends supported by ownCloud are:

- APCu: This is a local cache for systems running PHP 5.6 and up. APCu 4.0.6 and up is required. Alternatively you can use *the Zend OPCache*. However, **it is not a data cache**, only an opcode cache.
- Redis: This is a distributed cache for multi-server ownCloud installations. Version 2.2.6 or higher of the PHP Redis extension is required.
- Memcached: This is a distributed cache for multi-server ownCloud installations.

Note

You may use *both* a local and a distributed cache. The recommended ownCloud caches are APCu and Redis.

Note

If you do not install and enable a local memory cache you will see a warning on your ownCloud admin page. If you enable only a distributed cache in your config.php (`memcache.distributed`) and not a local cache (`memcache.local`) you will still see the cache warning.

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 Config.php Parameters) to select a different location.

Cache Types

APCu

PHP 5.6 and up include the Zend OPcache in core, and on most Linux distributions it is enabled by default. However, it *does not* bundle a data cache. Given that, we recommend that you use APCu instead. APCu is a data cache *and* is available in most Linux distributions.

Installing APCu

```
# On RedHat/CentOS/Fedora systems running PHP 5.6
yum install rh-php56-php-devel
pecl install apcu

# On RedHat/CentOS/Fedora systems running PHP 7.0
yum install rh-php70-php-devel
pecl install apcu

# On Debian/Ubuntu/Mint systems
apt-get install php-apcu
```

Note

On Ubuntu 14.04 LTS, the APCu version is 4.0.2. This is too old to use with ownCloud, which requires ownCloud 4.0.6+. You can install 4.0.7 from Ubuntu backports with the following command:

```
apt-get install php5-apcu/trusty-backports
```

After APCu's installed, enable the extension by creating a configuration file for it, using the following commands.

```
cat << EOF > /etc/opt/rh/rh-php70/php.d/20-apcu.ini
; APCu php extension
extension=apcu.so
EOF
```

With that done, assuming that you don't encounter any errors, restart Apache and the extension is ready to use.

Redis

Redis is an excellent modern memory cache to use for both distributed caching and as a local cache for Transactional File Locking, because it guarantees that cached objects are available for as long as they are needed.

The Redis PHP module must be at least version 2.2.6 or higher. 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 Installing Redis on other distributions.

Note

Debian Jessie users, please see this [GitHub discussion](#) if you have problems with LDAP authentication when using Redis.

Installing Redis on Debian-based Distributions

On Debian/Ubuntu/Mint run the following command:

```
apt-get install redis-server php5-redis
```

The installer will automatically launch Redis and configure it to launch at startup.

Note

If you're running ownCloud on Ubuntu 14.04, which does not package the required version of `php5-redis`, then work through [this guide on Tech and Me](#) to see how to install and configure it.

Installing Redis on RedHat, CentOS, and Fedora

On RedHat, CentOS, and Fedora run the following commands to install Redis:

```
yum install rh-php70-php-devel rh-redis32-redis  
pecl install redis
```

Unlike on Debian-based distributions, Redis will not start automatically on *RedHat*, *Centos*, and *Fedora*. Given that, you must use your service manager to both start Redis, and to launch it at boot time as a daemon. To do so, run the following commands:

```
systemctl start rh-redis32-redis  
systemctl enable rh-redis32-redis
```

You can verify that the Redis daemon is running using either of the following two commands:

```
ps ax | grep redis  
netstat -tlnp | grep redis
```

When it's running, enable the Redis extension by creating a configuration file for it, using the following commands.

```
cat << EOF > /etc/opt/rh/rh-php70/php.d/20-redis.ini  
; Redis php extension  
extension=redis.so  
EOF
```

After that, assuming that you don't encounter any errors, restart Apache and the extension is ready to use.

Additional notes for Redis vs. APCu on Memory Caching

APCu is faster at 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.

Installing Redis on other distributions

These instructions are adaptable for any distribution that does not package the supported version, or that does not package Redis at all, such as SUSE Linux Enterprise Server and RedHat Enterprise Linux.

Note

The [Redis PHP module](#) must be at least version 2.2.6.

On Debian/Mint/Ubuntu

Configuration

Use `apt-cache` to see the available `php5-redis` version, or the version of your installed package:

```
apt-cache policy php5-redis
```

On CentOS and Fedora

The `yum` command shows available and installed version information:

```
yum search php-pecl-redis
```

Clearing the Redis Cache

The Redis cache can be flushed from the command-line using [the redis-cli tool](#), as in the following example:

```
sudo redis-cli
SELECT <dbIndex>
FLUSHDB
```

`<dbIndex>` is the number of Redis database where the cache is stored. It is zero by default at ownCloud. To check what yours is currently set to, check the `dbindex` value in `config/config.php`. Here's an example of what to look for:

```
'redis': {
    'host' => 'localhost', // Can also be a unix domain socket => '/tmp/redis.sock'
    'port' => 6379,
    'timeout' => 0,
    'password' => '', // Optional, if not defined no password will be used.
    'dbindex' => 0 // Optional, if undefined SELECT will not run and will
                    // use Redis Server's default DB Index.
},
```

Further Reading

- <https://redis.io/commands/select>
- <https://redis.io/commands/flushdb>

Memcached

Memcached is a reliable old-timer for shared caching on distributed servers. It performs well with ownCloud with one exception: it is not suitable to use with Transactional File Locking. This is because it does not store locks, and data can disappear from the cache at any time. Given that, Redis is the best memory cache to use.

Note

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

Installing Memcached

On Debian/Ubuntu/Mint

On Debian/Ubuntu/Mint run the following command:

```
apt-get install memcached php5-memcached
```

Note

The installer will automatically start memcached and configure it to launch at startup.

On RedHat/CentOS/Fedora

On RedHat/CentOS/Fedora run the following command:

```
yum install memcached php-pecl-memcache
```

It will not start Memcached automatically after the installation or on subsequent reboots as a daemon, so you must do so yourself . To do so, run the following command:

```
systemctl start memcached  
systemctl enable memcached
```

You can verify that the Memcached daemon is running using one of the following commands:

```
ps ax | grep memcached  
netstat -tlnp | grep memcached
```

With the extension installed, you now need to configure it, by creating a configuration file for it. You can do so using the command below, substituting FILE_PATH with one from the list below the command.

```
cat << EOF > FILE_PATH  
; Memcached PHP extension  
extension=memcached.so  
EOF
```

Configuration File Paths

PHP Version	Filename
5.6	/etc/opt/rh/rh-php56/php.d/25-memcached.ini
7.0	/etc/opt/rh/rh-php70/php.d/25-memcached.ini

After that, assuming that you don't encounter any errors:

1. Restart your Web server
2. Add the appropriate entries to config.php (which you can find an example of below)
3. Refresh your ownCloud admin page

Clearing the Memcached Cache

The Memcached cache can be flushed from the command-line using a range of common Linux/UNIX tools, including netcat and telnet. The following example uses telnet to login, run the flush_all command, and logout:

```
telnet localhost 11211  
flush_all  
quit
```

For more information see:

- <https://github.com/memcached/memcached/wiki/Commands#flushall>

Configuring Memory Caching

Memory caches must be explicitly configured in ownCloud by:

1. Installing and enabling your desired cache (whether that be the PHP extension and/or the caching server).

Configuration

2. Adding the appropriate entry to ownCloud's config.php.

See Config.php Parameters for an overview of all possible config parameters. After installing and enabling your chosen memory cache, verify that it is active by running PHP Version and Information.

APCu Configuration

To use APCu, add this line to config.php:

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

With that done, refresh your ownCloud admin page, and the cache warning should disappear.

Redis Configuration

This example config.php configuration uses Redis for the local server cache:

```
'memcache.local' => '\OC\Memcache\Redis',
'redis' => [
    'host' => 'localhost',
    'port' => 6379,
],
'memcache.locking' => '\OC\Memcache\Redis', // Add this for best performance
```

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 ownCloud, use this example configuration:

```
'memcache.local' => '\OC\Memcache\Redis',
'redis' => [
    'host' => '/var/run/redis/redis.sock',
    'port' => 0,
],
```

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

Memcached Configuration

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

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

Configuration Recommendations Based on Type of Deployment

Small/Private Home Server

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

Small Organization, Single-server Setup

Use APCu for local caching, Redis for file locking

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

```
'port' => 6379,
],
```

Large Organization, Clustered Setup

Use Redis for everything except a local memory cache. Use the server's IP address or hostname so that it is accessible to other hosts:

```
'memcache.distributed' => '\OC\Memcache\Redis',
'memcache.locking' => '\OC\Memcache\Redis',
'memcache.local' => '\OC\Memcache\APCu',
'redis' => [
    'host' => 'server1',           // hostname example
    'host' => '12.34.56.78',      // IP address example
    'port' => 6379,
],
]
```

Configuring Transactional File Locking

Transactional File Locking prevents simultaneous file saving. To use it, you have to enable it in config.php as in the following example, which uses Redis as the cache backend:

```
'filelocking.enabled' => true,
'memcache.locking' => '\OC\Memcache\Redis',
'redis' => [
    'host' => 'localhost',
    'port' => 6379,
    'timeout' => 0.0,
    'password' => '', // Optional, if not defined no password will be used.
],
]
```

Note

For enhanced security it is recommended to configure Redis to require a password. See <http://redis.io/topics/security> for more information.

Caching Exceptions

If ownCloud is configured to use either Memcached or Redis as a memory cache, please be aware that you may encounter issues with functionality. When these occur, it is usually a result of PHP being incorrectly configured, or the relevant PHP extension not being available.

In the table below, you can see all of the known reasons for reduced or broken functionality related to caching.

Setup/Configuration	Result
If file locking is enabled, but the locking cache class is missing, then an exception will appear in the web UI	The application will not be usable
If file locking is enabled and the locking cache is configured, but the PHP module missing.	There will be a white page/exception in web UI. It will be a full page issue, and the application will not be usable
All enabled, but the Redis server is not running	The application will be usable. But any file operation will return a “ 500 Redis went away ” exception
If Memcache is configured for “local” and “distributed”, but the class is missing	There will be a white page and an exception written to the logs, This is because autoloading needs the missing class. So there is no way to show a page

Background Jobs

A system like ownCloud sometimes requires tasks to be done on a regular basis without requiring user interaction or hindering ownCloud's performance. For that reason, as a system administrator, you can configure background jobs (for example, database clean-ups) to be executed without any user interaction.

These jobs are typically referred to as [Cron Jobs](#). Cron jobs are commands or shell-based scripts that are scheduled to periodically run at fixed times, dates, or intervals. `cron.php` is an ownCloud internal process that runs such background jobs on demand.

ownCloud plug-in applications can register actions with `cron.php` automatically to take care of typical housekeeping operations. These actions can include garbage collecting of temporary files or checking for newly updated files using `filescan()` on externally mounted file systems.

You can decide how often jobs get processed, we recommend an interval of one minute.

Cron Jobs

You can schedule Cron jobs in three ways: [Cron](#), [Webcron](#), or [AJAX](#). These can all be configured in the admin settings menu. However, the recommended method is to use Cron. The following sections describe the differences between each method.

There are a number of things to keep in mind when choosing an automation option:

Firstly, while the default method is AJAX, though the preferred way is to use Cron. The reason for this distinction is that AJAX is easier to get up and running. As a result, it makes sense (often times) to accept it in the interests of expediency.

However, doing so is known to cause issues, such as backlogs and potentially not running every job on a heavily-loaded system. What's more, an increasing amount of ownCloud automation has been migrated from Ajax to Cron in recent versions. For this reason, we encourage you to not use it for too long — especially if your site is rapidly growing.

Secondly, while Webcron is better than Ajax, it too has limitations. For example, running Webcron will only remove a single item from the job queue, not all of them. Cron, however, will clear the entire queue.

Note

It's for this reason that we encourage you to use Cron — if at all possible.

Cron

Using the operating system Cron feature is the preferred method for executing regular tasks. This method enables the execution of scheduled jobs without the inherent limitations which the web server might have.

For example, to run a Cron job on a *nix system every minute, under the default web server user (often, `www-data` or `wwwrun`) you must set up the following Cron job to call the `cron.php` script:

```
# crontab -u www-data -e
* * * * * /usr/bin/php -f /path/to/your/owncloud/cron.php
```

You can verify if the cron job has been added and scheduled by executing:

```
# crontab -u www-data -l
* * * * * /usr/bin/php -f /path/to/your/owncloud/cron.php
```

Note

You have to make sure that `php` is found by `cron`, hence why we've deliberately added the full path to the PHP binary above (`/usr/bin/php`). On some systems it might be necessary to use `php-cli` instead of `php`.

Please refer to [the crontab man page](#) for the exact command syntax if you don't want to have it run every minute.

Note

There are other methods to invoke programs by the system regularly, e.g. [systemd timers](#)

Webcron

By registering your ownCloud cron.php script address as an external webcron service (for example, easyCron), you ensure that background jobs are executed regularly. To use this type of service, your external webcron service must be able to access your ownCloud server using the Internet. For example:

URL to call: `http[s]://<domain-of-your-server>/owncloud/cron.php`

AJAX

The AJAX scheduling method is the default option. However, it is also the *least* reliable. Each time a user visits the ownCloud page, a single background job is executed. The advantage of this mechanism, however, is that it does not require access to the system nor registration with a third party service. The disadvantage of this mechanism, when compared to the [Webcron](#) service, is that it requires regular visits to the page for it to be triggered.

Note

Especially when using the Activity App or external storages, where new files are added, updated, or deleted one of the other methods should be used.

Parallel Task Execution

Regardless of the approach which you take, since ownCloud 9.1, Cron jobs can be run in parallel. This is done by running cron.php multiple times. Depending on the process which you're automating, this may not be necessary. However, for longer-running tasks, such as those which are LDAP related, it may be very beneficial.

There is no way to do so via the ownCloud UI. But, the most direct way to do so, is by opening three console tabs and in each one run `php cron.php`. Each of these processes would acquire their own list of jobs to process without overlapping any other.

Available Background Jobs

A number of existing background jobs are available to be run just for specific tasks.

Note

These jobs are generally only needed on large instances and can be run as background jobs. If the number of users in your installation ranges between 1,000 and 3,000, or if you're using LDAP and it becomes a bottleneck, then admins can delete several entries in the `oc_jobs` table and replace them with the corresponding `occ` command, which you can see here:

- OCA\\DAV\\Command\\CleanupChunks -> `occ dav:cleanup-chunks`
- OCA\\DAV\\CardDAV\\SyncJob -> `occ dav:sync-system-addressbook`
- OCA\\Federation\\SyncJob -> `occ federation:sync-addressbooks`
- OCA\\Files_Trashbin\\BackgroundJob\\ExpireTrash -> `occ trashbin:expire`
- OCA\\Files_Versions\\BackgroundJob\\ExpireVersions -> `occ versions:expire`

If used, these should be scheduled to run on a daily basis.

While not exhaustive, these include:

CleanupChunks

The CleanupChunks job, contained in `OCA\DAV\Command\CleanupChunks`, will clean up outdated chunks (uploaded files) more than a certain number of days old. By default, the command cleans up chunks more than two days old. However, the command also accepts the number of days as an argument. It can be run, as follows, using the OCC command

```
occ dav:cleanup-chunks
```

ExpireTrash

The ExpireTrash job, contained in `OCA\Files_Trashbin\BackgroundJob\ExpireTrash`, will remove any file in the ownCloud trash bin which is older than the specified maximum file retention time. It can be run, as follows, using the OCC command:

```
occ trashbin:expire
```

ExpireVersions

The ExpireVersions job, contained in `OCA\Files_Versions\BackgroundJob\ExpireVersions`, will expire versions of files which are older than the specified maximum version retention time. It can be run, as follows, using the OCC command:

```
occ versions:expire
```

Warning

Please take care when adding `ExpireTrash` and `ExpireVersions` as [Cron](#) jobs. Make sure that they're not started in parallel on multiple machines. Running in parallel on a single machine is fine. But, currently, there isn't sufficient locking in place to prevent them from conflicting with each other if running in parallel across multiple machines.

SyncJob (CardDAV)

The CardDAV SyncJob, contained in `OCA\DAV\CardDAV\SyncJob`, syncs the local system address book, updating any existing contacts, and deleting any expired contacts. It can be run, as follows, using the OCC command:

```
occ dav:sync-system-addressbook
```

SyncJob (Federation)

OCAFederationSyncJob

It can be run, as follows, using the OCC command:

```
occ federation:sync-addressbooks
```

Config.php Parameters

ownCloud uses the `config/config.php` file to control server operations. `config/config.sample.php` lists all the configurable parameters within ownCloud, along with example or default values. This document provides a more detailed reference. Most options are configurable on your Admin page, so it is usually not necessary to edit `config/config.php`.

Note

The installer creates a configuration containing the essential parameters. Only manually add configuration parameters to `config/config.php` if you need to use a special value for a parameter. **Do not copy everything from `config/config.sample.php`. Only enter the parameters you wish to modify!**

Configuration

ownCloud supports loading configuration parameters from multiple files. You can add arbitrary files ending with .config.php in the config/ directory, for example you could place your email server configuration in email.config.php. This allows you to easily create and manage custom configurations, or to divide a large complex configuration file into a set of smaller files. These custom files are not overwritten by ownCloud, and the values in these files take precedence over config.php.

Default Parameters

These parameters are configured by the ownCloud installer, and are required for your ownCloud server to operate.

```
'instanceid' => '' ,
```

This is a unique identifier for your ownCloud installation, created automatically by the installer. This example is for documentation only, and you should never use it because it will not work. A valid `instanceid` is created when you install ownCloud. Needs to start with a letter.

```
'instanceid' => 'd3c944a9a' ,
```

```
'passwordsalt' => '' ,
```

The salt used to hash all passwords, auto-generated by the ownCloud installer. (There are also per-user salts.) If you lose this salt you lose all your passwords. This example is for documentation only, and you should never use it.

```
'trusted_domains' =>
array (
'demo.example.org' ,
'otherdomain.example.org' ,
) ,
```

Your list of trusted domains that users can log into. Specifying trusted domains prevents host header poisoning. Do not remove this, as it performs necessary security checks.

```
'cors.allowed-domains' => [
'https://foo.example.org' ,
] ,
```

The global list of CORS domains. All users can use tools running CORS requests from the listed domains.

```
'datadirectory' => '/var/www/owncloud/data' ,
```

Where user files are stored; this defaults to `data/` in the ownCloud directory. The SQLite database is also stored here, when you use SQLite.

(SQLite is not available in ownCloud Enterprise Edition)

```
'version' => '' ,
```

The current version number of your ownCloud installation. This is set up during installation and update, so you shouldn't need to change it.

```
'version.hide' => false ,
```

While hardening an ownCloud instance hiding the version information in `status.php` can be a legitimate step. Please consult the documentation before enabling this.

```
'show_server_hostname' => false ,
```

Optionally, show the hostname of the server in `status.php`. Defaults to hidden

```
'dbtype' => 'sqlite' ,
```

Identifies the database used with this installation. See also config option `supportedDatabases`

Available:

- sqlite (SQLite3 - Not in Enterprise Edition)
- mysql (MySQL/MariaDB)
- pgsql (PostgreSQL)
- oci (Oracle - Enterprise Edition Only)

```
'dbhost' => '',
```

Your host server name, for example localhost, hostname, hostname.example.com, or the IP address. To specify a port use hostname:####; to specify a Unix socket use localhost:/path/to/socket.

```
'dbname' => 'owncloud',
```

The name of the ownCloud database, which is set during installation. You should not need to change this.

```
'dbuser' => '',
```

The user that ownCloud uses to write to the database. This must be unique across ownCloud instances using the same SQL database. This is set up during installation, so you shouldn't need to change it.

```
'dbpassword' => '',
```

The password for the database user. This is set up during installation, so you shouldn't need to change it.

```
'dbtableprefix' => '',
```

Prefix for the ownCloud tables in the database.

```
'installed' => false,
```

Indicates whether the ownCloud instance was installed successfully; true indicates a successful installation, and false indicates an unsuccessful installation.

Default config.php Examples

When you use SQLite as your ownCloud database, your config.php looks like this after installation. The SQLite database is stored in your ownCloud data/ directory. SQLite is a simple, lightweight embedded database that is good for testing and for simple installations, but for production ownCloud systems you should use MySQL, MariaDB, or PostgreSQL.

```
<?php

$CONFIG = [
    'instanceid' => 'occ6f7365735',
    'passwordsalt' => '2c5778476346786306303',
    'trusted_domains' => [
        0 => 'localhost',
        1 => 'studio',
    ],
    'datadirectory' => '/var/www/owncloud/data',
    'dbtype' => 'sqlite3',
    'version' => '7.0.2.1',
    'installed' => true,
    'operation.mode' => 'single-instance',
];
```

This example is from a new ownCloud installation using MariaDB

```
<?php

$CONFIG = [
    'instanceid' => 'oc8c0fd71e03',
    'passwordsalt' => '515a13302a6b3950a9d0fdb970191a',
    'trusted_domains' => [
        0 => 'localhost',
```

```

1 => 'studio',
2 => '192.168.10.155'
],
'datadirectory' => '/var/www/owncloud/data',
'dbtype' => 'mysql',
'version' => '7.0.2.1',
'dbname' => 'owncloud',
'dbhost' => 'localhost',
'dbtableprefix' => 'oc_',
'dbuser' => 'oc_carla',
'dbpassword' => '67336bcdf7630dd80b2b81a413d07',
'installed' => true,
'operation.mode' => 'single-instance',
];

```

User Experience

These optional parameters control some aspects of the user interface. Default values, where present, are shown.

```
'default_language' => 'en',
```

This sets the default language on your ownCloud server, using ISO_639-1 language codes such as `en` for English, `de` for German, and `fr` for French. It overrides automatic language detection on public pages like login or shared items. User's language preferences configured under "personal -> language" override this setting after they have logged in.

```
'defaultapp' => 'files',
```

Set the default app to open on login. Use the app names as they appear in the URL after clicking them in the Apps menu, such as documents, calendar, and gallery. You can use a comma-separated list of app names, so if the first app is not enabled for a user then ownCloud will try the second one, and so on. If no enabled apps are found it defaults to the Files app.

```
'knowledgebaseenabled' => true,
```

`true` enables the Help menu item in the user menu (top right of the ownCloud Web interface). `false` removes the Help item.

```
'enable_avatars' => true,
```

`true` enables avatars, or user profile photos. These appear on the User page, on user's Personal pages and are used by some apps (contacts, mail, etc). `false` disables them.

```
'allow_user_to_change_display_name' => true,
```

`true` allows users to change their display names (on their Personal pages), and `false` prevents them from changing their display names.

```
'remember_login_cookie_lifetime' => 60*60*24*15,
```

Lifetime of the remember login cookie, which is set when the user clicks the `remember` checkbox on the login screen. The default is 15 days, expressed in seconds.

```
'session_lifetime' => 60 * 60 * 24,
```

The lifetime of a session after inactivity; the default is 24 hours, expressed in seconds.

```
'session_keepalive' => true,
```

Enable or disable session keep-alive when a user is logged in to the Web UI.

Enabling this sends a "heartbeat" to the server to keep it from timing out.

```
'token_auth_enforced' => false,
```

Enforce token authentication for clients, which blocks requests using the user password for enhanced security. Users need to generate tokens in personal settings which can be used as passwords on their clients.

Configuration

```
'login.alternatives' => [ ],
```

Allows to specify additional login buttons on the logon screen for e.g. SSO integration

```
'login.alternatives' => [
    ['href'           => 'https://www.testshib.org/Shibboleth.sso/ProtectNetwork?target=https%3A%2F%2Fmy.owncloud.tld%2Flogin%2Fssologin',
     'name'          => 'ProtectNetwork', 'img'           => '/img/PN_sign-in.gif'],
    ['href'           => 'https://www.testshib.org/Shibboleth.sso/OpenIdP.org?target=https%3A%2F%2Fmy.owncloud.tld%2Flogin%2Fssologin',
     'name'          => 'OpenIdP.org', 'img'           => '/img/openidp.png'],
],
'csrf.disabled' => false,
```

Disable ownCloud's built-in CSRF protection mechanism.

In some specific setups CSRF protection is handled in the environment, e.g., running F5 ASM. In these cases the built-in mechanism is not needed and can be disabled. Generally speaking, however, this config switch should be left unchanged.

WARNING: leave this as is if you're not sure what it does

```
'skeletondirectory' => '/path/to/owncloud/core/skeleton',
```

The directory where the skeleton files are located. These files will be copied to the data directory of new users. Leave empty to not copy any skeleton files.

```
'user_backends' => array(
    array(
        'class' => 'OC_User_IMAP',
        'arguments' => array('{imap.gmail.com:993/imap/ssl}INBOX')
    )
),
```

The `user_backends` app (which needs to be enabled first) allows you to configure alternate authentication backends. Supported backends are: IMAP (`OC_User_IMAP`), SMB (`OC_User_SMB`), and FTP (`OC_User_FTP`).

```
'lost_password_link' => 'https://example.org/link/to/password/reset',
```

If your user backend does not allow password resets (e.g. when it's a read-only user backend like LDAP), you can specify a custom link, where the user is redirected to, when clicking the "reset password" link after a failed login-attempt.

In case you do not want to provide any link, replace the url with 'disabled'

```
'accounts.enable_medial_search' => true,
```

Allow medial search on account properties like display name, user id, email, and other search terms. Allows finding 'Alice' when searching for 'lic'.

May slow down user search. Disable this if you encounter slow username search in the sharing dialog.

Mail Parameters

These configure the email settings for ownCloud notifications and password resets.

```
'mail_domain' => 'example.com',
```

The return address that you want to appear on emails sent by the ownCloud server, for example `oc-admin@example.com`, substituting your own domain, of course.

```
'mail_from_address' => 'owncloud',
```

FROM address that overrides the built-in `sharing-noreply` and `lostpassword-noreply` FROM addresses.

```
'mail_smtpdebug' => false,
```

Enable SMTP class debugging.

Configuration

```
'mail_smtpmode' => 'sendmail',
```

Which mode to use for sending mail: sendmail, smtp, qmail or php.

If you are using local or remote SMTP, set this to `smtp`.

If you are using PHP mail you must have an installed and working email system on the server. The program used to send email is defined in the `php.ini` file.

For the `sendmail` option you need an installed and working email system on the server, with `/usr/sbin/sendmail` installed on your Unix system.

For `qmail` the binary is `/var/qmail/bin/sendmail`, and it must be installed on your Unix system.

```
'mail_smtphost' => '127.0.0.1',
```

This depends on `mail_smtpmode`. Specify the IP address of your mail server host. This may contain multiple hosts separated by a semi-colon. If you need to specify the port number append it to the IP address separated by a colon, like this: `127.0.0.1:24`.

```
'mail_smtpport' => 25,
```

This depends on `mail_smtpmode`. Specify the port for sending mail.

```
'mail_smtptimeout' => 10,
```

This depends on `mail_smtpmode`. This sets the SMTP server timeout, in seconds. You may need to increase this if you are running an anti-malware or spam scanner.

```
'mail_smtpsecure' => '',
```

This depends on `mail_smtpmode`. Specify when you are using `ssl` or `tls`, or leave empty for no encryption.

```
'mail_smtpauth' => false,
```

This depends on `mail_smtpmode`. Change this to `true` if your mail server requires authentication.

```
'mail_smtpauthtype' => 'LOGIN',
```

This depends on `mail_smtpmode`. If SMTP authentication is required, choose the authentication type as `LOGIN` (default) or `PLAIN`.

```
'mail_smtpname' => '',
```

This depends on `mail_smtpauth`. Specify the username for authenticating to the SMTP server.

```
'mail_smtppassword' => '',
```

This depends on `mail_smtpauth`. Specify the password for authenticating to the SMTP server.

Proxy Configurations

```
'overwritehost' => '',
```

The automatic hostname detection of ownCloud can fail in certain reverse proxy and CLI/cron situations. This option allows you to manually override the automatic detection; for example `www.example.com`, or specify the port `www.example.com:8080`.

```
'overwriteprotocol' => '',
```

When generating URLs, ownCloud attempts to detect whether the server is accessed via `https` or `http`. However, if ownCloud is behind a proxy and the proxy handles the `https` calls, ownCloud would not know that `ssl` is in use, which would result in incorrect URLs being generated.

Valid values are `http` and `https`.

```
'overwritewebroot' => '',
```

ownCloud attempts to detect the webroot for generating URLs automatically.

Configuration

For example, if `www.example.com/owncloud` is the URL pointing to the ownCloud instance, the webroot is `/owncloud`. When proxies are in use, it may be difficult for ownCloud to detect this parameter, resulting in invalid URLs.

```
'overwritecondaddr' => '',
```

This option allows you to define a manual override condition as a regular expression for the remote IP address. For example, defining a range of IP addresses starting with `10.0.0.` and ending with `1 to 3: ^10\.\.0\.\. [1-3]`

```
'overwrite.cli.url' => '',
```

Use this configuration parameter to specify the base URL for any URLs which are generated within ownCloud using any kind of command line tools (cron or occ). The value should contain the full base URL: `https://www.example.com/owncloud`

```
'htaccess.RewriteBase' => '/',
```

To have clean URLs without `/index.php` this parameter needs to be configured.

This parameter will be written as "RewriteBase" on update and installation of ownCloud to your `.htaccess` file. While this value is often simply the URL path of the ownCloud installation it cannot be set automatically properly in every scenario and needs thus some manual configuration.

In a standard Apache setup this usually equals the folder that ownCloud is accessible at. So if ownCloud is accessible via "<https://mycloud.org/owncloud>" the correct value would most likely be `"/owncloud"`. If ownCloud is running under "<https://mycloud.org/>" then it would be `"/"`.

Note that the above rule is not valid in every case, as there are some rare setup cases where this may not apply. However, to avoid any update problems this configuration value is explicitly opt-in.

After setting this value run `occ maintenance:update:htaccess`. Now, when the following conditions are met ownCloud URLs won't contain `index.php`:

- `mod_rewrite` is installed
- `mod_env` is installed

```
'proxy' => '',
```

The URL of your proxy server, for example `proxy.example.com:8081`.

```
'proxyuserpwd' => '',
```

The optional authentication for the proxy to use to connect to the internet.

The format is: `username:password`.

Deleted Items (trash bin)

These parameters control the Deleted files app.

```
'trashbin_retention_obligation' => 'auto',
```

If the trash bin app is enabled (default), this setting defines the policy for when files and folders in the trash bin will be permanently deleted.

The app allows for two settings, a minimum time for trash bin retention, and a maximum time for trash bin retention. Minimum time is the number of days a file will be kept, after which it may be deleted. Maximum time is the number of days at which it is guaranteed to be deleted. Both minimum and maximum times can be set together to explicitly define file and folder deletion. For migration purposes, this setting is installed initially set to "auto", which is equivalent to the default setting in ownCloud 8.1 and before.

Available values:

- `auto`

default setting. keeps files and folders in the trash bin for 30 days and automatically deletes anytime after that if space is needed (note: files may not be deleted if space is not needed).

- `D, auto`

Configuration

keeps files and folders in the trash bin for D+ days, delete anytime if space needed (note: files may not be deleted if space is not needed)

- auto, D

delete all files in the trash bin that are older than D days automatically, delete other files anytime if space needed

- D1, D2

keep files and folders in the trash bin for at least D1 days and delete when exceeds D2 days

- disabled

trash bin auto clean disabled files and folders will be kept forever

```
'trashbin_purge_limit' => 50,
```

This setting defines percentage of free space occupied by deleted files that triggers auto purging of deleted files for this user

File versions

These parameters control the Versions app.

```
'versions_retention_obligation' => 'auto',
```

If the versions app is enabled (default), this setting defines the policy for when versions will be permanently deleted.

The app allows for two settings, a minimum time for version retention, and a maximum time for version retention. Minimum time is the number of days a version will be kept, after which it may be deleted. Maximum time is the number of days at which it is guaranteed to be deleted. Both minimum and maximum times can be set together to explicitly define version deletion. For migration purposes, this setting is installed initially set to "auto", which is equivalent to the default setting in ownCloud 8.1 and before.

Available values:

- auto

default setting. Automatically expire versions according to expire rules. Please refer to [../configuration/files/file_versioning](#) for more information.

- D, auto

keep versions at least for D days, apply expire rules to all versions that are older than D days

- auto, D

delete all versions that are older than D days automatically, delete other versions according to expire rules

- D1, D2

keep versions for at least D1 days and delete when exceeds D2 days

- disabled

versions auto clean disabled, versions will be kept forever

ownCloud Verifications

ownCloud performs several verification checks. There are two options, true and false.

```
'updatechecker' => true,
```

Check if ownCloud is up-to-date and shows a notification if a new version is available. This option is only applicable to ownCloud core. It is not applicable to app updates.

```
'updater.server.url' => 'https://updates.owncloud.com/server/' ,
```

URL that ownCloud should use to look for updates

```
'has_internet_connection' => true,
```

Is ownCloud connected to the Internet or running in a closed network?

```
'check_for_working_wellknown_setup' => true,
```

Configuration

Allows ownCloud to verify a working .well-known URL redirects. This is done by attempting to make a request from JS to <https://your-domain.com/.well-known/caldav/>

```
'config_is_read_only' => false,
```

In certain environments it is desired to have a read-only configuration file.

When this switch is set to `true` ownCloud will not verify whether the configuration is writable. However, it will not be possible to configure all options via the Web interface. Furthermore, when updating ownCloud it is required to make the configuration file writable again for the update process.

```
'operation.mode' => 'single-instance',
```

This defines the mode of operations. The default value is ‘single-instance’ which means that ownCloud is running on a single node, which might be the most common operations mode. The only other possible value for now is ‘clustered-instance’ which means that ownCloud is running on at least 2 nodes. The mode of operations has various impact on the behavior of ownCloud.

Logging

```
'log_type' => 'owncloud',
```

By default the ownCloud logs are sent to the `owncloud.log` file in the default ownCloud data directory.

If syslogging is desired, set this parameter to `syslog`. Setting this parameter to `errorlog` will use the PHP `error_log` function for logging.

```
'logfile' => '/var/log/owncloud.log',
```

Log file path for the ownCloud logging type.

Defaults to `[datadirectory]/owncloud.log`

```
'loglevel' => 2,
```

Loglevel to start logging at. Valid values are: 0 = Debug, 1 = Info, 2 = Warning, 3 = Error, and 4 = Fatal. The default value is Warning.

```
'syslog_tag' => 'ownCloud',
```

If you maintain different instances and aggregate the logs, you may want to distinguish between them. `syslog_tag` can be set per instance with a unique id. Only available if `log_type` is set to `syslog`.

The default value is `ownCloud`.

```
'log.syslog.format' => '[%reqId%][%remoteAddr%][%user%][%app%][%method%][%url%] %message%',
```

The syslog format can be changed to remove or add information.

In addition to the `%replacements%` below `%level%` can be used, but it is used as a dedicated parameter to the syslog logging facility anyway.

```
'log.conditions' => [
    [
        'shared_secret' => '57b58edb6637fe3059b3595cf9c41b9',
        'users' => ['user1'],
        'apps' => ['files_texteditor'],
        'logfile' => '/tmp/test.log'
    ],
    [
        'shared_secret' => '57b58edb6637fe3059b3595cf9c41b9',
        'users' => ['user1'],
        'apps' => ['gallery'],
        'logfile' => '/tmp/gallery.log'
    ],
],
```

Configuration

Log condition for log level increase based on conditions. Once one of these conditions is met, the required log level is set to debug. This allows to debug specific requests, users or apps

Supported conditions:

- **shared_secret:** if a request parameter with the name *log_secret* is set to this value the condition is met
- **users:** if the current request is done by one of the specified users, this condition is met
- **apps:** if the log message is invoked by one of the specified apps, this condition is met
- **logfile:** the log message invoked by the specified apps get redirected to this logfile, this condition is met Note: Not applicable when using syslog.

Defaults to an empty array.

```
'logdateformat' => 'F d, Y H:i:s',
```

This uses PHP.date formatting; see <http://php.net/manual/en/function.date.php>

```
'logtimezone' => 'Europe/Berlin',
```

The default timezone for logfiles is UTC. You may change this; see <http://php.net/manual/en/timezones.php>

```
'cron_log' => true,
```

Log successful cron runs.

```
'log_rotate_size' => false,
```

Enables log rotation and limits the total size of logfiles. The default is 0, or no rotation. Specify a size in bytes, for example 104857600 (100 megabytes = 100 * 1024 * 1024 bytes). A new logfile is created with a new name when the old logfile reaches your limit. If a rotated log file is already present, it will be overwritten.

Alternate Code Locations

Some of the ownCloud code may be stored in alternate locations.

```
'customclient_desktop' =>
    'https://owncloud.org/install/#install-clients',
'customclient_android' =>
    'https://play.google.com/store/apps/details?id=com.owncloud.android',
'customclient_ios' =>
    'https://itunes.apple.com/us/app/owncloud/id543672169?mt=8',
```

This section is for configuring the download links for ownCloud clients, as seen in the first-run wizard and on Personal pages.

Use the `apps_paths` parameter to set the location of the Apps directory, which should be scanned for available apps, and where user-specific apps should be installed from the Apps store. The path defines the absolute file system path to the app folder. The key `url` defines the HTTP Web path to that folder, starting from the ownCloud webroot. The key `writable` indicates if a Web server can write files to that folder.

Previews

ownCloud supports previews of image files, the covers of MP3 files, and text files. These options control enabling and disabling previews, and thumbnail size.

```
'enable_previews' => true,
```

By default, ownCloud can generate previews for the following filetypes:

Configuration

- Image files
- Covers of MP3 files
- Text documents

Valid values are `true`, to enable previews, or `false`, to disable previews

```
'preview_max_x' => 2048,
```

The maximum width, in pixels, of a preview. A value of `null` means there is no limit.

```
'preview_max_y' => 2048,
```

The maximum height, in pixels, of a preview. A value of `null` means there is no limit.

```
'preview_max_scale_factor' => 10,
```

If a lot of small pictures are stored on the ownCloud instance and the preview system generates blurry previews, you might want to consider setting a maximum scale factor. By default, pictures are upscaled to 10 times the original size. A value of `1` or `null` disables scaling.

```
'preview_max_filesize_image' => 50,
```

max file size for generating image previews with imagegd (default behaviour) If the image is bigger, it'll try other preview generators, but will most likely show the default mimetype icon

Value represents the maximum filesize in megabytes Default is 50 Set to -1 for no limit

```
'preview libreoffice_path' => '/usr/bin/libreoffice',
```

custom path for LibreOffice/OpenOffice binary

```
'preview_office_cl_parameters' =>
    '--headless --nologo --nofirststartwizard --invisible --norestore '.
    '--convert-to pdf --outdir ',
```

Use this if LibreOffice/OpenOffice requires additional arguments.

```
'enabledPreviewProviders' => array(
    'OC\Preview\PNG',
    'OC\Preview\JPEG',
    'OC\Preview\GIF',
    'OC\Preview\BMP',
    'OC\Preview\XBitmap',
    'OC\Preview\MP3',
    'OC\Preview\TXT',
    'OC\Preview\MarkDown'
),
```

Only register providers that have been explicitly enabled

The following providers are enabled by default:

- OC\Preview\PNG
- OC\Preview\JPEG
- OC\Preview\GIF
- OC\Preview\BMP
- OC\Preview\XBitmap
- OC\Preview\MarkDown
- OC\Preview\MP3
- OC\Preview\TXT

The following providers are disabled by default due to performance or privacy concerns:

Configuration

- OC\Preview\Illustrator
- OC\Preview\Movie
- OC\Preview\MSOffice2003
- OC\Preview\MSOffice2007
- OC\Preview\MSOfficeDoc
- OC\Preview\OpenDocument
- OC\Preview\PDF
- OC\Preview\Photoshop
- OC\Preview\Postscript
- OC\Preview\StarOffice
- OC\Preview\SVG
- OC\Preview\TIFF
- OC\Preview\Font

Note

Troubleshooting steps for the MS Word previews are available at the [../configuration/files/collaborative_documents_configuration](#) section of the Administrators Manual.

The following providers are not available in Microsoft Windows:

- OC\Preview\Movie
- OC\Preview\MSOfficeDoc
- OC\Preview\MSOffice2003
- OC\Preview\MSOffice2007
- OC\Preview\OpenDocument
- OC\Preview\StarOffice

Comments

Global settings for the Comments infrastructure

```
'comments.managerFactory' => '\OC\Comments\ManagerFactory',
```

Replaces the default Comments Manager Factory. This can be utilized if an own or 3rdParty CommentsManager should be used that – for instance – uses the filesystem instead of the database to keep the comments.

```
'systemtags.managerFactory' => '\OC\SystemTag\ManagerFactory',
```

Replaces the default System Tags Manager Factory. This can be utilized if an own or 3rdParty SystemTagsManager should be used that – for instance – uses the filesystem instead of the database to keep the tags.

Maintenance

These options are for halting user activity when you are performing server maintenance.

```
'maintenance' => false,
```

Enable maintenance mode to disable ownCloud

Configuration

If you want to prevent users from logging in to ownCloud before you start doing some maintenance work, you need to set the value of the maintenance parameter to true. Please keep in mind that users who are already logged-in are kicked out of ownCloud instantly.

```
'singleuser' => false,
```

When set to true, the ownCloud instance will be unavailable for all users who are not in the admin group.

SSL

```
'openssl' => array(
    'config' => '/absolute/location/of/openssl.cnf',
),
```

Extra SSL options to be used for configuration.

```
'enable_certificate_management' => false,
```

Allow the configuration of system wide trusted certificates

Memory caching backend configuration

Available cache backends:

- \OC\Memcache\APC Alternative PHP Cache backend
- \OC\Memcache\APCu APC user backend
- \OC\Memcache\ArrayCache In-memory array-based backend (not recommended)
- \OC\Memcache\Memcached Memcached backend
- \OC\Memcache\Redis Redis backend
- \OC\Memcache\XCache XCache backend

Advice on choosing between the various backends:

- APCu should be easiest to install. Almost all distributions have packages. Use this for single user environment for all caches.
- Use Redis or Memcached for distributed environments. For the local cache (you can configure two) take APCu.

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

Memory caching backend for locally stored data

- Used for host-specific data, e.g. file paths

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

Memory caching backend for distributed data

- Used for installation-specific data, e.g. database caching
- If unset_defaults to the value of memcache.local

```
'redis' => [
    'host' => 'localhost', // can also be a unix domain socket: '/tmp/redis.sock'
    'port' => 6379,
    'timeout' => 0.0,
    'password' => '', // Optional, if not defined no password will be used.
    'dbindex' => 0, // Optional, if undefined SELECT will not run and will use Redis Ser
],
```

Connection details for redis to use for memory caching in a single server configuration.

For enhanced security it is recommended to configure Redis to require a password. See <http://redis.io/topics/security> for more information.

```
'redis.cluster' => [
    'seeds' => [ // provide some/all of the cluster servers to bootstrap discovery, port
        'localhost:7000',
        'localhost:7001'
    ],
    'timeout' => 0.0,
    'read_timeout' => 0.0,
    'failover_mode' => \RedisCluster::FAILOVER_DISTRIBUTE
],

```

Connection details for a Redis Cluster

Only for use with Redis Clustering, for Sentinel-based setups use the single server configuration above, and perform HA on the hostname.

Redis Cluster support requires the php module phppredis in version 3.0.0 or higher.

Available failover modes:

- \RedisCluster::FAILOVER_NONE - only send commands to master nodes (default)
- \RedisCluster::FAILOVER_ERROR - failover to slaves for read commands if master is unavailable
- \RedisCluster::FAILOVER_DISTRIBUTE - randomly distribute read commands across master and slaves

```
'memcached_servers' => array(
    // hostname, port and optional weight. Also see:
    // http://www.php.net/manual/en/memcached.addservers.php
    // http://www.php.net/manual/en/memcached.addserver.php
    array('localhost', 11211),
    //array('other.host.local', 11211),
),

```

Server details for one or more memcached servers to use for memory caching.

```
'memcached_options' => array(
    // Set timeouts to 50ms
    \Memcached::OPT_CONNECT_TIMEOUT => 50,
    \Memcached::OPT_RETRY_TIMEOUT => 50,
    \Memcached::OPT_SEND_TIMEOUT => 50,
    \Memcached::OPT_RECV_TIMEOUT => 50,
    \Memcached::OPT_POLL_TIMEOUT => 50,

    // Enable compression
    \Memcached::OPT_COMPRESSION => true,

    // Turn on consistent hashing
    \Memcached::OPT_LIBKETAMA_COMPATIBLE => true,

    // Enable Binary Protocol
    \Memcached::OPT_BINARY_PROTOCOL => true,

    // Binary serializer will be enabled if the igbinary PECL module is available
    //\Memcached::OPT_SERIALIZER => \Memcached::SERIALIZER_IGBINARY,
),

```

Connection options for memcached, see <http://apprize.info/php/scaling/15.html>

```
'cache_path' => '',

```

Location of the cache folder, defaults to data/\$user/cache where \$user is the current user. When specified, the format will change to \$cache_path/\$user where \$cache_path is the configured cache directory and \$user is the user.

```
'cache_chunk_gc_ttl' => 86400, // 60*60*24 = 1 day

```

Configuration

TTL of chunks located in the cache folder before they're removed by garbage collection (in seconds). Increase this value if users have issues uploading very large files via the ownCloud Client as upload isn't completed within one day.

```
'dav.chunk_base_dir' => '',
```

Location of the chunk folder, defaults to `data/$user/uploads` where `$user` is the current user. When specified, the format will change to `$dav.chunk_base_dir/$user` where `$dav.chunk_base_dir` is the configured cache directory and `$user` is the user.

Sharing

Global settings for Sharing

```
'sharing.managerFactory' => '\OC\Share20\ProviderFactory',
```

Replaces the default Share Provider Factory. This can be utilized if own or 3rdParty Share Providers are used that – for instance – use the filesystem instead of the database to keep the share information.

```
'sharing.federation.allowHttpFallback' => false,
```

When talking with federated sharing server, allow falling back to HTTP instead of hard forcing HTTPS

All other configuration options

```
'dbdriveroptions' => array(
    PDO::MYSQL_ATTR_SSL_CA => '/file/path/to/ca_cert.pem',
    PDO::MYSQL_ATTR_INIT_COMMAND => 'SET wait_timeout = 28800'
),
```

Additional driver options for the database connection, eg. to enable SSL encryption in MySQL or specify a custom wait timeout on a cheap hoster.

```
'sqlite.journal_mode' => 'DELETE',
```

sqlite3 journal mode can be specified using this configuration parameter - can be 'WAL' or 'DELETE' see for more details <https://www.sqlite.org/wal.html>

```
'mysql.utf8mb4' => false,
```

During setup, if requirements are met (see below), this setting is set to true and MySQL can handle 4 byte characters instead of 3 byte characters.

If you want to convert an existing 3-byte setup into a 4-byte setup please set the parameters in MySQL as mentioned below and run the migration command:

```
./occ db:convert-mysql-charset
```

The config setting will be set automatically after a successful run.

Consult the documentation for more details.

MySQL requires a special setup for longer indexes (> 767 bytes) which are needed:

```
[mysqld] innodb_large_prefix=ON innodb_file_format=Barracuda innodb_file_per_table=ON
```

Tables will be created with

- character set: utf8mb4
- collation: utf8mb4_bin
- row_format: compressed

See:

<https://dev.mysql.com/doc/refman/5.7/en/charset-unicode-utf8mb4.html>

https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html#sysvar_innodb_large_prefix

https://mariadb.com/kb/en/mariadb/xtrabinnodb-server-system-variables/#innodb_large_prefix

<http://www.tocker.ca/2013/10/31/benchmarking-innodb-page-compression-performance.html>

<http://mechanics.flite.com/blog/2014/07/29/using-innodb-large-prefix-to-avoid-error-1071/>

```
'supportedDatabases' => array(
    'sqlite',
    'mysql',
    'pgsql',
    'oci',
),
)
```

Database types that are supported for installation.

Available:

- sqlite (SQLite3 - Not in Enterprise Edition)
- mysql (MySQL)
- pgsql (PostgreSQL)
- oci (Oracle - Enterprise Edition Only)

```
'tempdirectory' => '/tmp/owncloudtemp',
```

Override where ownCloud stores temporary files. Useful in situations where the system temporary directory is on a limited space ramdisk or is otherwise restricted, or if external storages which do not support streaming are in use.

The Web server user must have write access to this directory.

```
'hashingCost' => 10,
```

The hashing cost used by hashes generated by ownCloud. Using a higher value requires more time and CPU power to calculate the hashes

```
'blacklisted_files' => array('.htaccess'),
```

Blacklist a specific file or files and disallow the upload of files with this name. .htaccess is blocked by default.

WARNING: USE THIS ONLY IF YOU KNOW WHAT YOU ARE DOING.

```
'excluded_directories' =>
    array (
        '.snapshot',
        '~snapshot',
    ),
)
```

Exclude specific directory names and disallow scanning, creating and renaming using these names. Case insensitive.

Excluded directory names are queried at any path part like at the beginning, in the middle or at the end and will not be further processed if found. Please see the documentation for details and examples. Use when the storage backend supports eg snapshot directories to be excluded. **WARNING: USE THIS ONLY IF YOU KNOW WHAT YOU ARE DOING.**

```
'integrity.excluded.files' =>
    array (
        '.DS_Store',
        'Thumbs.db',
        '.directory',
        '.webapp',
        '.htaccess',
        '.user.ini',
    ),
)
```

Exclude files from the integrity checker command

```
'integrity.ignore.missing.app.signature' => [],
```

The list of apps that are allowed to have no signature.json

```
'share_folder' => '/',
```

Define a default folder for shared files and folders other than root.

Configuration

```
'cipher' => 'AES-256-CFB',
```

The default cipher for encrypting files. Currently AES-128-CFB and AES-256-CFB are supported.

```
'minimum_supported_desktop_version' => '2.2.4',
```

The minimum ownCloud desktop client version that will be allowed to sync with this server instance. All connections made from earlier clients will be denied by the server. Defaults to the minimum officially supported ownCloud version at the time of release of this server version.

When changing this, note that older unsupported versions of the ownCloud desktop client may not function as expected, and could lead to permanent data loss for clients or other unexpected results.

```
'quota_include_external_storage' => false,
```

EXPERIMENTAL: option whether to include external storage in quota calculation, defaults to false.

```
'filesystem_check_changes' => 0,
```

Specifies how often the local filesystem (the ownCloud data/ directory, and NFS mounts in data/) is checked for changes made outside ownCloud. This does not apply to external storages.

0 -> Never check the filesystem for outside changes, provides a performance increase when it's certain that no changes are made directly to the filesystem

1 -> Check each file or folder at most once per request, recommended for general use if outside changes might happen.

```
'part_file_in_storage' => true,
```

By default ownCloud will store the part files created during upload in the same storage as the upload target. Setting this to false will store the part files in the root of the users folder which might be required to work with certain external storage setups that have limited rename capabilities.

```
'mount_file' => '/var/www/owncloud/data/mount.json',
```

Where mount.json file should be stored, defaults to data/mount.json in the ownCloud directory.

```
'filesystem_cache_READONLY' => false,
```

When true, prevent ownCloud from changing the cache due to changes in the filesystem for all storage.

```
'secret' => '',
```

Secret used by ownCloud for various purposes, e.g. to encrypt data. If you lose this string there will be data corruption.

```
'trusted_proxies' => array('203.0.113.45', '198.51.100.128'),
```

List of trusted proxy servers

If you configure these also consider setting forwarded_for_headers which otherwise defaults to HTTP_X_FORWARDED_FOR (the X-Forwarded-For header).

```
'forwarded_for_headers' => array('HTTP_X_FORWARDED', 'HTTP_FORWARDED_FOR'),
```

Headers that should be trusted as client IP address in combination with trusted_proxies. If the HTTP header looks like 'X-Forwarded-For', then use 'HTTP_X_FORWARDED_FOR' here.

If set incorrectly, a client can spoof their IP address as visible to ownCloud, bypassing access controls and making logs useless!

Defaults to 'HTTP_X_FORWARDED_FOR' if unset

```
'max_filesize_animated_gifs_public_sharing' => 10,
```

max file size for animating gifs on public-sharing-site.

If the gif is bigger, it'll show a static preview

Value represents the maximum filesize in megabytes. Default is 10. Set to -1 for no limit.

Configuration

```
'filelocking.enabled' => true,
```

Enables transactional file locking.

This is enabled by default.

Prevents concurrent processes from accessing the same files at the same time. Can help prevent side effects that would be caused by concurrent operations. Mainly relevant for very large installations with many users working with shared files.

```
'filelocking.ttl' => 3600,
```

Set the lock's time-to-live in seconds.

Any lock older than this will be automatically cleaned up.

If not set this defaults to either 1 hour or the php max_execution_time, whichever is higher.

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

Memory caching backend for file locking

Because most memcache backends can clean values without warning using redis is highly recommended to *avoid data loss*.

```
'upgrade.disable-web' => false,
```

Disable the web based updater

```
'upgrade.automatic-app-update' => true,
```

Automatic update of market apps, set to "false" to disable.

```
'debug' => false,
```

Set this ownCloud instance to debugging mode

Only enable this for local development and not in production environments This will disable the minifier and outputs some additional debug information

```
'data-fingerprint' => '',
```

Sets the data-fingerprint of the current data served

This is a property used by the clients to find out if a backup has been restored on the server. Once a backup is restored run ./occ maintenance:data-fingerprint To set this to a new value.

Updating/Deleting this value can make connected clients stall until the user has resolved conflicts.

```
'copied_sample_config' => true,
```

This entry is just here to show a warning in case somebody copied the sample configuration. DO NOT ADD THIS SWITCH TO YOUR CONFIGURATION!

If you, brave person, have read until here be aware that you should not modify ANY settings in this file without reading the documentation.

```
'files_external_allow_create_new_local' => false,
```

Set this property to true if you want to enable the files_external local mount Option.

Default: false

```
'smb.logging.enable' => false,
```

Set this property to true if you want to enable debug logging for SMB access.

App config options

Retention for activities of the activity app:

```
'activity_expire_days' => 365,
```

Configuration

Every day a cron job is ran, which deletes all activities for all users which are older than the number of days that is set for `activity_expire_days`

```
'smb.logging.enable' => true,
```

This enables debug logging for SMB access. Use this carefully as it can generate a huge amount of log data.

Overriding Existing Parameter Values Using Environment Variables

ownCloud supports the ability to override the *web UI*, *command line*, and *Cron* environment settings by using environment variables. By doing so, you avoid the need to store credentials and other sensitive data in code. What's more, by using environment variables, you do not have to manage configurations (e.g., database connections) for different server environments, because environment variables store this information for you.

To override an existing setting, you need to export an environment variable which has the same name as the one which you want to override, prefixed with `OC_`. For example, if you wanted to override the value of `dbname`, you would set the environment variable `OC_dbname`.

Below are examples of setting an environment variable in the Apache and Nginx web servers, and for when running command line scripts.

Apache Web Server

```
# Inside a virtual host configuration
SetEnv OC_dbname owncloud_database_name
```

Nginx Web Server (php-fpm)

```
location / {
    fastcgi_param OC_dbname owncloud_database_name
}
```

Command Line

```
# export the variable into the environment before launching the Cron script
export OC_dbname=owncloud_database_name php -d variables_order=EGPCS cron.php
```

Email Configuration

ownCloud is capable of sending emails for a range of reasons. These include:

- Password reset emails
- Notifying users of new file shares
- Changes in files
- Activity notifications

To make use of them, users need to configure which notifications they want to receive. They can do this on their Personal pages.

Note

To be able to send emails, a functioning mail server must be available, whether locally in your network, or remotely.

Configuring an SMTP Server

To configure ownCloud to interact with an SMTP server, you can either update `config/config.php` by hand, or use the graphical Email Configuration Wizard, which updates `config/config.php` for you.

The Graphical Email Configuration Wizard

The wizard supports three mail server types: *SMTP*, *PHP*, and *Sendmail*. Use *SMTP* for a remote email server, and either *PHP* or *Sendmail* when your mail server is on the same machine as ownCloud.

Note

The *Sendmail* option refers to the *Sendmail* SMTP server, and any drop-in *Sendmail* replacement such as Postfix, Exim, or Courier. All of these include a *sendmail* binary, and are freely-interchangeable.

You need the following information from your mail server administrator to connect ownCloud to a remote SMTP server:

- Encryption type: None, SSL/TLS or STARTTLS.
- The From address you want your outgoing ownCloud mails to use.
- Whether authentication is required.
- Authentication method: None, Login, Plain, or NT LAN Manager.
- The server's IP address or fully-qualified domain name (FQDN).
- Login credentials, if required.

Email Server

This is used for sending out notifications. Saving...

Send mode: smtp Encryption: TLS
From address: owncloud @ alrac.net
Authentication method: Login (selected)
Authentication required:
Server address: : Port
Credentials:
Test email settings **Send email**

Your changes are saved immediately, and you can click the *Send Email* button to test your configuration. This sends a test message to the email address you configured on your Personal page. The test message says:

If you received this email, the settings seem to be correct.
--
ownCloud
web services under your control

Configuring PHP and Sendmail

Configuring PHP or Sendmail requires only that you select one of them, and then enter your desired return address.

Email Server

This is used for sending out notifications. Saving...

Send mode **sendmail**

From address **owncloud** @ **alrac.net**

Test email settings **Send email**

How do you decide which one to use? PHP mode uses your local `sendmail` binary. Use this if you want to use `php.ini` to control some of your mail server functions, such as setting *paths*, *headers*, or passing extra command options to the `sendmail` binary. These vary according to which server you are using, so consult your server's documentation to see what your options are.

In most cases the `smtp` option is best, because it removes the extra step of passing through PHP, and you can control all of your mail server options in one place, in your mail server configuration.

Setting Mail Server Parameters in config.php

If you prefer, you may set your mail server parameters in `config/config.php`. The following examples are for `SMTP`, `PHP`, `Sendmail`, and `Qmail`.

SMTP

If you want to send email using a local or remote SMTP server it is necessary to enter the name or IP address of the server, optionally followed by a colon separated port number, e.g. `:425`. If this value is not given the default port `25/tcp` will be used unless you change that by modifying the `mail_smtpport` parameter. Multiple servers can be entered, separated by semicolons:

```
<?php
"mail_smtpmode"      => "smtp",
"mail_smtphost"       => "smtp-1.server.dom;smtp-2.server.dom:425",
"mail_smtpport"        => 25,
```

Or:

```
<?php
"mail_smtpmode"      => "smtp",
"mail_smtphost"       => "smtp.server.dom",
"mail_smtpport"        => 425,
```

If a malware or SPAM scanner is running on the SMTP server it might be necessary that you increase the SMTP timeout to e.g., 30s:

```
<?php
"mail_smpttimeout"   => 30,
```

If the SMTP server accepts insecure connections, the default setting can be used:

```
<?php
"mail_smtpsecure"    => '',
```

If the SMTP server only accepts secure connections you can choose between the following two variants:

Configuration

SSL/TLS

A secure connection will be initiated using SSL/TLS via SMTPS on the default port 465/tcp:

```
<?php  
  
"mail_smtphost"      => "smtp.server.dom:465",  
"mail_smtpsecure"    => 'ssl',
```

STARTTLS

A secure connection will be initiated using STARTTLS via SMTP on the default port 25/tcp:

```
<?php  
  
"mail_smtphost"      => "smtp.server.dom",  
"mail_smtpsecure"    => 'tls',
```

An alternative is the port 587/tcp (recommended):

```
<?php  
  
"mail_smtphost"      => "smtp.server.dom:587",  
"mail_smtpsecure"    => 'tls',
```

Authentication

And finally it is necessary to configure if the SMTP server requires authentication, if not, the default values can be taken as is.

```
<?php  
  
"mail_smtpauth"      => false,  
"mail_smtpname"       => "",  
"mail_smtppassword"   => "",
```

If SMTP authentication is required you have to set the required username and password and can optionally choose between the authentication types **LOGIN** (default) or **PLAIN**.

```
<?php  
  
"mail_smtpauth"      => true,  
"mail_smtpauthtype"   => "LOGIN",  
"mail_smtpname"       => "username",  
"mail_smtppassword"   => "password",
```

PHP Mail

If you want to use PHP mail it is necessary to have an installed and working email system on your server. Which program in detail is used to send email is defined by the configuration settings in the **php.ini** file. On *nix systems this will most likely be Sendmail. ownCloud should be able to send email out of the box.

```
<?php  
  
"mail_smtpmode"      => "php",  
"mail_smtphost"       => "127.0.0.1",  
"mail_smtpport"       => 25,  
"mail_smpttimeout"    => 10,  
"mail_smtpsecure"    => "",  
"mail_smtpauth"       => false,  
"mail_smtpauthtype"   => "LOGIN",  
"mail_smtpname"       => "",  
"mail_smtppassword"   => "",
```

Sendmail

If you want to use the well known Sendmail program to send email, it is necessary to have an installed and working email system on your *nix server. The Sendmail binary (`/usr/sbin/sendmail`) is usually part of that system. ownCloud should be able to send email out of the box.

```
<?php  
  
"mail_smtpmode"      => "sendmail",  
"mail_smtphost"      => "127.0.0.1",  
"mail_smtpport"      => 25,  
"mail_smtptimeout"   => 10,  
"mail_smtpsecure"    => "",  
"mail_smtpauth"      => false,  
"mail_smtpauthtype"  => "LOGIN",  
"mail_smtpname"      => "",  
"mail_smtppassword"  => "",
```

Qmail

If you want to use the qmail program to send email, it is necessary to have an installed and working qmail email system on your server. The Sendmail binary (`/var/qmail/bin/sendmail`) will then be used to send email. ownCloud should be able to send email out of the box.

```
<?php  
  
"mail_smtpmode"      => "qmail",  
"mail_smtphost"      => "127.0.0.1",  
"mail_smtpport"      => 25,  
"mail_smtptimeout"   => 10,  
"mail_smtpsecure"    => "",  
"mail_smtpauth"      => false,  
"mail_smtpauthtype"  => "LOGIN",  
"mail_smtpname"      => "",  
"mail_smtppassword"  => "",
```

Send a Test Email

Regardless of how you have configured ownCloud to interact with an email server, to test your email configuration, save your email address in your personal settings and then use the **Send email** button in the *Email Server* section of the Admin settings page.

Using Self-Signed Certificates

When using self-signed certificates on the remote SMTP server the certificate must be imported into ownCloud. Please refer to Importing System-wide and Personal SSL Certificates for more information.

Troubleshooting

If you are unable to send email, try turning on debugging. Do this by enabling the `mail_smtpdebug` parameter in `config/config.php`.

```
<?php  
  
"mail_smtpdebug" => true;
```

Note

Immediately after pressing the **Send email** button, as described before, several `SMTP -> get_lines(): ...` messages appear on the screen. This is expected behavior and can be ignored.

Why is my web domain different from my mail domain?

The default domain name used for the sender address is the hostname where your ownCloud installation is served. If you have a different mail domain name you can override this behavior by setting the following configuration parameter:

```
<?php  
"mail_domain" => "example.com",
```

This setting results in every email sent by ownCloud (for example, the password reset email) having the domain part of the sender address appear as follows

```
no-reply@example.com
```

How can I find out if an SMTP server is reachable?

Use the ping command to check the server availability

```
ping smtp.server.dom
```

```
PING smtp.server.dom (ip-address) 56(84) bytes of data.  
64 bytes from your-server.local.lan (192.168.1.10): icmp_req=1 ttl=64  
time=3.64ms
```

How can I find out if the SMTP server is listening on a specific TCP port?

The best way to get mail server information is to ask your mail server admin. If you are the mail server admin, or need information in a hurry, you can use the netstat command. This example shows all active servers on your system, and the ports they are listening on. The SMTP server is listening on localhost port 25.

```
# netstat -pant
```

Active Internet connections (servers and established)						
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	ID/Program name
tcp	0	0	0.0.0.0:631	0.0.0.0:*	LISTEN	4418/cupsd
tcp	0	0	127.0.0.1:25	0.0.0.0:*	LISTEN	2245/exim4
tcp	0	0	127.0.0.1:3306	0.0.0.0:*	LISTEN	1524/mysqld

- 25/tcp is unencrypted smtp
- 110/tcp/udp is unencrypted pop3
- 143/tcp/udp is unencrypted imap4
- 465/tcp is encrypted smtps
- 993/tcp/udp is encrypted imaps
- 995/tcp/udp is encrypted pop3s

How can I determine if the SMTP server supports SMTPS?

A good indication that the SMTP server supports SMTPS is that it is listening on port **465**.

How can I determine what authorization and encryption protocols the mail server supports?

SMTP servers usually announce the availability of STARTTLS immediately after a connection has been established. You can easily check this using the telnet command.

Note

You must enter the marked lines to obtain the information displayed.

```
telnet smtp.domain.dom 25

Trying 192.168.1.10...
Connected to smtp.domain.dom.
Escape character is '^].
220 smtp.domain.dom ESMTP Exim 4.80.1 Tue, 22 Jan 2013 22:39:55 +0100
EHLO your-server.local.lan # <<< enter this command
250-smtp.domain.dom Hello your-server.local.lan [ip-address]
250-SIZE 52428800
250-8BITMIME
250-PIPELINING
250-AUTH PLAIN LOGIN CRAM-MD5 # <<< Supported auth protocols
250-STARTTLS # <<< Encryption is supported
250 HELP
QUIT # <<< enter this command
221 smtp.domain.dom closing connection
Connection closed by foreign host.
```

Enabling Debug Mode

If you are unable to send email, it might be useful to activate further debug messages by enabling the `mail_smtpdebug` parameter:

```
<?php  
"mail_smtpdebug" => true,
```

Note

Immediately after pressing the **Send email** button, as described before, several **SMTP -> get_lines(): ...** messages appear on the screen. This is expected behavior and can be ignored.

Using Email Templates

Most emails sent from ownCloud are based on editable email templates, which are a mixture of PHP and HTML. The currently available templates are:

Email	Format	Description	File Location
Activity notification mail	plain text	Notification of activities that users have enabled in the Notifications section of their Personal pages.	core/templates/mail.php
Lost password mail		Password reset email for users who lose their passwords.	core/templates/lostpassword/email.php
New user email	HTML plain text		settings/templates/email.new_user.php settings/templates/email.new_user_plain_text.php
Public link share email	HTML plain text	Notify users of new public link shares.	core/templates/mail.php core/templates/altmail.php
New file share email	HTML plain text	Notify users of new file shares.	core/templates/internalmail.php core/templates/internalaltmail.php

In addition to providing the email templates, this feature enables you to apply any pre-configured themes to the email. To modify an email template to users:

Configuration

1. Access the Admin page.
2. Scroll to the Mail templates section.
3. Select a template from the drop-down menu.
4. Make any desired modifications to the template.

The templates are written in PHP and HTML, and are already loaded with the relevant variables such as `username`, `share links`, and `filenames`. You can, if you are careful, edit these — even without knowing PHP or HTML. Don't touch any of the code, but it's OK to edit the text portions of the messages.

For example, this is the lost password mail template:

```
<?php  
  
echo str_replace(  
    '{link}',  
    ${'_['link']},  
    $l->t('Use the following link to reset your password: {link}')  
) ;
```

You could change the text portion of the template, Use the following link to reset your password: to say something else, such as:

Click the following link to reset your password.
If you did not ask for a password reset, ignore this message.

Again, be very careful to change nothing but the message text, because the tiniest coding error will break the template.

Note

You can edit the templates directly in the template text box, or you can copy and paste them to a text editor for modification and then copy and paste them back to the template text box for use when you are done.

Excluding Directories and Blacklisting Files

Definitions of terms

Blacklisted: Files that may harm the ownCloud environment like a foreign `.htaccess` file. Blacklisting prevents anyone from uploading blacklisted files to the ownCloud server.

Excluded: Existing directories on your ownCloud server, including external storage mounts, that are excluded from being processed by ownCloud. In effect they are invisible to ownCloud.

Both types are defined in `config.php`. Blacklisted files and excluded directories are not scanned by ownCloud, not viewed, not synced, and cannot be created, renamed, deleted, or accessed via direct path input from a file explorer. Even when a filepath is entered manually via a file explorer, the path cannot be accessed.

For example configurations please see `owncloud/config/config.sample.php`.

Impact on System Performance

If you have a filesystem mounted with 200,000 files and directories and 15 snapshots in rotation, you would now scan and process 200,000 elements plus $200,000 \times 15 = 3,000,000$ elements additionally. These additional 3,000,000 elements, 15 times more than the original quantity, would also be available for viewing and synchronisation. Because this is a big and unnecessary overhead, most times confusing to clients, further processing can be eliminated by using excluded directories.

Blacklisted Files

By default, ownCloud blacklists the file `.htaccess` to secure the running instance, which is important when using Apache as webserver. A foreign `.htaccess` file could overwrite rules defined by ownCloud. There is no explicit need to enter the file name `.htaccess` as parameter to the `blacklisted_files` array in `config.php`, but you can add more blacklisted file names if necessary.

Excluded Directories

Reason for excluding directories:

1. Enterprise storage systems, or special filesystems like ZFS and Btrfs are capable of snapshots. These snapshots are directories and keep point-in-time views of the data.
2. Snapshot directories are read-only.
3. There is no common naming for these directories, and most likely will never be. NetApp uses `.snapshot` and `~snapshot`, EMC eg `.ckpt`, HDS eg `.latest` and `~latest`, the ZFS filesystem uses `.zfs` and so on.
4. Viewing and scanning of these directories does not make any sense as these directories are used to ease backup, restores, and cloning
5. Directories which are part of the mounted filesystem, but must not be accessible via ownCloud.

Example:

If you have a snapshot-capable storage or filesystem where snapshots are enabled and presented to clients, each directory will contain a “special” visible directory named e.g. `.snapshot`. Depending on the system, you may find underneath a list of snapshots taken and in the next lower level the complete set of files and directories which were present when the snapshot was created. In most systems, this mechanism is true in all directory levels:

```
/.snapshot
  /nightly.0
    /home
    /dat
    /pictures
    file_1
    file_2
  /nightly.1
    /home
    /dat
    /pictures
    file_1
    file_2
  /nightly.2
    /home
    /dat
    /pictures
    file_1
    file_2
  ...
/home
/dat
/pictures
file_1
file_2
...
```

Example `excluded_directories` entries in `config.php` look like this:

```
'excluded_directories' => [
  '.snapshot',
  '~~snapshot',
  'dir1',
  'dir2',
],
```

Note that these are not pathnames, but directory names without any slashes. Excluding `dir1` excludes:

Configuration

```
/home/dir1  
/etc/stuff/dir1
```

But not:

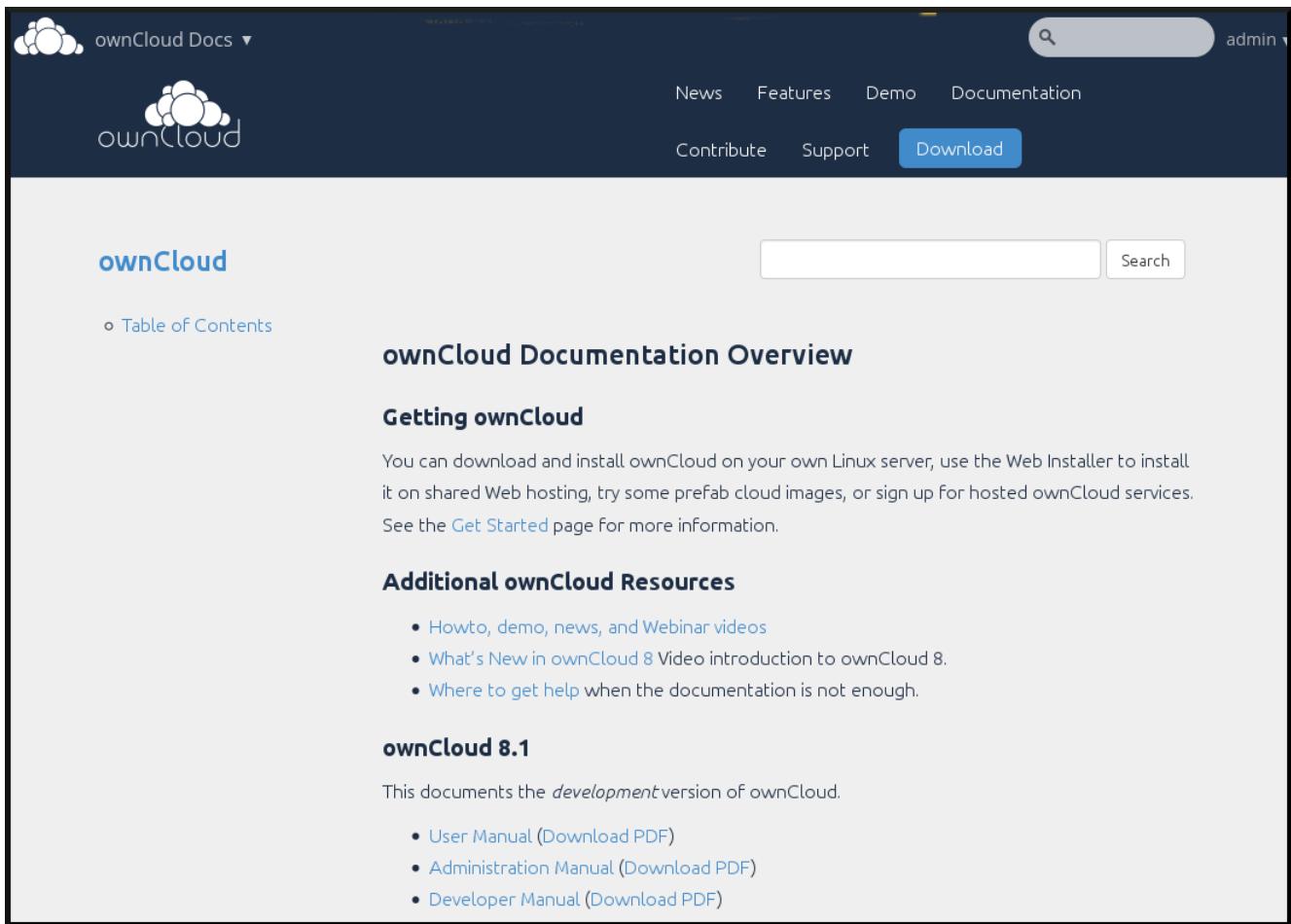
```
/home/.dir1  
/etc/stuff/mydir1
```

Example `blacklisted_files` entries in `config.php` look like this:

```
'blacklisted_files' => [  
    'hosts',  
    'evil_script.sh',  
,
```

Linking External Sites

You can embed external Web sites inside your ownCloud pages with the External Sites app, as this screenshot shows.



Click to enlarge

This is useful for quick access to important Web pages such as the ownCloud manuals and informational pages for your company, and for presenting external pages inside your custom ownCloud branding, if you use your own custom themes.

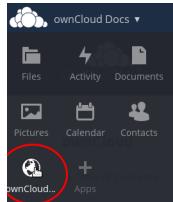
The External sites app is included in all versions of ownCloud. Go to **Apps > Not Enabled** to enable it. Then go to your ownCloud Admin page to create your links, which are saved automatically. There is a dropdown menu to select an icon, but there is only one default icon so you don't have to select one. Hover your cursor to the right of your links to make the trashcan icon appear when you want to remove them.

Configuration



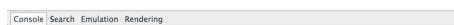
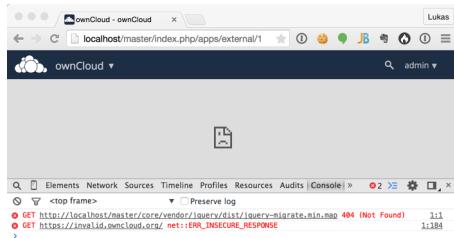
Click to enlarge

The links appear in the ownCloud dropdown menu on the top left after refreshing your page, and have globe icons.

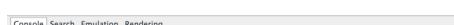
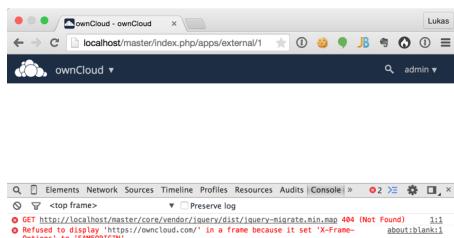


Your links may or may not work correctly due to the various ways that Web browsers and Web sites handle HTTP and HTTPS URLs, and because the External Sites app embeds external links in IFrames. Modern Web browsers try very hard to protect Web surfers from dangerous links, and safety apps like [Privacy Badger](#) and ad-blockers may block embedded pages. It is strongly recommended to enforce HTTPS on your ownCloud server; do not weaken this, or any of your security tools, just to make embedded Web pages work. After all, you can freely access them outside of ownCloud.

Most Web sites that offer login functionalities use the X-Frame-Options or Content-Security-Policy HTTP header which instructs browsers to not allow their pages to be embedded for security reasons (e.g. "Clickjacking"). You can usually verify the reason why embedding the website is not possible by using your browser's console tool. For example, this page has an invalid SSL certificate.



On this page, X-Frame-Options prevents the embedding.



There isn't much you can do about these issues, but if you're curious you can see what is happening.

Custom Client Download Repositories

You may configure the URLs to your own download repositories for your ownCloud desktop clients and mobile apps in config/config.php. This example shows the default download locations:

```
<?php
```

```
"customclient_desktop" => "https://owncloud.org/sync-clients/" ,  
"customclient_android" => "https://play.google.com/store/apps/details?id=com.owncloud.android" ,  
"customclient_ios" => "https://itunes.apple.com/us/app/owncloud/id543672169?mt=8" ,
```

Simply replace the URLs with the links to your own preferred download repos.

You may test alternate URLs without editing config/config.php by setting a test URL as an environment variable:

```
export OCC_UPDATE_URL=https://test.example.com
```

When you're finished testing you can disable the environment variable:

```
unset OCC_UPDATE_URL
```

Knowledge Base Configuration

The usage of ownCloud is more or less self explaining but nevertheless a user might run into a problem where he needs to consult the documentation or knowledge base. To ease access to the ownCloud documentation and knowledge base, a help menu item is shown in the settings menu by default.

Parameters

If you want to disable the ownCloud help menu item you can use the **knowledgebaseenabled** parameter inside the config/config.php.

```
<?php  
  
"knowledgebaseenabled" => true ,
```

Note

Disabling the help menu item might increase the number of support requests you have to answer in the future

Language Configuration

In normal cases ownCloud will automatically detect the language of the Web-GUI. If this does not work properly or you want to make sure that ownCloud always starts with a given language, you can use the **default_language** parameter.

Please keep in mind, that this will not effect a users language preference, which has been configured under "personal -> language" once he has logged in.

Please check settings/languageCodes.php for the list of supported language codes.

Parameters

```
<?php  
  
"default_language" => "en" ,
```

This parameters can be set in the config/config.php

Logging Configuration

Use your ownCloud log to review system status, or to help debug problems. You may adjust logging levels, and choose between using the ownCloud log or your syslog.

Parameters

Logging levels range from **DEBUG**, which logs all activity, to **FATAL**, which logs only fatal errors.

- **0:** DEBUG: Debug, informational, warning, and error messages, and fatal issues.
- **1:** INFO: Informational, warning, and error messages, and fatal issues.
- **2:** WARN: Warning, and error messages, and fatal issues.
- **3:** ERROR: Error messages and fatal issues.
- **4:** FATAL: Fatal issues only.

By default the log level is set to **2** (WARN). Use **DEBUG** when you have a problem to diagnose, and then reset your log level to a less-verbose level, as **DEBUG** outputs a lot of information, and can affect your server performance.

Logging level parameters are set in the config/config.php file, or on the Admin page of your ownCloud Web GUI.

ownCloud

All log information will be written to a separate log file which can be viewed using the log viewer on your Admin page. By default, a log file named **owncloud.log** will be created in the directory which has been configured by the **datadirectory** parameter in config/config.php.

The desired date format can optionally be defined using the **logdateformat** parameter in config/config.php. By default the **PHP date function** parameter "c" is used, and therefore the date/time is written in the format "2013-01-10T15:20:25+02:00". By using the date format in the example below, the date/time format will be written in the format "January 10, 2013 15:20:25".

```
"log_type" => "owncloud",
"logfile" => "owncloud.log",
"loglevel" => "3",
"logdateformat" => "F d, Y H:i:s",
```

syslog

All log information will be sent to your default syslog daemon.

```
"log_type" => "syslog",
"logfile" => "",
"loglevel" => "3",
```

The syslog format can be changed to remove or add information. In addition to the %replacements% below %level% can be used, but it is used as a dedicated parameter to the syslog logging facility anyway.

```
'log.syslog.format' => '[%reqId%][%remoteAddr%][%user%][%app%][%method%][%url%] %message%',
```

For the old syslog message format use:

```
'log.syslog.format' => '{%app%} %message%',
```

Conditional Logging Level Increase

You can configure the logging level to automatically increase to debug when the first condition inside a condition block is met. All conditions are optional !

- **shared_secret:** A unique token. If a http(s) request parameter named **log_secret** is added to the request and set to this token, the condition is met.
- **users:** If the current request is done by one of the specified users, this condition is met.
- **apps:** If the log message is invoked by one of the specified apps, this condition is met.
- **logfile:** The log message invoked gets redirected to this logfile when a condition above is met.

Notes regarding the logfile key:

1. If no logfile is defined, the standard logfile is used.

2. Not applicable when using syslog.

The following example demonstrates how all three conditions can look like.

The first one that matches triggers the condition block writing the log entry to the defined logfile.

```
'log.condition' => [
    'shared_secret' => '57b58edb6637fe3059b3595cf9c41b9',
    'users' => ['user1', 'user2'],
    'apps' => ['gallery'],
    'logfile' => '/tmp/test2.log'
],
```

Based on the conditional log settings above, following logs are written to the same logfile defined:

- Requests matching `log.secret` are debug logged.

```
curl -X PROPFIND -u sample-user:password \
https://your_domain/remote.php/webdav/?log_secret=57b58edb6637fe3059b3595cf9c41b9
```

- user1 and user2 gets debug logged.
- Access to app gallery gets debug logged.

Hardening and Security Guidance

ownCloud aims to ship with secure defaults that do not need to get modified by administrators. However, in some cases some additional security hardening can be applied in scenarios where the administrator has complete control over the ownCloud instance. This page assumes that you run ownCloud Server on Apache2 in a Linux environment.

Note

ownCloud will warn you in the administration interface if some critical security-relevant options are missing. However, it is still up to the server administrator to review and maintain system security.

Limit on Password Length

ownCloud uses the `bcrypt` algorithm, and thus for security and performance reasons, e.g., denial of service as CPU demand increases exponentially, it only verifies the first 72 characters of passwords. This applies to all passwords that you use in ownCloud: user passwords, passwords on link shares, and passwords on external shares.

Operating system

Give PHP read access to `/dev/urandom`

ownCloud uses a [RFC 4086 \("Randomness Requirements for Security"\)](#) compliant mixer to generate cryptographically secure pseudo-random numbers. This means that when generating a random number ownCloud will request multiple random numbers from different sources and derive from these the final random number.

The random number generation also tries to request random numbers from `/dev/urandom`, thus it is highly recommended to configure your setup in such a way that PHP is able to read random data from it.

Note

When having an `open_basedir` configured within your `php.ini` file, make sure to include `/dev/urandom`.

Enable hardening modules such as SELinux

It is highly recommended to enable hardening modules such as SELinux where possible. See SELinux Configuration to learn more about SELinux.

Deployment

Place data directory outside of the web root

It is highly recommended to place your data directory outside of the Web root (i.e. outside of `/var/www`). It is easiest to do this on a new installation.

Disable preview image generation

ownCloud is able to generate preview images of common filetypes such as images or text files. By default the preview generation for some file types that we consider secure enough for deployment is enabled by default. However, administrators should be aware that these previews are generated using PHP libraries written in C which might be vulnerable to attack vectors.

For high security deployments we recommend disabling the preview generation by setting the `enable_previews` switch to `false` in `config.php`. As an administrator you are also able to manage which preview providers are enabled by modifying the `enabledPreviewProviders` option switch.

Use HTTPS

Using ownCloud without using an encrypted HTTPS connection opens up your server to a man-in-the-middle (MITM) attack, and risks the interception of user data and passwords. It is a best practice, and highly recommended, to always use HTTPS on production servers, and to never allow unencrypted HTTP.

How to setup HTTPS on your Web server depends on your setup; please consult the documentation for your HTTP server. The following examples are for Apache.

Redirect all unencrypted traffic to HTTPS

To redirect all HTTP traffic to HTTPS administrators are encouraged to issue a permanent redirect using the 301 status code. When using Apache this can be achieved by adding a setting such as the following in the Apache VirtualHosts configuration containing the `<VirtualHost *:80>` entry:

```
Redirect permanent / https://example.com/
```

Enable HTTP Strict Transport Security

While redirecting all traffic to HTTPS is good, it may not completely prevent man-in-the-middle attacks. Thus administrators are encouraged to set the HTTP Strict Transport Security header, which instructs browsers to not allow any connection to the ownCloud instance using HTTP, and it attempts to prevent site visitors from bypassing invalid certificate warnings.

This can be achieved by setting the following settings within the Apache VirtualHost file containing the `<VirtualHost *:443>` entry:

```
<IfModule mod_headers.c>
  Header always set Strict-Transport-Security "max-age=15552000; includeSubDomains"
</IfModule>
```

If you don't have access to your Apache configuration it is also possible to add this to the main `.htaccess` file shipped with ownCloud. Make sure you're adding it below the line:

```
#### DO NOT CHANGE ANYTHING ABOVE THIS LINE ####
```

This example configuration will make all subdomains only accessible via HTTPS. If you have subdomains not accessible via HTTPS, remove `includeSubDomains`.

Note

This requires the `mod_headers` extension in Apache.

Proper SSL configuration

Configuration

Default SSL configurations by Web servers are often not state-of-the-art, and require fine-tuning for an optimal performance and security experience. The available SSL ciphers and options depend completely on your environment and thus giving a generic recommendation is not really possible.

We recommend using the [Mozilla SSL Configuration Generator](#) to generate a suitable configuration suited for your environment, and the free [Qualys SSL Labs Tests](#) gives good guidance on whether your SSL server is correctly configured.

Also ensure that HTTP compression is disabled to mitigate the BREACH attack.

Use a dedicated domain for ownCloud

Administrators are encouraged to install ownCloud on a dedicated domain such as cloud.domain.tld instead of domain.tld to gain all the benefits offered by the Same-Origin-Policy.

Ensure that your ownCloud instance is installed in a DMZ

As ownCloud supports features such as Federated File Sharing we do not consider Server Side Request Forgery (SSRF) part of our threat model. In fact, given all our external storage adapters this can be considered a feature and not a vulnerability.

This means that a user on your ownCloud instance could probe whether other hosts are accessible from the ownCloud network. If you do not want this you need to ensure that your ownCloud is properly installed in a segregated network and proper firewall rules are in place.

Serve security related Headers by the Web server

Basic security headers are served by ownCloud already in a default environment. These include:

- X-Content-Type-Options: nosniff
 - Instructs some browsers to not sniff the mimetype of files. This is used for example to prevent browsers from interpreting text files as JavaScript.
- X-XSS-Protection: 1; mode=block
 - Instructs browsers to enable their browser side Cross-Site-Scripting filter.
- X-Robots-Tag: none
 - Instructs search machines to not index these pages.
- X-Frame-Options: SAMEORIGIN
 - Prevents embedding of the ownCloud instance within an iframe from other domains to prevent Clickjacking and other similar attacks.

These headers are hard-coded into the ownCloud server, and need no intervention by the server administrator.

For optimal security, administrators are encouraged to serve these basic HTTP headers by the Web server to enforce them on response. To do this Apache has to be configured to use the .htaccess file and the following Apache modules need to be enabled:

- mod_headers
- mod_env

Administrators can verify whether this security change is active by accessing a static resource served by the Web server and verify that the above mentioned security headers are shipped.

Use Fail2ban

Another approach to hardening the server(s) on which your ownCloud installation rest is using an intrusion detection system. An excellent one is [Fail2ban](#). Fail2ban is designed to protect servers from brute force attacks. It works by monitoring log files (such as those for *ssh*, *web*, *mail*, and *log* servers) for certain patterns, specific to each server, and taking actions should those patterns be found.

Configuration

Actions include banning the IP from which the detected actions are being made from. This serves to both make the process more difficult as well as to prevent DDOS-style attacks. However, after a predefined time period, the banned IP is normally un-banned again.

This helps if the login attempts were genuine, so the user doesn't lock themselves out permanently. An example of such an action is users attempting to brute force login to a server via ssh. In this case, Fail2ban would look for something similar to the following in `/var/log/auth.log`.

```
Mar 15 11:17:37 yourhost sshd[10912]: input_userauth_request: invalid user audra [preauth]
Mar 15 11:17:37 yourhost sshd[10912]: pam_unix(sshd:auth): check pass; user unknown
Mar 15 11:14:51 yourhost sshd[10835]: PAM 2 more authentication failures; logname= uid=0 eui=
Mar 15 11:14:57 yourhost sshd[10837]: pam_unix(sshd:auth): authentication failure; logname=
Mar 15 11:14:59 yourhost sshd[10837]: Failed password for root from 221.194.44.231 port 4683
Mar 15 11:15:04 yourhost sshd[10837]: message repeated 2 times: [ Failed password for root f
Mar 15 11:15:04 yourhost sshd[10837]: Received disconnect from 221.194.44.231: 11: [preauth]
```

Note

If you're not familiar with what's going on, this snippet highlights a number of failed login attempts being made.

Using Fail2ban to secure an ownCloud login

On Ubuntu, you can install Fail2ban using the following commands:

```
apt update && apt upgrade
apt install fail2ban
```

Fail2ban installs several default filters for *Apache*, *NGINX*, and various other services, but none for ownCloud. Given that, we have to define our own filter. To do so, you first need to make sure that ownCloud uses your local timezone for writing log entries; otherwise, fail2ban cannot react appropriately to attacks. To do this, edit your `config.php` file and add the following line:

```
'logtimezone' => 'Europe/Berlin',
```

Note

Adjust the timezone to the one that your server is located in, based on [PHP's list of supported timezones](#).

This change takes effect as soon as you save `config.php`. You can test the change by:

1. Entering false credentials at your ownCloud login screen
2. Checking the timestamp of the resulting entry in ownCloud's log file.

Next, define a new Fail2ban filter rule for ownCloud. To do so, create a new file called `/etc/fail2ban/filter.d/owncloud.conf`, and insert the following configuration:

```
[Definition]
failregex={.*Login failed: \'.*\' \(\Remote IP: '<HOST>\'\')\}
ignoreregex =
```

This filter needs to be loaded when Fail2ban starts, so a further configuration entry is required to be added in `/etc/fail2ban/jail.d/defaults-debian.conf`, which you can see below:

```
[owncloud]
enabled = true
port = 80,443
protocol = tcp
filter = owncloud
maxretry = 3
```

Configuration

```
bantime = 10800
logpath = /var/owncloud_data/owncloud.log
```

This configuration:

1. Enables the filter rules for TCP requests on ports 80 and 443.
2. Bans IPs for 10800 seconds (3 hours).
3. Sets the path to the log file to analyze for malicious logins

Note

The most important part of the configuration is the `logpath` parameter. If this does not point to the correct log file, Fail2ban will either not work properly or refuse to start.

After saving the file, restart Fail2ban by running the following command:

```
service fail2ban restart
```

To test that the new ownCloud configuration has been loaded, use the following command:

```
fail2ban-client status
```

If “owncloud” is listed in the console output, the filter is both loaded and active. If you want to test the filter, run the following command, adjusting the path to your `owncloud.log`, if necessary:

```
fail2ban-regex /var/owncloud_data/owncloud.log /etc/fail2ban/filter.d/owncloud.conf
```

The output will look similar to the following, if you had one failed login attempt:

```
fail2ban-regex /var/www/owncloud_data/owncloud.log /etc/fail2ban/filter.d/owncloud.conf

Running tests
=====

Use    failregex file : /etc/fail2ban/filter.d/owncloud.conf
Use      log file : /var/www/owncloud_data/owncloud.log

Results
=====
Failregex: 1 total
|- #) [# of hits] regular expression
|  1) [1] {.*Login failed: \'.*\' \(\Remote IP: \'<HOST>\'\)'}
`-

Ignoreregex: 0 total

Date template hits:
|- [# of hits] date format
|  [40252] ISO 8601
`-

Lines: 40252 lines, 0 ignored, 1 matched, 40251 missed
```

The `Failregex` counter increments by 1 for every failed login attempt. To un-ban an IP, which was locked either during testing or unintentionally, use the following command:

```
fail2ban-client set owncloud unbanip <IP>
```

You can check the status of your ownCloud filter with the following command:

```
fail2ban-client status owncloud
```

This will produce an output similar to this:

```
Status for the jail: owncloud
|- filter
| |- File list:      /var/www/owncloud_data/owncloud.log
| |- Currently failed: 1
| `-- Total failed: 7
`- action
  |- Currently banned: 0
  | `-- IP list:
  `-- Total banned: 1
```

Password Policy

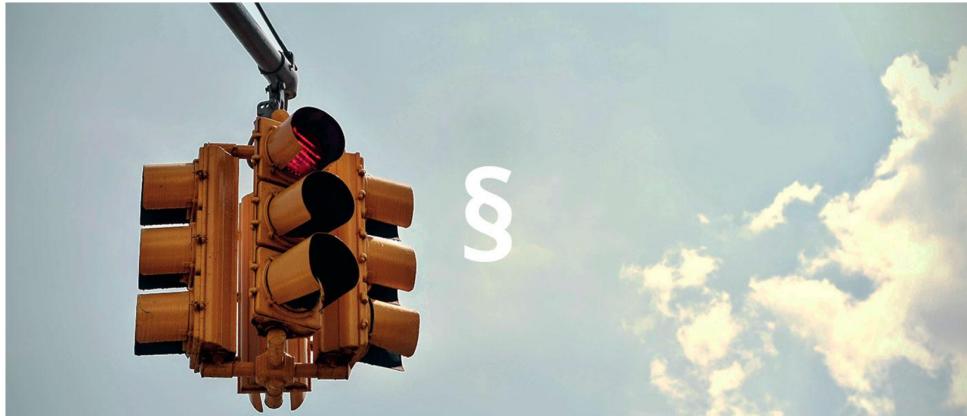
Password Policy

Password Policy

Define password policies for user and public link passwords

ENTERPRISE

made by ownCloud



This app is part of ownCloud Enterprise Edition. Take your ownCloud to the next level and start an ownCloud Enterprise Trial today!

[START ENTERPRISE TRIAL ➤](#)

ownCloud Enterprise users have the option of enabling [the Password Policy app](#). This application enables administrators to define password requirements, including: *minimum characters*, *numbers*, *capital letters*, and more, for all kinds of password endpoints, such as for user passwords and public link sharing passwords. Moreover the admin can enforce expiration dates for public link shares.

Share link password policy

Passwords should have at least:

- 8 minimum characters
- 1 uppercase letters
- 1 numbers
- 1 special characters

Define special characters #!



Link expiration:

- 7 days to expire link if password is set
- 7 days to expire link if password is not set

Save

ownCloud Community Edition users, however, can use [the Security app](#), available from the ownCloud Marketplace. It supports configuring a basic password policy, which includes:

1. Setting a password length
2. Whether to enforce at least one upper and lower case character, a numerical character and a special character.



Reverse Proxy Configuration

ownCloud can be run through a reverse proxy, which can cache static assets such as images, CSS, or Javascript files, move the load of handling HTTPS to a different server or load balance between multiple servers.

Defining Trusted Proxies

For security, you must explicitly define the proxy servers that ownCloud is to trust. Connections from trusted proxies will be specially treated to get the real client information, for use in access control and logging. Parameters are configured in config/config.php

Set the **trusted_proxies** parameter as an array of IP address to define the servers ownCloud should trust as proxies. This parameter provides protection against client spoofing, and you should secure those servers as you would your ownCloud server.

A reverse proxy can define HTTP headers with the original client IP address, and ownCloud can use those headers to retrieve that IP address. ownCloud uses the de-facto standard header 'X-Forwarded-For' by default, but this can be configured with the **forwarded_for_headers** parameter. This parameter is an array of PHP lookup strings, for example 'X-Forwarded-For' becomes 'HTTP_X_FORWARDED_FOR'. Incorrectly setting this parameter may allow clients to spoof their IP address as visible to ownCloud, even when going through the trusted proxy! The correct value for this parameter is dependent on your proxy software.

Overwrite Parameters

The automatic hostname, protocol or webroot detection of ownCloud can fail in certain reverse proxy situations. This configuration allows the automatic detection to be manually overridden.

If ownCloud fails to automatically detect the hostname, protocol or webroot you can use the **overwrite** parameters inside the config/config.php. The **overwritehost** parameter is used to set the hostname of the proxy. You can also specify a port. The **overwriteprotocol** parameter is used to set the protocol of the proxy. You can choose between the two options **http** and **https**. The **overwritewebroot** parameter is used to set the absolute web path of the proxy to the ownCloud folder. When you want to keep the automatic detection of one of the three parameters you can leave the value empty or don't set it. The **overwritecondaddr** parameter is used to overwrite the values dependent on the remote address. The value must be a **regular expression** of the IP addresses of the proxy. This is useful when you use a reverse SSL proxy only for https access and you want to use the automatic detection for http access.

Example

Multiple Domains Reverse SSL Proxy

If you want to access your ownCloud installation **http://domain.tld/owncloud** via a multiple domains reverse SSL proxy **https://ssl-proxy.tld/domain.tld/owncloud** with the IP address **10.0.0.1** you can set the following parameters inside the config/config.php.

```
<?php

$CONFIG = [
    "trusted_proxies" => ['10.0.0.1'],
    "overwritehost" => "ssl-proxy.tld",
    "overwriteprotocol" => "https",
    "overwritewebroot" => "/domain.tld/owncloud",
    "overwritecondaddr" => "^10\.\.0\.\.1\$",
];
}
```

With an Apache as reverse proxy (ssl-proxy.tld) you can use this configuration:

```
ProxyPass "/domain.tld/owncloud" "http://domain.tld/owncloud"
ProxyPassReverse "/domain.tld/owncloud" "http://domain.tld/owncloud"
```

Note

If you want to use the SSL proxy during installation you have to create the config/config.php otherwise you have to extend the existing **\$CONFIG** array.

Using Third Party PHP Components

ownCloud uses some third party PHP components to provide some of its functionality. These components are part of the software package and are contained in the **/3rdparty** folder.

Managing Third Party Parameters

When using third party components, keep the following parameters in mind:

- **3rdpartyroot** – Specifies the location of the 3rd-party folder. To change the default location of this folder, you can use this parameter to define the absolute file system path to the folder location.
- **3rdpartyurl** – Specifies the http web path to the 3rdpartyroot folder, starting at the ownCloud web root.

An example of what these parameters might look like is as follows:

```
<?php
```

```
"3rdpartyroot" => OC::$SERVERROOT."/3rdparty",
"3rdpartyurl"  => "/3rdparty",
```

Automatic Configuration Setup

If you need to install ownCloud on multiple servers, you normally do not want to set up each instance separately as described in Database Configuration. For this reason, ownCloud provides an automatic configuration feature.

To take advantage of this feature, you must create a configuration file, called `../owncloud/config/autoconfig.php`, and set the file parameters as required. You can specify any number of parameters in this file. Any unspecified parameters appear on the “Finish setup” screen when you first launch ownCloud.

The `../owncloud/config/autoconfig.php` is automatically removed after the initial configuration has been applied.

Parameters

When configuring parameters, you must understand that two parameters are named differently in this configuration file when compared to the standard config.php file.

autoconfig.php	config.php
directory	datadirectory
dbpass	dbpassword

Automatic Configurations Examples

The following sections provide sample automatic configuration examples and what information is requested at the end of the configuration.

Data Directory

Using the following parameter settings, the “Finish setup” screen requests database and admin credentials settings.

```
<?php
$AUTOCONFIG = array(
    "directory"      => "/www/htdocs/owncloud/data",
);
```

SQLite Database

Using the following parameter settings, the “Finish setup” screen requests data directory and admin credentials settings.

```
<?php
$AUTOCONFIG = array(
    "dbtype"         => "sqlite",
    "dbname"         => "owncloud",
    "dbtableprefix"  => "",
);
```

MySQL Database

Using the following parameter settings, the “Finish setup” screen requests data directory and admin credentials settings.

```
<?php
$AUTOCONFIG = array(
    "dbtype"         => "mysql",
    "dbname"         => "owncloud",
    "dbuser"         => "username",
    "dbpass"         => "password",
    "dbhost"         => "localhost",
);
```

```
"dbtableprefix" => "",  
);
```

Note

Keep in mind that the automatic configuration does not eliminate the need for creating the database user and database in advance, as described in Database Configuration.

PostgreSQL Database

Using the following parameter settings, the “Finish setup” screen requests data directory and admin credentials settings.

```
<?php  
$AUTOCONFIG = array(  
    "dbtype"      => "pgsql",  
    "dbname"      => "owncloud",  
    "dbuser"      => "username",  
    "dbpass"      => "password",  
    "dbhost"      => "localhost",  
    "dbtableprefix" => "",  
);
```

Note

Keep in mind that the automatic configuration does not eliminate the need for creating the database user and database in advance, as described in Database Configuration.

All Parameters

Using the following parameter settings, because all parameters are already configured in the file, the ownCloud installation skips the “Finish setup” screen.

```
<?php  
$AUTOCONFIG = array(  
    "dbtype"      => "mysql",  
    "dbname"      => "owncloud",  
    "dbuser"      => "username",  
    "dbpass"      => "password",  
    "dbhost"      => "localhost",  
    "dbtableprefix" => "",  
    "adminlogin"   => "root",  
    "adminpass"    => "root-password",  
    "directory"    => "/www/htdocs/owncloud/data",  
);
```

Note

Keep in mind that the automatic configuration does not eliminate the need for creating the database user and database in advance, as described in Database Configuration.

ownCloud Server Tuning

Using Cron to Perform Background Jobs

See Background Jobs for a description and the benefits.

Enable JavaScript and CSS Asset Management

See `js_css_asset_management_configuration` for a description and the benefits.

Enable Memory Caching

Caching improves performance by storing data, code, and other objects in memory. Memory cache configuration for ownCloud is no longer automatically available from ownCloud 8.1 but must be installed and configured separately. ownCloud supports [Redis](#), [APCu](#), and [Memcached](#) as memory caching backends. See Memory Caching, for further details.

Use Redis-based Transactional File Locking

File locking is enabled by default, using the database locking backend. However, this places a significant load on your database. See the section Transactional File Locking for how to configure ownCloud to use Redis-based Transactional File Locking.

Redis Tuning

Redis tuning improves both file locking (if used) and memory caching (when using Redis). Here is a brief guide for tuning Redis to improve the performance of your ownCloud installation, when working with sizeable instances.

TCP-Backlog

If you raised the TCP-backlog setting, the following warning appears in the Redis logs:

```
WARNING: The TCP backlog setting of 20480 cannot be enforced because /proc/sys/net/core/soma...
```

If so, please consider that newer versions of Redis have their own TCP-backlog value set to 511, and that you have to increase if you have many connections. In high requests-per-second environments, you need a significant backlog to avoid slow clients connection issues.

Note

The Linux kernel will silently truncate the TCP-backlog setting to the value of `/proc/sys/net/core/somaxconn`. So make sure to raise both the value of `somaxconn` and `tcp_max_syn_backlog`, to get the desired effect.

To fix this warning, set the value of `net.core.somaxconn` to 65535 in `/etc/rc.local`, so that it persists upon reboot, by running the following command.

```
sudo echo sysctl -w net.core.somaxconn=65535 >> /etc/rc.local
```

After the next reboot, 65535 connections will be allowed, instead of the default value.

Transparent Huge Pages (THP)

If you are experiencing latency problems with Redis, the following warning may appear in your Redis logs:

```
WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This creates b...
```

If so, unfortunately, when a Linux kernel has [Transparent Huge Pages](#) enabled, Redis incurs a significant latency penalty after the fork call is used, to persist information to disk. Transparent Huge Pages are the cause of the following issue:

1. A fork call is made, resulting in two processes with shared huge pages being created.

2. In a busy instance, a few event loops cause commands to target a few thousand pages, causing the copy-on-write of almost the entire process memory.
3. Big latency and memory usage result.

As a result, make sure to disable Transparent Huge Pages using the following command:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

Redis Latency Problems

If you are having issues with Redis latency, please refer to [the official Redis guide](#) on how to handle them.

Database Tuning

Using MariaDB/MySQL Instead of SQLite

MySQL or MariaDB are preferred because of the [performance limitations of SQLite with highly concurrent applications](#), like ownCloud.

See the section Database Configuration for how to configure ownCloud for MySQL or MariaDB. If your installation is already running on SQLite then it is possible to convert to MySQL or MariaDB using the steps provided in Converting Database Type.

Tune MariaDB/MySQL

A comprehensive guide to tuning MySQL and MariaDB is outside the scope of the ownCloud documentation. However, here are three links that can help you find further information:

- [MySQLTuner](#).
- [Percona Tools for MySQL](#)
- [Optimizing and Tuning MariaDB](#).

Tune PostgreSQL

A comprehensive guide to tuning PostgreSQL is outside the scope of the ownCloud documentation. However, here are three links that can help you find further information:

- [Five Steps to PostgreSQL Performance](#)
- [Tuning the autovacuum proceess for tables with huge update workloads \(oc_filecache\)](#)

SSL / Encryption App

SSL (HTTPS) and file encryption/decryption can be offloaded to a processor's AES-NI extension. This can both speed up these operations while lowering processing overhead. This requires a processor with the [AES-NI instruction set](#).

Here are some examples how to check if your CPU / environment supports the AES-NI extension:

- For each CPU core present: `grep flags /proc/cpuinfo` or as a summary for all cores: `grep -m 1 ^flags /proc/cpuinfo` If the result contains any aes, the extension is present.
- Search eg. on the Intel web if the processor used supports the extension [Intel Processor Feature Filter](#) You may set a filter by "AES New Instructions" to get a reduced result set.
- For versions of openssl >= 1.0.1, AES-NI does not work via an engine and will not show up in the openssl engine command. It is active by default on the supported hardware. You can check the openssl version via `openssl version -a`
- If your processor supports AES-NI but it does not show up eg via grep or coreinfo, it is maybe disabled in the BIOS.
- If your environment runs virtualized, check the virtualization vendor for support.

Webserver Tuning

Tune Apache

Enable HTTP/2 Support

If you want to improve the speed of an ownCloud installation, while at the same time increasing its security, you can enable [HTTP/2 support for Apache](#). Please be aware that [most browsers require HTTP/2 to be used with SSL enabled](#).

Apache Processes

An Apache process uses around 12MB of RAM. Apache should be configured so that the maximum number of `HTTPD` processes times 12MB is lower than the amount of RAM. Otherwise the system begins to swap and the performance goes down.

Use KeepAlive

The `KeepAlive` directive enables persistent HTTP connections, allowing multiple requests to be sent over the same TCP connection. Enabling it reduces latency by as much as 50%. In combination with the periodic checks of the sync client the following settings are recommended:

```
KeepAlive On
KeepAliveTimeout 100
MaxKeepAliveRequests 200
```

Hostname Lookups

```
# cat /etc/httpd/conf/httpd.conf
...
HostnameLookups off
```

Log files

Log files should be switched off for maximum performance. To do that, comment out the `CustomLog` directive. However, keep `ErrorLog` set, so errors can be tracked down.

Enable index.php-less URLs

Since ownCloud 9.0.3 you need to explicitly configure and enable index.php-less URLs (e.g. <https://example.com/apps/files/> instead of <https://example.com/index.php/apps/files/>). The following documentation provides the needed steps to configure this for the Apache Web server.

Prerequisites

Before being able to use index.php-less URLs you need to enable the `mod_rewrite` and `mod_env` Apache modules. Furthermore a configured `AllowOverride All` directive within the vhost of your Web server is needed. Please have a look at the Apache manual for how to enable and configure these.

Furthermore these instructions are only working when using Apache together with the `mod_php` Apache module for PHP. Other modules like `php-fpm` or `mod_fastcgi` are unsupported.

Finally the user running your Web server (e.g. `www-data`) needs to be able to write into the `.htaccess` file shipped within the ownCloud root directory (e.g. `/var/www/owncloud/.htaccess`). If you have applied Set Strong Directory Permissions the user might be unable to write into this file and the needed update will fail. You need to revert this strong permissions temporarily by following the steps described in [Setting Permissions for Updating](#).

Configuration steps

The first step is to configure the `overwrite.cli.url` and `htaccess.RewriteBase config.php` options (See [Config.php Parameters](#)). If you're accessing your ownCloud instance via <https://example.com/> the following two options need to be added / configured:

```
'overwrite.cli.url' => 'https://example.com',
'htaccess.RewriteBase' => '/',
```

Configuration

If the instance is accessed via `https://example.com/owncloud` the following configuration is needed:

```
'overwrite.cli.url' => 'https://example.com/owncloud',
'htaccess.RewriteBase' => '/owncloud',
```

As a second step ownCloud needs to enable index.php-less URLs. This is done:

- during the next update of your ownCloud instance
- by manually running the `occ maintenance:update:htaccess` (See Using the `occ` Command)

Afterwards your instance should have index.php-less URLs enabled.

Troubleshooting

If accessing your ownCloud installation fails after following these instructions and you see messages like this in your ownCloud log:

```
The requested uri('login') cannot be processed by the script '/owncloud/index.php'
```

make sure that you have configured the two `config.php` options listed above correctly.

User Management

User Management

On the User management page of your ownCloud Web UI you can:

- Create new users
- View all of your users in a single scrolling window
- Filter users by group
- See what groups they belong to
- Edit their full names and passwords
- See their data storage locations
- View and set quotas
- Create and edit their email addresses
- Send an automatic email notification to new users
- Delete them with a single click

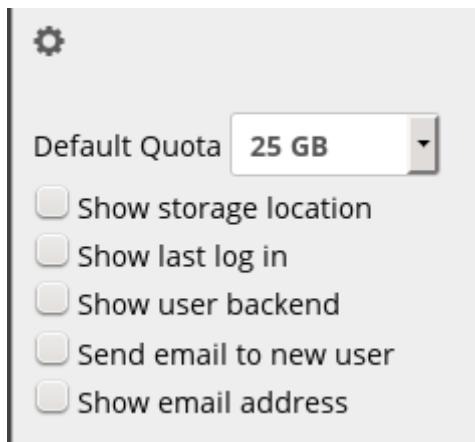
The default view displays basic information about your users.

Username	Password	Groups	Create	Search Users	
Username	Full Name	Password	Groups	Group Admin for	Quota
A admin	admin	●●●●●●	admin	no group	Default
layla	layla	●●●●●●	users, artists	artists	10 GB
molly	molly	●●●●●●	users	no group	Default
ritasue	ritasue	●●●●●●	artists	users	10 GB
stashcat	stashcat	●●●●●●	users, admin	no group	5 GB

The Group filters on the left sidebar lets you quickly filter users by their group memberships, and create new groups.

+ Add Group	
Everyone	5
Admins	2
users	3
artists	2

Click the gear icon on the lower left sidebar to set a default storage quota, and to display additional fields: **Show storage location**, **Show last log in**, **Show user backend**, **Send email to new users**, and **Show email address**.



User accounts have the following properties:

Login Name (Username)

The unique ID of an ownCloud user, and it cannot be changed.

Full Name

The user's display name that appears on file shares, the ownCloud Web interface, and emails. Admins and users may change the Full Name anytime. If the Full Name is not set it defaults to the login name.

Password

The admin sets the new user's first password. Both the user and the admin can change the user's password at anytime.

Groups

You may create groups, and assign group memberships to users. By default new users are not assigned to any groups.

Group Admin

Group admins are granted administrative privileges on specific groups, and can add and remove users from their groups.

Quota

The maximum disk space assigned to each user. Any user that exceeds the quota cannot upload or sync data. You have the option to include external storage in user quotas.

Creating a New User

To create a user account:

Configuration

- Enter the new user's **Login Name** and their initial **Password**
- Optionally, assign **Groups** memberships
- Click the **Create** button

The screenshot shows a user creation form. At the top, there are fields for 'Username' (set to 'terry') and 'Full Name'. Below this is a table listing existing users: admin, layla, molly, and ritasue. To the right of the table is a dropdown menu titled 'users' with three options: 'admin', 'artists', and 'users'. The 'users' option has a checked checkbox. A 'Create' button is located at the top right of the form area.

Username	Full Name
admin	admin
layla	layla
molly	molly
ritasue	ritasue

users

admin

artists

users

+ add group

Login names may contain letters (a-z, A-Z), numbers (0-9), dashes (-), underscores (_), periods (.) and at signs (@). After creating the user, you may fill in their **Full Name** if it is different than the login name, or leave it for the user to complete.

If you have checked **Send email to new user** in the control panel on the lower left sidebar, you may also enter the new user's email address, and ownCloud will automatically send them a notification with their new login information. You may edit this email using the email template editor on your Admin page (see Email Configuration).

Reset a User's Password

You cannot recover a user's password, but you can set a new one:

- Hover your cursor over the user's **Password** field
- Click on the **pencil icon**
- Enter the user's new password in the password field, and remember to provide the user with their password

If you have encryption enabled, there are special considerations for user password resets. Please see Encryption Configuration.

Renaming a User

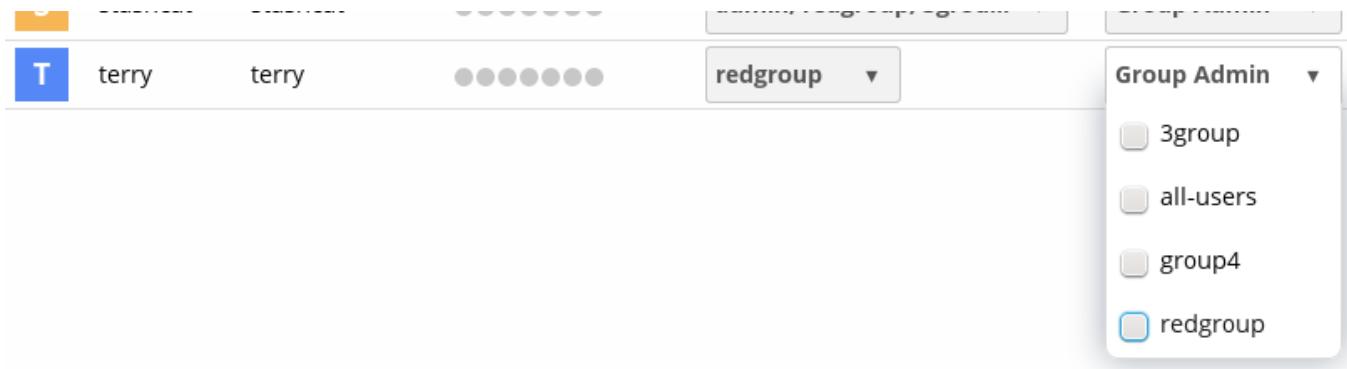
Each ownCloud user has two names: a unique **Login Name** used for authentication, and a **Full Name**, which is their display name. You can edit the display name of a user, but you cannot change the login name of any user.

To set or change a user's display name:

- Hover your cursor over the user's **Full Name** field
- Click on the **Pencil icon**
- Enter the user's new display name

Granting Administrator Privileges to a User

ownCloud has two types of administrators: **Super Administrators** and **Group Administrators**. Group administrators have the rights to create, edit and delete users in their assigned groups. Group administrators cannot access system settings, or add or modify users in the groups that they are not **Group Administrators** for. Use the dropdown menus in the **Group Admin** column to assign group admin privileges.



Super Administrators have full rights on your ownCloud server, and can access and modify all settings. To assign the **Super Administrators** role to a user, simply add them to the `admin` group.

Managing Groups

You can assign new users to groups when you create them, and create new groups when you create new users. You may also use the **Add Group** button at the top of the left pane to create new groups. New group members will immediately have access to file shares that belong to their new groups.

Setting Storage Quotas

Click the gear on the lower left pane to set a default storage quota. This is automatically applied to new users. You may assign a different quota to any user by selecting from the **Quota** dropdown, selecting either a preset value or entering a custom value. When you create custom quotas, use the normal abbreviations for your storage values such as 500 MB, 5 GB, 5 TB, and so on.

You now have a configurable option in `config.php` that controls whether external storage is counted against user's quotas. This is still experimental, and may not work as expected. The default is to not count external storage as part of user storage quotas. If you prefer to include it, then change the default `false` to `true`:

```
'quota_include_external_storage' => false,
```

Metadata (such as thumbnails, temporary files, and encryption keys) takes up about 10% of disk space, but is not counted against user quotas. Users can check their used and available space on their Personal pages. Only files that originate with users count against their quotas, and not files shared with them that originate from other users. For example, if you upload files to a different user's share, those files count against your quota. If you re-share a file that another user shared with you, that file does not count against your quota, but the originating user's.

Encrypted files are a little larger than unencrypted files; the unencrypted size is calculated against the user's quota.

Deleted files that are still in the trash bin do not count against quotas. The trash bin is set at 50% of quota. Deleted file aging is set at 30 days. When deleted files exceed 50% of quota then the oldest files are removed until the total is below 50%.

When version control is enabled, the older file versions are not counted against quotas.

When a user creates a public share via URL, and allows uploads, any uploaded files count against that user's quota.

Deleting users

Deleting a user is easy: hover your cursor over their name on the **Users** page until a trashcan icon appears at the far right. Click the trashcan, and they're gone. You'll see an undo button at the top of the page, which remains until you refresh the page. When the undo button is gone you cannot recover the deleted user.

All of the files owned by the user are deleted as well, including all files they have shared. If you need to preserve the user's files and shares, you must first download them from your ownCloud Files page, which compresses them into a zip file, or use a sync client to copy them to your local computer. See [File Sharing](#) to learn how to create persistent file shares that survive user deletions.

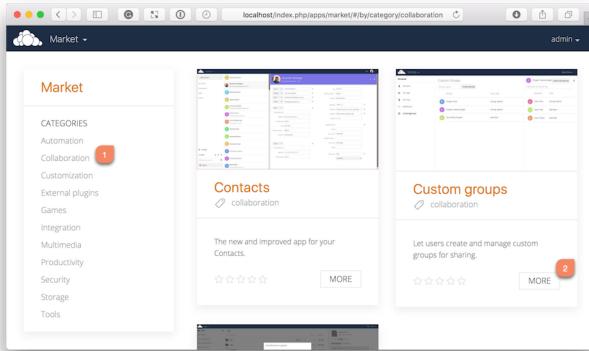
Enabling Custom Groups

Configuration

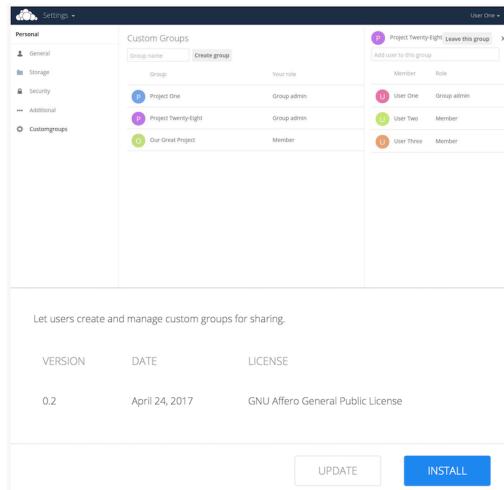
In previous versions of ownCloud, files and folders could only be shared with individual users or groups created by administrators. This wasn't the most efficient way to work. From ownCloud 10.0, users can create groups on-the-fly, through a feature called "Custom Groups", enabling them to share content in a more flexible way.

To enable Custom Groups:

1. From the ownCloud Market, which you can find in version 10.0 under the Apps menu, click "**Market**".
2. Click "**Collaboration**" (1), to filter the list of available options and click the "**Custom groups**" application (2).



3. Click "**INSTALL**" in the bottom right-hand corner of the Custom Groups application.



With this done, Custom Group functionality will be available in your ownCloud installation.

Resetting a Lost Admin Password

The normal ways to recover a lost password are:

1. Click the password reset link on the login screen; this appears after a failed login attempt. This works only if you have entered your email address on your Personal page in the ownCloud Web interface, so that the ownCloud server can email a reset link to you.
2. Ask another ownCloud server admin to reset it for you.

If neither of these is an option, then you have a third option, and that is using the `occ` command. `occ` is in the `owncloud` directory, for example `/var/www/owncloud/occ`. `occ` has a command for resetting all user passwords, `user:resetpassword`. It is best to run `occ` as the HTTP user, as in this example on Ubuntu Linux:

```
$ sudo -u www-data php /var/www/owncloud/occ user:resetpassword admin
Enter a new password:
Confirm the new password:
Successfully reset password for admin
```

If your ownCloud username is not `admin`, then substitute your ownCloud username.

Configuration

You can find your HTTP user in your HTTP configuration file. These are the default Apache HTTP user:group on Linux distros:

- Centos, Red Hat, Fedora: apache:apache
- Debian, Ubuntu, Linux Mint: www-data:www-data
- openSUSE: wwwrun:www

See Using the `occ` Command to learn more about using the `occ` command.

Resetting a User Password

The ownCloud login screen displays a **Wrong password. Reset it?** message after a user enters an incorrect password, and then ownCloud automatically resets their password. However, if you are using a read-only authentication backend such as LDAP or Active Directory, this will not work. In this case you may specify a custom URL in your `config.php` file to direct your user to a server than can handle an automatic reset:

```
'lost_password_link' => 'https://example.org/link/to/password/reset',
```

User Authentication with IMAP, SMB, and FTP

You may configure additional user backends in ownCloud's configuration file (`config/config.php`) using the following syntax:

```
<?php

"user_backends" => [
    0 => [
        "class"      => ...,
        "arguments"  => [
            0 => ...
        ],
    ],
],
```

Note

A non-blocking or correctly configured SELinux setup is needed for these backends to work, if SELinux is enabled on your server.. Please refer to the SELinux documentation for further details.

Currently the [External user support app](#) (`user_external`), which is not enabled by default, provides three backends. These are:

- [IMAP](#)
- [SMB](#)
- [FTP](#)

See [Installing and Managing Apps](#) for more information.

IMAP

Provides authentication against IMAP servers.

Option	Value/Description
Class	OC_User_IMAP.
Arguments	A mailbox string as defined in the PHP documentation .
Dependency	PHP's IMAP extension . See Manual Installation on Linux for instructions on how to install it.

Example

```
<?php

"user_backends" => [
    0 => [
        "class"      => "OC_User_IMAP",
        "arguments"  => [
            // The IMAP server to authenticate against
            '{imap.gmail.com:993/imap/ssl}',
            // The domain to send email from
            'example.com'
        ],
    ],
],
],
```

Warning

The second arguments parameter ensures that only users from that domain are allowed to login. When set, after a successful login, the domain will be stripped from the email address and the rest used as an ownCloud username. For example, if the email address is guest.user@example.com, then guest.user will be the username used by ownCloud.

SMB

Provides authentication against Samba servers.

Option	Value/Description
Class	OC_User_SMB.
Arguments	The samba server to authenticate against.
Dependency	PECL's smbclient extension or smbclient.

Example

```
<?php

"user_backends" => [
    0 => [
        "class"      => "OC_User_SMB",
        "arguments"  => [
            0 => 'localhost'
        ],
    ],
],
],
```

FTP

Provides authentication against FTP servers.

Option	Value/Description
Class	OC_User_FTP.
Arguments	The FTP server to authenticate against.
Dependency	PHP's FTP extension . See Manual Installation on Linux for instructions on how to install it.

Example

```
<?php

"user_backends" => [
    0 => [
        "class"      => "OC_User_FTP",
        "arguments"  => [
            0 => 'localhost'
        ],
    ],
],
```

User Authentication with LDAP**Warning**

Please check both the advanced and expert configurations carefully before using in production

ownCloud ships with an LDAP application which allows LDAP users (including Active Directory) to appear in your ownCloud user listings. These users will authenticate to ownCloud with their LDAP credentials, so you don't have to create separate ownCloud user accounts for them. You will manage their ownCloud group memberships, quotas, and sharing permissions just like any other ownCloud user.

Note

The PHP LDAP module is required. It is supplied by `php7.1-ldap` on Debian/Ubuntu and `php-ldap` on CentOS/Red Hat/Fedora. Please check for the correct version, based on your installation of PHP.

The LDAP application supports:

- LDAP group support
- File sharing with ownCloud users and groups
- Access via WebDAV and ownCloud Desktop Client
- Versioning, external Storage and all other ownCloud features
- Seamless connectivity to Active Directory, with no extra configuration required
- Support for primary groups in Active Directory
- Auto-detection of LDAP attributes such as base DN, email, and the LDAP server port number
- Only read access to your LDAP (edit or delete of users on your LDAP is not supported)

Warning

The LDAP app is not compatible with the User backend using remote HTTP servers app. You cannot use both of them at the same time.

Note

A non-blocking or correctly configured [SELinux](#) setup is needed for the LDAP backend to work. Please refer to the SELinux Configuration.

Configuration

First, enable the LDAP user and group backend app on the Apps page in ownCloud. Then, go to your Admin page to configure it. The LDAP configuration panel has four tabs. A correctly completed first tab (“Server”) is mandatory to access the other tabs. A green indicator lights when the configuration is correct. Hover your cursor over the fields to see some pop-up tooltips.

Server Tab

Start with the Server tab. You may configure multiple servers if you have them. At a minimum you must supply the LDAP server’s hostname. If your server requires authentication, enter your credentials on this tab. ownCloud will then attempt to auto-detect the server’s port and base DN. The base DN and port are mandatory, so if ownCloud cannot detect them you must enter them manually.

LDAP

The screenshot shows the LDAP configuration interface. The top navigation bar includes tabs for Server, Users, Login Attributes, Groups, Advanced, and Expert. The Server tab is active. Below the tabs, there's a dropdown menu for selecting a server configuration, currently set to "1. Server". There are buttons for adding (+), deleting (-), and editing (edit icon). The main configuration area consists of several input fields: "Host" (empty), "Port" (set to 389), "User DN" (empty), "Password" (empty), and "One Base DN per line" (containing "dc=directory,dc=my-company,dc=com"). To the right of these fields are "Detect Base DN" and "Test Base DN" buttons. A checkbox labeled "Manually enter LDAP filters (recommended for large directories)" is unchecked. At the bottom of the form, a message states "Configuration incomplete" next to "Continue" and "Help" buttons.

Server configuration:

Configure one or more LDAP servers. Click the “Delete Configuration” button to remove the active configuration.

Host:

The host name or IP address of the LDAP server. It can also be an **ldaps://** URI. If you enter the port number, it speeds up server detection.

Examples:

- directory.my-company.com
- ldaps://directory.my-company.com
- directory.my-company.com:9876

Port:

The port on which to connect to the LDAP server. The field is disabled in the beginning of a new configuration. If the LDAP server is running on a standard port, the port will be detected automatically. If you are using a non-standard port, ownCloud will attempt to detect it. If this fails you must enter the port number manually.

Example:

- 389

User DN:

The name as DN of a user who has permissions to do searches in the LDAP directory. Leave it empty for anonymous access. We recommend that you have a special LDAP system user for this.

Example:

Configuration

- uid=owncloudsystemuser,cn=sysusers,dc=my-company,dc=com

Password:

The password for the user given above. Empty for anonymous access.

Base DN:

The base DN of LDAP, from where all users and groups can be reached. You may enter multiple base DNs, one per line. Base DNs for users and groups can be set in the Advanced tab. This field is mandatory. ownCloud attempts to determine the Base DN according to the provided User DN or the provided Host, and you must enter it manually if ownCloud does not detect it.

Example:

- dc=my-company,dc=com

User Filter

Use this to control which LDAP users are listed as ownCloud users on your ownCloud server. In order to control which LDAP users can login to your ownCloud server use the Login filter. Those LDAP users who have access but are not listed as users (if there are any) will be hidden users. You may bypass the form fields and enter a raw LDAP filter if you prefer.

The screenshot shows the 'User Filter' configuration page. At the top, there is a navigation bar with tabs: Server (selected), Users, Login Attributes, Groups, Advanced, and Expert. Below the navigation bar, there is a section titled 'Limit ownCloud access to users meeting these criteria:'. It contains two dropdown menus: 'Only these object classes:' and 'Only from these groups:'. A note below the first dropdown says: 'The most common object classes for users are organizationalPerson, person, user, and inetOrgPerson. If you are not sure which object class to select, please consult your directory admin.' There is also a link 'Edit LDAP Query' and a text input field 'Edit LDAP Query'. At the bottom of the page, there is a button 'Verify settings and count users' and a footer with links: 'Configuration incomplete', 'Back', 'Continue', and 'Help'.

Only those object classes:

ownCloud will determine the object classes that are typically available for user objects in your LDAP. ownCloud will automatically select the object class that returns the highest amount of users. You may select multiple object classes.

Only from those groups:

If your LDAP server supports the memberof-overlay in LDAP filters, you can define that only users from one or more certain groups are allowed to appear in user listings in ownCloud. By default, no value will be selected. You may select multiple groups.

Note

Group membership is configured by adding *memberUid*, *uniqueMember* or *member* attributes to an ldap group (see Group Member association) below. In order to efficiently look up the groups a user who is a member of the LDAP server must support a *memberof*-overlay. It allows using the virtual *memberOf* or *isMemberOf* attributes of an LDAP user in the user filter. If your LDAP server does not support the *memberof*-overlay in LDAP filters, the input field is disabled. Please contact your LDAP administrator.

- Active Directory uses *memberOf* and is enabled by default.
- OpenLDAP uses *memberOf*. [Reverse Group Membership Maintenance](#) needs to be enabled.
- Oracle uses *isMemberOf* and is enabled by default.

Edit raw filter instead:

Clicking on this text toggles the filter mode and you can enter the raw LDAP filter directly. Example:

```
(&(objectClass=inetOrgPerson)(memberOf=cn=owncloudusers,ou=groups,dc=example,dc=com))
```

x users found:

This is an indicator that tells you approximately how many users will be listed in ownCloud. The number updates automatically after any changes.

Login Filter

The settings in the Login Filter tab determine which LDAP users can log in to your ownCloud system and which attribute or attributes the provided login name is matched against (e.g., LDAP/AD username, email address). You may select multiple user details. You may bypass the form fields and enter a raw LDAP filter if you prefer.

You may override your User Filter settings on the User Filter tab by using a raw LDAP filter.

Server	Users	Login Attributes	Groups	Advanced	Expert
--------	-------	-------------------------	--------	----------	--------

When logging in, ownCloud will find the user based on the following attributes:

LDAP / AD Username:

LDAP / AD Email

Address:

Other Attributes:

[Edit LDAP Query](#)

Configuration incomplete

LDAP Username:

If this value is checked, the login value will be compared to the username in the LDAP directory. The corresponding attribute, usually *uid* or *samaccountname* will be detected automatically by ownCloud.

LDAP Email Address:

If this value is checked, the login value will be compared to an email address in the LDAP directory; specifically, the *mailPrimaryAddress* and *mail* attributes.

Other Attributes:

This multi-select box allows you to select other attributes for the comparison. The list is generated automatically from the user object attributes in your LDAP server.

Edit raw filter instead:

Clicking on this text toggles the filter mode and you can enter the raw LDAP filter directly. The "%uid" placeholder is replaced with the login name entered by the user upon login.

Examples:

- only username:

```
(&(objectClass=inetOrgPerson)(memberOf=cn=owncloudusers,ou=groups,dc=example,dc=com))
```

- username or email address:

```
((&(objectClass=inetOrgPerson)(memberOf=cn=owncloudusers,ou=groups,dc=example,dc=com))
```

Group Filter

By default, no LDAP groups will be available in ownCloud. The settings in the group filter tab determine which groups will be available in ownCloud. You may also elect to enter a raw LDAP filter instead.

Groups meeting these criteria are available in ownCloud:

Only these object classes: Select object classes

Only from these groups: Select groups

[Edit LDAP Query](#)

Edit LDAP Query

Verify settings and count groups

Configuration incomplete [Back](#) [Help](#)

Only those object classes:

ownCloud will determine the object classes that are typically available for group objects in your LDAP server. ownCloud will only list object classes that return at least one group object. You can select multiple object classes. A typical object class is `group`, or `posixGroup`.

Only from those groups:

ownCloud will generate a list of available groups found in your LDAP server. From these groups, you can select the group or groups that get access to your ownCloud server.

Edit raw filter instead:

Clicking on this text toggles the filter mode and you can enter the raw LDAP filter directly.

Example:

- `objectClass=group`
- `objectClass=posixGroup`

y groups found:

This tells you approximately how many groups will be available in ownCloud. The number updates automatically after any change.

Advanced Settings

The LDAP Advanced Setting section contains options that are not needed for a working connection. This provides controls to disable the current configuration, configure replica hosts, and various performance-enhancing options.

The Advanced Settings are structured into three parts:

- Connection Settings
- Directory Settings
- Special Attributes

Connection Settings

LDAP

The screenshot shows the LDAP configuration interface with the following details:

- Server Tab:** Contains tabs for Server, Users, Login Attributes, Groups, Advanced (selected), and Expert.
- Connection Settings Tab:** Contains the following configuration options:
 - Configuration:**
 - Active:** A checked checkbox.
 - Backup (Replica) Host:** An input field.
 - Backup (Replica) Port:** An input field.
 - Disable Main Server:** An unchecked checkbox.
 - Turn off SSL certificate validation:** An unchecked checkbox.
 - Cache Time-To-Live:** An input field containing "600".
 - Directory Settings:** A collapsed section.
 - Special Attributes:** A collapsed section.
- Buttons:** Test Configuration and Help.

Configuration Active:

Enables or Disables the current configuration. By default, it is turned off. When ownCloud makes a successful test connection it is automatically turned on.

Backup (Replica) Host:

If you have a backup LDAP server, enter the connection settings here. ownCloud will then automatically connect to the backup when the main server cannot be reached. The backup server must be a replica of the main server so that the object UUIDs match.

Example:

- directory2.my-company.com

Backup (Replica) Port:

The connection port of the backup LDAP server. If no port is given, but only a host, then the main port (as specified above) will be used.

Example:

- 389

Disable Main Server:

You can manually override the main server and make ownCloud only connect to the backup server. This is useful for planned downtimes.

Turn off SSL certificate validation:

Turns off SSL certificate checking. Use it for testing only!

Cache Time-To-Live:

A cache is introduced to avoid unnecessary LDAP traffic, for example caching usernames so they don't have to be looked up for every page, and speeding up loading of the Users page. Saving the configuration empties the cache. The time is given in seconds.

Note that almost every PHP request requires a new connection to the LDAP server. If you require fresh PHP requests we recommend defining a minimum lifetime of 15s or so, rather than completely eliminating the cache.

Examples:

- Ten minutes: 600
- One hour: 3600

See the Caching section below for detailed information on how the cache operates.

Directory Settings

The screenshot shows the ownCloud configuration interface with the 'Groups' tab selected. The 'Directory Settings' section is expanded, displaying the following configuration:

- User Display Name Field: displayname
- 2nd User Display Name Field: (empty)
- Base User Tree: (empty) with a 'Base User Tree' button
- User Search Attributes: Optional; one attribute per line (empty)
- Group Display Name Field: cn
- Base Group Tree: (empty)
- Group Search Attributes: Optional; one attribute per line (empty)
- Group-Member association: uniqueMember
- Dynamic Group Member URL: (empty)
- Nested Groups:
- Paging chunksize: 500

Below the configuration area, there are buttons for 'Special Attributes', 'Test Configuration', and 'Help'.

User Display Name Field:

The attribute that should be used as display name in ownCloud.

Examples:

- displayName
- givenName
- sn

2nd User Display Name Field:

An optional second attribute displayed in brackets after the display name, for example using the `mail` attribute displays as `Molly Foo (molly@example.com)`.

Examples:

- `mail`
- `userPrincipalName`
- `sAMAccountName`

Base User Tree:

The base DN of LDAP, from where all users can be reached. This must be a complete DN, regardless of what you have entered for your Base DN in the Basic setting. You can specify multiple base trees, one on each line.

Examples:

- `cn=programmers,dc=my-company,dc=com`
- `cn=designers,dc=my-company,dc=com`

User Search Attributes:

These attributes are used when searches for users are performed, for example in the share dialogue. The user display name attribute is the default. You may list multiple attributes, one per line.

If an attribute is not available on a user object, the user will not be listed, and will be unable to login. This also affects the display name attribute. If you override the default you must specify the display name attribute here.

Examples:

- `displayName`
- `mail`

Group Display Name Field:

The attribute that should be used as ownCloud group name. ownCloud allows a limited set of characters (`a-zA-Z0-9 .-_@`). Once a group name is assigned it cannot be changed.

Examples:

- `cn`

Base Group Tree:

The base DN of LDAP, from where all groups can be reached. This must be a complete DN, regardless of what you have entered for your Base DN in the Basic setting. You can specify multiple base trees, one in each line.

Examples:

- `cn=barcelona,dc=my-company,dc=com`
- `cn=madrid,dc=my-company,dc=com`

Group Search Attributes:

These attributes are used when a search for groups is done, for example in the share dialogue. By default the group display name attribute as specified above is used. Multiple attributes can be given, one in each line.

If you override the default, the group display name attribute will not be taken into account, unless you specify it as well.

Examples:

- `cn`
- `description`

Group Member association:

The attribute that is used to indicate group memberships, i.e., the attribute used by LDAP groups to refer to their users.

ownCloud detects the value automatically. You should only change it if you have a very valid reason and know what you are doing.

Examples:

- member with FDN for Active Directory or for objectclass groupOfNames groups
- memberUid with RDN for objectclass posixGroup groups
- uniqueMember with FDN for objectclass groupOfUniqueNames groups

Note

The Group Member association is used to efficiently query users of a certain group, eg., on the userManagement page or when resolving all members of a group share.

Dynamic Group Member URL

The LDAP attribute that on group objects contains an LDAP search URL that determines what objects belong to the group. An empty setting disables dynamic group membership functionality. See [Configuring Dynamic Groups](#) for more details.

Nested Groups:

This makes the LDAP connector aware that groups could be stored inside existing group records. By default a group will only contain users, so enabling this option isn't necessary. However, if groups are contained inside groups, and this option is not enabled, any groups contained within other groups will be ignored and not returned in search results.

Paging Chunk Size:

This sets the maximum number of records able to be returned in a response when ownCloud requests data from LDAP. If this value is greater than the limit of the underlying LDAP server (such as 3000 for Microsoft Active Directory) the LDAP server will reject the request and the search request will fail. Given that, it is important to set the requested chunk size to a value no larger than that which the underlying LDAP server supports.

Special Attributes

Server	Users	Login Attributes	Groups	Advanced	Expert
--------	-------	------------------	--------	----------	--------

► Connection Settings

► Directory Settings

▼ Special Attributes

Quota Field	
Quota Default	
Email Field	
User Home Folder Naming Rule	

[Test Configuration](#) [Help](#)

Quota Field:

The name of the LDAP attribute to retrieve the user quota limit from, e.g., ownCloudQuota. Note: any quota set in LDAP overrides quotas set in ownCloud's user management page.

Quota Default:

Override ownCloud's default quota for *LDAP* users who do not have a quota set in the Quota Field, e.g., 15 GB. Please bear in mind the following, when using these fields to assign user quota limits. It should help to alleviate any, potential, confusion.

1. After installation ownCloud uses an unlimited quota by default.
2. Administrators can modify this value, at any time, in the user management page.
3. However, when an LDAP quota is set it will override any values set in ownCloud.
4. If an LDAP per/attribute quota is set, it will override the LDAP Quota Default value.

Note

Administrators are not allowed to modify the user quota limit in the user management page when steps 3 or 4 are in effect. At this point, updates are only possible via LDAP.

See the [LDAP Schema for OwnCloud Quota](#)

Email Field:

Set the user's email from an LDAP attribute, e.g., `mail`. Leave it empty for default behavior.

User Home Folder Naming Rule:

By default, the ownCloud server creates the user directory in your ownCloud data directory and gives it the ownCloud username, e.g., `/var/www/owncloud/data/5a9df029-322d-4676-9c80-9fc8892c4e4b`, if your data directory is set to `/var/www/owncloud/data`.

It is possible to override this setting and name it after an LDAP attribute value, e.g., `attr:cn`. The attribute can return either an absolute path, e.g., `/mnt/storage43/alice`, or a relative path which must not begin with a `/`, e.g., `CloudUsers/CookieMonster`. This relative path is then created inside the data directory (e.g., `/var/www/owncloud/data/CloudUsers/CookieMonster`).

Since ownCloud 8.0.10 and up the home folder rule is enforced. This means that once you set a home folder naming rule (get a home folder from an LDAP attribute), it must be available for all users. If it isn't available for a user, then that user will not be able to login. Also, the filesystem will not be set up for that user, so their file shares will not be available to other users. For older versions you may enforce the home folder rule with the `occ` command, like this example on Ubuntu:

```
sudo -u www-data php occ config:app:set user_ldap enforce_home_folder_naming_rule --value
```

Since ownCloud 10.0 the home folder naming rule is only applied when first provisioning the user. This prevents data loss due to re-provisioning the users home folder in case of unintentional changes in LDAP.

Expert Settings

Server	Users	Login Attributes	Groups	Advanced	Expert
--------	-------	------------------	--------	----------	--------

Internal Username

By default the internal username will be created from the UUID attribute. It makes sure that the username is unique and characters do not need to be converted. The internal username has the restriction that only these characters are allowed: [\a-\zA-\Z0-\9_.@-]. Other characters are replaced with their ASCII correspondence or simply omitted. On collisions a number will be added/increased. The internal username is used to identify a user internally. It is also the default name for the user home folder. It is also a part of remote URLs, for instance for all *DAV services. With this setting, the default behavior can be overridden. To achieve a similar behavior as before ownCloud 5 enter the user display name attribute in the following field. Leave it empty for default behavior. Changes will have effect only on newly mapped (added) LDAP users.

Internal Username

Attribute:

Override UUID detection

By default, the UUID attribute is automatically detected. The UUID attribute is used to doubtlessly identify LDAP users and groups. Also, the internal username will be created based on the UUID, if not specified otherwise above. You can override the setting and pass an attribute of your choice. You must make sure that the attribute of your choice can be fetched for both users and groups and it is unique. Leave it empty for default behavior. Changes will have effect only on newly mapped (added) LDAP users and groups.

UUID Attribute for Users:

UUID Attribute for Groups:

Username-LDAP User Mapping

Usernames are used to store and assign (meta) data. In order to precisely identify and recognize users, each LDAP user will have an internal username. This requires a mapping from username to LDAP user. The created username is mapped to the UUID of the LDAP user. Additionally the DN is cached as well to reduce LDAP interaction, but it is not used for identification. If the DN changes, the changes will be found. The internal username is used all over. Clearing the mappings will have leftovers everywhere. Clearing the mappings is not configuration sensitive, it affects all LDAP configurations! Never clear the mappings in a production environment, only in a testing or experimental stage.

Warning

In the Expert Settings fundamental behavior can be adjusted to your needs. The configuration should be well-tested before starting production use.

Internal Username:

The internal username is the identifier in ownCloud for LDAP users. By default it will be created from the UUID attribute. The UUID attribute ensures that the username is unique, and that characters do not need to be converted. Only these characters are allowed: [\a-\zA-\Z0-\9_.@-]. Other characters are replaced with their ASCII equivalents, or are simply omitted.

The LDAP backend ensures that there are no duplicate internal usernames in ownCloud, i.e., that it is checking all other activated user backends (including local ownCloud users). On collisions a random number (between 1000 and 9999) will be attached to the retrieved value. For example, if "alice" exists, the next username may be "alice_1337".

The internal username is the default name for the user home folder in ownCloud. It is also a part of remote URLs, for instance for all *DAV services.

You can override all of this with the Internal Username setting. Leave it empty for default behavior. Changes will affect only newly mapped LDAP users.

Examples:

- uid

Override UUID detection

By default, ownCloud auto-detects the UUID attribute. The UUID attribute is used to uniquely identify LDAP users and groups. The internal username will be created based on the UUID, if not specified otherwise.

You can override the setting and pass an attribute of your choice. You must make sure that the attribute of your choice can be fetched for both users and groups and it is unique. Leave it empty for default behavior. Changes will have effect only on newly mapped LDAP users and groups.

It also will have effect when a user's or group's DN changes and an old UUID was cached, which will result in a new user. Because of this, the setting should be applied before putting ownCloud in production use and clearing the bindings (see the User and Group Mapping section below).

Examples:

- cn

Username-LDAP User Mapping

ownCloud uses usernames as keys to store and assign data. In order to precisely identify and recognize users, each LDAP user will have a internal username in ownCloud. This requires a mapping from ownCloud username to LDAP user. The created username is mapped to the UUID of the LDAP user. Additionally the DN is cached as well to reduce LDAP interaction, but it is not used for identification. If the DN changes, the change will be detected by ownCloud by checking the UUID value.

The same is valid for groups. The internal ownCloud name is used all over in ownCloud. Clearing the Mappings will have leftovers everywhere. Never clear the mappings in a production environment, but only in a testing or experimental server.

Clearing the mappings is not configuration sensitive, it affects all LDAP configurations!

Testing the configuration

The “**Test Configuration**” button checks the values as currently given in the input fields. You do not need to save before testing. By clicking on the button, ownCloud will try to bind to the ownCloud server using the settings currently given in the input fields. If the binding fails you'll see a yellow banner with the error message:

“The configuration is invalid. Please have a look at the logs for further details.”

When the configuration test reports success, save your settings and check if the users and groups are fetched correctly on the Users page.

Syncing Users

While users who match the login and user filters can log in, only synced users will be found in the sharing dialog. Whenever users log in their display name, email, quota, avatar and search attributes will be synced to ownCloud. If you want to keep the metadata up to date you can set up a cron job, using the occ command. Versions of ownCloud before 10.0 imported all users when the users page was loaded, but this is no longer the case.

We recommend creating a Cron job, to automate regularly syncing LDAP users with your ownCloud database.

How Often Should the Job Run?

This depends on the amount of users and speed of the update, but we recommend *at least* once per day. You can run it more frequently, but doing so may generate too much load on the server.

ownCloud Avatar integration

ownCloud supports user profile pictures, which are also called avatars. If a user has a photo stored in the jpegPhoto or thumbnailPhoto attribute on your LDAP server, it will be used as their avatar. In this case the user cannot alter their avatar (on their Personal page) as it must be changed in LDAP. jpegPhoto is preferred over thumbnailPhoto.

Profile picture



Your avatar is provided by your original account.

If the `jpegPhoto` or `thumbnailPhoto` attribute is not set or empty, then users can upload and manage their avatars on their ownCloud Personal pages. Avatars managed in ownCloud are not stored in LDAP.

The `jpegPhoto` or `thumbnailPhoto` attribute is fetched once a day to make sure the current photo from LDAP is used in ownCloud. LDAP avatars override ownCloud avatars, and when an LDAP avatar is deleted then the most recent ownCloud avatar replaces it.

Photos served from LDAP are automatically cropped and resized in ownCloud. This affects only the presentation, and the original image is not changed.

Troubleshooting, Tips and Tricks

SSL Certificate Verification (LDAPS, TLS)

A common mistake with SSL certificates is that they may not be known to PHP. If you have trouble with certificate validation make sure that

- You have the certificate of the server installed on the ownCloud server
- The certificate is announced in the system's LDAP configuration file, usually `/etc/ldap/ldap.conf`.
- Using LDAPS, also make sure that the port is correctly configured (by default 636)
- If you get the error "Lost connection to LDAP server" or "**No connection to LDAP server**" double check the connection parameters and try connecting to LDAP with tools like `ldapsearch`. If using ldaps or TLS make sure the certificate is readable by the user that is used to serve ownCloud.

Microsoft Active Directory

Compared to earlier ownCloud versions, no further tweaks need to be done to make ownCloud work with Active Directory. ownCloud will automatically find the correct configuration in the set-up process.

memberOf / Read MemberOf permissions

If you want to use `memberOf` within your filter you might need to give your querying user the permissions to use it. For Microsoft Active Directory this is described [here](#).

Duplicating Server Configurations

In case you have a working configuration and want to create a similar one or "snapshot" configurations before modifying them you can do the following:

1. Go to the "**Server**" tab
2. On "**Server Configuration**" choose "**Add Server Configuration**"
3. Answer the question "**Take over settings from recent server configuration?**" with "**yes**".
4. (optional) Switch to "**Advanced**" tab and uncheck "**Configuration Active**" in the "**Connection Settings**", so the new configuration is not used on Save
5. Click on "**Save**"

Now you can modify and enable the configuration.

Performance tips

Caching

Using caching to speed up lookups. See Memory Caching. The ownCloud cache is populated on demand, and remains populated until the “**Cache Time-To-Live**” for each unique request expires. User logins are not cached, so if you need to improve login times set up a slave LDAP server to share the load.

You can adjust the “**Cache Time-To-Live**” value to balance performance and freshness of LDAP data. All LDAP requests will be cached for 10 minutes by default, and you can alter this with the “**Cache Time-To-Live**” setting. The cache answers each request that is identical to a previous request, within the time-to-live of the original request, rather than hitting the LDAP server.

The “**Cache Time-To-Live**” is related to each single request. After a cache entry expires there is no automatic trigger for re-populating the information, as the cache is populated only by new requests, for example by opening the User administration page, or searching in a sharing dialog.

There is one trigger which is automatically triggered by a certain background job which keeps the user-group-mappings up-to-date, and always in cache.

Under normal circumstances, all users are never loaded at the same time. Typically the loading of users happens while page results are generated, in steps of 30 until the limit is reached or no results are left. For this to work on an oC-Server and LDAP-Server, “**Paged Results**” must be supported, which assumes PHP >= 5.6.

ownCloud remembers which user belongs to which LDAP-configuration. That means each request will always be directed to the right server unless a user is defunct, for example due to a server migration or unreachable server. In this case the other servers will also receive the request.

LDAP indexing

Turn on indexing. Deciding which attributes to index depends on your configuration and which LDAP server you are using. See [The openLDAP tuning guide](#) for openLDAP, and [How to Index an Attribute in Active Directory](#) for Active Directory.

Use precise base DNs

The more precise your base DN, the faster LDAP can search because it has fewer branches to search.

Use precise filters

Use good filters to further define the scope of LDAP searches, and to intelligently direct your server where to search, rather than forcing it to perform needlessly-general searches.

ownCloud LDAP Internals

Some parts of how the LDAP backend works are described here.

User and Group Mapping

In ownCloud the user or group name is used to have all relevant information in the database assigned. To work reliably a permanent internal user name and group name is created and mapped to the LDAP DN and UUID. If the DN changes in LDAP it will be detected, and there will be no conflicts.

Those mappings are done in the database table `ldap_user_mapping` and `ldap_group_mapping`. The user name is also used for the user's folder (except if something else is specified in *User Home Folder Naming Rule*), which contains files and meta data.

As of ownCloud 5 the internal user name and a visible display name are separated. This is not the case for group names, yet, i.e., a group name cannot be altered.

That means that your LDAP configuration should be good and ready before putting it into production. The mapping tables are filled early, but as long as you are testing, you can empty the tables any time. Do not do this in production.

Handling with Backup Server

When ownCloud is not able to contact the main LDAP server, ownCloud assumes it is offline and will not try to connect again for the time specified in "Cache Time-To-Live". If you have a backup server configured ownCloud will connect to it instead. When you have scheduled downtime, check "Disable Main Server" to avoid unnecessary connection attempts.

User Provisioning API

The Provisioning API application enables a set of APIs that external systems can use to create, edit, delete and query user attributes, query, set and remove groups, set quota and query total storage used in ownCloud. Group admin users can also query ownCloud and perform the same functions as an admin for groups they manage. The API also enables an admin to query for active ownCloud applications, application info, and to enable or disable an app remotely. HTTP requests can be used via a Basic Auth header to perform any of the functions listed above. The Provisioning API app is enabled by default.

The base URL for all calls to the share API is `owncloud_base_url/ocs/v1.php/cloud`.

Instruction Set For Users

users / adduser

Create a new user on the ownCloud server. Authentication is done by sending a basic HTTP authentication header.

Syntax: `ocs/v1.php/cloud/users`

- HTTP method: POST
- POST argument: userid - string, the required username for the new user
- POST argument: password - string, the required password for the new user

Status codes:

- 100 - successful
- 101 - invalid input data
- 102 - username already exists
- 103 - unknown error occurred whilst adding the user

Example

- POST `http://admin:secret@example.com/ocs/v1.php/cloud/users` -d `userid="Frank" -d password="frankspassword"`
- Creates the user Frank with password `frankspassword`

XML Output

```
<?xml version="1.0"?>
<ocs>
<meta>
<status>ok</status>
<statuscode>100</statuscode>
<message/>
</meta>
<data/>
</ocs>
```

users / getusers

Retrieves a list of users from the ownCloud server. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: ocs/v1.php/cloud/users

- HTTP method: GET
- url arguments: search - string, optional search string
- url arguments: limit - int, optional limit value
- url arguments: offset - int, optional offset value

Status codes:

- 100 - successful

Example

- GET `http://admin:secret@example.com/ocs/v1.php/cloud/users?search=Frank`
- Returns list of users matching the search string.

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data>
    <users>
      <element>Frank</element>
    </users>
  </data>
</ocs>
```

users /getuser

Retrieves information about a single user. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: ocs/v1.php/cloud/users/{userid}

- HTTP method: GET

Status codes:

- 100 - successful

Example

- GET `http://admin:secret@example.com/ocs/v1.php/cloud/users/Frank`
- Returns information on the user Frank

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <status>ok</status>
    <statuscode>100</statuscode>
    <message/>
  </meta>
  <data>
    <enabled>true</enabled>
    <quota>
```

Configuration

```
<free>81919008768</free>
<used>5809166</used>
<total>81924817934</total>
<relative>0.01</relative>
</quota>
<email>user@example.com</email>
<displayname>Frank</displayname>
<home>/mnt/data/files/Frank</home>
<two_factor_auth_enabled>false</two_factor_auth_enabled>
</data>
</ocs>
```

users / edituser

Edits attributes related to a user. Users are able to edit email, displayname and password; admins can also edit the quota value. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/users/{userid}`

- HTTP method: PUT
- PUT argument: key, the field to edit (email, quota, display, password)
- PUT argument: value, the new value for the field

Status codes:

- 100 - successful
- 101 - user not found
- 102 - invalid input data

Examples

- `PUT http://admin:secret@example.com/ocs/v1.php/cloud/users/Frank -d key="email" -d value="franksnewemail@example.org"`
Updates the email address for the user Frank
- `PUT http://admin:secret@example.com/ocs/v1.php/cloud/users/Frank -d key="quota" -d value="100MB"`
Updates the quota for the user Frank

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data/>
</ocs>
```

users / deleteuser

Deletes a user from the ownCloud server. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/users/{userid}`

- HTTP method: DELETE

Statuscodes:

- 100 - successful

Configuration

- 101 - failure

Example

- **DELETE** `http://admin:secret@example.com/ocs/v1.php/cloud/users/Frank`
- Deletes the user Frank

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data/>
</ocs>
```

users / getgroups

Retrieves a list of groups the specified user is a member of. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/users/{userid}/groups`

- HTTP method: GET

Status codes:

- 100 - successful

Example

- **GET** `http://admin:secret@example.com/ocs/v1.php/cloud/users/Frank/groups`
- Retrieves a list of groups of which Frank is a member

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data>
    <groups>
      <element>admin</element>
      <element>group1</element>
    </groups>
  </data>
</ocs>
```

users / addtogroup

Adds the specified user to the specified group. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/users/{userid}/groups`

- HTTP method: POST

Configuration

- POST argument: groupid, string - the group to add the user to
- Status codes:

- 100 - successful
- 101 - no group specified
- 102 - group does not exist
- 103 - user does not exist
- 104 - insufficient privileges
- 105 - failed to add user to group

Example

- POST http://admin:secret@example.com/ocs/v1.php/cloud/users/Frank/groups -d groupid="newgroup"
- Adds the user Frank to the group newgroup

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data/>
</ocs>
```

users / removefromgroup

Removes the specified user from the specified group. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: ocs/v1.php/cloud/users/{userid}/groups

- HTTP method: DELETE
- POST argument: groupid, string - the group to remove the user from

Status codes:

- 100 - successful
- 101 - no group specified
- 102 - group does not exist
- 103 - user does not exist
- 104 - insufficient privileges
- 105 - failed to remove user from group

Example

- DELETE http://admin:secret@example.com/ocs/v1.php/cloud/users/Frank/groups -d groupid="newgroup"
- Removes the user Frank from the group newgroup

XML Output

Configuration

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data/>
</ocs>
```

users / createsubadmin

Makes a user the subadmin of a group. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: ocs/v1.php/cloud/users/{userid}/subadmins

- HTTP method: POST
- POST argument: groupid, string - the group of which to make the user a subadmin

Status codes:

- 100 - successful
- 101 - user does not exist
- 102 - group does not exist
- 103 - unknown failure

Example

- POST `https://admin:secret@example.com/ocs/v1.php/cloud/users/Frank/subadmins -d groupid="group"`
- Makes the user Frank a subadmin of the group group

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data/>
</ocs>
```

users / removesubadmin

Removes the subadmin rights for the user specified from the group specified. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: ocs/v1.php/cloud/users/{userid}/subadmins

- HTTP method: DELETE
- DELETE argument: groupid, string - the group from which to remove the user's subadmin rights

Status codes:

- 100 - successful
- 101 - user does not exist
- 102 - user is not a subadmin of the group / group does not exist
- 103 - unknown failure

Example

- `DELETE https://admin:secret@example.com/ocs/v1.php/cloud/users/Frank/subadmins -d groupid="oldgroup"`
- Removes Frank's subadmin rights from the oldgroup group

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data/>
</ocs>
```

users / getsubadmingroups

Returns the groups in which the user is a subadmin. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/users/{userid}/subadmins`

- HTTP method: GET

Status codes:

- 100 - successful
- 101 - user does not exist
- 102 - unknown failure

Example

- `GET https://admin:secret@example.com/ocs/v1.php/cloud/users/Frank/subadmins`
- Returns the groups of which Frank is a subadmin

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <status>ok</status>
    <statuscode>100</statuscode>
    <message/>
  </meta>
  <data>
    <element>testgroup</element>
  </data>
</ocs>
```

Instruction Set For Groups

groups / getgroups

Retrieves a list of groups from the ownCloud server. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/groups`

Configuration

- HTTP method: GET
- url arguments: search - string, optional search string
- url arguments: limit - int, optional limit value
- url arguments: offset - int, optional offset value

Status codes:

- 100 - successful

Example

- GET `http://admin:secret@example.com/ocs/v1.php/cloud/groups?search=adm`
- Returns list of groups matching the search string.

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data>
    <groups>
      <element>admin</element>
    </groups>
  </data>
</ocs>
```

groups / addgroup

Adds a new group. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: ocs/v1.php/cloud/groups

- HTTP method: POST
- POST argument: `groupid`, string - the new groups name

Status codes:

- 100 - successful
- 101 - invalid input data
- 102 - group already exists
- 103 - failed to add the group

Example

- POST `http://admin:secret@example.com/ocs/v1.php/cloud/groups -d groupid="newgroup"`
- Adds a new group called `newgroup`

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
```

Configuration

```
<status>ok</status>
</meta>
<data/>
</ocs>
```

groups / getgroup

Retrieves a list of group members. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: ocs/v1.php/cloud/groups/{groupid}

- HTTP method: GET

Status codes:

- 100 - successful

Example

- POST `http://admin:secret@example.com/ocs/v1.php/cloud/groups/admin`
- Returns a list of users in the admin group

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data>
    <users>
      <element>Frank</element>
    </users>
  </data>
</ocs>
```

groups / getsubadmins

Returns subadmins of the group. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: ocs/v1.php/cloud/groups/{groupid}/subadmins

- HTTP method: GET

Status codes:

- 100 - successful
- 101 - group does not exist
- 102 - unknown failure

Example

- GET `https://admin:secret@example.com/ocs/v1.php/cloud/groups/mygroup/subadmins`
- Return the subadmins of the group: mygroup

XML Output

```
<?xml version="1.0"?>
<ocs>
```

Configuration

```
<meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
</meta>
<data>
  <element>Tom</element>
</data>
</ocs>
```

groups / deletegroup

Removes a group. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/groups/{groupid}`

- HTTP method: DELETE

Status codes:

- 100 - successful
- 101 - group does not exist
- 102 - failed to delete group

Example

- `DELETE http://admin:secret@example.com/ocs/v1.php/cloud/groups/mygroup`
- Delete the group `mygroup`

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data/>
</ocs>
```

Instruction Set For Apps

apps / getapps

Returns a list of apps installed on the ownCloud server. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/apps/`

- HTTP method: GET
- url argument: filter, string - optional (enabled or disabled)

Status codes:

- 100 - successful
- 101 - invalid input data

Example

Configuration

- GET `http://admin:secret@example.com/ocs/v1.php/cloud/apps?filter=enabled`
- Gets enabled apps

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data>
    <apps>
      <element>files</element>
      <element>provisioning_api</element>
    </apps>
  </data>
</ocs>
```

apps / getappinfo

Provides information on a specific application. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/apps/{appid}`

- HTTP method: GET

Status codes:

- 100 - successful

Example

- GET `http://admin:secret@example.com/ocs/v1.php/cloud/apps/files`
- Get app info for the files app

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
  <data>
    <info/>
    <remote>
      <files>appinfo/remote.php</files>
      <webdav>appinfo/remote.php</webdav>
      <filesync>appinfo/filesync.php</filesync>
    </remote>
    <public/>
    <id>files</id>
    <name>Files</name>
    <description>File Management</description>
    <licence>AGPL</licence>
    <author>Robin Appelman</author>
    <require>4.9</require>
    <shipped>true</shipped>
  </data>
</ocs>
```

Configuration

```
<standalone></standalone>
<default_enable></default_enable>
<types>
  <element>filesystem</element>
</types>
</data>
</ocs>
```

apps / enable

Enable an app. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/apps/{appid}`

- HTTP method: POST

Status codes:

- 100 - successful

Example

- POST `http://admin:secret@example.com/ocs/v1.php/cloud/apps/files_texteditor`
- Enable the `files_texteditor` app

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
</ocs>
```

apps / disable

Disables the specified app. Authentication is done by sending a Basic HTTP Authorization header.

Syntax: `ocs/v1.php/cloud/apps/{appid}`

- HTTP method: DELETE

Status codes:

- 100 - successful

Example

- DELETE `http://admin:secret@example.com/ocs/v1.php/cloud/apps/files_texteditor`
- Disable the `files_texteditor` app

XML Output

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <statuscode>100</statuscode>
    <status>ok</status>
  </meta>
</ocs>
```

ownCloud Roles

ownCloud supports eight user roles. These are:

- [Anonymous](#)
- [Guest](#)
- [Standard User](#)
- [Federated User](#)
- [ownCloud Group Administrator](#)
- [ownCloud Administrator](#)
- [System Administrator](#)
- [Auditor](#)

The following information is not an in-depth guide, but more of a high-level overview of each type.

Anonymous

- Is not a regular user.
- Has access to specific content made available via public links. - Can be password-protected (optional, enforced, policy-enforced). - Can have an expiration date (optional, enforced, enforced dependent on password).
- Has no personal space
- Has no file ownership (ownership of uploaded/created files is directed to sharer).
- Has no use of clients.
- Quota is that of the sharer.
- Permissions are those granted by the sharer for specific content, e.g., *view-only*, *edit*, and *File Drop*.
- Can only use file and viewer apps, such as [PDF Viewer](#) and [Collabora Online](#).

Guest

- Is a regular user with restricted permissions, identified via e-mail address.
- Has no personal space.
- Has no file ownership (ownership of uploaded/created files is directed to sharer).
- Has access to shared space. The permissions are granted by the sharer.
- Is not bound to the inviting user.
 - Can log in as long as shares are available.
 - Becomes deactivated when no shares are left; this is the shared with guests filter.
 - Reactivated when a share is received.
 - Administrators will be able to automate user cleanup (“**disabled for x days**”).
- Can use all clients.
- Fully auditable in the enterprise edition.
- Can be promoted to group administrator or administrator, but will still have no personal space.
- Apps are specified by the admin (whitelist).

Note

The Shared with Guests Filter

This filter makes it easy for sharers to view and remove their shares with a guest, which also removes their responsibility for guests. When all of a guest's shares are removed, the guest is then disabled and can no longer login.

Standard User

- Is a regular user (from LDAP, ownCloud user backend, or another backend).
- Has personal space. Permissions are granted by the administrator.
- Shared space: Permissions as granted by sharer.
- Apps: All enabled, might be restricted by group membership.

Federated User

- Is not an internal user.
- Can trust a federated system.
- Has access to shared space through users on the considered ownCloud system.
- Can share data with the considered system (accept-/rejectable).

ownCloud Group Administrator

- Is a regular user, such as from LDAP, an ownCloud user backend, or another backend.
- Can manage users in their groups, such as adding and removing them, and changing quota of users in the group.
- Can add new users to their groups and can manage guests.
- Can enable and disable users.
- Can impersonate users in their groups.
- Custom group creation may be restricted to group admins.

ownCloud Administrator

- Is a regular user (from LDAP, ownCloud user backend, or another backend).
- Can configure ownCloud features via the UI, such as sharing settings, app-specific configurations, and external storages for users.
- Can manage users, such as adding and removing, enabling and disabling, quota and group management.
- Can restrict app usage to groups, where applicable.
- Configurable access to log files.
- Mounting of external shares and local shares (of external file systems) is disabled by default.

System Administrator

- Is not an ownCloud user.
- Has access to ownCloud code (e.g., config.php and apps folders) and command-line tool (occ).
- Configures and maintains the ownCloud environment (*PHP, Webserver, DB, Storage, Redis, Firewall, Cron, and LDAP*, etc.).

Maintenance

- Maintains ownCloud, such as updates, backups, and installs extensions.
- Can manage users and groups, such as via occ.
- Has access to the master key when storage encryption is used.
- **Storage admin:** Encryption at rest, which prevents the storage administrator from having access to data stored in ownCloud.
- **DB admin:** Calendar/Contacts etc. DB entries not encrypted.

Auditor

- Is not an ownCloud user.
- Conducts usage and compliance audits in enterprise scenarios.
- App logs (especially [Auditlog](#)) can be separated from ownCloud log. This separates the Auditor and Sysadmin roles. An audit.log file can be enabled, which the Sysadmin can't access.
- **Best practice:** parse separated log to an external analyzing tool.

Maintenance

Maintenance Mode Configuration

You must put your ownCloud server into maintenance mode before performing upgrades, and for performing troubleshooting and maintenance. Please see [Using the occ Command](#) to learn how to put your server into the various maintenance modes (`maintenance:mode`, `maintenance:singleuser`, and `maintenance:repair`) with the `occ` command.

`maintenance:mode` locks the sessions of logged-in users and prevents new logins. This is the mode to use for upgrades. You must run `occ` as the HTTP user, like this example on Ubuntu Linux:

```
$ sudo -u www-data php occ maintenance:mode --on
```

You may also put your server into this mode by editing config/config.php. Change "`maintenance`" => `false` to "`maintenance`" => `true`:

```
<?php  
"maintenance" => true,
```

Then change it back to `false` when you are finished.

Backing up ownCloud

When you backup your ownCloud server, there are four things that you need to copy:

1. Your config/ directory.
2. Your data/ directory.
3. Your ownCloud database.
4. Your custom theme files, if you have any. (See [Theming ownCloud](#))

When you install your ownCloud server from our [Open Build Service](#) packages (or from distro packages, which we do not recommend) **do not backup your ownCloud server files**, which are the other files in your `owncloud/` directory such as `core/`, `3rdparty/`, `apps/`, `lib/`, and all the rest of the ownCloud files. If you restore these files from backup they may not be in sync with the current package versions, and will fail the code integrity check. This may also cause other errors, such as white pages.

When you install ownCloud from the source tarballs this will not be an issue, and you can safely backup your entire ownCloud installation, with the exception of your ownCloud database. Databases cannot be copied, but you must use the database tools to make a correct database dump.

To restore your ownCloud installation from backup, see [Restoring ownCloud](#).

Backing Up the config/ and data/ Directories

Simply copy your config/ and data/ folder to a place outside of your ownCloud environment. This example uses rsync to copy the two directories to /oc-backupdir:

```
rsync -Aax config data /oc-backupdir/
```

There are many ways to backup normal files, and you may use whatever method you are accustomed to.

Backup Database

You can't just copy a database, but must use the database tools to make a correct database dump.

MySQL/MariaDB

MySQL or MariaDB, which is a drop-in MySQL replacement, is the recommended database engine. To backup MySQL/MariaDB:

```
mysqldump --single-transaction -h [server] -u [username] -p [password] [db_name] > owncloud-db.bak
```

Example:

```
mysqldump --single-transaction -h localhost -u username -p password owncloud > owncloud-db.bak
```

SQLite

```
sqlite3 data/owncloud.db .dump > owncloud-dbbackup_`date +"%Y%m%d" `.bak
```

PostgreSQL

```
PGPASSWORD="password" pg_dump [db_name] -h [server] -U [username] -f owncloud-dbbackup_`date +"%Y%m%d" `.bak
```

Restoring Files From Backup When Encryption Is Enabled

If you need to restore files from backup, which were backed up when encryption was enabled, here's how to do it.

Note

This is effective from at least version v8.2.7 of ownCloud onwards. Also, this is **not officially supported**. ownCloud officially supports either restoring the full backup or restoring nothing — not restoring individual parts of it.

1. Restore the file from backup.
2. Restore the file's encryption keys from backup.
3. Run `occ files:scan`; this makes the scanner find it. Note that, in the DB it will (1) have the "size" set to the encrypted size, which is wrong (and bigger) and (2) the "encrypted" flag will be set to 0.
4. Update the "encrypted" flag to 1 in the DB to all `files` under `files/path`, but **not** directories. Setting the flag to 1 tells the encryption application that the file is encrypted and needs to be processed.

Note

There's no need to update the encrypted flag for files in either "files_versions" or "files_trashbin", because these aren't scanned or found by `occ files:scan`.

5. Download the file once as the user; the file's size will be corrected automatically.

This process might not be suitable across all environments. If it's not suitable for yours, you might need to run an OCC command that does the scanning. But, that will require the user's password or recovery key.

How to Upgrade Your ownCloud Server

We recommend that you keep your ownCloud server up to date. When an update is available for your ownCloud server, you will see a notification at the top of your ownCloud Web interface. When you click the notification, it will bring you here.

Before beginning an upgrade, please keep the following points in mind:

- Review the release notes for important information about the needed migration steps during that upgrade to help ensure a smooth upgrade process.
- Skipping major releases is not supported. However *you can* migrate from 9.0.9 straight to 10.0.
- Downgrading is not supported.
- Upgrading is disruptive, as your ownCloud server will be put into maintenance mode.
- Large installations may take several hours to complete the upgrade.
- Downgrading **is not supported** as it risks corrupting your data. If you want to revert to an older ownCloud version, make a new, fresh installation and then restore your data from backup. Before doing this, file a support ticket (if you have paid support) or ask for help in the ownCloud forums to resolve your issue without downgrading.

Prerequisites

We strongly recommend that you always maintain regular backups as well as make a fresh backup before every upgrade. We also recommend that you review any installed third-party apps for compatibility with the new ownCloud release. Ensure that they are all disabled before beginning the upgrade. After the upgrade is complete re-enable any which are compatible with the new release.

Warning

Install unsupported apps at your own risk.

Upgrade Options

There are three ways to upgrade your ownCloud server:

1. **(Recommended)** Perform a manual upgrade, using the latest ownCloud release.
2. Use your distribution's package manager, in conjunction with our official ownCloud repositories. **Note:** This approach should not be used unattended nor in clustered setups.
3. Use the Updater App. This is needed in scenarios where the admin does not have access to the command line. It is recommended for shared hosting environments and for users who want an easy way to track different release channels.

Note

Enterprise customers will use their Enterprise software repositories to maintain their ownCloud servers, rather than the Open Build Service. Please see [Installing & Upgrading ownCloud Enterprise Edition](#) for more information.

Upgrade ownCloud From Packages

Upgrade Quickstart

The best method for keeping ownCloud current on Linux servers is by configuring your system to use ownCloud's [Open Build Service](#) repository. Then stay current by using your Linux package manager to install fresh ownCloud packages. After installing upgraded packages you must run a few more steps to complete the upgrade. These are the basic steps to upgrading ownCloud:

Warning

Make sure that you don't skip a major release when upgrading via repositories. For example you can't upgrade from 8.1.x to 9.0.x directly as you would skip the 8.2.x major release. See [Upgrading Across Skipped Releases](#) for more information.

- Disable all third-party apps.
- Make a fresh backup.
- Upgrade your ownCloud packages.
- Run occ upgrade (The optional parameter to skip migration tests was removed from oC 9.2. See [Test the Upgrade](#) for background information).
- Apply strong permissions to your ownCloud directories.
- Take your ownCloud server out of maintenance mode.
- Re-enable third-party apps.

Warning

When upgrading from oC 9.0 to 9.1 with existing Calendars or Addressbooks please have a look at the 9.0 release notes for important information about the needed migration steps during that upgrade.

Upgrade Tips

Upgrading ownCloud from our [Open Build Service](#) repository is just like any normal Linux upgrade. For example, on Debian or Ubuntu Linux this is the standard system upgrade command:

```
apt-get update && apt-get upgrade
```

Or you can upgrade just ownCloud with this command:

```
apt-get update && apt-get install owncloud
```

On Fedora, CentOS, and Red Hat Linux use `yum` to see all available updates:

```
yum check-update
```

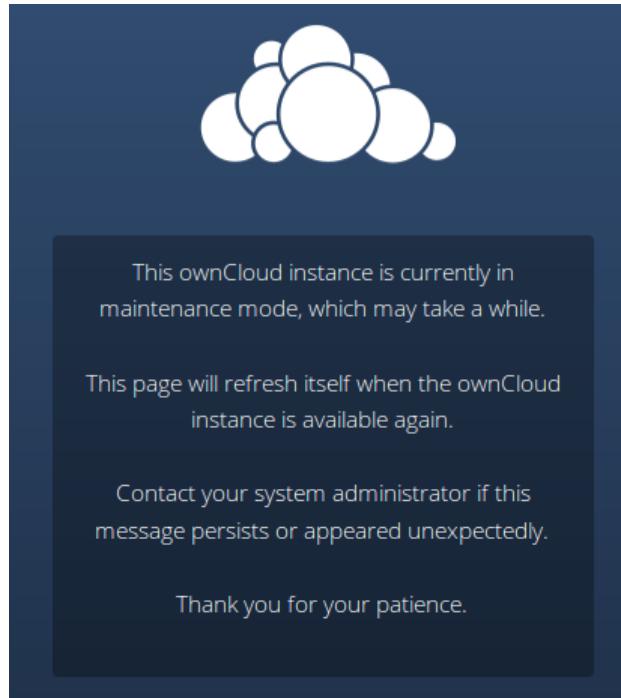
You can apply all available updates with this command:

```
yum update
```

Or update only ownCloud:

```
yum update owncloud
```

Your Linux package manager only downloads the current ownCloud packages. Then your ownCloud server is immediately put into maintenance mode. You may not see this until you refresh your ownCloud page.



Then use `occ` to complete the upgrade. You must run `occ` as your HTTP user. This example is for Debian/Ubuntu:

```
sudo -u www-data php occ upgrade
```

This example is for CentOS/RHEL/Fedora:

```
sudo -u apache php occ upgrade
```

The optional parameter to skip migration tests during this step was removed in oC 9.2. See [Test the Upgrade](#) for background information.

See [Using the occ Command](#) to learn more.

Setting Strong Directory Permissions

After upgrading, verify that your ownCloud directory permissions are set according to [Set Strong Directory Permissions](#).

Upgrading Across Skipped Releases

It is best to update your ownCloud installation with every new point release (e.g. 8.1.10), and to never skip any major release (e.g. don't skip 8.2.x between 8.1.x and 9.0.x). If you have skipped any major release you can bring your ownCloud current with these steps:

1. Add the repository of your current version (e.g. 8.1.x)
2. Upgrade your current version to the latest point release (e.g. 8.1.10) via your package manager
3. Run the `occ upgrade` routine (see [Upgrade Quickstart](#) above)
4. Add the repository of the next major release (e.g. 8.2.x)
5. Upgrade your current version to the next major release (e.g. 8.2.8) via your package manager
6. Run the `occ upgrade` routine (see [Upgrade Quickstart](#) above)
7. Repeat from step 4 until you reach the last available major release (e.g. 9.1.x)

You'll find repositories of previous ownCloud major releases in the [ownCloud Server Changelog](#).

Upgrading ownCloud with the Updater App

The Updater app automates many of the steps of upgrading an ownCloud installation. It is useful for installations that do not have root access, such as shared hosting, for installations with a smaller number of users and data, and it automates updating manual installations.

Warning

When upgrading from oC 9.0 to 9.1 with existing Calendars or Adressbooks please have a look at the Release Notes of oC 9.0 for important info about this migration.

New in 9.0, the Updater app has command-line options.

Note

The Updater app is **not enabled and not supported** in ownCloud Enterprise edition.

The Updater app is **not included** in the [Linux packages on our Open Build Service](#), but only in the [tar and zip archives](#). When you install ownCloud from packages you should keep it updated with your package manager.

Downgrading is not supported and risks corrupting your data! If you want to revert to an older ownCloud version, install it from scratch and then restore your data from backup. Before doing this, file a support ticket (if you have paid support) or ask for help in the ownCloud forums to see if your issue can be resolved without downgrading.

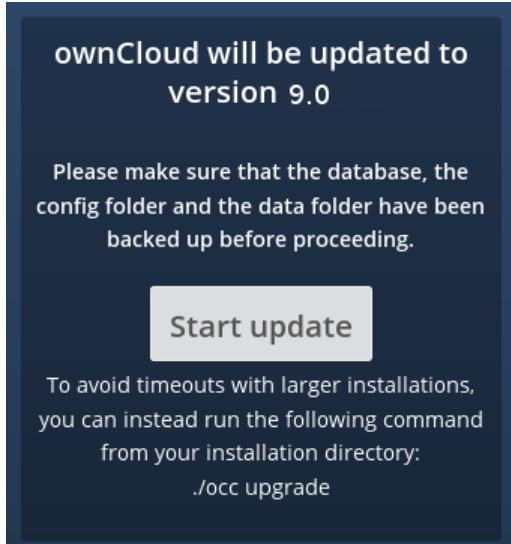
You should maintain regular backups (see Backing up ownCloud), and make a backup before every update. The Updater app does not backup your database or data directory.

The Updater app performs these operations:

- Creates an `updater_backup` directory under your ownCloud data directory
- Downloads and extracts updated package content into the `updater_backup/packageVersion` directory
- Makes a copy of your current ownCloud instance, except for your data directory, to `updater_backup/currentVersion-randomstring`
- Moves all directories except `data` and `config` from the current instance to `updater_backup/tmp`
- Moves all directories from `updater_backup/packageVersion` to the current version
- Copies your old `config.php` to the new `config/` directory

Using the Updater app to update your ownCloud installation is just a few steps:

1. You should see a notification at the top of any ownCloud page when there is a new update available.
2. Even though the Updater app backs up important directories, you should always have your own current backups (See Backing up ownCloud for details.)
3. Verify that the HTTP user on your system can write to your whole ownCloud directory; see the Setting Permissions for Updating section below.
4. Navigate to your Admin page and click the **Update Center** button under Updater. This takes you to the Updater control panel.
5. Click Update, and carefully read the messages. If there are any problems it will tell you. The most common issue is directory permissions; your HTTP user needs write permissions to your whole ownCloud directory. (See Set Strong Directory Permissions.) Another common issue is SELinux rules (see SELinux Configuration.) Otherwise you will see messages about checking your installation and making backups.
6. Click Proceed, and then it performs the remaining steps, which takes a few minutes.
7. If your directory permissions are correct, a backup was made, and downloading the new ownCloud archive succeeded you will see the following screen. Click the Start Update button to complete your update:



Note

If you have a large ownCloud installation and have shell access, you should use the `occ upgrade` command, running it as your HTTP user, instead of clicking the Start Update button, in order to avoid PHP timeouts.

This example is for Ubuntu Linux:

```
$ sudo -u www-data php occ upgrade
```

The optional parameter to skip migration tests during this step was removed in oC 9.2. See [Test the Upgrade](#) for more information.

8. It runs for a few minutes, and when it is finished displays a success message, which disappears after a short time.

Refresh your Admin page to verify your new version number. In the Updater section of your Admin page you can see the current status and backups. These are backups of your old and new ownCloud installations, and do not contain your data files. If your update works and there are no problems you can delete the backups from this screen.

If the update fails, then you must update manually. (See [Manually upgrading](#).)

Setting Permissions for Updating

For hardened security, we highly recommend setting the permissions on your ownCloud directory as strictly as possible, immediately after the initial installation. However, these strict permissions will prevent the Updater app from working, as it needs your whole ownCloud directory to be owned by the HTTP user.

So to set the appropriate permissions for updating, run the code below. Replace the `ocpath` variable with the path to your ownCloud directory, and replace the `htuser` and `htgroup` variables with your HTTP user and group.

```
#!/bin/bash
# Sets permissions of the owncloud instance for updating

ocpath='/var/www/owncloud'
htuser='www-data'
htgroup='www-data'

chown -R ${htuser}: ${htgroup} ${ocpath}
```

You can find your HTTP user in your HTTP server configuration files. Or you can use PHP Version and Information (Look for the **User/Group** line).

- The HTTP user and group in Debian/Ubuntu is `www-data`.
- The HTTP user and group in Fedora/CentOS is `apache`.

- The HTTP user and group in Arch Linux is http.
- The HTTP user in openSUSE is wwwrun, and the HTTP group is www.

After the update is completed, re-apply the strong directory permissions immediately.

Command Line Options

The Updater app includes command-line options to automate updates, to create checkpoints and to roll back to older checkpoints. You must run it as your HTTP user. This example on Ubuntu Linux displays command options:

```
sudo -u www-data php updater/application.php list
```

See usage for commands, like this example for the upgrade:checkpoint command:

```
sudo -u www-data php updater/application.php upgrade:checkpoint -h
```

You can display a help summary:

```
sudo -u www-data php updater/application.php --help
```

When you run it without options it runs a system check:

```
sudo -u www-data php owncloud/updater/application.php
ownCloud updater 1.0 - CLI based ownCloud server upgrades
Checking system health.
- file permissions are ok.
Current version is 9.0.0.12
No updates found online.
Done
```

Create a checkpoint:

```
sudo -u www-data php updater/application.php upgrade:checkpoint --create
Created checkpoint 9.0.0.12-56d5e4e004964
```

List checkpoints:

```
sudo -u www-data php updater/application.php upgrade:checkpoint --list
```

Restore an earlier checkpoint:

```
sudo -u www-data php owncloud/updater/application.php upgrade:checkpoint
--restore=9.0.0.12-56d5e4e004964
```

Add a line like this to your crontab to automatically create daily checkpoints:

```
2 15 * * * sudo -u www-data php /path/to/owncloud/updater/application.php
upgrade:checkpoint --create > /dev/null 2>&1
```

updater.secret value in config.php

When running the updater, you will be prompted to add a hashed secret into your config.php file. On the updater web interface, you then need to enter the unhashed secret into the web form.

In case you forgot your password/secret, you can re-create it by changing config.php. You can run this on your shell:

```
php -r 'echo password_hash("Enter a random password here", PASSWORD_DEFAULT). "\n";'
```

Please replace Enter a random password here with your own. Then add this into your config.php:

```
'updater.secret' => 'The value you got from the above hash command',
```

Manual ownCloud Upgrade

Note

If you're not comfortable performing a manual upgrade, you can also use your Linux distribution's package manager, or use the Updater App.

Backup Your Existing Installation

First, backup the following items:

- The ownCloud server data directory
- The config.php file
- All 3rd party apps
- The ownCloud server database

```
# This example assumes Ubuntu Linux and MariaDB
cp -rv /var/www/owncloud /opt/backup/owncloud && mysqldump <db_name> > /opt/backup/backup-fi
```

Review Third-Party Apps

Review any installed third-party apps for compatibility with the new ownCloud release. Ensure that they are all disabled before beginning the upgrade. After the upgrade is complete re-enable any which are compatible with the new release.

Warning

Install unsupported apps at your own risk.

Check ownCloud's Mandatory Requirements

ownCloud's mandatory requirements (such as PHP versions and extensions) can change from one version to the next. Ensure that you review them and update your server(s), if required, before upgrading ownCloud.

Enable Maintenance Mode

Put your server in maintenance mode and disable Cron jobs. Doing so prevents new logins, locks the sessions of logged-in users, and displays a status screen so that users know what is happening.

There are two ways to enable maintenance mode. The preferred method is to use the occ command — which you must run as your webserver user. The other way is by entering your config.php file and changing 'maintenance' => false, to 'maintenance' => true.,

```
# Enable maintenance mode using the occ command.
sudo -u www-data php occ maintenance:mode --on

# Disable Cron jobs
sudo service cron stop
```

Stop the Webserver

With those steps completed, stop your webserver.

```
sudo service apache2 stop
```

Download the Latest Installation

Download the latest ownCloud server release from owncloud.org/install/ into an empty directory **outside** of your current installation.

Note

Enterprise users must download their new ownCloud archives from their accounts on <https://customer.owncloud.com/owncloud/>.

Setup the New Installation

Not all installations are the same, so we encourage you to take one of two paths to upgrade your ownCloud installation. These are *the standard upgrade* and *the power user upgrade*.

If you're reasonably new to ownCloud, or not too familiar with upgrading an ownCloud installation, please follow the standard upgrade. Otherwise, take the approach that you're most comfortable with, likely the power user upgrade.

Note

Regardless of which approach that you take, they will both assume that your existing ownCloud installation is located in the default location: /var/www/owncloud.

The Standard Upgrade

Delete all files and folders in your existing ownCloud directory (/var/www/owncloud) — **except** data and config.

Attention!

Don't keep the apps directory.

With those files and folders deleted, extract the archive of the latest ownCloud server, over the top of your existing installation.

```
# Extract the .tar.bz2 archive  
tar -jxf owncloud-10.0.3.tar.bz2 -C /var/www/  
  
# Extract the zip archive  
unzip -q owncloud-10.0.3.zip -d /var/www/
```

The Power User Upgrade

Rename your current ownCloud directory, for example, from `owncloud` to `owncloud-old`. Extract the unpacked ownCloud server directory and its contents to the location of your original ownCloud installation.

```
# Assumes that the new release was unpacked into /tmp/  
mv /tmp/owncloud /var/www/
```

With the new source files now in place of the old ones, next copy the `config.php` file from your old ownCloud directory to your new ownCloud directory.

```
cp /var/www/owncloud-old/config/config.php /var/www/owncloud/config/config.php
```

If you keep your `data/` directory *inside* your `owncloud/` directory, copy it from your old version of ownCloud to your new version. If you keep it *outside* of your `owncloud/` directory, then you don't have to do anything with it, because its location is configured in your original `config.php`, and none of the upgrade steps touch it.

Disable Core Apps

Before the upgrade can run, several apps need to be disabled, if they're enabled, before the upgrade can succeed. These are: *activity*, *files_pdfviewer*, *files_texteditor*, and *gallery*. The following command provides an example of how to do so.

```
sudo -u www-data php occ app:disable activity
sudo -u www-data php occ app:disable files_pdfviewer
sudo -u www-data php occ app:disable files_texteditor
sudo -u www-data php occ app:disable gallery
```

Market and Marketplace App Upgrades

Before getting too far into the upgrade process, please be aware of how the Market app and its configuration options affect the upgrade process.

- The Market app is not upgraded if it is either disabled (because `appstoreenabled` is set to `false`) or it is not available.
- If `upgrade.automatic-app-update` is set to `false` apps installed from the Marketplace are not automatically upgraded.

In addition to these two points, if there are installed apps (whether compatible or incompatible with the next version, or missing source code) and the Market app is enabled, but there is no available internet connection, then these apps will need to be manually updated once the upgrade is finished.

Start the Upgrade

With the apps disabled and the webserver started, launch the upgrade process from the command line.

```
# Here is an example on CentOS Linux
sudo -u www-data php occ upgrade
```

Note

The optional parameter to skip migration tests during this step was removed in oC 10.0. See [Test the Upgrade](#) for background information. See [Using the occ Command](#) to learn more about the `occ` command.

The upgrade operation can take anywhere from a few minutes to a few hours, depending on the size of your installation. When it is finished you will see either a success message, or an error message which indicates why the process did not complete successfully.

Copy Old Apps

If you are using 3rd party applications, look in your new `/var/www/owncloud/apps/` directory to see if they are there. If not, copy them from your old `apps/` directory to your new one, and make sure that the directory permissions are the same as for the other ones.

Disable Maintenance Mode

Assuming your upgrade succeeded, next disable maintenance mode. The simplest way is by using `occ` from the command line.

```
sudo -u www-data php occ maintenance:mode --off
```

Restart the Webserver

With all that done, restart your web server.

```
sudo service apache2 start
```

Finalize the Installation

Maintenance

With maintenance mode disabled, login and:

- Re-enable cron jobs
- Check that the version number reflects the new installation. It's visible at the bottom of your Admin page.
- Check that your other settings are correct.
- Go to the Apps page and review the core apps to make sure the right ones are enabled.
- Re-enable your third-party apps.
- Apply strong permissions to your ownCloud directories.

Test the Upgrade

Previous versions of ownCloud included a migration test. ownCloud first ran a migration simulation by copying the ownCloud database and performing the upgrade on the copy, to ensure that the migration would succeed.

Then the copied tables were deleted after the upgrade was completed. This doubled the upgrade time, so admins could skip this test (by risking a failed upgrade) with `php occ upgrade --skip-migration-test`.

The migration test has been removed from ownCloud 9.2. ownCloud server admins should have current backups before migration, and rely on backups to correct any problems from the migration.

Reverse Upgrade

If you need to reverse your upgrade, see Restoring ownCloud.

Troubleshooting

When upgrading ownCloud and you are running MySQL or MariaDB with binary logging enabled, your upgrade may fail with these errors in your MySQL/MariaDB log:

An unhandled exception has been thrown:

```
exception 'PDOException' with the message 'SQLSTATE[HY000]: General error: 1665
Cannot execute statement: impossible to write to binary log since
BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine limited to row-based'
```

Please refer to MySQL / MariaDB with Binary Logging Enabled on how to correctly configure your environment.

Occasionally, *files do not show up after an upgrade*. A rescan of the files can help:

```
sudo -u www-data php console.php files:scan --all
```

See [the owncloud.org support page](#) for further resources for both home and enterprise users.

Sometimes, ownCloud can get *stuck in a upgrade*. This is usually due to the process taking too long and encountering a PHP time-out. Stop the upgrade process this way:

```
sudo -u www-data php occ maintenance:mode --off
```

Then start the manual process:

```
sudo -u www-data php occ upgrade
```

If this does not work properly, try the repair function:

```
sudo -u www-data php occ maintenance:repair
```

Restoring ownCloud

When you install ownCloud from packages, follow these steps to restore your ownCloud installation. Start with a fresh ownCloud package installation in a new, empty directory. Then restore these items from your backup (see Backing up ownCloud):

1. Your config/ directory.
2. Your data/ directory.

3. Your ownCloud database.
4. Your custom theme files, if you have any. (See [Theming ownCloud](#))

When you install ownCloud from the source tarballs you may safely restore your entire ownCloud installation from backup, with the exception of your ownCloud database. Databases cannot be copied, but you must use the database tools to make a correct restoration.

When you have completed your restoration, see [Setting Strong Permissions](#).

Restore Directories

Simply copy your configuration and data folder to your ownCloud environment. You could use this command, which restores the backup example in [Backing up ownCloud](#):

```
rsync -Aax config data /var/www/owncloud/
```

There are many ways to restore normal files from backups, and you may use whatever method you are accustomed to.

Restore Database

Note

This guide assumes that your previous backup is called “owncloud-dbbackup.bak”

MySQL

MySQL is the recommended database engine. To restore MySQL:

```
mysql -h [server] -u [username] -p[password] [db_name] < owncloud-dbbackup.bak
```

SQLite

```
rm data/owncloud.db
sqlite3 data/owncloud.db < owncloud-dbbackup.bak
```

PostgreSQL

```
PGPASSWORD="password" pg_restore -c -d owncloud -h [server] -U [username]
owncloud-dbbackup.bak
```

Migrating to a Different Server

If the need arises, ownCloud can be migrated to a different server. A typical use case would be a hardware change or a migration from the Enterprise appliance to a physical server. All migrations have to be performed with ownCloud in maintenance mode. Online migration is supported by ownCloud only when implementing industry-standard clustering and high-availability solutions **before** ownCloud is installed for the first time.

To start, let's work through a potential use case. A configured ownCloud instance runs reliably on one machine, but for some reason the instance needs to be moved to a new machine. Depending on the size of the ownCloud instance the migration might take several hours.

For the purpose of this use case, it is assumed that:

1. The end users reach the ownCloud instance via a virtual hostname (such as a DNS CNAME record) which can be pointed at the new location.
2. The authentication method (e.g., LDAP) remains the same after the migration.

Warning

During the migration, do not make any changes to the original system, except for putting it into maintenance mode. This ensures, should anything unforeseen happen, that you can go back to your existing installation and resume availability of your ownCloud installation while debugging the problem.

How to Migrate

Firstly, set up the new machine with your desired Linux distribution. At this point you can either install ownCloud manually via the compressed archive, or with your Linux package manager.

Then, on the original machine turn on maintenance mode and then stop ownCloud. After waiting for 6 - 7 minutes for all sync clients to register that the server is in maintenance mode, ref:*stop the web server <maintenance_commands_label>* that is serving ownCloud.

After that, create a database dump from the database, copy it to the new machine, and import it into the new database.

Then, copy only your data, configuration, and database files from your original ownCloud instance to the new machine (See Backing up ownCloud and Restore Directories).

Warning

You must keep the data/ directory's original file path during the migration. However, you can change it before you begin the migration, or after the migration's completed.

The data files should keep their original timestamp otherwise the clients will re-download all the files after the migration. This step might take several hours, depending on your installation. This can be done on a number of sync clients, such as by using rsync with -t option

With ownCloud still in maintenance mode and before changing the DNS CNAME record, start up the database and web server on the new machine. Then point your web browser to the migrated ownCloud instance and confirm that:

1. You see the maintenance mode notice
2. That a log file entry is written by both the web server and ownCloud
3. That no error messages occur.

If all of these things occur, then take ownCloud out of maintenance mode and repeat. After doing this, log in as an admin and confirm that ownCloud functions as normal.

At this point, change the DNS CNAME entry to point your users to the new location. And with the CNAME entry updated, you now need to update the trusted domains.

Managing Trusted Domains

All URLs used to access your ownCloud server must be whitelisted in your config.php file, under the trusted_domains setting. Users are allowed to log into ownCloud only when they point their browsers to a URL that is listed in the trusted_domains setting.

Note

This setting is important when changing or moving to a new domain name. You may use IP addresses and domain names.

A typical configuration looks like this:

```
'trusted_domains' => [  
    0 => 'localhost',
```

```
1 => 'server1.example.com',
2 => '192.168.1.50',
],
```

The loopback address, 127.0.0.1, is automatically whitelisted, so as long as you have access to the physical server you can always log in. In the event that a load-balancer is in place, there will be no issues as long as it sends the correct X-Forwarded-Host header.

Example Migration

Now, let's step through an example migration. For this example to work, you will need the following on both the servers that you will use for the migration:

- Ubuntu 16.04
- SSH with PermitRootLogin set to yes

Preparation

Before you can perform a migration, you have to prepare. To do this, first make sure SSH is installed:

```
apt install ssh -y
```

Next, edit ssh-config and enable root ssh login.

```
nano /etc/ssh/sshd_config
PermitRootLogin yes
```

And then restart SSH.

```
service ssh restart
```

Lastly, install ownCloud on the new server.

Migration

Enable Maintenance Mode

The first step is to enable maintenance mode. To do that, use the following commands:

```
cd /var/www/owncloud/
sudo -u www-data php occ maintenance:mode --on
```

After that's done, wait for 6-7 minutes and stop Apache:

```
service apache2 stop
```

Transfer the Database

Now, you have to transfer the database from the old server to the new one. To do that, first backup the database.

```
cd /var/www/owncloud/
mysqldump --single-transaction -h localhost -u admin -ppassword owncloud > owncloud-dbbackup.bak
```

Then, export the database to the new server.

```
rsync -Aaxt owncloud-dbbackup.bak root@new_server_address:/var/www/owncloud
```

With that completed, import the database on new server.

```
mysql -h localhost -u admin -ppassword owncloud < owncloud-dbbackup.bak
```

Note

You can find the values for the mysqldump command in your config.php, in your owncloud root directory. [server]= dbhost, [username]= dbuser, [password]= dbpassword, and [db_name]= dbname.

Note

For InnoDB tables only The –single-transaction flag will start a transaction before running. Rather than lock the entire database, this will let mysqldump read the database in the current state at the time of the transaction, making for a consistent data dump.

Note

For Mixed MyISAM / InnoDB tables Either dumping your MyISAM tables separately from InnoDB tables or use –lock-tables instead of –single-transaction to guarantee the database is in a consistent state when using mysqldump.

Transfer Data and Configure the New Server

```
rsync -Aavxt config data root@new_server_address:/var/www/owncloud
```

Warning

If you want to move your data directory to another location on the target server, it is advised to do this as a second step. Please see the data directory migration document How To Manually Move a Data Directory for more details.

Finish the Migration

Now it's time to finish the migration. To do that, on the new server, first verify that ownCloud is in maintenance mode.

```
sudo -u www-data php occ maintenance:mode
```

Next, start up the database and web server on the new machine.

```
service mysql start  
service apache2 start
```

With that done, point your web browser to the migrated ownCloud instance, and confirm that you see the maintenance mode notice, and that no error messages occur. If both of these occur, take ownCloud out of maintenance mode.

```
sudo -u www-data php occ maintenance:mode --off
```

And finally, log in as admin and confirm normal function of ownCloud. If you have a domain name, and you want an SSL certificate, we recommend [certbot](#).

Reverse the Changes to ssh-config

Now you need to reverse the change to ssh-config. Specifically, set `PermitRootLogin` to `no` and restart ssh. To do that, run the following command:

```
service ssh restart
```

Update DNS and Trusted Domains

Finally, update the DNS' CNAME entry to point to your new server. If you have not only migrated physically from server to server but have also changed your ownCloud server's domain name, you also need to update the domain in the Trusted Domain setting in `config.php`, on the target server.

How To Manually Move a Data Directory

If you need to move your ownCloud data directory from its current location to somewhere else, here is a manual process that you can take to make it happen.

Note

This example assumes that:

- The current folder is: `/var/www/owncloud/data`
- The new folder is: `/mnt/owncloud`
- You're using Apache as your webserver

1. Stop Apache
2. Use rsync to sync the files from the current folder to the new one
3. Create a symbolic link from the new directory to the old one
4. Double-check the [directory permissions](#) on the new directory
5. Restart Apache

To save time, here's the commands which you can copy and use:

```
apachectl -k stop
rsync -avz /var/www/owncloud/data /mnt/owncloud
ln -s /mnt/owncloud /var/www/owncloud/data
apachectl -k graceful
```

Note

If you're on CentOS/Fedora, try `systemctl restart httpd`. If you're on Debian/Ubuntu try `sudo systemctl restart apache2`. To learn more about the `systemctl` command, please refer to [the systemd essentials guide](#)

Fix Hardcoded Database Path Variables

Update the oc_storages table

If you want to manually change the location of the data folder in the database, run the SQL below:

```
UPDATE oc_storages SET id='local::/mnt/owncloud'
WHERE id='local::/var/www/owncloud/data/' ;
```

Update the oc_accounts table

You next need to update the home column in the `oc_accounts` table. This column contains the absolute path for user folders, e.g., `/mnt/data/files/admin`. Assuming that the new home directory is: `/mnt/data/files/super-admin`, and that the user's id is 1, you could change it using the following SQL statement:

```
UPDATE oc_accounts SET home=' /mnt/data/files/super-admin'
WHERE id=1;
```

Note

Please don't copy and paste this example verbatim — nor any of the others. It, and the others, are provided only as guides to what you should or could do.

Update the oc_jobs table

The next area to check is the `oc_jobs` table. The logrotate process may have hard-coded a non-standard (or old) value for the data path. To check it, run the SQL below and see if any results are returned:

```
SELECT * FROM oc_jobs
WHERE class = 'OC\Log\Rotate';
```

If any are, run the SQL below to update them, changing the value as appropriate.

```
UPDATE oc_jobs SET argument = '/your/new/data/path'
WHERE id = <id of the incorrect record>;
```

Fix Application Settings

One thing worth noting is that individual apps may reference the data directory separate from the core system configuration. If so, then you will need to find which applications do this, and change them as needed.

For example, if you listed the application configuration by running `occ config:list`, then you might see output similar to that below:

```
{
  "apps": {
    "fictitious": {
      "enabled": "yes",
      "installed_version": "2.3.2",
      "types": "filesystem",
      "datadir": "var/www/owncloud/data"
    }
  }
}
```

Here, the “fictitious” application references the data directory as being set to `var/www/owncloud/data`. So you would have to change the value by using the `config:app:set` option. Here’s an example of how you would update the setting:

```
occ config:app:set --value /mnt/owncloud fictitious datadir
```

Issues and Troubleshooting

General Troubleshooting

If you have trouble installing, configuring or maintaining ownCloud, please refer to our community support channels:

- The ownCloud Forums

Note

The ownCloud forums have a [FAQ category](#) where each topic corresponds to typical mistakes or frequently occurring issues

- [The ownCloud User mailing list](#)
- The ownCloud IRC chat channel `irc://#owncloud@freenode.net` on freenode.net, also accessible via [webchat](#)

Please understand that all these channels essentially consist of users like you helping each other out. Consider helping others out where you can, to contribute back for the help you get. This is the only way to keep a community like ownCloud healthy and sustainable!

If you are using ownCloud in a business or otherwise large scale deployment, note that ownCloud Inc. offers the [Enterprise Edition](#) with commercial support options.

Bugs

If you think you have found a bug in ownCloud, please:

- Search for a solution (see the options above)
- Double-check your configuration

If you can't find a solution, please use our [bugtracker](#). You can generate a configuration report with the `occ config` command, with passwords automatically obscured.

General Troubleshooting

Check the ownCloud System Requirements, especially supported browser versions.

When you see warnings about `code integrity`, refer to Code Signing.

Disable 3rdparty / non-shipped apps

It might be possible that 3rd party / non-shipped apps are causing various different issues. Always disable 3rd party apps before upgrades, and for troubleshooting. Please refer to the Apps Commands on how to disable an app from command line.

ownCloud Logfiles

In a standard ownCloud installation the log level is set to `Normal`. To find any issues you need to raise the log level to `All` in your `config.php` file, or to `Everything` on your ownCloud Admin page. Please see Logging Configuration for more information on these log levels.

Some logging - for example JavaScript console logging - needs debugging enabled. Edit `config/config.php` and change `'debug' => false`, to `'debug' => true`. Be sure to change it back when you are finished.

For JavaScript issues you will also need to view the javascript console. All major browsers have developer tools for viewing the console, and you usually access them by pressing F12. For Firefox we recommend to installing the Firebug extension.

Note

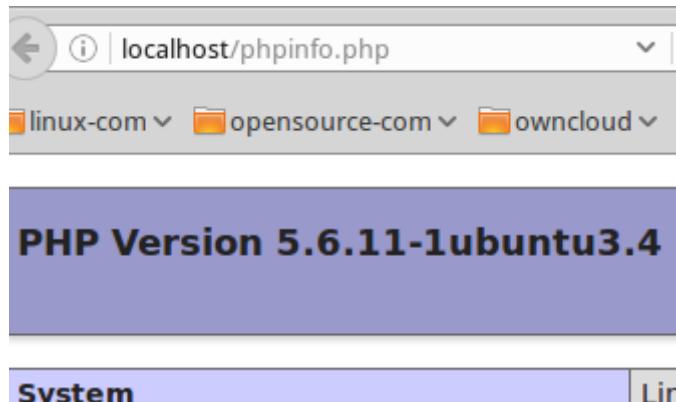
The logfile of ownCloud is located in the data directory `owncloud/data/owncloud.log`.

PHP Version and Information

You will need to know your PHP version and configurations. To do this, create a plain-text file named **phpinfo.php** and place it in your Web root, for example `/var/www/html/phpinfo.php`. (Your Web root may be in a different location; your Linux distribution documentation will tell you where.) This file contains just this line:

```
<?php phpinfo(); ?>
```

Open this file in a Web browser by pointing your browser to `localhost/phpinfo.php`:



Your PHP version is at the top, and the rest of the page contains abundant system information such as active modules, active .ini files, and much more. When you are finished reviewing your information you must delete `phpinfo.php`, or move it outside of your Web directory, because it is a security risk to expose such sensitive data.

Debugging Sync Issues

Warning

The data directory on the server is exclusive to ownCloud and must not be modified manually.

Disregarding this can lead to unwanted behaviours like:

- Problems with sync clients
- Undetected changes due to caching in the database

If you need to directly upload files from the same server please use a WebDAV command line client like `cadaver` to upload files to the WebDAV interface at:

`https://example.com/owncloud/remote.php/dav`

Common problems / error messages

Some common problems / error messages found in your logfiles as described above:

- SQLSTATE[HY000] [1040] Too many connections -> You need to increase the connection limit of your database, please refer to the manual of your database for more information.
- SQLSTATE[HY000]: General error: 5 database is locked -> You're using SQLite which can't handle a lot of parallel requests. Please consider converting to another database like described in Converting Database Type.
- SQLSTATE[HY000]: General error: 2006 MySQL server has gone away -> Please refer to Troubleshooting for more information.
- SQLSTATE[HY000] [2002] No such file or directory -> There is a problem accessing your SQLite database file in your data directory (`data/owncloud.db`). Please check the permissions of this folder/file or if it exists at all. If you're using MySQL please start your database.
- Connection closed / Operation cancelled or expected filesize 4734206 got 458752 -> This could be caused by wrong KeepAlive settings within your Apache config. Make sure that KeepAlive is set to On and also try to raise the limits of KeepAliveTimeout and MaxKeepAliveRequests. On Apache

with mod_php using a different Multi-Processing Module (MPM) then prefork could be another reason. Further information is available [in the forums](#).

- No basic authentication headers were found -> This error is shown in your data/owncloud.log file. Some Apache modules like mod_fastcgi, mod_fcgid or mod_proxy_fcgi are not passing the needed authentication headers to PHP and so the login to ownCloud via WebDAV, CalDAV and CardDAV clients is failing. More information on how to correctly configure your environment can be found [at the forums](#).

OAuth2

ownCloud clients cannot connect to the ownCloud server

If ownCloud clients cannot connect to your ownCloud server, check to see if PROPFIND requests receive HTTP/1.1 401 Unauthorized responses. If this is happening, more than likely your webserver configuration is stripping out the bearer authorization header.

If you're using the Apache web server, add the following SetEnvIf directive to your Apache configuration, whether in the general Apache config, in a configuration include file, or in ownCloud's .htaccess file.

```
SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1
```

Alternatively, if you're using NGINX, add the following configuration to your NGINX setup:

```
# Adding this allows the variable to be accessed with $_SERVER['Authorization']
fastcgi_param Authorization $http_authorization;
```

Missing Data Directory

During the normal course of operations, the ownCloud data directory may be temporarily unavailable for a variety of reasons. These can include network timeouts on mounted network disks, unintentional unmounting of the partition on which the directory sits, or a corruption of the RAID setup. If you have experienced this, here's how ownCloud works and what you can expect.

During normal operation, ownCloud's data directory contains a hidden file, named .ocdata. The purpose of this file is for setups where the data folder is mounted (such as via NFS) and for some reason the mount disappeared. If the directory isn't available, the data folder would, in effect, be completely empty and the ".ocdata" would be missing. When this happens, ownCloud will return a [503 Service not available](#) error, to prevent clients believing that the files are gone.

Troubleshooting Web server and PHP problems

Logfiles

When having issues the first step is to check the logfiles provided by PHP, the Web server and ownCloud itself.

Note

In the following the paths to the logfiles of a default Debian installation running Apache2 with mod_php is assumed. On other Web servers, Linux distros or operating systems they can differ.

- The logfile of Apache2 is located in /var/log/apache2/error.log.
- The logfile of PHP can be configured in your /etc/php5/apache2/php.ini. You need to set the directive log_errors to On and choose the path to store the logfile in the error_log directive. After those changes you need to restart your Web server.
- The logfile of ownCloud is located in the data directory /var/www/owncloud/data/owncloud.log.

Web Server and PHP Modules

Note

Lighttpd is not supported with ownCloud, and some ownCloud features may not work at all on Lighttpd.

There are some Web server or PHP modules which are known to cause various problems like broken up-/downloads. The following shows a draft overview of these modules:

1. Apache

- libapache2-mod-php5filter (use libapache2-mod-php5 instead)
- mod_dav
- mod_deflate
- mod_evasive
- mod_pagespeed
- mod_proxy_html (can cause broken PDF downloads)
- mod_reqtimeout
- mod_security
- mod_spdy together with libapache2-mod-php5 / mod_php (use fcgi or php-fpm instead)
- mod_xsendfile / X-Sendfile (causing broken downloads if not configured correctly)

2. NGINX

- ngx_pagespeed
- HttpDavModule
- X-Sendfile (causing broken downloads if not configured correctly)

3. PHP

- eAccelerator

Troubleshooting WebDAV

General troubleshooting

ownCloud uses SabreDAV, and the SabreDAV documentation is comprehensive and helpful.

See:

- [SabreDAV FAQ](#)
- [Web servers](#) (Lists lighttpd as not recommended)
- [Working with large files](#) (Shows a PHP bug in older SabreDAV versions and information for mod_security problems)
- [0 byte files](#) (Reasons for empty files on the server)
- [Clients](#) (A comprehensive list of WebDAV clients, and possible problems with each one)
- [Finder, OS X's built-in WebDAV client](#) (Describes problems with Finder on various Web servers)

There is also a well maintained FAQ thread available at the [ownCloud Forums](#) which contains various additional information about WebDAV problems.

Error 0x80070043 “The network name cannot be found.” while adding a network drive

The windows native WebDAV client might fail with the following error message:

```
Error 0x80070043 "The network name cannot be found." while adding a network drive
```

A known workaround for this issue is to update your web server configuration.

Apache

You need to add the following rule set to your main web server or virtual host configuration, or the `.htaccess` file in your document root.

```
# Fixes Windows WebDav client error 0x80070043 "The network name cannot be found."
RewriteEngine On
RewriteCond %{HTTP_USER_AGENT} ^(DavClnt)$
RewriteCond %{REQUEST_METHOD} ^(OPTIONS)$
RewriteRule .* - [R=401,L]
```

Troubleshooting Contacts & Calendar

Service discovery

Some clients - especially on iOS/Mac OS X - have problems finding the proper sync URL, even when explicitly configured to use it.

If you want to use CalDAV or CardDAV clients together with ownCloud it is important to have a correct working setup of the following URLs:

```
https://example.com/.well-known/carddav
https://example.com/.well-known/caldav
```

Those need to be redirecting your clients to the correct DAV endpoints. If running ownCloud at the document root of your Web server the correct URL is:

```
https://example.com/remote.php/dav
```

and if running in a subfolder like `owncloud`:

```
https://example.com/owncloud/remote.php/dav
```

For the first case the `.htaccess` file shipped with ownCloud should do this work for you when running Apache. You only need to make sure that your Web server is using this file.

If your ownCloud instance is installed in a subfolder called `owncloud` and you're running Apache create or edit the `.htaccess` file within the document root of your Web server and add the following lines:

```
Redirect 301 /.well-known/carddav /owncloud/remote.php/dav
Redirect 301 /.well-known/caldav /owncloud/remote.php/dav
```

Now change the URL in the client settings to just use:

```
https://example.com
```

instead of e.g.

```
https://example.com/owncloud/remote.php/dav/principals/username.
```

There are also several techniques to remedy this, which are described extensively at the [Sabre DAV website](#).

Unable to update Contacts or Events

If you get an error like:

```
PATCH https://example.com/remote.php/dav HTTP/1.0 501 Not Implemented
```

it is likely caused by one of the following reasons:

Using Pound reverse-proxy/load balancer

As of writing this Pound doesn't support the HTTP/1.1 verb. Pound is easily [patched](#) to support HTTP/1.1.

Misconfigured Web server

Your Web server is misconfigured and blocks the needed DAV methods. Please refer to Troubleshooting WebDAV above for troubleshooting steps.

Client Sync Stalls

One known reason is stray locks. These should expire automatically after an hour. If stray locks don't expire (identified by e.g. repeated `file.txt is locked` and/or `Exception\\\\FileLocked` messages in your `data/owncloud.log`), make sure that you are running system cron and not Ajax cron (See Background Jobs). See <https://github.com/owncloud/core/issues/22116> and <https://central.owncloud.org/t/file-is-locked-how-to-unlock/985> for some discussion and additional info of this issue.

Other issues

Some services like *Cloudflare* can cause issues by minimizing JavaScript and loading it only when needed. When having issues like a not working login button or creating new users make sure to disable such services first.

Code Signing

ownCloud supports code signing for the core releases, and for ownCloud applications. Code signing gives our users an additional layer of security by ensuring that nobody other than authorized persons can push updates.

It also ensures that all upgrades have been executed properly, so that no files are left behind, and all old files are properly replaced. In the past, invalid updates were a significant source of errors when updating ownCloud.

FAQ

Why Did ownCloud Add Code Signing?

By supporting Code Signing we add another layer of security by ensuring that nobody other than authorized persons can push updates for applications, and ensuring proper upgrades.

Do We Lock Down ownCloud?

The ownCloud project is open source and always will be. We do not want to make it more difficult for our users to run ownCloud. Any code signing errors on upgrades will not prevent ownCloud from running, but will display a warning on the Admin page. For applications that are not tagged "Official" the code signing process is optional.

Not Open Source Anymore?

The ownCloud project is open source and always will be. The code signing process is optional, though highly recommended. The code check for the core parts of ownCloud is enabled when the ownCloud release version branch has been set to stable.

For custom distributions of ownCloud it is recommended to change the release version branch in `version.php` to something else than "stable".

Is Code Signing Mandatory For Apps?

Code signing is optional for all third-party applications.

Fixing Invalid Code Integrity Messages

A code integrity error message ("There were problems with the code integrity check. More information...") appears in a yellow banner at the top of your ownCloud Web interface:

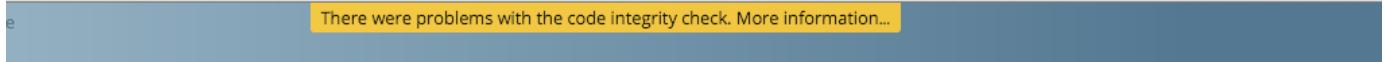
There were problems with the code integrity check. More information...

Note

The yellow banner is only shown for admin users.

Clicking on this link will take you to your ownCloud admin page, which provides the following options:

1. Link to this documentation entry.
2. Show a list of invalid files.
3. Trigger a rescan.



There were problems with the code integrity check. More information...

1 2 3

Security & setup warnings

- Some files have not passed the integrity check. Further information on how to resolve this issue can be found in our documentation. ([List of invalid files...](#) / [Rescan...](#))

To debug issues caused by the code integrity check click on “List of invalid files...”, and you will be shown a text document listing the different issues. The content of the file will look similar to the following example:

```
Technical information
=====
The following list covers which files have failed the integrity check. Please read the previous linked documentation to learn more about the errors and how to fix them.

Results
=====
- core
  - INVALID_HASH
    - /index.php
    - /version.php
  - EXTRA_FILE
    - /test.php
- calendar
  - EXCEPTION
    - OC\IntegrityCheck\Exceptions\InvalidSignatureException
    - Signature data not found.
- tasks
  - EXCEPTION
    - OC\IntegrityCheck\Exceptions\InvalidSignatureException
    - Certificate has been revoked.

Raw output
=====
Array
(
    [core] => Array
        (
            [INVALID_HASH] => Array
                (
                    [/index.php] => Array
                        (
                            [expected] =>
f1c5e2630d784bc9cb02d5a28f55d6f24d06dae2a0fee685f3
c2521b050955d9d452769f61454c9ddfa9c308146ade10546c
fa829794448eaaffbc9a04a29d216
                            [current] =>
ce08bf30bcbb879a18b49239a9bec6b8702f52452f88a9d321
42cad8d2494d5735e6bfa0d8642b2762c62ca5be49f9bf4ec2

```

```
        )
    )

    [/version.php] => Array
    (
        [expected] =>
c5a03bacae8dedf8b239997901balffffd2fe51271d13a00cc4
b34b09cca5176397a89fc27381ccb1f72855fa18b69b6f87d7
d5685c3b45aeee373b09be54742ea
        [current] =>
88a3a92c11db91declac3be0e1c87f862c95ba6ffaaaa3f2c3
b8f682187c66f07af3a3b557a868342ef4a271218felc1e300
c478e6c156c5955ed53c40d06585
    )

)

[EXTRA_FILE] => Array
(
    [/test.php] => Array
    (
        [expected] =>
        [current] =>
09563164f9904a837f9ca0b5f626db56c838e5098e0ccc1d8b
935f68fa03a25c5ec6f6b2d9e44a868e8b85764dafd1605522
b4af8db0ae269d73432e9a01e63a
    )

)

)

[calendar] => Array
(
    [EXCEPTION] => Array
    (
        [class] => OC\IntegrityCheck\Exceptions\InvalidSignature
Exception
        [message] => Signature data not found.
    )

)

[tasks] => Array
(
    [EXCEPTION] => Array
    (
        [class] => OC\IntegrityCheck\Exceptions\InvalidSignatureException
        [message] => Certificate has been revoked.
    )

)
```

In above error output it can be seen that:

1. In the ownCloud core (that is, the ownCloud server itself) the files “index.php” and “version.php” do have the wrong version.
 2. In the ownCloud core the unrequired extra file “/test.php” has been found.
 3. It was not possible to verify the signature of the calendar application.

4. The certificate of the task application was revoked.

You have to do the following steps to solve this:

1. Upload the correct "index.php" and "version.php" files from e.g. the archive of your ownCloud version.
2. Delete the "test.php" file.
3. Contact the developer of the application. A new version of the app containing a valid signature file needs to be released.
4. Contact the developer of the application. A new version of the app signed with a valid signature needs to be released.

For other means on how to receive support please take a look at <https://owncloud.org/support/>. After fixing these problems verify by clicking "Rescan...".

Note

When using a FTP client to upload those files make sure it is using the Binary transfer mode instead of the ASCII transfer mode.

Rescans

Rescans are triggered at installation, and by updates. You may run scans manually with the `occ` command. The first command scans the ownCloud core files, and the second command scans the named app. There is not yet a command to manually scan all apps:

```
occ integrity:check-core  
occ integrity:check-app $appid
```

See Using the `occ` Command to learn more about using `occ`.

Errors

Warning

Please don't modify the mentioned `signature.json` itself.

The following errors can be encountered when trying to verify a code signature.

- `INVALID_HASH`
 - The file has a different hash than specified within `signature.json`. This usually happens when the file has been modified after writing the signature data.
- `MISSING_FILE`
 - The file cannot be found but has been specified within `signature.json`. Either a required file has been left out, or `signature.json` needs to be edited.
- `EXTRA_FILE`
 - The file does not exist in `signature.json`. This usually happens when a file has been removed and `signature.json` has not been updated. It also happens if you have placed additional files in your ownCloud installation folder.
- `EXCEPTION`
 - Another exception has prevented the code verification. There are currently these following exceptions:
 - Signature data not found. `

- The app has mandatory code signing enforced but no `signature.json` file has been found in its `appinfo` folder.
- Certificate is not valid.
 - The certificate has not been issued by the official ownCloud Code Signing Root Authority.
- Certificate is not valid for required scope. (Requested: %s, current: %s)
 - The certificate is not valid for the defined application. Certificates are only valid for the defined app identifier and cannot be used for others.
- Signature could not get verified.
 - There was a problem with verifying the signature of `signature.json`.
- Certificate has been revoked.
 - The certificate which was used to sign the application was revoked.

Impersonating Users

Sometimes you may need to use your ownCloud installation as another user, whether to help users debug an issue or to get a better understanding of what they see when they use their ownCloud account. The ability to do so is a feature delivered via an ownCloud app called [Impersonate](#).

Note

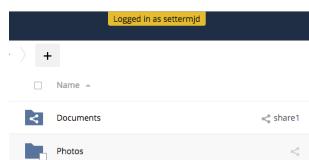
This functionality is available only to administrators.

Impersonating a User

When installed, you can then impersonate users; in effect, you will be logged in as said user. To do so, go to the Users list, where you will now see a new column available called “[Impersonate](#)”, as in the screenshot below.

Username		Groups		Create
Username	Impersonate	Full Name	Password	Groups
A admin		dmin	*****	admin
M matthew_setter_gmail_com		matthew	*****	guest_app
S settermijd		settermijd	*****	share1, share2, share3, sha...
S share1		share1	*****	no group

Click the gray head icon next to the user that you want to impersonate. Doing so will log you in as that user, temporarily pausing your current session. You will see a notification at the top of the page that confirms you’re now logged in as (or impersonating) that user.



Anything that you see until you log out will be what that user would see.

Ending an Impersonation

When you’re ready to stop impersonating the user, log out and you will return to your normal user session.

Restrict Impersonation to Groups & Group administrators

As a security measure, the application lets ownCloud administrators restrict the ability to impersonate users to administrators of specific groups. When enabled and configured, only a group’s administrator can impersonate members of their group.

Enterprise Features

For example, if an ownCloud administrator restricts user impersonation only to the group: ‘group1’, then **only** ‘group1’’s administrators can impersonate users belonging to ‘group1’. No other users can impersonate other users.

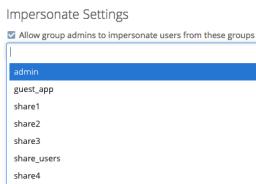
Note

ownCloud administrators can always impersonate all users of an ownCloud instance when the application is installed.

To enable this option, in the administrator settings panel (administrator -> Settings -> Admin) in the “**Additional**” section, you’ll see a section titled: “**Impersonate Settings**”; which you can see below.



Check the checkbox under that, and a textbox will appear. If you click in the textbox, you will see a list of available groups on your ownCloud installation. As you type, the list will filter down to only ones that match the text entered, as you can see below.



Choose one or more groups from the list, and they will be added to the textbox.

Enterprise Features

Installation

Installing & Upgrading ownCloud Enterprise Edition

The recommended method for installing and maintaining your ownCloud Enterprise edition is with your Linux package manager. Configure your package manager to use the ownCloud Enterprise repository, import the signing key, and then install and update ownCloud packages like any other software package.

Please refer to the `README - ownCloud Package Installation.txt` document in your account at Customer.owncloud.com account for instructions on setting up your Linux package manager.

After you have completed your initial installation of ownCloud as detailed in the README, follow the instructions in The Installation Wizard to finish setting up ownCloud.

To upgrade your Enterprise server, refer to How to Upgrade Your ownCloud Server.

Manual Installation

Download the ownCloud archive from your account at <https://customer.owncloud.com/owncloud>, then follow the instructions at Manual Installation on Linux.

SELinux

Linux distributions that use SELinux need to take some extra steps so that ownCloud will operate correctly under SELinux. Please see SELinux Configuration for some recommended configurations.

License Keys

Introduction

You’ll need to install a license key to use ownCloud Enterprise Edition. There are two types of license keys: one is a free 30-day trial key. The other is a full license key for Enterprise customers.

Enterprise Features

You can [download and try ownCloud Enterprise for 30 days for free](#), which auto-generates a free 30-day key. When this key expires your ownCloud installation is not removed, so when you become an Enterprise customer you can enter your new key to regain access. See [How to Buy ownCloud](#) for sales and contact information.

Configuration

Once you get your Enterprise license key, it needs to be copied to your ownCloud configuration file, config/config.php like this example:

```
'license-key' => 'test-20150101-XXXXXXXXXXXXXXXXXXXXXXXXXXXX-YYYYYY' ,
```

Each running instance of ownCloud requires a license key. Keys will work across upgrades without issue, so new keys will not be required when you upgrade your ownCloud Enterprise to a new version.

Supported ownCloud Enterprise Edition Apps

See Supported Apps in ownCloud for a list of supported apps.

Note

3rd party and unsupported apps must be disabled before performing a system upgrade. Then install the upgraded versions, and after the upgrade is complete re-enable them.

Oracle Database Setup

This document will cover the setup and preparation of the ownCloud server to support the use of Oracle as a backend database.

Outline of Steps

This document will cover the following steps:

- Setup of the ownCloud user in Oracle: This involves setting up a user space in Oracle for setting up the ownCloud database.
- Installing the Oracle Instant Client on the Web server (facilitating the connection to the Oracle Database).
- Compiling and installing the Oracle PHP Plugin oci8 module
- Pointing ownCloud at the Oracle database in the initial setup process

The document assumes that you already have your Oracle instance running, and have provisioned the needed resources. It also assumes that you have installed ownCloud with all of the prerequisites.

Configuring Oracle

Setting up the User Space for ownCloud

Step one, if it has not already been completed by your DBA, provision a user space on the Oracle instance for ownCloud. This can be done by logging in as a DBA and running the script below:

```
CREATE USER owncloud IDENTIFIED BY password;
ALTER USER owncloud DEFAULT TABLESPACE users TEMPORARY TABLESPACE temp QUOTA unlimited ON users;
GRANT create session, create table, create procedure, create sequence, create trigger, create view;
```

Substitute an actual password for `password`. Items like `TableSpace`, `Quota` etc., will be determined by your DBA (database administrator).

Add OCI8 Client Packages

Installation of the OCI8 client is dependent on your distribution. Given that, please use the relevant section below to find the relevant instructions to install the client.

Ubuntu

If you're using Ubuntu, we recommend that you use *this very thorough guide* from the Ubuntu Community Wiki to install the OCI8 extension.

Note

This *should* work for other Debian-based distributions, however your mileage may vary.

RedHat / Centos / Fedora

To install the *OCI8 extension* on a RedHat-based distribution, you first need to download two Oracle Instant Client packages:

- Instant Client Package - Basic (`oracle-instantclient12.2-basic-12.2.0.1.0-1.x86_64.rpm`)
- Instant Client Package - SDK (`oracle-instantclient12.2-devel-12.2.0.1.0-1.x86_64.rpm`)

Then, to install them, use the following commands:

```
rpm --install oracle-instantclient12.2-basic-12.2.0.1.0-1.x86_64.rpm \
      oracle-instantclient12.2-devel-12.2.0.1.0-1.x86_64.rpm
```

Install the OCI8 PHP Extension

With the Oracle packages installed you're now ready to install PHP's OCI8 extension.

Note

Provide: `instantclient,/usr/lib/oracle/12.2/client64/lib` when requested, or let it auto-detect the location (if possible).

```
pecl install oci8
```

With the extension installed, you now need to configure it, by creating a configuration file for it. You can do so using the command below, substituting `FILE_PATH` with one from the list below the command.

```
cat << EOF > FILE_PATH
; Oracle Instant Client Shared Object extension
extension=oci8.so
EOF
```

Configuration File Paths

Debian & Ubuntu

PHP Version	Filename
5.6	/etc/php/5.6/apache2/conf.d/20-oci.ini
7.0	/etc/php/7.0/apache2/conf.d/20-oci.ini
7.1	/etc/php/7.1/apache2/conf.d/20-oci.ini

RedHat, Centos, & Fedora

PHP Version	Filename
-------------	----------

5.6	/etc/opt/rh/rh-php56/php.d/20-oci8.ini
7.0	/etc/opt/rh/rh-php70/php.d/20-oci8.ini

Validating the Extension

With all that done, confirm that it's been installed and available in your PHP distribution, run the following command:

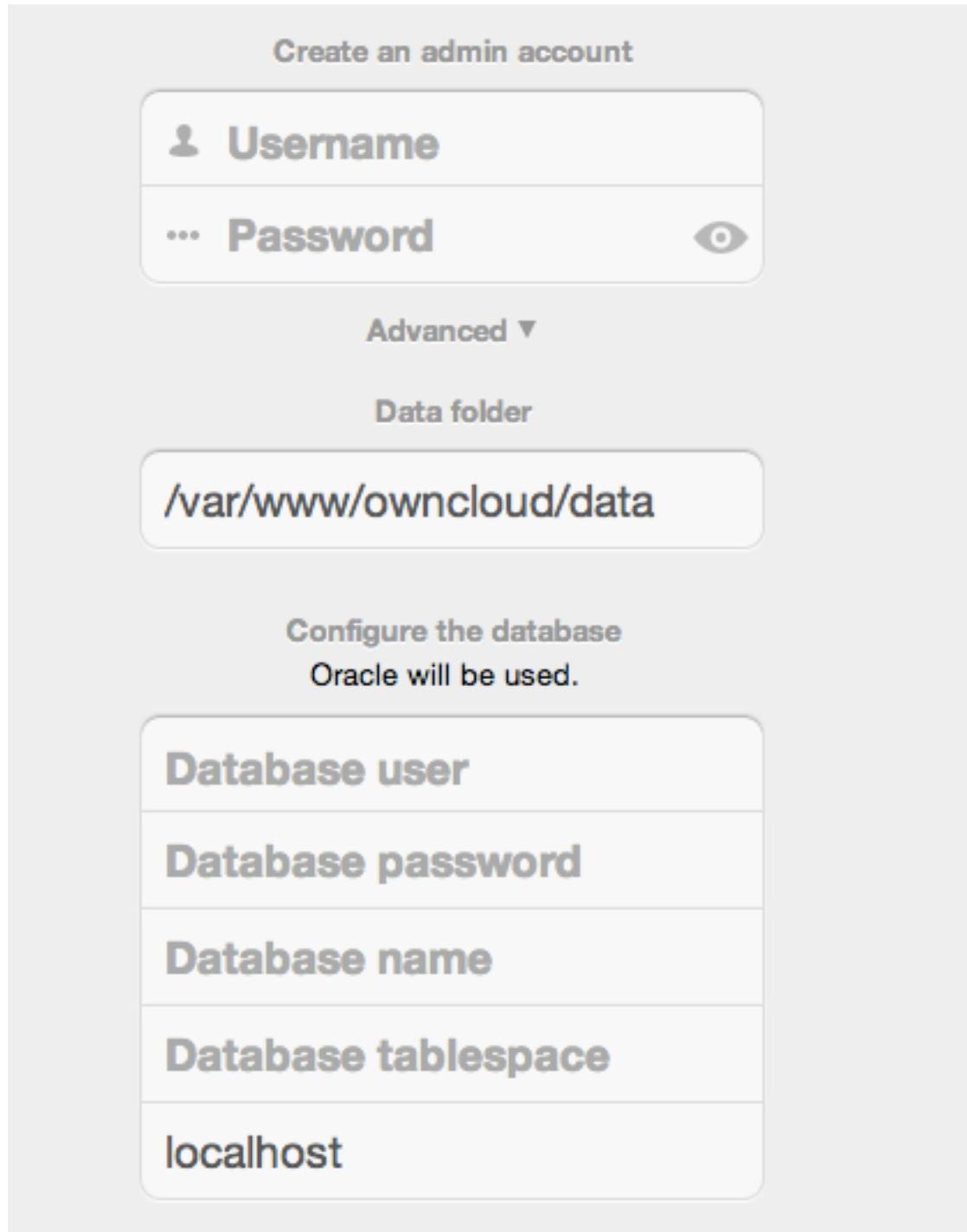
```
php -m | grep -i oci8
```

When the process has completed, assuming that you don't encounter any errors, restart Apache and the extension is ready to use.

Configure ownCloud

The next step is to configure the ownCloud instance to point to the Oracle Database, again this document assumes that ownCloud has previously been installed.

Configuration Wizard



Database user

This is the user space created in step 2.1. In our Example this would be owncloud.

Database password

Again this is defined in the script from section 2.1 above, or pre-configured and provided to you by your DBA.

Database Name

Represents the database or the service that has been pre-configured on the TSN Listener on the Database Server. This should also be provided by the DBA. In this example, the default setup in the Oracle install was orcl (there is a TSN Listener entry for orcl on our database server).

This is not like setting up with MySQL or SQL Server, where a database based on the name you give is created. The oci8 code will call this specific service and it must be active on the TSN Listener on your Oracle Database server.

Database Table Space

Provided by the DBA. In this example the users table space (as is seen in the user creation script above), was used.

Configuration File

Assuming all of the steps have been followed to completion, the first run wizard should complete successfully, and an operating instance of ownCloud should appear.

The configuration file should look something like this:

```
<?php

$CONFIG = [
    'instanceid' => 'abcdefghijkl',
    'passwordsalt' => '01234567890123456789',
    'datadirectory' => '/var/data',
    'dbtype' => 'oci',
    'version' => '8.2.x.y',
    'dbname' => 'orcl',
    'dbhost' => '192.168.1.57',
    'dbtableprefix' => 'oc_',
    'dbuser' => 'owncloud1',
    'dbpassword' => '*****',
    'installed' => true,
];
```

Useful SQL Commands

Is my Database Reachable?

On the machine where your Oracle database is installed, type:

```
sqlplus username
```

```
SQL> select * from v$version;

BANNER
-----
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
PL/SQL Release 11.2.0.2.0 - Production
CORE 11.2.0.2.0      Production
TNS for Linux: Version 11.2.0.2.0 - Production
NLSRTL Version 11.2.0.2.0 - Production

SQL> exit
```

Show Database Users:

```
Oracle     : SELECT * FROM all_users;
```

Show available Databases:

```
Oracle     : SELECT name FROM v$database; (requires DBA privileges)
```

Show ownCloud Tables in Database:

```
Oracle     : SELECT table_name FROM user_tables;
```

Quit Database:

```
Oracle     : quit
```

Firewall Configuration

File Firewall

The File Firewall GUI enables you to manage firewall rule sets. You can find it in your ownCloud admin page, under Admin -> Security. The File Firewall lets you control access and sharing in fine detail, by creating rules for allowing or denying access restrictions based on: *group*, *upload size*, *client devices*, *IP address*, *time of day*, as well as many more criteria. In addition to these restriction options, the File Firewall app also supports rules based on regular expressions.

How the File Firewall Works

Each firewall rule set consists of one or more conditions. If a request matches all of the conditions, in at least one rule set, then the request is blocked by the firewall. Otherwise, the request is allowed by the firewall.

Note

The File Firewall app cannot lock out administrators from the web interface when rules are misconfigured.

Using the File Firewall

Figure 1 shows an empty firewall configuration panel. Set your logging level to **Blocked Requests Only** for debugging, and create a new rule set by clicking the **Add Group** button. After setting up your rules you must click the **Save Rules** button.

File Firewall

Requests are checked against all groups of rules that are defined below. A request is blocked when at least one group matches the request. A group matches a request when all rule conditions in the group evaluate to true.

The screenshot shows a web-based configuration interface for a file firewall. At the top, there is a 'Group Name' input field containing '----'. Below it is a dropdown menu with a single entry '----'. To the right of the dropdown is a 'Delete' button with a crossed-out 'x' icon. At the bottom of the main panel are three buttons: '+ Add rule', '+ Add group', and another 'Delete' button. Below the main panel, there is a 'Logging' section with two buttons: 'Save Rules' and 'Blocked Requests Only' (which is currently selected). The entire interface is contained within a light gray border.

Figure 1: Empty File Firewall configuration panel

Figure 2 shows two rules. The first rule, **No Support outside office hours**, prevents members of the support group from logging into the ownCloud Web interface from 5pm-9am, and also blocks client syncing. The second rule prevents members of the “qa-team” group from accessing the Web UI from IP addresses that are outside of the local network.

File Firewall

Requests are checked against all groups of rules that are defined below. A request is blocked when at least one group matches the request. A group matches a request when all rule conditions in the group evaluate to true.

The screenshot shows the configuration interface for a File Firewall. It displays two examples of rules:

- Example 1:** No support outside of the office. This rule has two conditions:
 - User Group is support
 - Request Time between 05:00 pm +0545 and 09:00 am +0545
- Example 2:** No QA outside of the office. This rule has two conditions:
 - User Group is qa-team
 - IP Range (IPv4) is not 192.168.1.0/24

At the bottom, there are buttons for **+ Add rule**, **+ Add group**, and **x Delete**.

Figure 2: Two example rules that restrict logins per user group

All other users are not affected, and can log in anytime from anywhere.

Available Conditions

User Group

The user (is|is not) a member of the selected group.

User Agent

The User-Agent of the request (matches|does not match) the given string.

User Device

A shortcut for matching all known (android | ios | desktop) sync clients by their User Agent string.

Request Time

The time of the request (has to|must not) be in a single range from beginning time to end time.

Request URL

The **full page URL** (has to|must not) (match|contain|begin with|end) with a given string.

Request Type

The request (is|is not) a (WebDAV|public share link|other) request.

Request IP Range (IPv4) and IP Range (IPv6)

The request's REMOTE_ADDR header (is|is not) matching the given IP range.

File Size Upload

When a file is uploaded the size has to be (less|less or equal|greater|greater or equal) to the given size.

File Mimetype Upload

When a file is uploaded the mimetype (is|is not|begins with|does not begin with|ends with|does not end with) the given string.

System File Tag

One of the parent folders or the file itself (is|is not) tagged with a System tag.

Regular Expression

The File Firewall supports regular expressions, allowing you to create custom rules using the following conditions:

- IP Range (IPv4)
- IP Range (IPv6)
- User agent
- User group
- Request URL

You can combine multiple rules into one rule, e.g., if a rule applies to both the support and the qa-team you could write your rule like this:

```
Regular Expression > ^(support|qa-team)$ > is > User group
```

Warning

We do not recommend modifying the configuration values directly in your config.php. These use JSON encoding, so the values are difficult to read and a single typo will break all of your rules.

Controlling Access to Folders

The easiest way to block access to a folder, starting with ownCloud 9.0, is to use a system tag. A new rule type was added which allows you to block access to files and folders, where at least one of the parents has a given tag.

Now you just need to add the tag to the folder or file, and then block the tag with the File Firewall. This example blocks access to any folder with the tag "Confidential" from outside access.

Block by System Tag:

```
System file tag:    is      "Confidential"  
IP Range (IPv4):   is not  "192.168.1.0/24"
```

File Firewall

Requests are checked against all groups of rules that are defined below. A request is blocked when at least one group matches the request. A group matches a request when all rule conditions in the group evaluate to true.

The screenshot shows a configuration interface for a firewall rule. At the top, there is a button labeled "Block confidential file". Below it, there are two rule definitions:

- System file tag**: Set to "is" and "Confidential".
- IP Range (IPv4)**: Set to "is not" and "192.168.1.0/24".

At the bottom right, there are three buttons: "+ Add rule", "+ Add group", and "Delete".

Custom Configuration for Branded Clients

If you are using branded ownCloud clients, you may define `firewall.branded_clients` in your `config.php` to identify your branded clients in the firewall “**User Device**” rule.

The configuration is a `User-Agent => Device map`. `Device` must be one of the following:

- android
- android_branded
- ios
- ios_branded
- desktop
- desktop_branded

The User-Agent is always compared all lowercase. By default the agent is compared with `equals`. When a trailing or leading asterisk, `*`, is found, the agent is compared with `starts with` or `ends with`. If the agent has both a leading and a trailing `*`, the string must appear anywhere. For technical reasons the User-Agent string must be at least 4 characters, including wildcards. When you build your branded client you have the option to create a custom User Agent.

In this example configuration you need to replace the example User Agent strings, for example '`android_branded`', with your own User Agent strings:

```
// config.php

'firewall.branded_clients' => array(
    'my ownbrander android user agent string' => 'android_branded',
    'my ownbrander second android user agent string' => 'android_branded',
    'my ownbrander ios user agent string' => 'ios_branded',
    'my ownbrander second ios user agent string' => 'ios_branded',
    'my ownbrander desktop user agent string' => 'desktop_branded',
    'my ownbrander second desktop user agent string' => 'desktop_branded',
),
```

The Web UI dropdown then expands to the following options:

- Android Client - always visible
- iOS Client - always visible
- Desktop Client - always visible
- Android Client (Branded) - visible when at least one `android_branded` is defined
- iOS Client (Branded) - visible when at least one `ios_branded` is defined

- Desktop Client (Branded) - visible when at least one desktop_branded is defined
- All branded clients - visible when at least one of android_branded, ios_branded or desktop_branded is defined
- All non-branded clients - visible when at least one of android_branded, ios_branded or desktop_branded is defined
- Others (Browsers, etc.) - always visible

Then these options operate this way:

- The * Client options only match android, ios and desktop respectively.
- The * Client (Branded) options match the *_branded agents equivalent.
- All branded clients matches: android_branded, ios_branded and desktop_branded
- All non-branded clients matches: android, ios and desktop

Ransomware Protection

Ransomware is [an ever-present threat](#), both for large enterprises as well as for individuals. Once infected, a whole hard disk (or just parts of it) can become encrypted, leading to unrecoverable data loss.

Once this happens, attackers usually ask victims to pay a ransom, often via cryptocurrencies such as Bitcoin, in exchange for the decryption key required to decrypt their data.

While paying the ransom works in some cases, it is not recommended, as there is no guarantee that the attackers will supply the key after payment is made. To help mitigate such threats and ensure ongoing access to user data, ownCloud provides the Ransomware Protection app.

Important

It is essential to be aware that user data needs to be synchronized with your ownCloud Server using the ownCloud Desktop synchronization client. Data that is not synchronized and stored in ownCloud cannot be protected.

About Ransomware Protection

The app is tasked with *detecting*, *preventing*, and *reverting* anomalies. Anomalies are file operations (including *create*, *update*, *delete*, and *move*) not intentionally conducted by the user. It aims to do so in two ways: [protection](#), and [protection](#).

Prevention: Blocking Common Ransomware File Extensions

Like other forms of cyberattack, ransomware has a range of diverse characteristics. On the one hand it makes them hard to detect and on the other it makes them even harder to prevent. Recent ransomware attacks either encrypt a user's files and add a specific file extension to them (e.g., ".crypt"), or they replace the original files with an encrypted copy and add a particular file extension.

File Extension Blacklist

The first line of defense against such threats is a blacklist that blocks write access to file extensions known to originate from ransomware.

Ransomware Protection ships with a [static extension list](#) of around 1,500 file extensions. As new extensions are regularly created, this list also needs to be regularly reviewed and updated. Future releases of Ransomware Protection will include an updated list and the ability to update the list via syncing with FSRM's [API](#) by using [occ](#).

Important

Please check the provided ransomware blacklist! It is **strongly recommended** to check the provided ransomware blacklist to ensure that it fits your needs. In some cases, the patterns might be too generic and result in false positives.

File Blocking

The second line of defense is file blocking. As files are uploaded, they are compared against the file extension blacklist. If a match is found, the upload is denied.

Note

File blocking is always enabled.

Account Locking

The third line of defense is account locking. If a client uploads a file matching a pattern in the ransomware blacklist, the account is locked (set as read-only) for client access (*create*, *change*, *move*, and *delete* operations). Doing this prevents further, malicious, changes.

Following this, clients receive an error (403 Access Forbidden) which notifies the user that the account is locked by Ransomware Protection.

Note

Write access (e.g., moving and deleting files) is still possible for users when they log in with their web browser.

When an account is locked, administrators can unlock the account using the `occ ransomguard:unlock` command. Administrators can also manually lock user accounts, using the `occ ransomguard:lock` command.

Note

When an account is locked, it will still be fully usable from the ownCloud web UI. However, ownCloud clients (as well as other WebDAV clients) will see the account as set to read-only mode.

Users will see a yellow notification banner in the ownCloud web UI directing them to “Personal Settings -> Security” (“*Ransomware detected: Your account is locked (read-only) for client access to protect your data. Click here to unlock.*”), where additional information is displayed and users can unlock their account when ransomware issues are resolved locally.

Note

Locking is enabled by default. If this is not desired, an administrator can disable it in the “Admin -> Security” panel.

Protection: Data Retention and Rollback

While Ransomware Prevention mitigates risks of a range of ransomware attacks, it is not a future-proof solution, because ransomware is becoming ever-more sophisticated. There are known attacks that change file extensions randomly or keep them unchanged which makes them harder to detect.

Ultimately there is a consensus that only one solution can provide future-proof protection from ransomware attacks: retaining data and providing the means to roll back to a particular point in time.

ownCloud Ransomware Protection will, therefore, record all changes on an ownCloud Server and allow administrators to rollback user data to a particular point in time, making use of ownCloud's integrated Versioning and Trash bin features.

Doing so allows all user data that is synchronized with the server to be rolled back to its state before the attack occurred. A combination of Ransomware prevention and protection reduces risks to a minimum acceptable level.

Other Elements of Ransomware Protection

Name	Command (if applicable)	Description
Ransomware Prevention (Blocker)		First line of defense against ransomware attacks. Ransomware Protection uses a file name pattern blacklist to prevent uploading files that have file extensions associated with ransomware (e.g. ".crypt") thereby preserving the original files on the ownCloud Server.
Ransomguard Scanner	occ ransomguard:scan <timestamp> <user>	A command to scan the ownCloud database for changes in order to discover anomalies in a user's account and their origin. It enables an administrator to determine the point in time where undesired actions happened as a prerequisite for restoration.
Ransomguard Restorer	occ ransomguard:restore <timestamp> <user>	A command for administrators to revert all operations in a user account that occurred after a certain point in time.
Ransomguard Lock	occ ransomguard:lock <user>	Set a user account as read-only for ownCloud and other WebDAV clients. This prevents any further changes to the account.
Ransomguard Unlock	occ ransomguard:unlock <user>	Unlock a user account which was set to read-only.

Note

<timestamp> must be in the *Linux timestamp format*.

Requirements

Mandatory

- File Firewall rule (previous approach for ransomware protection).** If you have configured the File Firewall rule which was provided as a preliminary protection mechanism, please remove it. The functionality (Blocking) is covered by Ransomware Protection in an improved way.
- Ransomware Protection.** Ransomware protection needs to be in operation before an attack occurs, as it needs to record file operations to be able to revert them, in case of an attack.
- ownCloud Versions App.** Required to restore older file versions. The capabilities of Ransomware Protection depend on its configuration regarding version retention.
- ownCloud Trash Bin App.** Required to restore deleted files. The capabilities of Ransomware Protection depend on its configuration regarding trash bin retention.

Optional

1. Activity app. For viewing activity logs.

Limitations

- Ransomware Protection works with master-key based storage encryption. With credential-based storage encryption, only Ransomware Prevention (Blocking) works.
- Rollback is not based on snapshots:
 - The [trash bin retention policy](#) may delete files, making them unrecoverable. To avoid this, set `trashbin_retention_obligation` to disabled, or choose a conservative policy for trash bin retention. However, please be aware that this may increase storage requirements.
 - Trash bin items may be deleted by the user making them unrecoverable by Ransomware Protection => Users need to know this.
 - Versions have [a built-in “thin-out” policy](#) which makes it possible that required file versions are unrecoverable by Ransomware Protection. To help avoid this, set `versions_retention_obligation` to disabled or choose a conservative policy for version retention. Please be aware that this might increase your storage needs.
 - A specific version of a file that is needed for rollback might have been manually restored, making this version potentially unrecoverable by Ransomware Protection. Currently, after restoration the restored version *is not a version anymore*, e.g., the version is not present in versioning.
- Recovery capabilities in received shared folders are currently limited. Changed file contents and deletions can be restored but MOVE operations can't. The case when a ransomware attack renames files in a received shared folder is therefore not yet covered.
- Contents in secondary storages, such as *Windows network drives*, *Dropbox*, and *Google Drive*, are unrecoverable by Ransomware Protection, because they do not have versioning or trash bin enabled in ownCloud.
- Rolling files forward is not *currently* supported or tested. Therefore it is vital to:
 - Carefully decide the point in time to rollback to.
 - To have proper backups to be able to conduct the rollback again, if necessary.

File Management

Advanced File Tagging With the Workflow App

The Workflow App provides advanced management of file tagging. The app has three parts: Tag Manager, Automatic Tagging, and Retention.

The Workflow App should be enabled by default (Apps page), and the three configuration modules visible on your ownCloud Admin page.

See [Tagging Files](#) in the ownCloud User manual to learn how to apply and filter tags on files.

Tag Manager

The Tag Manager is for creating new tags, editing existing tags, and deleting tags. Tags may be marked as **Visible**, **Restricted**, or **Invisible**.

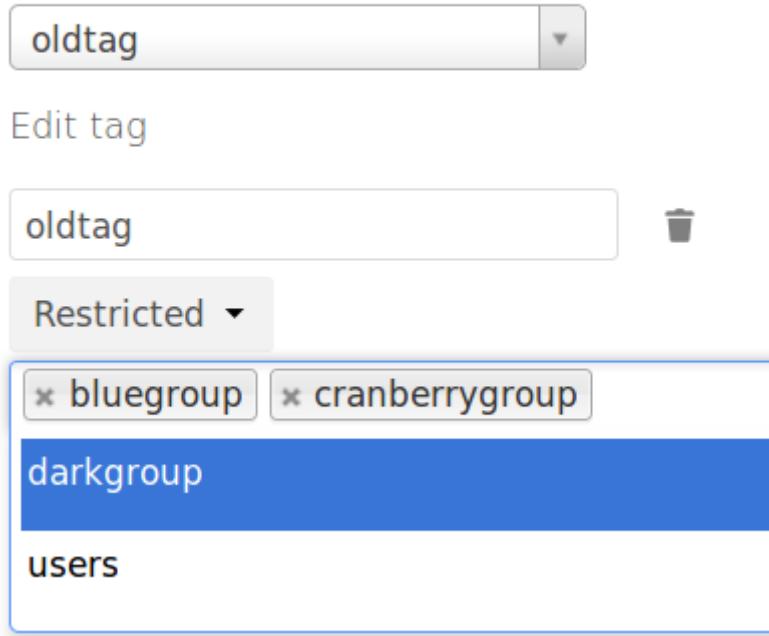
Visible means that all users may see, rename, and apply these tags to files and folders.

Restricted means tags are assignable and editable only to the user groups that you select. Other users can filter files by restricted tags, but cannot tag files with them or rename them. The tags are marked (restricted).

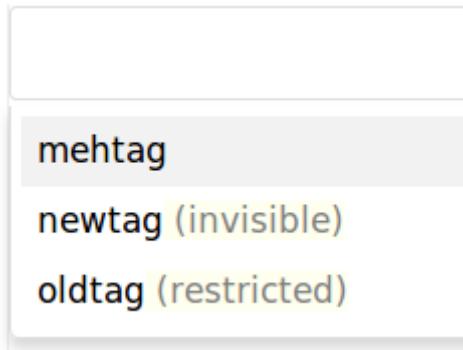
Invisible means visible only to ownCloud admins.

Use the **Collaborative tag management** module on your ownCloud admin page to edit and create tags.

Collaborative tag management



This is what your tags look like in the **Tags** view on your files page. Non-admin users will not see invisible tags, but they will see visible and restricted tags.



Automatic Tagging

The Automatic Tagging module operates on newly-uploaded files. Create a set of conditions, and then when a file or folder matches those conditions it is automatically tagged. The tag must already have been created with the Tag Manager.

For example, you can assign the invisible tag **iOS Uploads** to all files uploaded from iOS devices. This tag is visible only to admins.

Automatic tagging

Automatically tag newly uploaded files, matching the conditions, with the following tags:

iOS files  

Conditions:

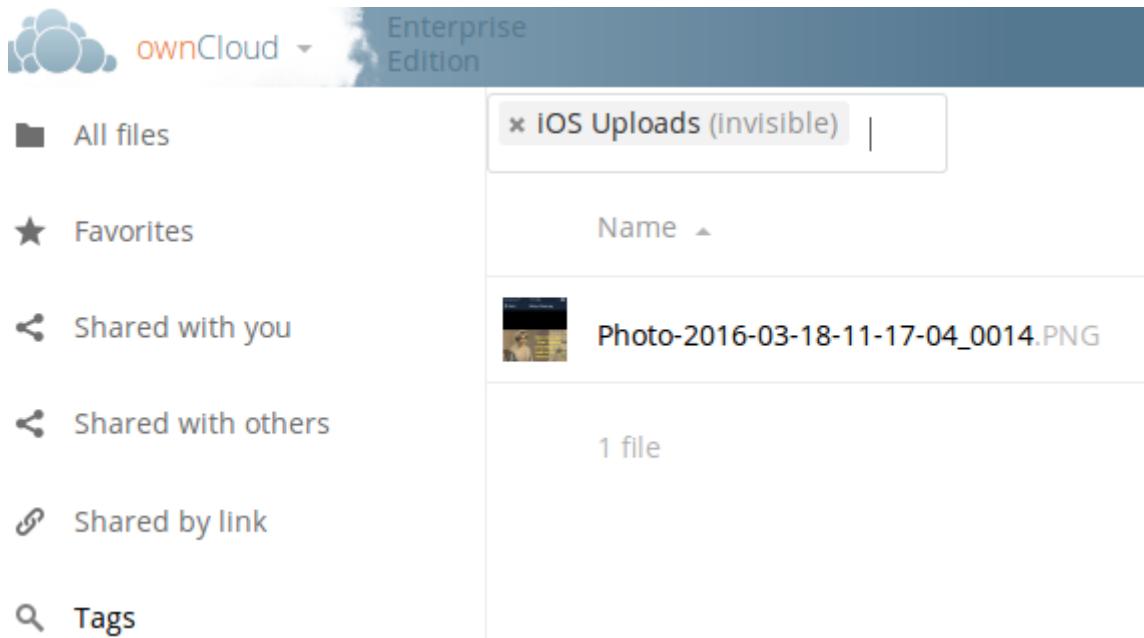
Device type is iOS Client

Add tags:

iOS Uploads (invisible)

[+ Add new rule](#)

When files with this tag are shared with you, you can view them with the Tags filter on the Files page.



The screenshot shows the ownCloud Enterprise Edition interface. On the left, there's a sidebar with links: All files, Favorites, Shared with you, Shared with others, Shared by link, and Tags. The Tags link is highlighted. On the right, a file list is shown with a search bar at the top containing the tag name: "iOS Uploads (invisible)". The results show one file: "Photo-2016-03-18-11-17-04_0014.PNG". Below the file list, it says "1 file".

Automatic Tagging is especially useful with the Retention module.

Retention

The Retention module is your housecleaning power tool, because it automatically deletes files after a time period that you specify. Select which tag to set a time limit on, and then set your time limit. File age is calculated from the file mtime (modification time).

Retention periods

Delete files tagged with the following tags after the given time:

iOS Uploads (Invisible)   

[+ Add new rule](#)

Enterprise Features

For best performance, retention tags should be applied high in your file hierarchy. If subfolders have the same tags as their parent folders, their tags must also be processed, so it will take a little longer.

Retention Engines

There are two retention engines that further allow you to fine-tune your retention settings: **TagBasedRetention** and **UserBasedRetention**. **TagBasedRetention** is the default.

TagBasedRetention: This checks files that have a particular tag assigned. Then it checks (depth-first) the children of the tagged item, before continuing with the other tagged items. Children that have already been checked will not be checked a second time.

This is optimised for processing smaller numbers of files that have multiple retention tags.

UserBasedRetention: Examines files per user. It first iterates over all files and folders (siblings first), then examines the tags for those items and checks their respective retention periods. This is optimised for many files with few retention tags.

To select UserBasedRetention, add this line to your ee.config.php:

```
'workflow.retention_engine' => userbased,
```

External Storage

Enterprise-Only Authentication Options

In ownCloud 9.0+, there are five authentication backends for external storage mounts:

- Username and password
- Log-in credentials, save in session
- Log-in credentials, save in database
- User entered, store in database
- Global credentials

The first two are common to all editions of ownCloud, and the last three are only in the Enterprise edition. These are available to:

- FTP
- ownCloud
- SFTP
- SMB/CIFS
- WebDAV
- Windows Network Drive

Username and password

This is the default; a login entered by the admin when the external mount is created. The login is stored in the database, which allows sharing, and background jobs, such as file scanning, to operate.

Log-in credentials, save in session

Credentials are only stored in the session and not captured in the database. Files cannot be shared, as credentials are not stored.

Log-in credentials, save in database

Credentials are stored in the database, and files can be shared.

User entered, store in database

Users provide their own login credentials, rather than using admin-supplied credentials. User credentials are stored in the database, and files can be shared.

Global credentials

Re-usable credentials entered by the admin, files can be shared.

Enterprise Features

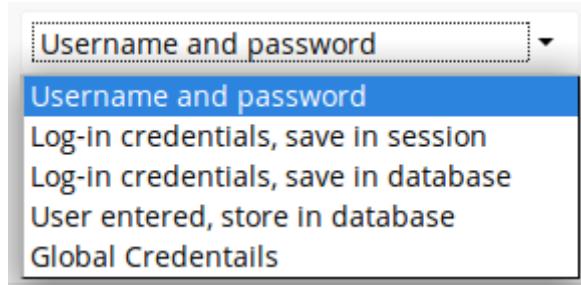
Global credentials are entered in a separate form.

External Storage

Global credentials for external storages

Username	Password	Save
----------	----------	------

Use the dropdown selector to choose the authentication backend when you create a new external mount.



LDAP Home Connector

The LDAP Home Connector App enables you to configure your ownCloud server to display your users' Windows home directories on their Files pages, just like any other folder. Typically, Windows home directories are stored on a network server in a root folder, such as Users, which then contains individual folders for each user.

You must already have the LDAP app enabled and a working LDAP/Active Directory configuration in ownCloud.

Next, configure the root Windows home directory to be mounted on your ownCloud server. Then use the LDAP Home Connector and LDAP app to connect it to ownCloud.

Mount Home Directory

Create an entry in /etc/fstab for the remote Windows root home directory mount. Store the credentials to access the home directory in a separate file, for example /etc/credentials, with the username and password on separate lines, like this:

```
username=winhomeuser  
password=winhomepassword
```

Then add a line like this to /etc/fstab, substituting your own server address and filenames:

```
//192.168.1.58/share /mnt/share cifs credentials=/etc/credentials,uid=33,gid=33
```

Configure the LDAP Home Connector

Enable the LDAP Home Connector app. Then go to the LDAP Home Connector form on your ownCloud admin page. In the **Display folder as:** field enter the name as you want it to appear on your users' File pages.

Then in the **Attribute name:** field enter the LDAP attribute name that will contain the home directory. Use any LDAP attribute that is not already in use, then save your changes.

LDAP User Home

Display folder as:	Windows Home Directory
Attribute name:	userSharedFolder
Save	

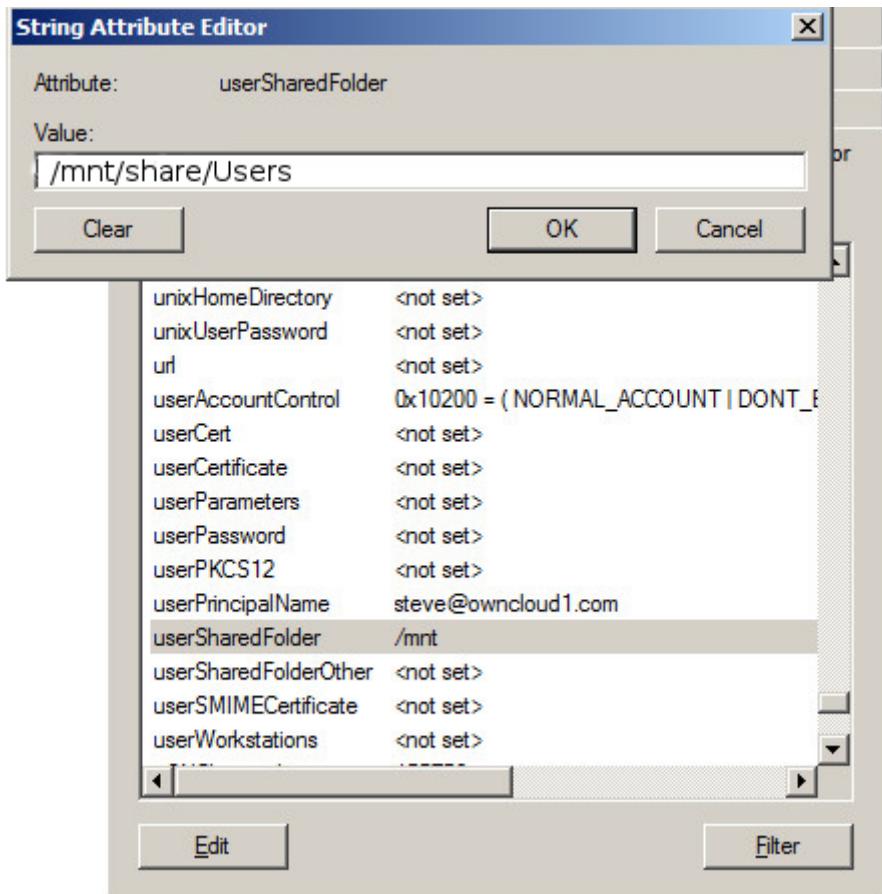
Configure the LDAP Server

In Active Directory, open the user profile. Scroll to the **Extensions** section and open the **Attribute Editor** tab

The screenshot shows the Windows Active Directory User Properties dialog box. The 'Attribute Editor' tab is selected in the tabs at the top. Below it, a table lists various attributes and their values. The 'userSharedFolder' attribute is listed with a value of '(never)'. At the bottom of the table are 'Edit' and 'Filter' buttons.

Attribute	Value
accountExpires	(never)
accountNameHistory	<not set>
aCSPolicyName	<not set>
adminCount	<not set>
adminDescription	<not set>
adminDisplayName	<not set>
altSecurityIdentities	<not set>
assistant	<not set>
attributeCertificateAttr...	<not set>
audio	<not set>
badPasswordTime	(never)
badPwdCount	0
businessCategory	<not set>
c	<not set>

Scroll to the attribute being used (UserSharedFolder in this instance), and click **Edit**. Enter the users home directory.



Save your changes, and you are finished.

Configuring SharePoint Integration

Native SharePoint support has been added to the ownCloud Enterprise edition as a secondary storage location for SharePoint 2007, 2010 and 2013. When this is enabled, users can access and sync all of their SharePoint content via ownCloud, whether in the desktop sync, mobile or Web interfaces. Updated files are bi-directionally synced automatically. SharePoint shares are created by the ownCloud admin, and optionally by any users who have SharePoint credentials.

The ownCloud SharePoint plugin uses SharePoint document lists as remote storage folders. ownCloud respects SharePoint access control lists (ACLs), so ownCloud sharing is intentionally disabled for SharePoint mountpoints. This is to preserve SharePoint ACLs and ensure content is properly accessed as per SharePoint rules.

The plugin uses the Simple Object Access Protocol (SOAP) and WebDAV for the uploads and downloads to talk to SharePoint servers. Your ownCloud server must have `php-soap` or `php5-soap` installed. Linux packages and ownCloud appliances will install `php5-soap` as a required dependency.

The supported authentication methods are:

- Basic Auth
- NTLM (Recommended)

Creating a SharePoint Mount

Enable the SharePoint app, and then enter the Admin panel to set up SharePoint connections in the SharePoint Drive Configuration section.

Enter your SharePoint Listing credentials. These credentials are not stored in the database, but are used only during plugin setup to list the Document Libraries available per SharePoint site.

SharePoint Configuration

Listing credentials. These fields are only used to list available SharePoint document list. They are not stored.

administrator

Global credentials. These fields can be used for each of the SharePoint mounts

administrator

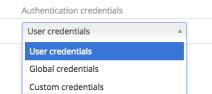
Global credentials is optional. If you fill in these fields, these credentials will be used on all SharePoint mounts where you select: **Use global credentials** as the authentication credentials.

Local Folder Name	Available for	SharePoint Site Url	Document Library
sharepoint1	All users	https://example.com	folder1
sharepoint2	All users	https://example2.com	folder2

Enter your ownCloud mountpoint in the Local Folder Name column. This is the name of the folder that each user will see on the ownCloud filesystem. You may use an existing folder, or enter a name to create a new mount point

Select who will have access to this mountpoint, by default All users, or a user or a group.

Enter your SharePoint server URL, then click the little refresh icon to the left of the Document Library field. If your credentials and URL are correct you'll get a dropdown list of available SharePoint libraries. Select the document library you want to mount.



Select which kind of Authentication credentials you want to use for this mountpoint. If you select **Custom credentials** you will have to enter the the credentials on this line. Otherwise, the global credentials or the user's own credentials will be used. Click Save, and you're done

Enabling Users

You may allow your users to create their own SharePoint mounts on their Personal pages, and allow sharing on these mounts.

- Allow users to mount their own SharePoint document libraries
- Allow users to share content in SharePoint mount points

Note

Speed up load times by disabling file previews in config.php, because the previews are generated by downloading the remote files to a temp file. This means ownCloud will spend a lot of time creating previews for all of your SharePoint content. To disable file previews, add the following line to the ownCloud config file found in /owncloud/config/config.php:

```
'enable_previews' => false,
```

Troubleshooting

Unsharing

SharePoint unsharing is handled in the background via Cron. If you remove the sharing option from a SharePoint mount, it will take a little time for the share to be removed, until the Cron job runs.

Logging

Turn on SharePoint app logging by modifying config/config.php, setting sharepoint.logging.enable to true, as in the example below.

```
'sharepoint.logging.enable' => true,
```

Mount Points

Global mount points can't be accessed: You have to fill out your SharePoint credentials as User on the personal settings page, or in the popup menu. These credentials are used to mount all global mount points.

Personal mount points can't be accessed: You have to fill your SharePoint credentials as User on the personal settings page in case your personal mount point doesn't have its own credentials.

A user can't update the credentials: Verify that the correct credentials are configured, and the correct type, either global or custom.

Installing and Configuring the External Storage: Windows Network Drives App

The [External Storage: Windows Network Drives](#) app creates a control panel in your Admin page for seamlessly integrating Windows and Samba/CIFS shared network drives as external storages.

Any Windows file share and Samba servers on Linux and other Unix-type operating systems use the SMB/CIFS file-sharing protocol. The files and directories on the SMB/CIFS server will be visible on your Files page just like your other ownCloud files and folders.

They are labeled with a little four-pane Windows-style icon, and the left pane of your Files page includes a Windows Network Drive filter. Figure 1 shows a new Windows Network Drive share marked with red warnings.

These indicate that ownCloud cannot connect to the share because it requires the user to login, it is not available, or there is an error in the configuration.

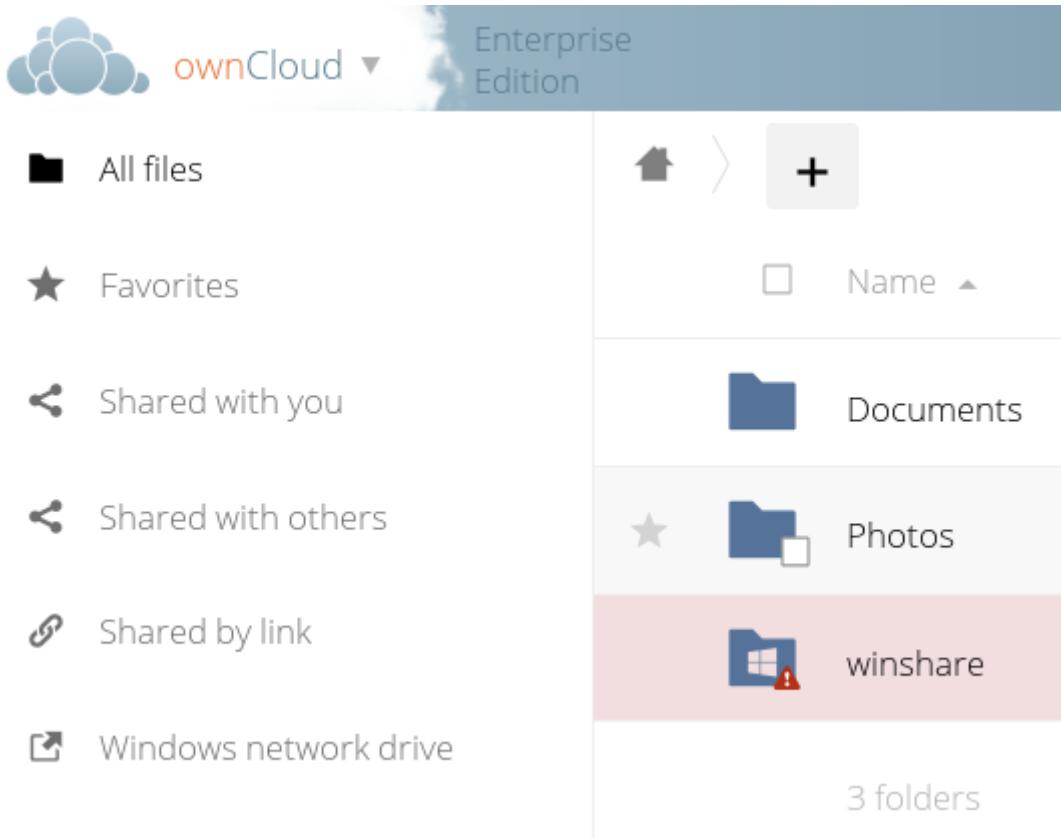


Figure 1: Windows Network Drive share on your Files page.

Files are synchronized bi-directionally, and you can create, upload, and delete files and folders. ownCloud server admins can create Windows Network Drive mounts and optionally allow users to set up their own personal Windows Network Drive mounts.

Enterprise Features

Depending on the authentication method, passwords for each mount are encrypted and stored in the ownCloud database, using a long random secret key stored in `config.php`, which allows ownCloud to access the shares when the users who own the mounts are not logged in. Or, passwords are not stored and available only for the current session, which adds security.

Installation

Install the [External Storage: Windows Network Drives app](#) from the ownCloud Market App or ownCloud Marketplace. For it to work, there are a few dependencies to install.

- A Samba client. This is included in all Linux distributions. On Debian, Ubuntu, and other Debian derivatives it is called `smbclient`. On SUSE, Red Hat, CentOS, and other Red Hat derivatives it is `samba-client`.
- `php-smbclient` (version 0.8.0+). It should be included in most Linux distributions. You can use [eduardok/libsmbclient-php](#), if your distribution does not provide it.
- `which` and `stdbuf`. These should be included in most Linux distributions.

Example

Assuming that your ownCloud installation is on Ubuntu, then the following commands will install the required dependencies:

```
# Install core packages
sudo apt-get update -y
sudo apt-get install -y smbclient php-smbclient coreutils
```

Other method using PECL is:

```
# Install php-smbclient using PECL
pecl install smbclient

# Install it from source
git clone git://github.com/eduardok/libsmbclient-php.git
cd libsmbclient-php ; phpize
./configure
make
sudo make install
```

Note

Regardless of the method you use, remember to check if an `smbclient.ini` file exists in `/etc/php/<your php version>/mods-available` and contains the following line:

```
extension="smbclient.so"
```

If so, then make it available via by running the following command:

```
sudo phpenmod -v ALL smbclient
```

Creating a New Share

When you create a new WND share you need three things:

- The login credentials for the share
- The server address, the share name; and
- The folder you want to connect to

Note

Treat all the parameters as being case-sensitive. Although some parts of the app might work properly, regardless of case, other parts might have problems if case isn't respected.

1. Enter the ownCloud mount point for your new WND share. This must not be an existing folder.
2. Then select your authentication method; See Enterprise-Only Authentication Options for complete information on the five available authentication methods.

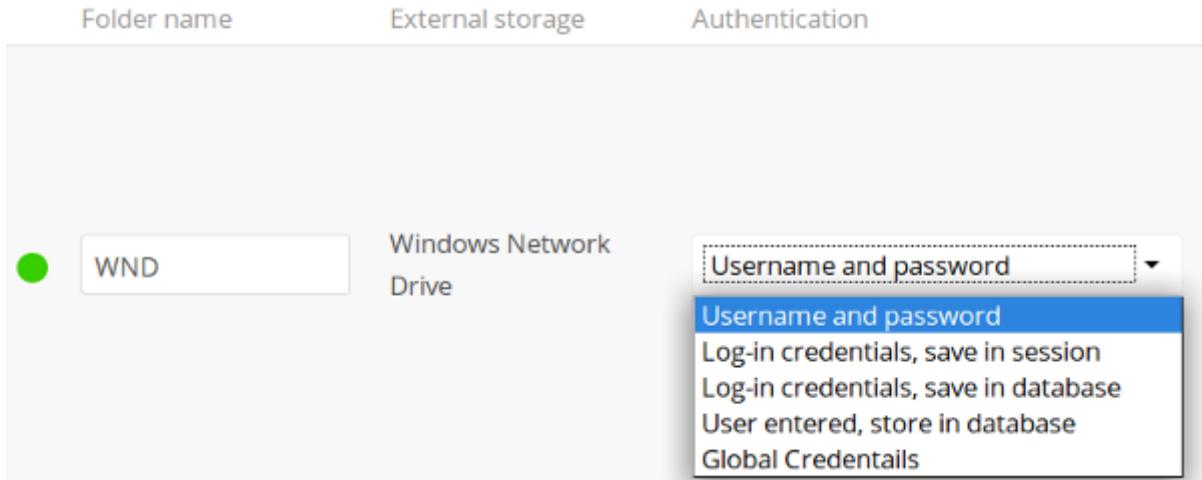


Figure 2: WND mountpoint and authorization credentials.

3. Enter the address of the server that contains the WND share.
4. The Windows share name.
5. The root folder of the share. This is the folder name, or the \$user variable for user's home directories. Note that the LDAP Internal Username Attribute must be set to the samaccountname for either the share or the root to work, and the user's home directory needs to match the samaccountname. (See User Authentication with LDAP.)
6. Login credentials.
7. Select users or groups with access to the share. The default is all users.
8. Click the gear icon for additional mount options. Note that previews are enabled by default, while sharing is not (see figure 2). Sharing is not available for all authorization methods; see Enterprise-Only Authentication Options. For large storages with many files, you may want to disable previews, because this can significantly increase performance.

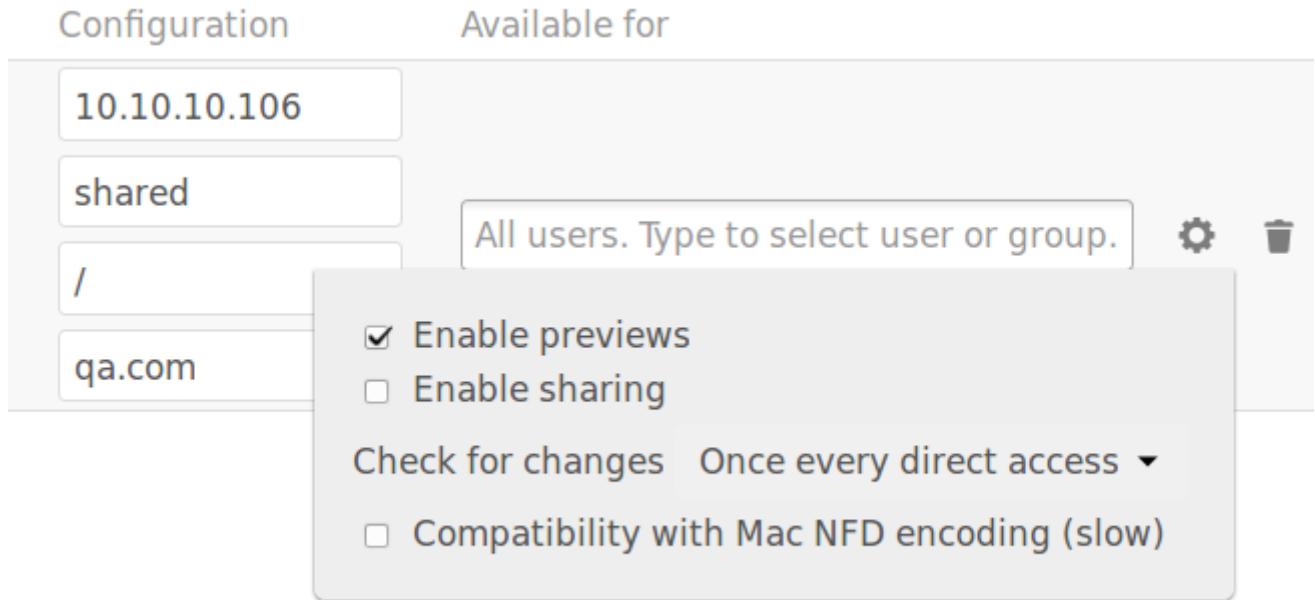


Figure 3: WND server, credentials, and additional mount options.

Your changes are saved automatically.

Note

When you create a new mountpoint using Login credentials, you must log out of ownCloud and then log back in so you can access the share. You only have to do this the first time.

Personal WND Mounts

Users create their own WND mounts on their Personal pages. These are created the same way as Admin-created shares. Users have four options for login credentials:

- Username and password
- Log-in credentials, save in session
- Log-in credentials, save in database
- Global credentials

libsmbclient Issues

If your Linux distribution ships with `libsmbclient 3.x`, which is included in the Samba client, you may need to set up the `HOME` variable in Apache to prevent a segmentation fault. If you have `libsmbclient 4.1.6` and higher it doesn't seem to be an issue, so you won't have to change your `HOME` variable. To set up the `HOME` variable on Ubuntu, modify the `/etc/apache2/envvars` file:

```
unset HOME
export HOME=/var/www
```

In Red Hat/CentOS, modify the `/etc/sysconfig/httpd` file and add the following line to set the `HOME` variable in Apache:

```
export HOME=/usr/share/httpd
```

By default, CentOS has activated SELinux, and the `httpd` process can not make outgoing network connections. This will cause problems with the `curl`, `ldap` and `samba` libraries. You'll need to get around this to make this work. First, check the status:

```
getsebool -a | grep httpd  
httpd_can_network_connect --> off
```

Then enable support for network connections:

```
setsebool -P httpd_can_network_connect 1
```

In openSUSE, modify the /usr/sbin/start_apache2 file:

```
export HOME=/var/lib/apache2
```

Restart Apache, open your ownCloud Admin page and start creating SMB/CIFS mounts.

Windows Network Drive Listener

The SMB protocol supports registering for notifications of file changes on remote Windows SMB storage servers. Notifications are more efficient than polling for changes, as polling requires scanning the whole SMB storage. ownCloud supports SMB notifications with an `occ` command, `occ wnd:listen`.

Note

The notifier only works with remote storage on Windows servers. It does not work reliably with Linux servers due to technical limitations.

Your `smbclient` version needs to be 4.x, as older versions do not support notifications. The ownCloud server needs to know about changes to files on integrated storage so that the changed files will be synced to the ownCloud server, and to desktop sync clients.

Files changed through the ownCloud Web Interface, or sync clients are automatically updated in the ownCloud file cache, but this is not possible when files are changed directly on remote SMB storage mounts.

To create a new SMB notification, start a listener on your ownCloud server with `occ wnd:listen`. The listener marks changed files, and a background job updates the file metadata.

Windows network drive connections and setup of `occ wnd:listen` often does not always work the first time. If you encounter issues using it, then try the following troubleshooting steps:

1. Check the connection with `smbclient` on the command line of the ownCloud server

Take the example of attempting to connect to the share named `MyData` using `occ wnd:listen`. Running the following command would work:

```
sudo -u www-data ./occ wnd:listen MyHost MyData svc_owncloud password
```

However, running this command would not:

```
sudo -u www-data ./occ wnd:listen MyHost mydata svc_owncloud password
```

Setting Up the WND Listener

The WND listener for ownCloud 10 includes two different commands that need to be executed:

- `wnd:listen`
- `wnd:process-queue`

wnd:listen

This command listens and stores notifications in the database coming from one specific host and share. It is intended to be run as a service. The command requires the host and share, which the listener will listen to, and the Windows/Samba account that will listen. The command does not produce any output by default, unless errors happen.

Note

You can increase the command's verbosity by using `-vvv`. Doing so displays what the listener is doing, including a timestamp and the notifications received.

Note

Although the exact permissions required for the Windows account are unknown, read-only should be enough.

The simplest way to start the `wnd:listen` process manually, perhaps for initial testing, is as follows

```
sudo -u www-data ./occ wnd:listen <host> <share> <username>
```

The password is an optional parameter and you'll be asked for it if you didn't provide it, as in the example above. In order to start the `wnd:listen` without any user interaction, provide the password as the user's 4th parameter, as in the following example:

```
sudo -u www-data ./occ wnd:listen <host> <share> <username> <password>
```

For additional options to provide the password, check Password Options

Note that in any case there won't be any processing of the password by default. This means that spaces or newline chars won't be removed unless explicitly told. Use the `--password-trim` option in those cases.

You should be able to run any of those commands, and/or wrap them into a `systemd` service or any other startup service, so that the `wnd:listen` command is automatically started during boot, if you need it.

wnd:process-queue

This command processes the stored notifications for a given host and share. This process is intended to be run periodically as a Cron job, or via a similar mechanism. The command will process the notifications stored by the `wnd:listen` process, showing only errors by default. If you need more information, increase the verbosity by calling `wnd:process-queue -vvv`.

As a simple example, you can check the following:

```
sudo -u www-data ./occ wnd:process-queue <host> <share>
```

You can run that command, even if there are no notifications to be processed.

As said, you can wrap that command in a Cron job so it's run every 5 minutes for example.

Basic Setup for One ownCloud Server

First, go to the admin settings and set up the required WND mounts. Be aware though, that there are some limitations. These are:

- We need access to the Windows account password for the mounts to update the file cache properly. This means that "*login credentials, saved in session*" won't work with the listener. "*login credentials, saved in DB*" should work and could be the best replacement.
- The `$user` placeholder in the share, such as `//host/$user/path/to/root`, for providing a share which is accessible per/user won't work with the listener. This is because the listener won't scale, as you'll need to setup one listener per/share. As a result, you'll end up with too many listeners. An alternative is to provide a common share for the users and use the `$user` placeholder in the root, such as `//host/share/$user/folder`.

Second, start the `wnd:listen` process if it's not already started, ideally running it as a service. If it isn't running, no notification are stored. The listener stores the notifications. Any change in the mount point configuration, such as adding or removing new mounts, and logins by new users, won't affect the behavior, so there is no need to restart the listener in those cases.

In case you have several mount point configurations, note that each listener attaches to one host and share. If there are several mount configurations targeting different shares, you'll need to spawn one listener for each. For example,

if you have one configuration with 10.0.0.2/share1 and another with 10.0.0.2/share2, you'll need to spawn 2 listeners, one for the first configuration and another for the second.

Third, run the wnd:process-queue periodically, usually via a Cron job. The command processes all the stored notifications for a specific host and share. If you have several, you could set up several Cron jobs, one for each host and share with different intervals, depending on the load or update urgency. As a simple example, you could run the command every 2 minutes for one server and every 5 minutes for another.

As said, the command processes all the stored notifications, squeeze them and scan the resulting folders. The process might crash if there are too many notifications, or if it has too many storages to update. The --chunk-size option will help by making the command process all the notifications in buckets of that size.

On the one hand the memory usage is reduced, on the other hand there is more network activity. We recommend using the option with a value high enough to process a large number of notifications, but not so large to crash the process. Between 200 and 500 should be fine, and we'll likely process all the notifications in one go.

Password Options

There are several ways to supply a password:

1. Interactively in response to a password prompt.

```
sudo -u www-data ./occ wnd:listen <host> <share> <username>
```

2. Sent as a parameter to the command.

```
sudo -u www-data ./occ wnd:listen <host> <share> <username> <password>
```

3. Read from a file, using the --password-file switch to specify the file to read from. Note that the password must be in plain text inside the file, and neither spaces nor newline characters will be removed from the file by default, unless the --password-trim option is added. The password file must be readable by the apache user (or www-data)

```
sudo -u www-data ./occ wnd:listen <host> <share> <username> \
--password-file=/my/secret/password/file
```

```
sudo -u www-data ./occ wnd:listen <host> <share> <username> \
--password-file=/my/secret/password/file --password-trim
```

Note

If you use the --password-file switch, the entire contents of the file will be used for the password, so please be careful with newlines.

Warning

If using --password-file make sure that the file is only readable by the apache / www-data user and inaccessible from the web. This prevents tampering or leaking of the information. The password won't be leaked to any other user using ps.

4. Using 3rd party software to store and fetch the password. When using this option, the 3rd party app needs to show the password as plaintext on standard output.

3rd Party Software Examples

```
cat /tmp/plainpass | sudo -u www-data ./occ wnd:listen <host> <share> <username> --password-
```

This provides a bit more security because the /tmp/plainpass password should be owned by root and only root should be able to read the file (0400 permissions); Apache, particularly, shouldn't be able to read it. It's expected that root will be the one to run this command.

```
base64 -d /tmp/encodedpass | sudo -u www-data ./occ wnd:listen <host> <share> <username> \  
--password-file=--
```

Similar to the previous example, but this time the contents are encoded in [Base64 format](#) (there's not much security, but it has additional obfuscation).

Third party password managers can also be integrated. The only requirement is that they have to provide the password in plain text somehow. If not, additional operations might be required to get the password as plain text and inject it in the listener.

As an example:

You can use “pass” as a password manager. You can go through <http://xmodulo.com/manage-passwords-command-line-linux.html> to setup the keyring for whoever will fetch the password (probably root) and then use something like the following

```
pass the-password-name | sudo -u www-data ./occ wnd:listen <host> <share> <username> --passw
```

Password Option Precedence

If both the argument and the option are passed, e.g., `occ wnd:listen <host> <share> <username> <password> --password-file=/tmp/pass`, then the `--password-file` option will take precedence.

Optimizing `wnd:process-queue`

Note

Do not use this option if the process-queue is fast enough. The option has some drawbacks, specifically regarding password changes in the backend.

`wnd:process-queue` creates all the storages that need to be updated from scratch. To do so, we need to fetch all the users from all the backends (currently only the ones that have logged in at least once because the others won't have the storages that we'll need updates).

To optimize this, `wnd:process-queue` make use of two switches: “`--serializer-type`” and “`--serializer-params`”. These serialize storages for later use, so that future executions don't need to fetch the users, saving precious time — especially for large organizations.

Switch	Allowed Values
<code>--serializer-type</code>	<code>file</code> . Other valid values may be added in the future, as more implementations are requested.
<code>--serializer-params</code>	Depends on <code>--serializer-type</code> , because those will be the parameters that the chosen serializer will use. For the <code>file</code> serializer, you need to provide a file location in the host FS where the storages will be serialized. You can use <code>--serializer-params file=/tmp/file</code> as an example.

While the specific behavior will depend on the serializer implementation, the overall behavior can be simplified as follows:

If the serializer's data source (such as a *file*, a *database table*, or some *Redis keys*) has storage data, it uses that data to create the storages; otherwise, it creates the storages from scratch.

After the storages are created, notifications are processed for the storages. If the storages have been created from scratch, those storages are written in the data source so that they can be read on the next run.

Note

It's imperative to periodically clean up the data source to fetch fresh data, such as for new storages and updated passwords. There isn't a generic command to do this from ownCloud, because it depends on the specific serializer type. Though this option could be provided at some point if requested.

The File Serializer

The file serializer is a serializer implementation that can be used with the `wnd:process-queue` command. It requires an additional parameter where you can specify the location of the file containing the serialized storages.

There are several things you should know about this serializer:

- The generated file contains the encrypted passwords for accessing the backend. This is necessary in order to avoid re-fetching the user information, when next accessing the storages.
- The generated file is intended to be readable and writable **only** for the web server user. Other users shouldn't have access to this file. Do not manually edit the file. You can remove the file if it contains obsolete information.

Usage Recommendations

Number of Serializers

Only one file serializer should be used per server and share, as the serialized file has to be per server and share. Consider the following usage scenario:

- If you have three shares: `10.0.2.2/share1`, `10.0.2.2/share2`, and `10.0.10.20/share2`, then you should use three different calls to `wnd:process-queue`, changing the target file for the serializer for each one.

Since the serialized file has to be per server and share, the serialized file has some checks to prevent misuse. Specifically, if we detect you're trying to read the storages for another server and share from the file, the contents of the file won't be read and will fallback to creating the storage from scratch. At this point, we'll then update the contents of that file with the new storage.

Doing so, though, creates unneeded competition, where several process-queue will compete for the serializer file. For example, let's say that you have two process-queues targeting the same serializer file. After the first process creates the file the second process will notice that the file is no longer available. As a result, it will recreate the file with new content.

At this point the first process runs again and notices that the file isn't available and recreate the file again. When this happens, the serializer file's purpose isn't fulfilled. As a result, we recommend the use of a different file per server and share.

File Clean Up

The file will need to cleaned up from time to time. The easiest way to do this is to remove the file when it is no longer needed. The file will be regenerated with fresh data the next execution if the serializer option is set.

Interaction Between Listener, Serializer, and Windows Password Lockout

Windows supports [password lockout policies](#). If one is enabled on the server where an ownCloud share is located, and a user fails to enter their password correctly several times, they may be locked out and unable to access the share.

This is a [known issue](#) that prevents these two inter-operating correctly. Currently, the only viable solution is to ignore that feature and use the `wnd:listen` and `wnd:process-queue`, without the serializer options.

There is also an additional issue to take into account though, which is that parallel runs of `wnd:process-queue` might lead to a user lockout. The reason for this is that several `wnd:process-queue` might use the same wrong password because it hasn't been updated by the time they fetch it.

As a result, it's recommended to force the execution serialization of that command to prevent this issue. You might want to use [Anacron](#), which seems to have an option for this scenario, or wrap the command with [flock](#).

If you need to serialize the execution of the `wnd:process-queue`, check the following example with [flock](#)

```
flock -n /my/lock/file sudo -u www-data ./occ wnd:process-queue <host> <share>
```

In that case, flock will try get the lock of that file and won't run the command if it isn't possible. For our case, and considering that file isn't being used by any other process, it will run only one `wnd:process-queue` at a time. If someone tries to run the same command a second time while the previous one is running, the second will fail and won't be executed. Check [flock's documentation](#) for details and other options.

Multiple Server Setup

Setups with several servers might have some difficulties in some scenarios:

- The `wnd:listen` component *might* be duplicated among several servers. This shouldn't cause a problem, depending on the limitations of the underlying database engine. The supported database engines should be able to handle concurrent access and de-duplication.
- The `wnd:process-queue` *should* also be able to run from any server, however limitations for concurrent executions still apply. As a result, you might need to serialized command execution of the `wnd:process-queue` among the servers (to avoid for the password lockout), which might not be possible or difficult to achieve. You might want to execute the command from just one specific server in this case.
- `wnd:process-queue + serializer`. First, check the above section to know the interactions with the password lockout. Right now, the only option you have to set it up is to store the target file in a common location for all the server. We might need to provide a specific serializer for this scenario (based on Redis or DB)

Basic Command Execution Examples

```
sudo -u www-data ./occ ``wnd:listen`` host share username password  
sudo -u www-data ./occ ``wnd:process-queue`` host share  
sudo -u www-data ./occ ``wnd:process-queue`` host share -c 500  
sudo -u www-data ./occ ``wnd:process-queue`` host share -c 500 \  
  --serializer-type file \  
  --serializer-params file=/opt/oc/store  
sudo -u www-data ./occ ``wnd:process-queue`` host2 share2 -c 500 \  
  --serializer-type File \  
  --serializer-params file=/opt/oc/store2
```

To set it up, make sure the listener is running as a system service:

```
sudo -u www-data ./occ ``wnd:listen`` host share username password
```

Setup a Cron job or similar with something like the following two commands:

```
sudo -u www-data ./occ wnd:process-queue host share -c 500 \  
  --serializer-type file \  
  --serializer-params file=/opt/oc/store1  
  
rm -f /opt/oc/store1 # With a different schedule
```

The first run will create the `/opt/oc/store1` with the serialized storages, the rest of the executions will use that file. The second Cron job, the one removing the file, will force the `wnd:process-queue` to refresh the data.

It's intended to be run in a different schedule, so there are several executions of the `wnd:process-queue` fetching the data from the file. Note that the file can be removed manually at any time if it's needed (for example, the admin has reset some passwords, or has been notified about password changing).

Configuring S3 and OpenStack Swift Objects as Primary Storage

In ownCloud Enterprise edition, you can configure S3 objects as primary storage. This replaces the default ownCloud `owncloud/data` directory. You may still need to keep the `owncloud/data` directory for these reasons:

- The ownCloud log file is saved in the data directory
- Legacy apps may not support using anything but the `owncloud/data` directory

You can move your logfile by changing its location in `config.php`. You may still need `owncloud/data` for backwards compatibility with some apps.

Implications

This configuration needs to be applied before the first login of any user – including the admin user. Otherwise, you will run into an error condition, because ownCloud cannot find the user's files anymore.

ownCloud in object store mode expects exclusive access to the object store container, because it only stores the binary data for each file. The metadata are kept in the local database for performance reasons.

The current implementation is incompatible with any app that uses direct file I/O and circumvents the ownCloud virtual filesystem. That includes Encryption and Gallery. Gallery stores thumbnails directly in the filesystem, and Encryption causes severe overhead because key files need to be fetched in addition to any requested file.

Configuration

Look in `config.sample.php` for a example configurations. Copy the relevant part to your `config.php` file. Any object store needs to implement `\OCP\Files\ObjectStore\IObjectStore` and can be passed parameters in the constructor with the arguments key:

```
'objectstore' => [
    'class' => 'Implementation\\Of\\OCP\\Files\\ObjectStore\\IObjectStore',
    'arguments' => [
        ...
    ],
],
```

Amazon S3

The S3 backend mounts a bucket of the Amazon S3 object store into the virtual filesystem. The class to be used is `OCA\ObjectStore\S3`:

```
'objectstore' => [
    'class' => 'OCA\ObjectStore\S3',
    'arguments' => [
        // replace with your bucket
        'bucket' => 'owncloud',
        'autocreate' => true,
        // uncomment to enable server side encryption
        //'serversideencryption' => 'AES256',
        'options' => [
            // version and region are required
            'version' => '2006-03-01',
            // change to your region
            'region' => 'eu-central-1',
            'credentials' => [
                // replace key and secret with your credentials
                'key' => 'EJ39ITYZEUH5BGWDRUFY',
                'secret' => 'M5MrXTRjkyMaxXPe2FRXMTfTfbKEnZCu+7uRTVSj',
            ],
        ],
    ],
],
```

Ceph S3

The S3 backend can also be used to mount the bucket of a ceph object store via the s3 API into the virtual filesystem. The class to be used is `OCA\ObjectStore\S3`:

```
'objectstore' => [
  'class' => 'OCA\ObjectStore\S3',
  'arguments' => [
    // replace with your bucket
    'bucket' => 'owncloud',
    'autocreate' => true,
    'options' => [
      // version and region are required
      'version' => '2006-03-01',
      'region' => '',
      // replace key, secret and bucket with your credentials
      'credentials' => [
        // replace key and secret with your credentials
        'key' => 'EJ39ITYZEUH5BGWDRUFY',
        'secret' => 'M5MrXTRjkyMaxXPe2FRXMTfTfbKENZCu+7uRTVSj',
      ],
      // replace the ceph endpoint with your rgw url
      'endpoint' => 'http://cephhost:8000/',
      // Use path style when talking to ceph
      'command.params' => [
        'PathStyle' => true,
      ],
    ],
  ],
],
```

S3 Multibucket Configuration

ownCloud 9.1+ has multibucket support for S3 object stores:

```
'objectstore_multibucket' => [
  'class' => 'OCA\ObjectStore\S3',
  'arguments' => [
    'autocreate' => true,
    'options' => [
      'version' => '2006-03-01',
      'region' => 'eu-central-1',
      'credentials' => [
        'key' => 'EJ39ITYZEUH5BGWDRUFY',
        'secret' => 'M5MrXTRjkyMaxXPe2FRXMTfTfbKENZCu+7uRTVSj',
      ],
    ],
  ],
],
```

OpenStack Swift

The Swift backend mounts a container on an OpenStack Object Storage server into the virtual filesystem. The class to be used is \\\OC\\\Files\\\ObjectStore\\\Swift:

```
'objectstore' => [
  'class' => 'OC\Files\ObjectStore\Swift',
  'arguments' => [
    'username' => 'demo',
    'password' => 'password',
    'container' => 'owncloud',
    'autocreate' => true,
    'region' => 'RegionOne',
    'url' => 'http://devstack:5000/v2.0',
    'tenantName' => 'demo',
    'serviceName' => 'swift',
```

Enterprise Features

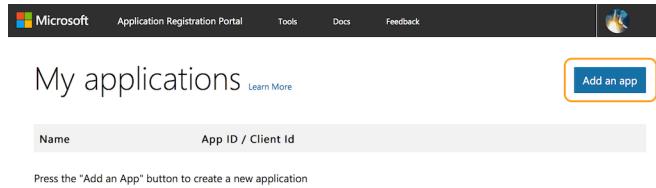
```
// url Type, optional, public, internal or admin
'urlType' => 'internal'
],
]
```

How to Create and Configure Microsoft OneDrive

To use Microsoft OneDrive as an external storage option in ownCloud, you need to do two things:

1. Create an application configuration
2. Configure a mount point in ownCloud

Create an Application Configuration



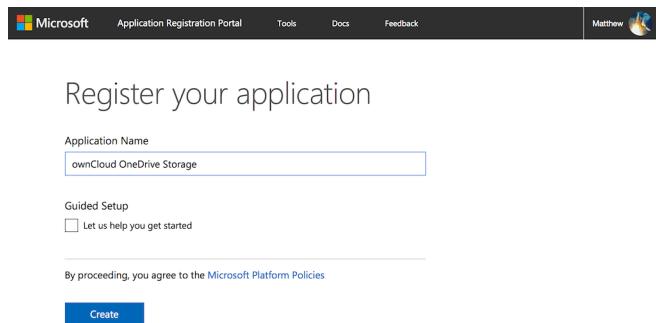
The screenshot shows the Microsoft Application Registration Portal. At the top, there's a navigation bar with links for Microsoft, Application Registration Portal, Tools, Docs, and Feedback. On the right side of the header is a user profile icon. Below the header, the main area is titled "My applications" with a "Learn More" link. There's a search bar and a table with columns for "Name" and "App ID / Client Id". A large blue button labeled "Add an app" is prominently displayed at the bottom right of the table area. A small note below the table says "Press the 'Add an app' button to create a new application".

To create a new application:

- Open <https://apps.dev.microsoft.com/> in your browser of choice and click “Create App”.
- Under “Properties”, set the application’s name.
- Click “Create”.

With the application created, you can then add a range of further settings. However, only a few of them are required for use with ownCloud.

Application Password



The screenshot shows the "Register your application" page. At the top, there's a Microsoft logo and a "Create App" button. The main form has fields for "Application Name" (containing "ownCloud OneDrive Storage"), "Guided Setup" (with a checkbox for "Let us help you get started"), and a "Create" button. Below the form, there's a note about agreeing to Microsoft Platform Policies.

Under “Application Secrets”, click “Generate New Password”, which generates a password and displays it in a popup window. It is required later during when configuring a mount point.

Note

Copy the password to your preferred password manager, as it is only displayed **once**.

Redirect URLs

Under “Platforms”, click “Add Platform” and choose “Web” in the popup window which appears. Only one redirect URL field is visible at first, so click “Add URL” to add another one.

Enterprise Features

With two fields available, add two redirect URLs; one for `settings/admin` and one for `settings/personal`, as you can see in the image below.

Platforms

Add Platform

Web

Allow Implicit Flow

Redirect URLs [Add URL](#)

Logout URL [Edit](#)

Delete

Application Permissions

Microsoft Graph Permissions

The settings you set here may vary depending on whether you get a token from our V1 or V2 endpoint. [What's the difference?](#)

Delegated Permissions [Add](#) [About delegated permissions](#)

User.Read [X](#)

[Application Permissions](#) [Add](#) [About application permissions](#)

Under “*Microsoft Graph Permissions*”, click “*Add*” next to “*Application Permissions*”. This opens a popup window where you can choose the required permissions. Add at least the following four:

- `Files.Read.All`
- `Files.ReadWrite.All`
- `IdentityRiskEvent.Read.All`
- `User.Read.All`

With those settings added, click “Save”, located right at the bottom of the page.

Configure a Mount Point in ownCloud

You can add as many OneDrive mount points as you want. To do so:

1. Add a new storage, selecting “One Drive” for external storage.
2. Set the credentials of your OneDrive application, and then accept the permissions.
3. If everything is accepted, the mount points should appear, with a green status icon on the far left-hand side.

External Storage

Enable external storage

Global credentials for external storage

Username Password Save

Folder name	External storage	Authentication	Configuration	Available for
OneDrive_old	One Drive	OneDrive OAuth2 ▾	https://[REDACTED].1.e31-4948	All users. Type to select user or group. Edit Delete
OneDrive	One Drive	OneDrive OAuth2 ▾	https://[REDACTED].7659-4150	All users. Type to select user or group. Edit Delete
OneDrive_u2_app4	One Drive	OneDrive OAuth2 ▾	https://[REDACTED].4236-0104	All users. Type to select user or group. Edit Delete
OneDrive1_u1_app3	One Drive	OneDrive OAuth2 ▾	https://[REDACTED].446-0725-0000-0000	All users. Type to select user or group. Edit Delete
OneDrive1_u1_app6	One Drive	OneDrive OAuth2 ▾	https://[REDACTED].1.e31-4948	All users. Type to select user or group. Edit Delete
OneDrive_u2_app6	One Drive	OneDrive OAuth2 ▾	https://[REDACTED].1.e31-4948	All users. Type to select user or group. Edit Delete

Folder name [Add storage](#)

To be able to use the `occ` command `files_onedrive:subscribe`, you need to have the variable `overwrite.cli.url` set in `config/config.php`, as in this example:

```
'overwrite.cli.url' => 'https://example.org:63984/index.php',
```

Note

The HTTPS prefix, port, and /index.php suffix are mandatory.

User Management

Shibboleth Integration

Introduction

The ownCloud Shibboleth user backend application integrates ownCloud with a Shibboleth Service Provider (SP) and allows operations in federated and single-sign-on (SSO) infrastructures. Setting up Shibboleth has two big steps:

1. Enable and configure the Apache Shibboleth module.
2. Enable and configure the ownCloud Shibboleth app.

The Apache Shibboleth module

Currently supported installations are based on the [native Apache integration](#). The individual configuration of the service provider is highly dependent on the operating system, as well as on the integration with the Identity Providers (IdP), and require case-by-case analysis and installation.

A good starting point for the service provider installation can be found in [the official Shibboleth Wiki](#).

A successful installation and configuration will populate Apache environment variables with at least a unique user id which is then used by the ownCloud Shibboleth app to login a user.

Apache Configuration

This is an example configuration as installed and operated on a Linux server running the Apache 2.4 Web server. These configurations are highly operating system specific and require a high degree of customization.

The ownCloud instance itself is installed in /var/www/owncloud/. The following aliases are defined in an Apache virtual host directive:

Further Shibboleth specific configuration as defined in /etc/apache2/conf.d/shib.conf:

```
# Load the Shibboleth module.
LoadModule mod_shib /usr/lib64/shibboleth/mod_shib_24.so

# Ensure handler will be accessible
<Location /Shibboleth.sso>
    AuthType None
    Require all granted
</Location>

# always fill env with shib variable for logout url
<Location />
    AuthType shibboleth
    ShibRequestSetting requireSession false
    Require shibboleth
</Location>

# authenticate only on the login page
<Location ~ "^(/index.php)?/login">
    # force internal users to use the IdP
    <If ">R '192.168.1.0/24'">
        AuthType shibboleth
        ShibRequestSetting requireSession true
    </If>
</Location>
```

```
        require valid-user
</If>
# allow basic auth for eg. guest accounts
<Else>
    AuthType shibboleth
    ShibRequestSetting requireSession false
    require shibboleth
</Else>
</Location>

# shib session for css, js and woff not needed
<Location ~ "\.*\.(css|js|woff)">
    AuthType None
    Require all granted
</Location>
```

To allow users to login via the IdP, add a login alternative with the `login.alternatives` option in `config.php`.

Depending on the ownCloud Shibboleth app mode, you may need to revisit this configuration.

The ownCloud Shibboleth App

After enabling the Shibboleth app on your Apps page, you need to choose the app mode and map the necessary Shibboleth environment variables to ownCloud user attributes on your Admin page.

Shibboleth

App Mode **Not active**

Environment
Autoprovision Users
Single sign-on only

Use **Shib-Session-ID** as Shibboleth session

Use **eppn** as uid

Use **eppn** as email

Use **eppn** as display name

Server Environment:

htaccessWorking	true
HTTP_HOST	docker.oc.solidgear.es:53738
HTTP_USER_AGENT	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:42.0) Gecko/20100101 Firefox/42.0
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE	en-US,en;q=0.5
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_DNT	1
HTTP_COOKIE	PHPSESSID=nlkrv949lmuo7dpkgb91gbe13; ocy0:1
HTTP_CONNECTION	keep-alive
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE	<address>Apache/2.4.7 (Ubuntu) Server at docker.oc.solidgear.es
SERVER_SOFTWARE	Apache/2.4.7 (Ubuntu)
SERVER_NAME	docker.oc.solidgear.es
SERVER_ADDR	172.17.1.245
SERVER_PORT	53738
REMOTE_ADDR	166.176.185.154
DOCUMENT_ROOT	/opt/owncloud
REQUEST_SCHEME	http

figure 1: Enabling Shibboleth on the ownCloud Admin page

Choosing the App Mode

After enabling the app it will be in **Not active** mode, which ignores a Shibboleth session and allows you to login as an administrator and inspect the currently available Apache environment variables. Use this mode to set up the environment mapping for the other modes, and in case you locked yourself out of the system. You can also change the app mode and environment mappings by using the `occ` command, like this example on Ubuntu Linux:

```
$ sudo -u www-data php occ shibboleth:mode notactive
$ sudo -u www-data php occ shibboleth:mapping --uid login
```

In **Single sign-on only** mode the app checks if the environment variable for the Shibboleth session, by default **Shib-Session-Id**, is set. If that is the case it will take the value of the environment variable as the `uid`, by default

eppn, and check if a user is known by that uid. In effect, this allows another user backend, e.g., the LDAP app, to provide the displayname, email and avatar.

Note

As an example the IdP can send the **userPrincipalName** which the Apache Shibboleth module writes to a custom Apache environment variable called `login`. The ownCloud Shibboleth app reads that `login` environment variable and tries to find an LDAP user with that `username`. For this to work **userPrincipalName** needs to be added to the **Additional Search Attributes** in the LDAP directory settings on the advanced tab. We recommend using a scoped login attribute like **userPrincipalName** or **mail** because otherwise the search might find multiple users and prevent login.

Note

In many scenarios Shibboleth is not intended to hide the user's password from the service provider, but only to implement SSO. If that is the case it is sufficient to protect the ownCloud base url with Shibboleth. This will send Web users to the IdP but allow desktop and mobile clients to continue using username and password, preventing popups due to an expired Shibboleth session lifetime.

In **Autoprovision Users** mode the app will not ask another user backend, but instead provision users on the fly by reading the two additional environment variables for display name and email address.

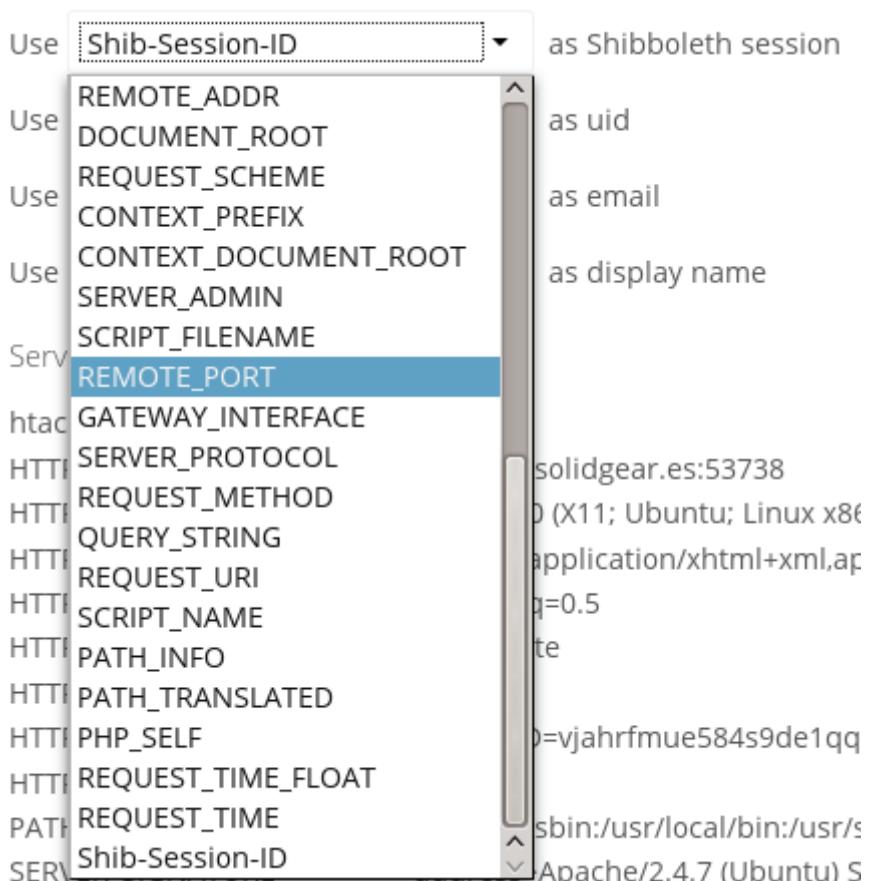


figure 2: Mapping Shibboleth environment configuration variables to ownCloud user attributes

In ownCloud 8.1 the Shibboleth environment variable mapping was stored in `apps/user_shibboleth/config.php`. This file was overwritten on upgrades, preventing a seamless upgrade procedure. In ownCloud 8.2+ the variables are stored in the ownCloud database, making Shibboleth automatically upgradeable.

Mapping ownCloud User IDs

From 3.1.2 you can now specify a mapper that is used on inbound ownCloud user IDs, to adjust them before usage in ownCloud. You can set the mapper using `occ`:

```
$ sudo -u www-data php occ config:app:set user_shibboleth uid_mapper --value="OCA\User_Shibboleth\Mapper\NoOpMapper"
```

You may view the currently configured mapper using:

```
$ sudo -u www-data php occ shibboleth:mapping
```

The following mappers are provided with the app:

- OCA\User_Shibboleth\Mapper\NoOpMapper - the default, does not alter the uid
- OCA\User_Shibboleth\Mapper\ADFSMapper - splits the uid around a ; character and takes the first piece
- OCA\User_Shibboleth\Mapper\GUIDInMemoryMapper - maps in binary GUIDs to strings

Shibboleth with Desktop and Mobile Clients

The ownCloud Desktop Client can interact with an ownCloud instance running inside a Shibboleth Service Provider by using OAuth2 tokens to authenticate.

The ownCloud Android and iOS mobile apps also work with OAuth2 tokens.

WebDAV Support

Users of standard WebDAV clients can generate an App Password on the Personal settings page. Use of App Passwords may be enforced with the `token_auth_enforced` option in `config.php`

Known Limitations

Encryption

File encryption can only be used together with Shibboleth when the master key-based encryption is used because the per-user encryption requires the user's password to unlock the private encryption key. Due to the nature of Shibboleth the user's password is not known to the service provider.

Other Login Mechanisms

You can allow other login mechanisms (e.g. LDAP or ownCloud native) by creating a second Apache virtual host configuration. This second location is not protected by Shibboleth, and you can use your other ownCloud login mechanisms.

Session Timeout

Session timeout on Shibboleth is controlled by the IdP. It is not possible to have a session length longer than the length controlled by the IdP. In extreme cases this could result in re-login on mobile clients and desktop clients every hour.

UID Considerations and Windows Network Drive compatibility

To log in LDAP users via SAML for Single Sign On the user in LDAP must be uniquely resolvable by searching for the username that was sent in the SAML token. For this to work the `ldap` attribute containing the username needs to be added to the **Additional Search Attributes** in the LDAP directory settings on the advanced tab. We recommend using a scoped login attribute like `userPrincipalName` or `mail` because otherwise the search might find multiple users and prevent login.

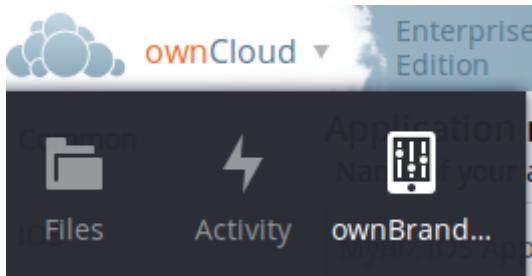
`user_shibboleth` will do the authentication, and `user_ldap` will provide user details such as `email` and `displayname`.

Creating Branded ownCloud Clients

Creating Branded Client Apps

Overview

ownBrander is an ownCloud build service that is exclusive to Enterprise customers for creating branded Android and iOS ownCloud sync apps, and branded ownCloud desktop sync clients. You build your apps with the ownBrander app on your Customer.owncloud.com account, and within 24-48 hours the completed, customized apps are loaded into your account. You must supply your own artwork, and you'll find all the specifications and required elements in ownBrander.



Building a Branded Desktop Sync Client

See [Building Branded ownCloud Clients](#) for instructions on building your own branded desktop sync client, and for setting up an automatic update service.

Your users may run both a branded and un-branded desktop sync client side-by-side. Both clients run independently of each other, and do not share account information or files.

Building a Branded iOS App

Building and distributing your branded iOS ownCloud app involves a large number of interdependent steps. The process is detailed in the [Building Branded ownCloud Clients](#) manual. Follow these instructions exactly and in order, and you will have a nice branded iOS app that you can distribute to your users.

Building an Android App

Building and distributing your branded Android ownCloud app is fairly simple, and the process is detailed in [Building Branded ownCloud Clients](#).

Custom Client Download Repositories

See [Custom Client Download Repositories](#) to learn how test and configure custom download repository URLs for your branded clients.

Logging Apps

Enterprise Logging Apps

The **Log user and file sharing actions** app (`apps/admin_audit`) records the file sharing activity of your users, file tagging, and user logins and logouts.



Log user and file sharing actions 0.7

by ownCloud Inc. / Bart Visscher (Commercial-licensed)

✓ Official

Show description ...

Disable

Your logging level must be set to at least **Info, warnings, errors, and fatal issues** on your ownCloud admin page, or 'loglevel' => 1 in config.php.

View your logfiles on your admin page. Click the **Download logfile** button to dump the plain text log, or open the logfile directly in a text editor. The default location is owncloud/data/owncloud.log.

See Logging Configuration and Advanced File Tagging With the Workflow App for more information on logging and tagging.

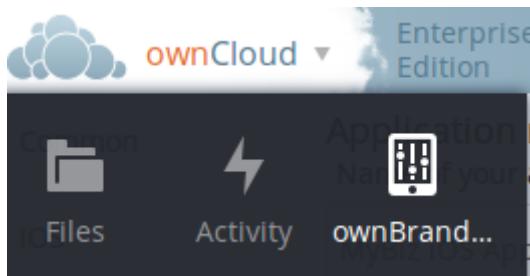
Server Branding

Custom Theming ownCloud

Overview

ownBrander is an ownCloud build service that is exclusive to Enterprise edition customers for creating branded ownCloud clients and servers. You may brand your ownCloud server using ownBrander to easily build a **custom theme**, using your own logo and artwork. ownCloud has always been theme-able, but it was a manual process that required editing CSS and PHP files. Now Enterprise customers can use ownBrander, which provides an easy graphical wizard.

You need an Enterprise subscription, an account on Customer.owncloud.com, and the ownBrander app enabled on your account. When you complete the steps in the wizard the ownBrander service builds your new branded theme, and in 24-48 hours you'll see it in your account.



When you open the ownBrander app, go to the Web tab. You will see an introduction and the wizard, which starts with uploading your logo. You will need a number of images in specific sizes and formats, and the wizard tells you what you need. Example images are on the right, and you can click to enlarge them.

Login logo images



Login page logo

This is the image shown on the login page just above the Username and Password fields (svg format) (width: 252px height: 122px) [i](#)

[Upload](#)



Login page logo

This is the image shown on the login page just above the Username and Password fields. This image is used when the browser does not support svg, we recommend this to be the same as the previous one (Login page logo) (png format) (width: 252px height: 122px) [i](#)

[Delete image](#)

[Upload](#)



Logo icon



Note

If you see errors when you upload SVG files, such as “Incorrect extension.File type image/svg+xml is not correct”, “This SVG is invalid”, or “Error uploading file: Incorrect size”, try opening the file in [Inkscape](#) then save as “Plain SVG” and upload your SVG image again.

The wizard has two sections. The first section contains all the required elements: logos and other artwork, colors, naming, and your enterprise URL. The Suggested section contains optional items such as additional logo placements and custom URLs.

When you are finished, click the **Generate Web Server** button. If you want to change anything, go ahead and change it and click the **Generate Web Server** button. This will override your previous version, if it has not been created yet. In 24-48 hours you'll find your new branded theme in the **Web** folder in your [Customer.owncloud.com](#) account.

Inside the **Web** folder you'll find a **themes** folder. Copy this to your `owncloud/themes` directory. You may name your **themes** folder anything you want, for example `myBrandedTheme`. Then configure your ownCloud server to use your branded theme by entering it in your `config.php` file:

```
"theme" => "myBrandedTheme"
```

If anything goes wrong with your new theme, comment out this line to re-enable the default theme until you fix your branded theme. The branded theme follows the same file structure as the default theme, and you may further customize it by editing the source files. .. Note:: Always edit only your custom theme files. Never edit the default

theme files.

The ownCloud X Appliance

What is the Appliance?

If you don't know a lot about Linux, only have a small IT staff, or are your IT staff — even if that's only in your spare time — the ownCloud X Appliance will let you get started using ownCloud quickly and easily.

The Appliance:

- Provides a pre-packaged, easy to deploy ownCloud, ready for you in most popular virtual machine formats, including *ESX*, *VirtualBox*, *KVM* and *VMware*.
- Contains the ownCloud 10 virtual image, and all the additional software you need to get up and running on ownCloud in minutes; this includes: *ownCloud X Server and Enterprise Apps*, *Apache 2*, *PHP*, and *MySQL*.
- Scales up to 500 users. Depending on the intensity and pattern of use, this can vary from 400 up to 600 users.

Note

Some configurations, such as SAML IDPs, or LDAP or AD instances, may need additional configuration to connect.

How to Install the Appliance

The install process is a little involved, but not too much. To keep it succinct, you need to:

- Download and Install the appliance
- Step through **the configuration wizard**
- Activate the configured appliance

Important

You need **Internet access** to use the appliance. The appliance has to be activated with a license that you will receive from Univention via email. This license has to be imported in the appliance via the **web interface**. The appliance also needs access to a DHCP server so that it can receive an IP address and be accessible.

After that, you can access the running instance of ownCloud and further configure it to suit your needs.

Download the Appliance

First off, you need to download the ownCloud X Appliance from [the ownCloud download page](#) and click "DOWNLOAD NOW". This will display a form, which you can see a sample of below, which you'll need to fill out. It will ask you for the following details:

- Email address
- Download version (*ESXi*, *VirtualBox*, *VMware*, *KVM*)
- Your first, last, and company names, and your country of origin

The ownCloud X Appliance

The image contains two screenshots of the ownCloud X Trial Appliance download page. The left screenshot shows a form with fields for 'Email' (matthew@matthewsetter.com) and 'Download Version' (ownCloud 10.0.1 Trial Appliance (VirtualBox)). It also includes a checkbox for accepting terms and conditions. The right screenshot shows a summary page with a search bar, language selection (EN), and a link to the changelog for the latest stable version (10.0.1).

After you've filled out the form, click “**DOWNLOAD OWN CLOUD**” to begin the download of the virtual appliance.

Note

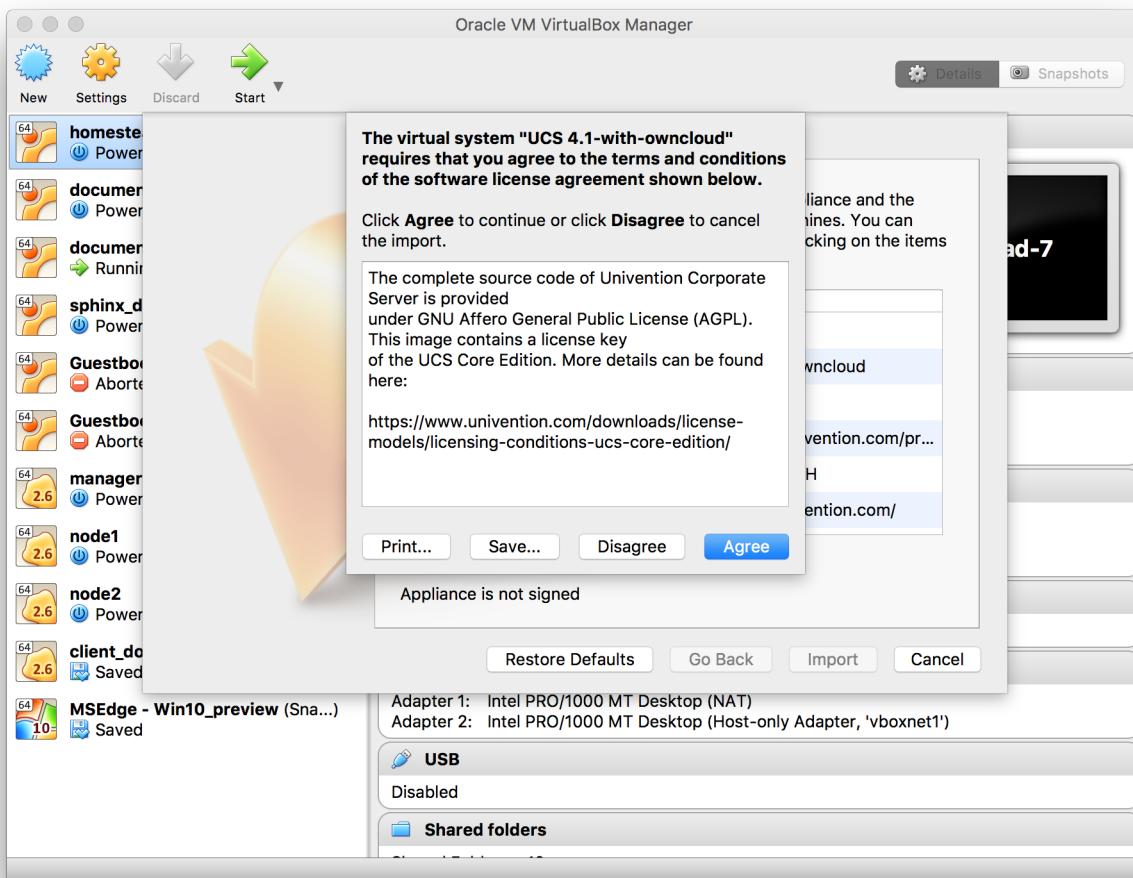
The virtual appliance files are around 1.4GB in size, so may take some time, depending on your network bandwidth.

Note

You can also download it from [the owncloud.org page](#).

Install the Appliance

Once you've downloaded the virtual appliance file, import it into your virtualization software, accept the T's & C's of the license agreement, and launch it. The example below shows this being done using VirtualBox.



Note

If you try to install an ownCloud appliance in your domain after removing an existing one, please remember to remove the original one from your DNS configuration.

Important

Don't Forget the **IP Address** and the **Administrator Password**. You will need them to use the Appliance.

Start the Appliance

Once imported, start the appliance. Doing so launches the installer wizard which helps you specify the core configuration. This includes:

Localization settings: Here, you can specify the language, timezone, and keyboard layout. Domain and network configuration: These settings can be either obtained automatically, via a DHCP lookup, or provided manually.

Domain setup: This lets you manage users and permissions directly within the ownCloud installation in the virtual appliance, or to make use of an existing Active Directory or UCS domain.

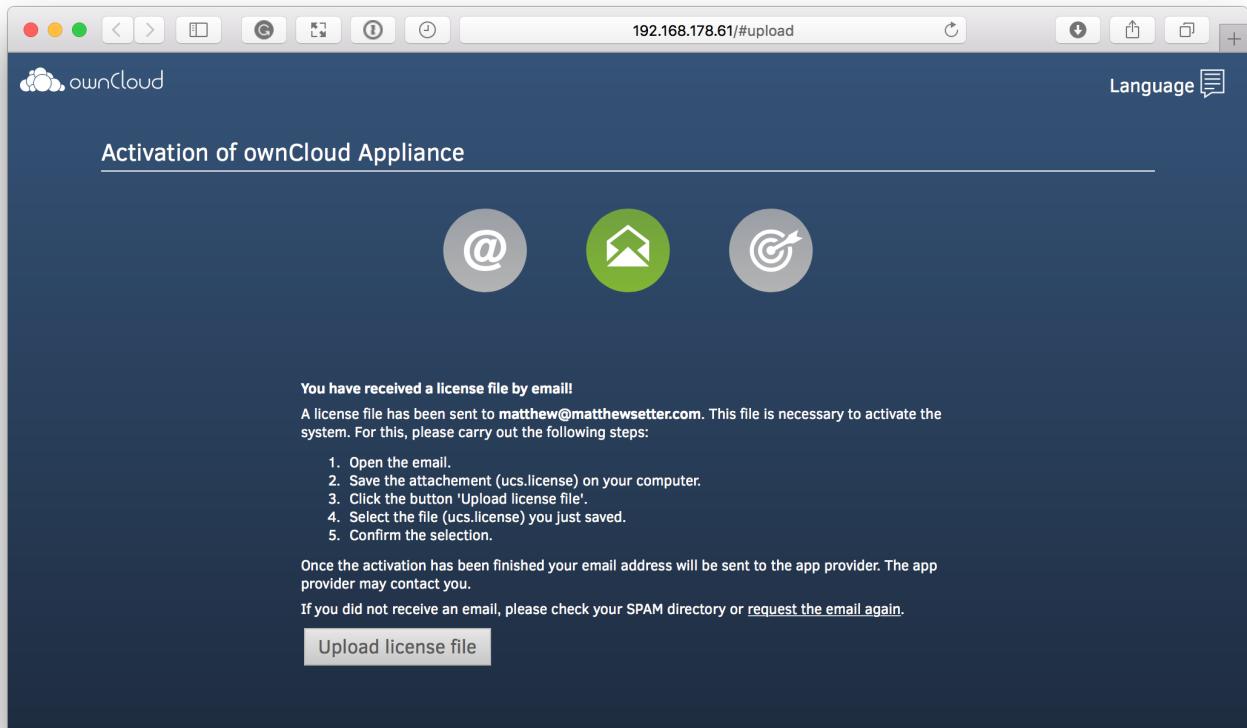
Account information: This lets you specify your organisation's name, the email address (used for receiving the license which you'll need to activate the appliance), and the administrator password. Note, this password is for the administrator (or root user) of the virtual machine, not for the ownCloud installation.

Host settings: This lets you specify the fully-qualified domain name of the virtual appliance, as well as an LDAP Base DN.

Once you've provided all of the required information, you can then finish the wizard, which will finish building the virtual appliance. Make sure that you double-check the information provided, so that you don't have to start over.

Activate the Appliance

When the wizard completes, the virtual machine will be almost ready to use. You then need only retrieve the license file from the email which was sent to you and upload it. The page to do that from can be found by opening your browser to the IP address of the virtual appliance, as you can see below. The installer may instruct you to use `https://` to access the activation page. If this gives an error in the browser, then remove the `https://`.

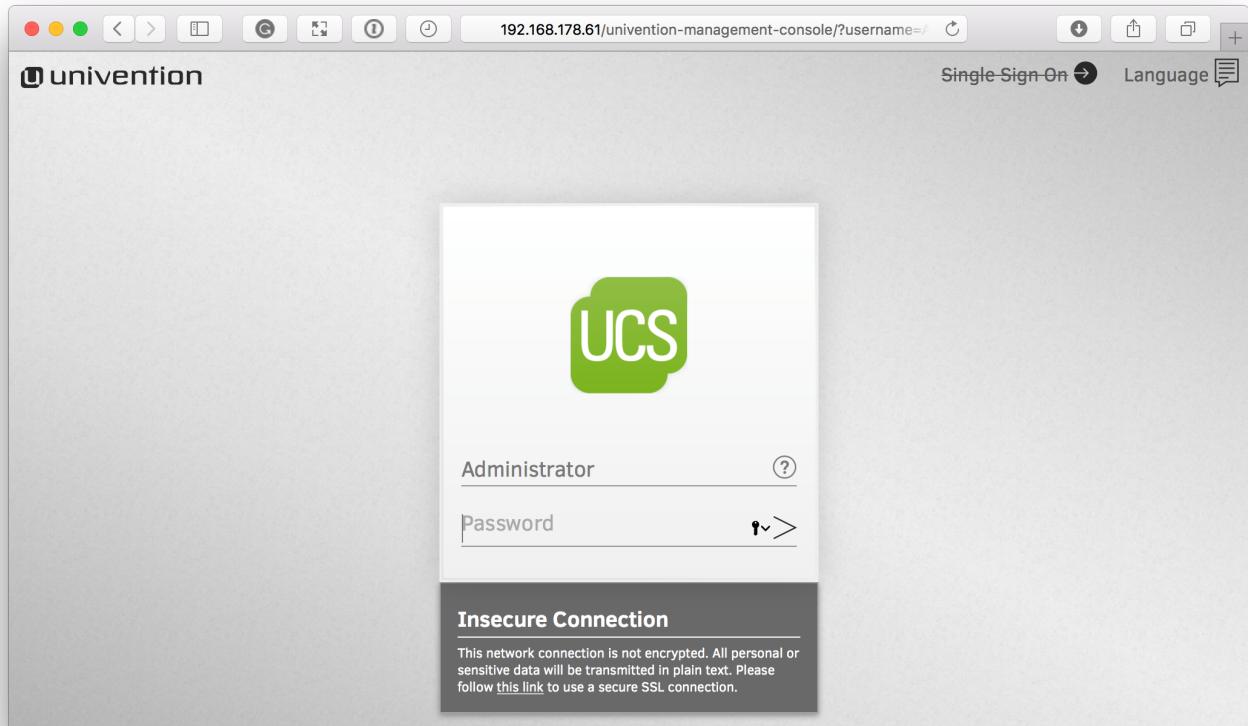


Administer the Appliance

Once activated, you should be redirected to the portal, which you can see below.

The screenshot shows the ownCloud Portal interface. At the top, there is a header with the ownCloud logo, a search icon, a lock icon, a menu icon, and the Univention logo. Below the header, the "Applications" section is displayed, featuring a card for "ownCloud" with its logo, name, and URL (ucs-123.owncloud.in...). The "Administration" section follows, containing four cards: "System and domain settings" (with a green gear icon), "Admin Manual" (with the ownCloud logo), "User Manual" (with the ownCloud logo), and "Univention Blog" (with a red 'U' icon).

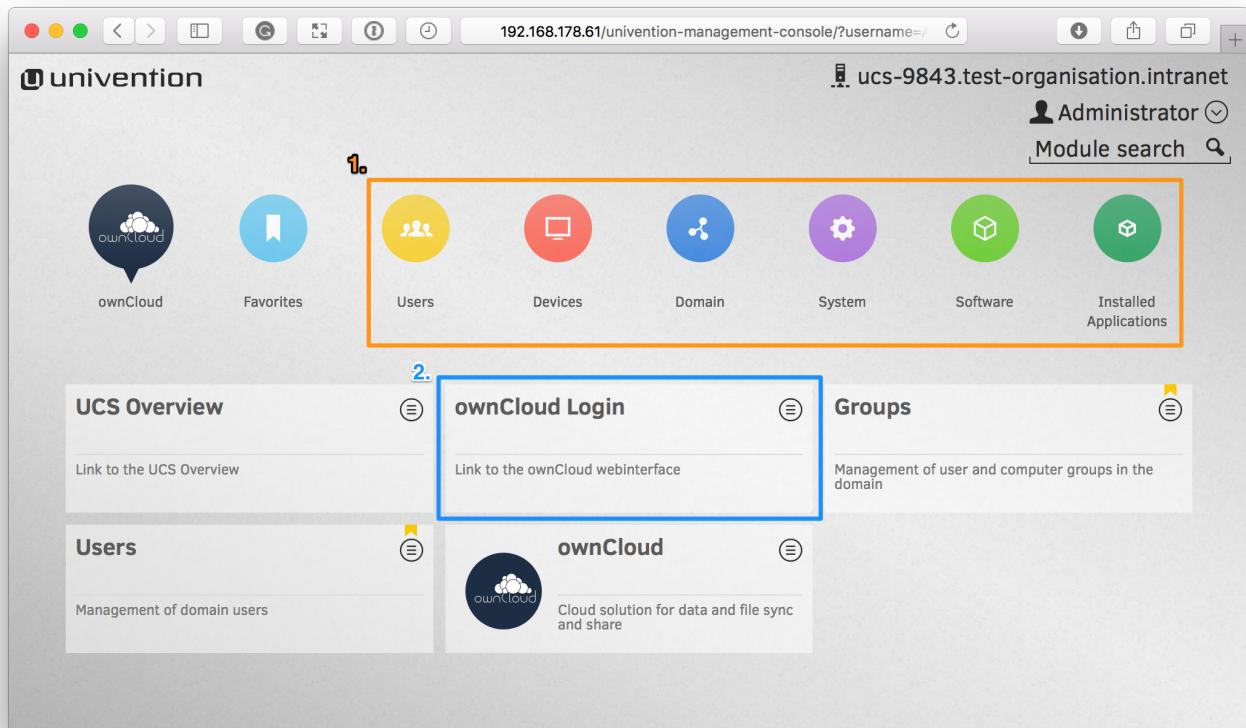
If you want to create new users and groups, or download apps from the Univention appcenter click on the “**System and domain settings**”. Login as the “**Administrator**” using the password that you supplied during the configuration wizard earlier.



Note

If you are not redirected to the appliance login page, you can open it using the following url:
`https://<ip address of the virtual machine>/univention-management-console.`

After you've done so, you will now be at the Univention management console, which you can see below.



The management console allows you to manage the virtual appliance (1), covering such areas as: *users*, *devices*, *domains*, and *software*. You will also be able to access the ownCloud web interface (2).

Note

The default username for the ownCloud is: `owncloud` and so is the password. The password is **not** the password you supplied during the configuration wizard.

Note

For security reasons `rpcbind` should be disabled in the appliance. An open, from the internet accessible portmapper service like `rpcbind` can be used by an attacker to perform DDoS-Reflection-Attacks. Furthermore, the attacker can obtain information about your system, for example running rpc-services, or existing network shares. The german IT security agency "BSI" reported that systems with an open `rpcbind` service were used to perform DDoS-Reflection-Attacks against other systems. If you want to create NFS shares on the appliance and give someone permission to access them, then you can enable `rpcbind` again.

ownCloud Appliance Login Information

Welcome to the ownCloud Appliance. Here are the login credentials.

```
username: owncloud  
password: owncloud
```

Login to the Appliance via command line or SSH with the root account.

```
username: root  
password: <Administrator password>
```

Login into the ownCloud docker container with this Univention command:

```
univention-app shell owncloud
```

ownCloud's data directory is under the following path:

```
/var/lib/univention-appcenter/apps/owncloud/data
```

How to Update ownCloud

There are two options to update an ownCloud installation hosted on an ownCloud X Appliance:

- Use the Univention Management Console
- Use the Command Line

Warning

Do not use the ownCloud built in web updater!

Use the Univention Management Console

Using the Univention Management Console, there are two paths to upgrade an existing ownCloud installation:

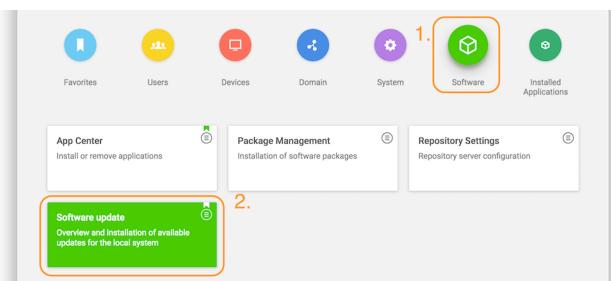
- In-place Upgrade (for 10.0 users)
- Uninstall the Existing Version and Install the New Version (for 9.1 users)

In-place Upgrade (for 10.0 users)

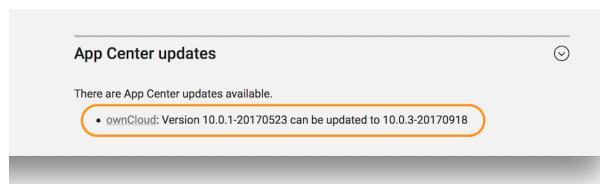
Note

Existing certificates and themes persist after an upgrade

To perform an in-place upgrade, after logging in to the Univention server, under “Administration”, click the first option labeled “System and domain settings”. This takes you to the Univention Management Console. From there, click the “Software” shortcut (1), and then click “Software update” (2).

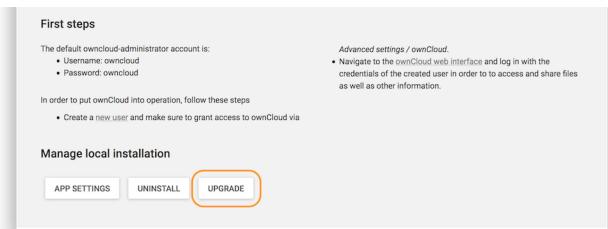


This will load the Software update management panel, after a short time scanning for available updates. If an update is available, under “App Center updates” you will see “**There are App Center updates available**”. If one is, as in the image below, click “ownCloud” which takes you to the ownCloud application.

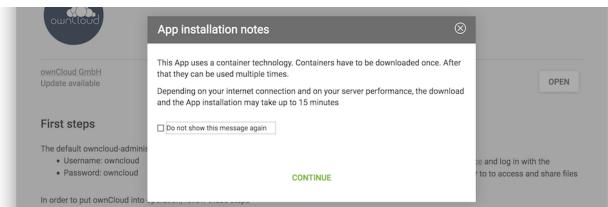


When there, part-way down the page you'll see the “Manage local installation” section. Under there, click “**UPGRADE**”.

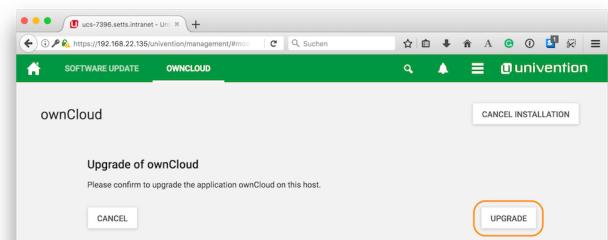
The ownCloud X Appliance



Before the upgrade starts, a prompt appears titled “**App Installation notes**”. This is nothing to be concerned about. So check the checkbox “**Do not show this message again**”. Then click “**CONTINUE**”.



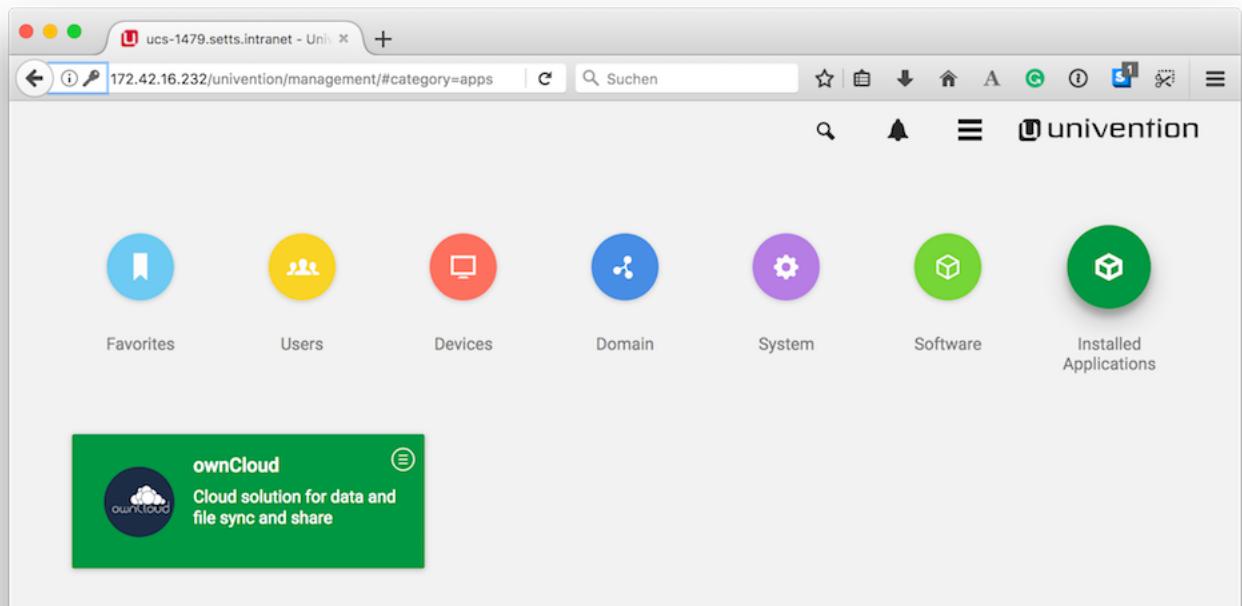
Next an upgrade confirmation page appears. To accept the confirmation, click “**UPGRADE**” on the far right-hand side of the confirmation page.



This launches the upgrade process, which requires no manual intervention. When the upgrade completes, the ownCloud app page will be visible again, but without the “**UPGRADE**” button. Now, login to ownCloud by clicking the “**OPEN**” button, on the far right-hand side of the page.

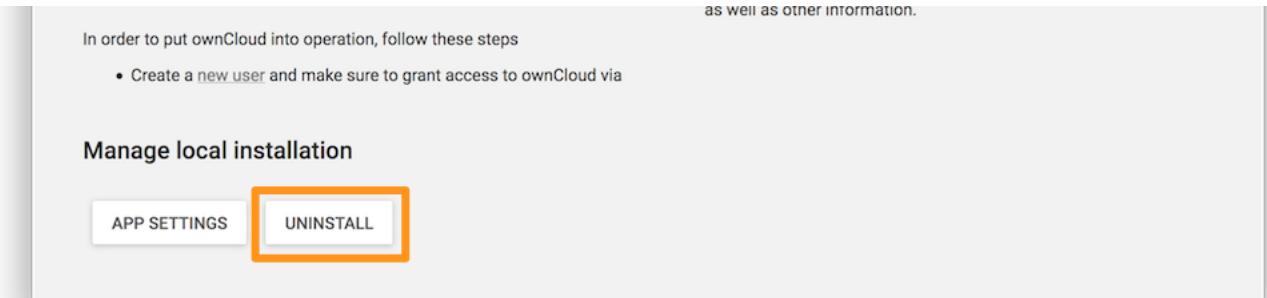
Uninstall the Existing Version and Install the New Version (for 9.1 users)

Open your ownCloud X Appliance and go to the “**System and Domain Settings**” dashboard. Then, after logging in, click “**Installed Applications**”, and then click ownCloud.

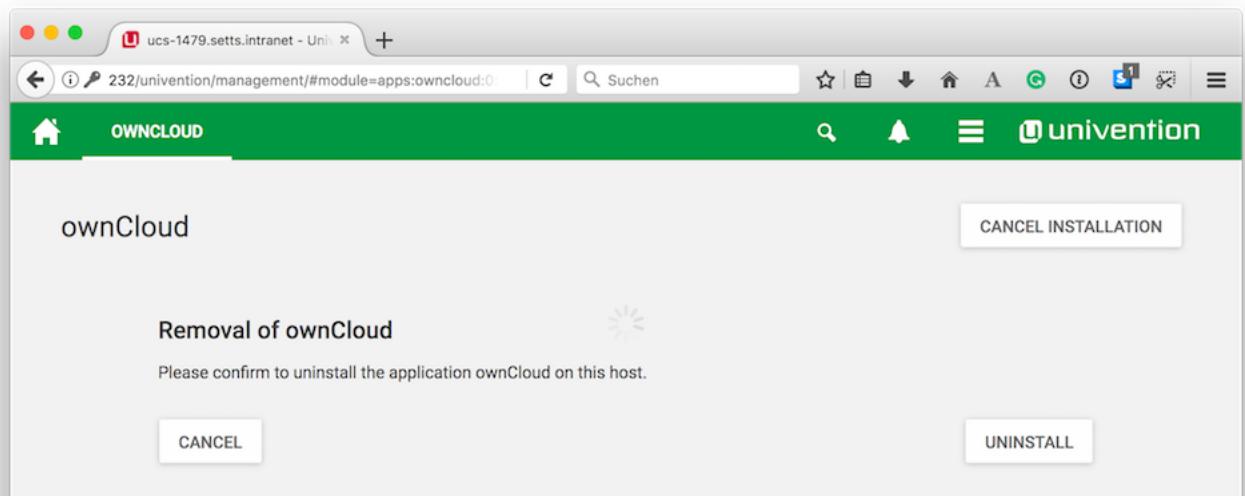


The ownCloud X Appliance

This takes you to the ownCloud app settings page. From there, begin uninstalling ownCloud by clicking “UNINSTALL” under “Manage local installations”



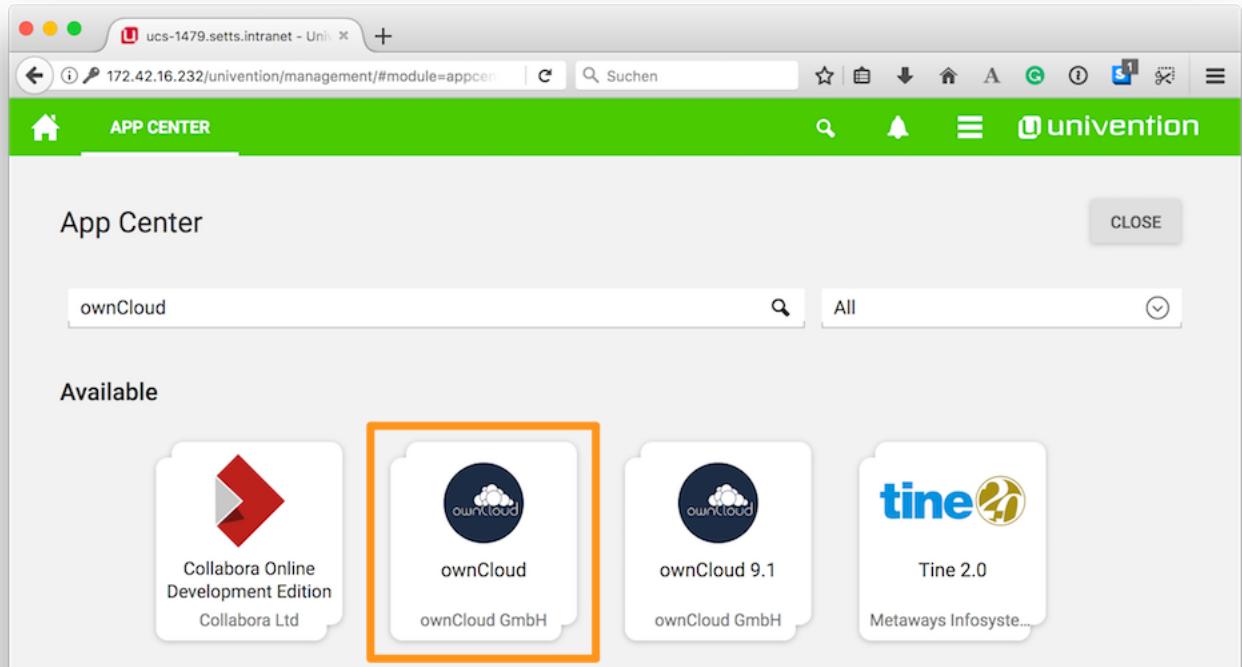
This takes you to an uninstall confirmation page. On that page, click UNINSTALL on the lower left-hand side of the page.



Follow the process until it's finished. Then, click on “Close” in the upper right corner.

Note

Your data and users will remain.

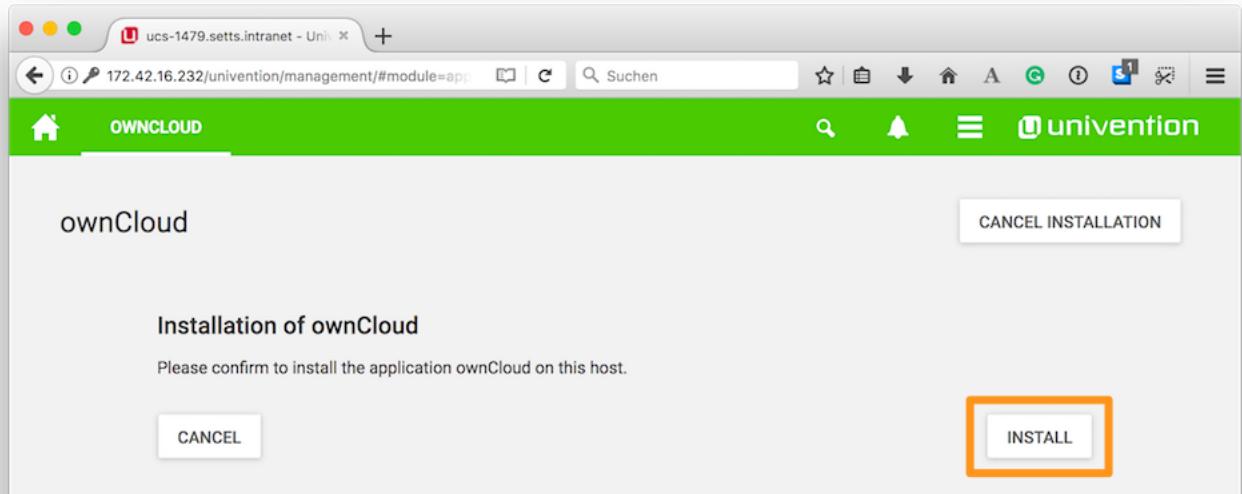


Following that, go to “**Software - Appcenter**”, and search for “*ownCloud*”. At the moment, two matching results will be returned. Pick the one that does not contain a version number.

To confirm the version number, scroll to the bottom of the page, and in the More information section, look for the version string, next to Installed version, as in the screenshot below.

A screenshot of a web page showing the "More information" section for the "ownCloud" app. The section includes fields for App provider (ownCloud GmbH), Contact (ucs@owncloud.com), More information (ownCloud), Support (Available support options), and Installed version (10.0.1-20170523). The "Installed version" field is highlighted with an orange box. To the right, there is an "App Rating" section with three stars and links for Vendor supported, Popularity award, and Editor's award. A green circular button with an upward arrow is located in the bottom right corner.

If it is the right version, click “**INSTALL**”. Then the License Agreement is displayed. If you agree to it, click “**ACCEPT LICENSE**”. This will display an installation confirmation screen. To confirm the installation, click “**INSTALL**”.



The installation will then be carried out. When it is finished, you will have the latest version of ownCloud installed.

Note

Your data and users will persist.

Use the Command Line

As with the Univention Management Console, there are two paths to upgrade an existing ownCloud installation from the command line:

- Upgrading From Version 10.0.1 to 10.0.3
- Upgrading From Versions Prior to 10.0

Upgrading From Version 10.0.1 to 10.0.3

Upgrading from the command line is also available. To do so, login to your ownCloud X Appliance, either via ssh or directly on the server. Once logged in, check if there is an upgrade available.

You can use the command `univention-app info`. This command lists information about the current state of every installed App.

```
root@ucs-9446:~# univention-app info
UCS: 4.2-1 errata165
App Center compatibility: 4
Installed: 4.1/owncloud=10.0.1-20170523
Upgradable: owncloud
```

If an upgrade is available, you then need to run the `univention-app upgrade`, as in the example below.

```
univention-app upgrade owncloud
```

You will have to enter your Administrator password to start the upgrade. This command takes some time to complete, primarily based on the appliance's network connection speed. However, it should not take more than a few minutes.

After the upgrade has completed (if it was successful) as a sanity check, run `univention-app info`, to confirm the currently installed version of ownCloud. As in the example below, you should see that the installed version is now higher than before, and that ownCloud is no longer upgradable.

```
root@ucs-9446:~# univention-app info
UCS: 4.2-1 errata165
App Center compatibility: 4
```

Installed: 4.1/owncloud=10.0.3-20170918
Upgradable:

Upgrading From Versions Prior to 10.0

If you're running a version of ownCloud prior to 10.0, the above in-place upgrade doesn't work. This is because the earlier versions of ownCloud are installed with a different application to the 10.x version. More specifically, the versions of the ownCloud app, prior to 10, have a version suffix in the name. For example the ownCloud 8.2 app is named `owncloud82`.

Given that, you first have to uninstall the existing version and then install the 10.x version. To do so, run the following commands:

```
# Assumes that owncloud82 is the currently installed version
univention-app remove owncloud82
univention-app update
univention-app install owncloud
```

And after the upgrade and updates are completed, you can then login to ownCloud and verify the upgrade.

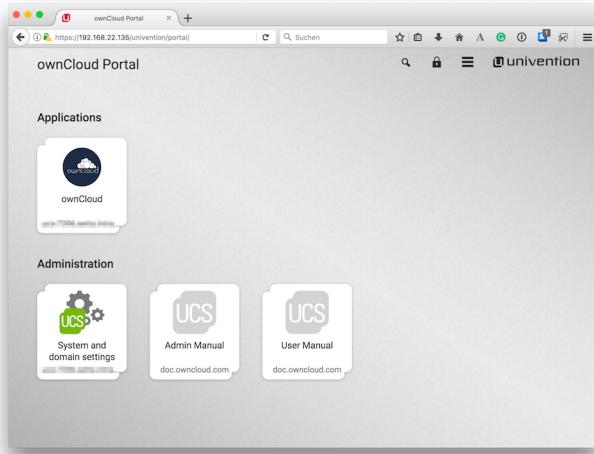
Managing UCS

Adding Users and Groups in UCS for ownCloud

If you want to add users and groups to your ownCloud installation via the UCS (Univention Corporate Server) UI, here's a concise guide showing how.

Login to the Univention Management Console

After logging in to the Univention server, under “Administration”, click the first option, labeled “**System and domain settings**”.

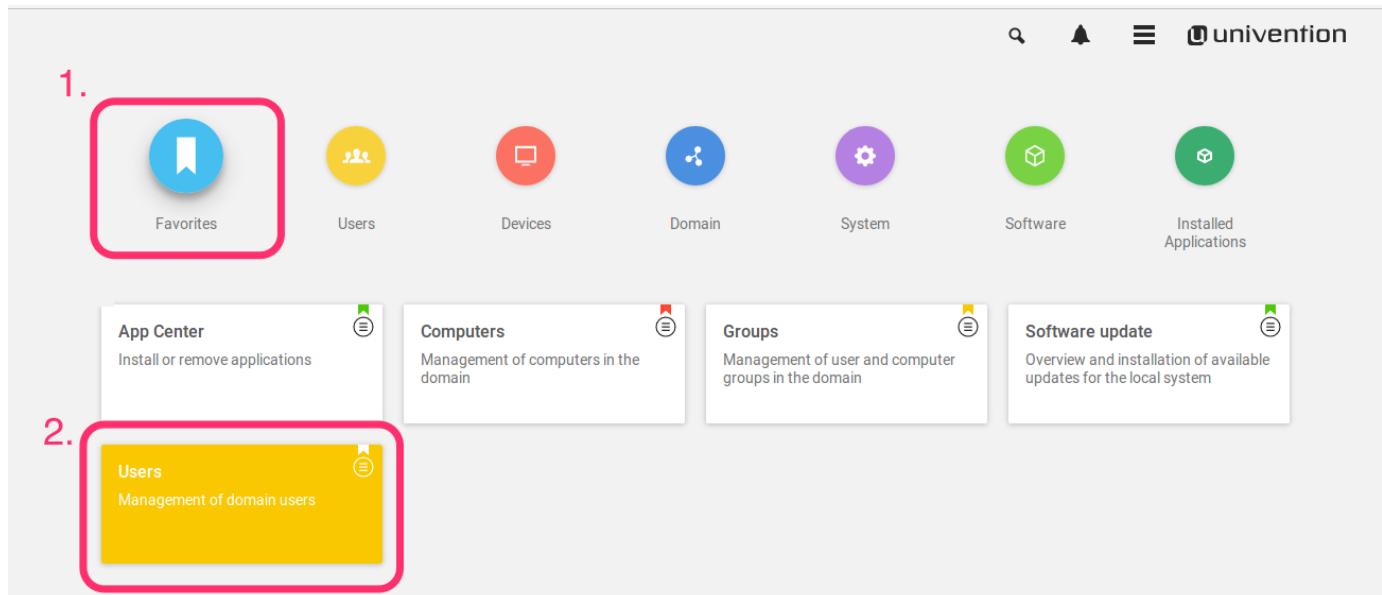


This takes you to the Univention Management Console.

Create the User

Once there, click “**Users**”.

The ownCloud X Appliance



In the screen that appears, add a new user by clicking “ADD” in the top left-hand corner of the users table.

This screenshot shows the 'Users' table dialog. At the top left is a search bar with a magnifying glass icon and a 'CLOSE' button at the top right. Below the search bar is a table header with columns for 'Name' and 'Path'. The table body contains three rows of user data, each with a checkbox and a small profile icon. To the left of the table, there is a form for adding a new user with fields for 'Name' and 'Title'. A red arrow labeled '1.' points to the 'ADD' button in this form. In the top left corner of the table area, there is another 'ADD' button. The status bar at the bottom right of the table indicates '0 users of 3 selected.'

This opens up a new user dialog, where you can supply the relevant details for the new user. Enter a username and optionally a first name, last name, and a title. Then click “NEXT”.

Add a new user.

1.

Title First name Last name *

User name *

CANCEL ADVANCED **NEXT**

In the next dialog that appears, enter and confirm the password. You can, optionally, choose some further options, if desired. Then click “**CREATE USER**”.

Add a new user.

1.

.....

Password * Password (retype) *

Change password on next login ②

Override password check

Account disabled

CANCEL ADVANCED BACK **CREATE USER**

The new user will have been created, so click the “**CLOSE**” button, in the top right-hand corner, to go back to “**Favorites**”.

The ownCloud X Appliance

The screenshot shows the 'Users' section of the ownCloud X Appliance. At the top left is a search bar with placeholder text 'Search users...'. To its right is a magnifying glass icon and a double arrow icon. On the far right is a 'CLOSE' button. Below the search bar is a table with four columns: 'Name' (with an upward arrow icon), 'Path', and two other columns which are mostly blank. The table contains four rows, each with a checkbox and a user icon. The first row is 'Administrator' with path 'intranet.owncloud:/users'. The second row is 'Carlos' with path 'intranet.owncloud:/users'. The third row is 'Dmitry' with path 'intranet.owncloud:/users'. The fourth row is 'Peter' with path 'intranet.owncloud:/users'. A red arrow labeled '1.' points to the 'Peter' row. Another red arrow labeled '2.' points to the 'CLOSE' button.

Create the Group

Now it's time to create a new group. Click “Groups”, which is located between “Computers” and “Software Update”.

The screenshot shows the main dashboard of the ownCloud X Appliance. At the top is a navigation bar with icons for search, notifications, and user profile, followed by the text 'univention'. Below the navigation bar are seven circular icons with labels: 'Favorites' (blue), 'Users' (yellow), 'Devices' (red), 'Domain' (blue), 'System' (purple), 'Software' (green), and 'Installed Applications' (dark green). In the center, there are three main management sections: 'App Center' (Install or remove applications), 'Computers' (Management of computers in the domain), 'Groups' (Management of user and computer groups in the domain, highlighted with a yellow box and a red border), and 'Software update' (Overview and installation of available updates for the local system). A red box highlights the 'Groups' section. A red arrow labeled '1.' points from the 'Groups' section to the 'ADD' button on the left side of the groups table. Another red arrow labeled '2.' points to the 'CLOSE' button in the top right corner of the 'Groups' interface.

From there, click “ADD”, located on the left-hand side of the groups table.

Groups

Search groups...

1.

<input type="checkbox"/>	Name	Path
<input type="checkbox"/>	Backup Join	intranet.owncloud:/groups
<input type="checkbox"/>	Basketball	intranet.owncloud:/groups
<input type="checkbox"/>	Computers	intranet.owncloud:/groups

0 groups of 13 selected.

CLOSE

In the next dialog that appears, first enter the name of the group and optionally a description. Then, under “**Members of this group**”, click “**ADD**”.

Groups

CUSTOMIZE THIS PAGE CREATE GROUP BACK

General
ownCloud
[Advanced settings] [Options] [Policies]

Basic settings **1.**

Group account

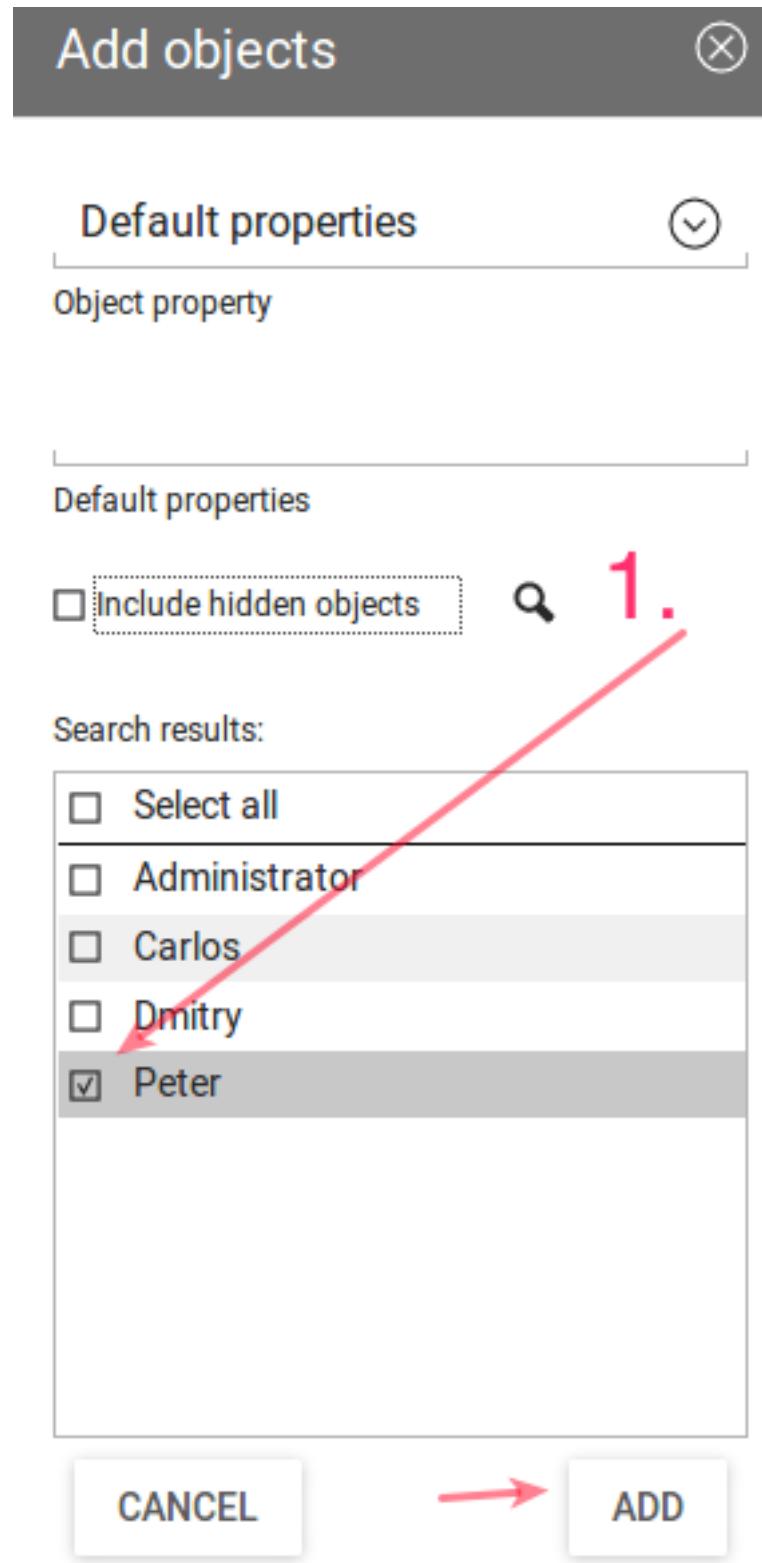
Name* Description

Members of this group

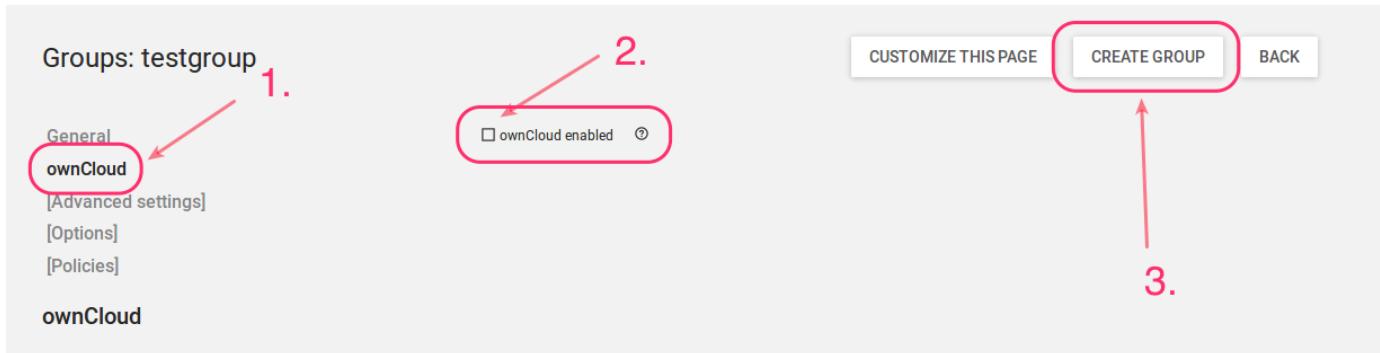
Users REMOVE

2.

This opens up an “**Add objects**” (or “Add new group”) dialog. Find the user, in the list at the bottom, that you want to add to the group, check the checkbox next to their name, and click “**ADD**”.



After that, click on “ownCloud” in the left-hand side navigation, and check the option “ownCloud enabled”. And lastly, click “CREATE GROUP”.



With that done, the new user and group are now available in your ownCloud installation.

Note

Depending on your installation, you will either see these changes immediately or you will have to wait for the user sync to be done. This happens every 10 minutes by default.

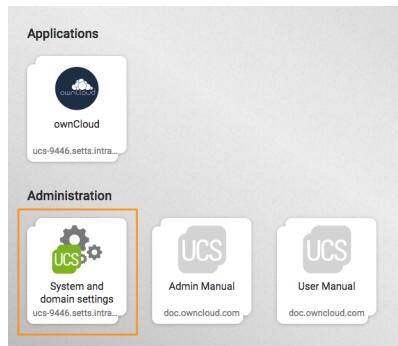
Install Antivirus Software in the ownCloud Appliance

This guide details how to enable a virus scanner in the ownCloud Appliance. It is composed of two parts:

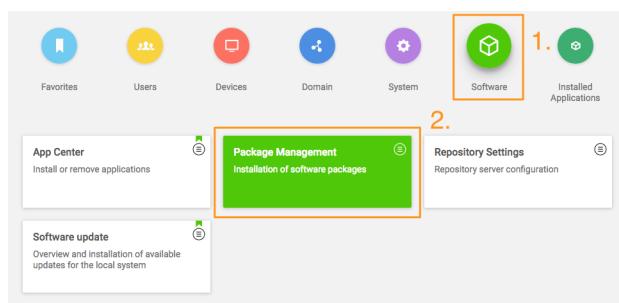
1. Install ClamAV and related components
2. Configure ownCloud to use ClamAV

Install ClamAV and Related Components

First, start the appliance and go to “System and domain settings”.

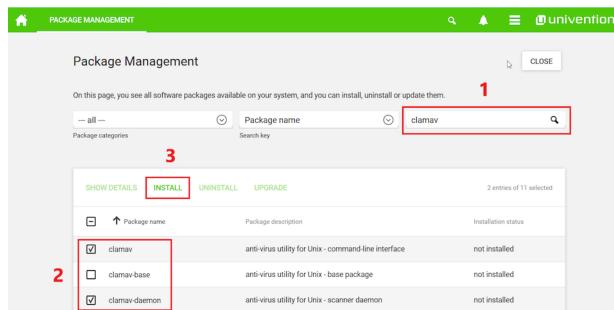


When there, log in with the administrator account. After you have done that, click “Software” and open “Package Management”, as in the screenshot below.

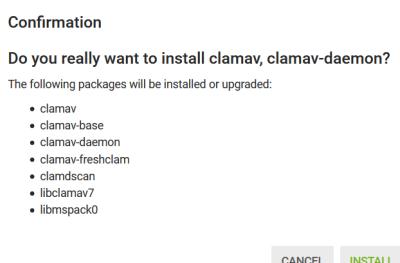


From there, you first need to install ClamAV. To do this, in the third field, next to the one containing the text “**Package name**”, type in the phrase: “**clamav**” (1). Doing so filters the list of packages to only those matching that phrase. In the filtered list of packages, check the checkboxes next to “**clamav**” (2), “**clamav-freshclam**”, and “**clamav-daemon**”.

After doing that, click “**INSTALL**” (3) above the listed packages, next to “**SHOW DETAILS**”, to install them.



After you do so, a confirmation dialog appears, as in the screenshot below, asking for confirmation to install the packages. Confirm the choice by again clicking “**INSTALL**”.



The installation should only take a few minutes.

Configure ownCloud to Use ClamAV

You next need to configure ClamAV in your ownCloud instance. Please refer to the ClamAV documentation for instructions on how to do that.

If you try to update the ClamAV virus database manually, by entering `freshclam`, and see the error below, it means that `freshclam` is already updating the database.

```
ERROR: /var/log/clamav/freshclam.log is locked by another process
ERROR: Problem with internal logger (UpdateLogFile = /var/log/clamav/freshclam.log).
```

Updates are run based on the configured time interval in the applicable Cron job. In the example below, the update would run every 47 minutes:

```
# m h dom mon dow command
47 * * * * /usr/bin/freshclam --quiet
```

If there are errors running the `freshclam` process, check if a process is blocking the log file, by running the following command:

```
lsof /var/log/clamav/freshclam.log
```

If you want to refresh the ClamAV database manually, follow these steps:

```
# Gently end the freshclam process with this command:
sudo pkill -15 -x freshclam

# Start the refresh process again with this command:
sudo freshclam
```

How to add certificates

If you want to use your own SSL certificates for the appliance, you have to follow these three steps:

1. Create the certificates and deposit them on your appliance.
2. Connect to your appliance either directly on the command line of your virtual machine or via ssh connection to your appliance.

3. Execute the following commands:

```
ucr set apache2/ssl/certificate="/etc/myssl/cert.pem"  
ucr set apache2/ssl/key="/etc/myssl/private.key"
```

Note

Remember to adjust the path and filename to match your certificate.

Once you've completed these steps, restart Apache using the following command:

```
service apache2 restart
```

Now your certificates will be used to access your appliance.

Active Directory Integration

In case you have tested the appliance with your Active Directory environment, removed the appliance and now want to include it again - you might run into some issues.

The solution is to clean up the previous DNS entries in your Domain Controller.

After that, you should be able to include the appliance again in your Active Directory environment.

Backup

If you remove the ownCloud app or update it - a backup is created automatically.

The backup remains on the host system and can be restored.

It is stored in

```
/var/lib/univention-appcenter/backups/
```

The file name is

```
appcenter-backup-owncloud:date
```

In it, you find your data and conf folders.

Your database backup is in

```
/var/lib/univention-appcenter/backups/data/backups
```

FAQ

How do I transfer files from one user to another?

See [transferring files to another user](#).

How do I deal with problems caused by using self-signed SSL certificates?

See [the security section of the OCC command](#).

I'm the admin and I lost my password! What do I do now!

See [the reset admin password documentation](#).

What is a Federated System?

A Federated System is another ownCloud or [OpenCloudMesh](#) supporting cloud service.