

Background jobs



- [Parameters](#)
 - [maintenance_window_start](#)
 - [Cron jobs](#)
 - [AJAX](#)
 - [Webcron](#)
 - [Cron](#)
 - [systemd](#)
-

A system like Nextcloud sometimes requires tasks to be done on a regular basis without the need for user interaction or hindering Nextcloud performance. For that purpose, as a system administrator, you can define background jobs (for example, database clean-ups) which are executed without any need for user interaction.

These jobs are typically referred to as *cron jobs*. Cron jobs are commands or shell-based scripts that are scheduled to run periodically at fixed times, dates, or intervals. `cron.php` is a Nextcloud internal process that runs such background jobs on demand.

Nextcloud apps register actions with `cron.php` automatically to take care of typical housekeeping operations, such as garbage collecting of temporary files or checking for newly updated files using `filescan()` for externally mounted file systems.

Parameters

`maintenance_window_start`



This setting is only taken into account in `cron` mode.

In the `config/config.php` file you can specify this config. Some background jobs only run once a day. When an hour is defined (timezone is UTC) for this config, the background jobs which advertise themselves as not time-sensitive will be delayed during the "working" hours and only run in the 4 hours after the given time. This is e.g. used for activity expiration, suspicious login training, and update checks.

A value of 1 e.g. will only run these background jobs between 01:00am UTC and 05:00am UTC:

```
'maintenance_window_start' => 1,
```

If you don't care when these jobs run, you can set the value to `100`, but beware that resource intensive jobs may then run unnecessarily during high usage periods. This may lead to slower performance and a lower quality user experience.

This setting may also be set directly via `occ` just like any other configuration parameter:

```
occ config:system:set maintenance_window_start --type=integer --value=1
```

Cron jobs

You can schedule cron jobs in three ways -- using AJAX, Webcron, or cron. The default method is to use AJAX. However, the recommended method is to use cron. The following sections describe the differences between each method.

AJAX

Use case: Single user instance

The AJAX scheduling method is the default option. Unfortunately, however, it is also the least reliable. Each time a user visits the Nextcloud page, a single background job is executed. The advantage of this mechanism 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.

“ ”

Especially when using the Activity app or external storages, where new files are added, updated or deleted, or when **multiple users** use the server, it is recommended to use `cron`.

Webcron

Use case: Very small instance (1–5 users depending on the usage)

By registering your Nextcloud `cron.php` script address at an external webcron service (for example, [easyCron](#)), you ensure that background jobs are executed regularly. To use this type of service with your server, you must be able to access your server using the Internet. For example:

```
URL to call: http[s]://<domain-of-your-server>/nextcloud/cron.php
```

“ ”

Since WebCron is still executed via the web, the webserver in most cases limits the resources on the execution. To avoid interrupts inside jobs only 1 job is executed per call. When webcron is called once every 5 minutes this limits your instance to 288 background jobs per day, which is only suitable for very small instances. For bigger instances, it is recommended to use `cron`.

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 the Web server might have.

To run a cron job on a *nix system, every 5 minutes, 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
```

And append this line:

```
*/5 * * * * php -f /var/www/nextcloud/cron.php
```

You can verify if the cron job has been added and scheduled by executing:

```
# crontab -u www-data -l
```

Which returns:

```
[snip]
```

```
*/5 * * * * php -f /var/www/nextcloud/cron.php
```

“ ”

You have to replace the path `/var/www/nextcloud/cron.php` with the path to your current Nextcloud installation.

“ ”

On some systems it might be required to call **php-cli** instead of **php**.

“ ”

Please refer to the crontab man page for the exact command syntax.

systemd

If systemd is installed on the system, a systemd timer could be an alternative to a cronjob.

This approach requires two files: **nextcloudcron.service** and **nextcloudcron.timer**. Create these two files in `/etc/systemd/system/`.

nextcloudcron.service should look like this:

```
[Unit]
Description=Nextcloud cron.php job

[Service]
User=www-data
ExecCondition=php -f /var/www/nextcloud/occ status -e
ExecStart=/usr/bin/php -f /var/www/nextcloud/cron.php
KillMode=process
```

Replace the user `www-data` with the user of your http server and `/var/www/nextcloud/cron.php` with the location of **cron.php** in your nextcloud directory.

The ExecCondition checks that the nextcloud instance is operating normally before running the background job, and skips it if otherwise.

The `KillMode=process` setting is necessary for external programs that are started by the cron job to keep running after the cron job has finished.

Note that the **.service** unit file does not need an `[[Install]]` section. Please check your setup because we recommended it in earlier versions of this admin manual.

nextcloudcron.timer should look like this:

```
[Unit]
Description=Run Nextcloud cron.php every 5 minutes

[Timer]
OnBootSec=5min
OnUnitActiveSec=5min
Unit=nextcloudcron.service

[Install]
WantedBy=timers.target
```

The important parts in the timer-unit are `OnBootSec` and `OnUnitActiveSec`. `OnBootSec` will start the timer 5 minutes after boot, otherwise, you would have to start it manually after every boot. `OnUnitActiveSec` will set a 5-minute timer after the service-unit was last activated.

Now all that is left is to start and enable the timer by running this command:

```
systemctl enable --now nextcloudcron.timer
```

When the option `--now` is used with `enable`, the respective unit will also be started.



Selecting the option `Cron` in the admin menu for background jobs is not mandatory, because once cron.php is executed from the command line or cron service it will set it automatically to `Cron`.

#1

cocobook-sync 2026-10-28 16:26:56 UTC
cocobook-sync 2026-10-28 16:26:56 UTC