systemd/Timers - ArchWiki

Archived: 2026-04-05 17:26:02 UTC

Timers are systemd unit files whose name ends in *.timer* that control *.service* files or events. Timers can be used as an alternative to cron (read #As a cron replacement). Timers have built-in support for calendar time events, monotonic time events, and can be run asynchronously.

Timer units

Timers are *systemd* unit files with a suffix of *.timer*. Timers are like other unit configuration files and are loaded from the same paths but include a `[Timer]` section which defines when and how the timer activates. Timers are defined as one of two types:

- **Realtime timers** (a.k.a. wallclock timers) activate on a calendar event, the same way that cronjobs do. The option `OnCalendar=` is used to define them.
- **Monotonic timers** activate after a time span relative to a varying starting point. They stop if the computer is temporarily suspended or shut down. There are number of different monotonic timers but all have the form: `On`*`Type`*`Sec=`. Common monotonic timers include `OnBootSec` and `OnUnitActiveSec`.

For a full explanation of timer options, see the systemd.timer(5). The argument syntax for calendar events and time spans is defined in systemd.time(7).

Note *systemd* offers the target `timers.target` which sets up all timers that should be active after boot (see systemd.special(7) for details). To use it, add `WantedBy=timers.target` to the `[Install]` section of your timer and enable the timer unit.

Service units

For each *.timer* file, a matching *.service* file exists (e.g. `foo.timer` and `foo.service`). The *.timer* file activates and controls the *.service* file. The *.service* does not require an `[Install]` section as it is the *timer* units that are enabled. If necessary, it is possible to control a differently-named unit using the `Unit=` option in the timer's `[Timer]` section.

Management

To use a *timer* unit enable and start it like any other unit (remember to add the *.timer* suffix). To view all started timers, run:

```
$ systemctl list-timers
```

```
NEXT                         LEFT        LAST                         PASSED    UNIT
Thu 2014-07-10 19:37:03 CEST  11h left    Wed 2014-07-09 19:37:03 CEST  12h ago    systemd-tmpfiles-
```

```
Fri 2014-07-11 00:00:00 CEST  15h left    Thu 2014-07-10 00:00:13 CEST  8h ago     logrotate.timer
```

Note

- To list all timers (including inactive), use `systemctl list-timers --all`.
- The status of a service started by a timer will likely be inactive unless it is currently being triggered.
- If a timer gets out of sync, it may help to delete its `stamp-*` file in `/var/lib/systemd/timers` (or `~/.local/share/systemd/` in case of user timers). These are zero length files which mark the last time each timer was run. If deleted, they will be reconstructed on the next start of their timer.

Examples

A service unit file can be scheduled with a timer out-of-the-box. The following examples schedule `foo.service` to be run with a corresponding timer called `foo.timer`.

Monotonic timer

A timer which will start 15 minutes after boot and again every week while the system is running.

```
/etc/systemd/system/foo.timer
```

```
[Unit]
Description=Run foo weekly and on boot

[Timer]
OnBootSec=15min
OnUnitActiveSec=1w

[Install]
WantedBy=timers.target
```

Realtime timer

A timer which starts once a week (at 12:00am on Monday). When activated, it triggers the service immediately if it missed the last start time (option `Persistent=true`), for example due to the system being powered off:

```
/etc/systemd/system/foo.timer
```

```
[Unit]
Description=Run foo weekly

[Timer]
OnCalendar=weekly
Persistent=true
```

```
[Install]
WantedBy=timers.target
```

When more specific dates and times are required, `OnCalendar` events uses the following format:

```
DayOfWeek Year-Month-Day Hour:Minute:Second
```

An asterisk may be used to specify any value and commas may be used to list possible values. Two values separated by `..` indicate a contiguous range.

In the below example the service is run the first four days of each month at 12:00 PM, but *only* if that day is a Monday or a Tuesday.

```
OnCalendar=Mon,Tue *-*-01..04 12:00:00
```

To run a service on the first Saturday of every month, use:

```
OnCalendar=Sat *-*-1..7 18:00:00
```

When using the `DayOfWeek` part, at least one weekday has to be specified. If you want something to run every day at 4am, use:

```
OnCalendar=*-*-* 4:00:00
```

To run a service at different times, `OnCalendar` may be specified more than once. In the example below, the service runs at 22:30 on weekdays and at 20:00 on weekends.

```
OnCalendar=Mon..Fri 22:30
OnCalendar=Sat,Sun 20:00
```

You can also specify a timezone at the end of the directive (use `timedatectl list-timezones` to list accepted values)

```
OnCalendar=*-*-* 02:00:00 Europe/Paris
```

More information is available in systemd.time(7).

Tip

- `OnCalendar` time specifications can be tested in order to verify their validity and to calculate the next time the condition would elapse when used on a timer unit file with the `calendar` option of the *systemd-*

analyze utility. For example, one can use `systemd-analyze calendar weekly` or `systemd-analyze calendar "Mon,Tue *-*-01..04 12:00:00"`. Add `--iterations=N` to ask for more iterations to be printed.
- The `faketime` command is especially useful to test various scenarios with the above command; it comes with the libfaketime package.
- Special event expressions like `daily` and `weekly` refer to *specific start times* and thus any timers sharing such calendar events will start simultaneously. Timers sharing start events can cause poor system performance if the timers' services compete for system resources. The `RandomizedDelaySec` option in the `[Timer]` section avoids this problem by randomly staggering the start time of each timer. See systemd.timer(5).
- Add the option `AccuracySec=1us` to the `[Timer]` section, to avoid the inaccuracy of the *1m* default value of `AccuracySec`. Also see systemd.timer(5).
- Some options (`WakeSystem`) may require specific system capabilities and prevent a timer from starting, resulting in the following error messages: "Failed to enter waiting state: Operation not supported" and "Failed with result 'resources'.".

Transient timer units

One can use `systemd-run` to create transient *.timer* units. That is, one can set a command to run at a specified time without having a service file. For example the following command touches a file after 30 seconds:

```
# systemd-run --on-active=30 /bin/touch /tmp/foo
```

One can also specify a pre-existing service file that does not have a timer file. For example, the following starts the systemd unit named `someunit.service` after 12.5 hours have elapsed:

```
# systemd-run --on-active="12h 30m" --unit someunit.service
```

See systemd-run(1) for more information and examples.

As a cron replacement

Although cron is arguably the most well-known job scheduler, *systemd* timers can be an alternative.

Benefits

The main benefits of using timers come from each job having its own *systemd* service. Some of these benefits are:

- Jobs can be easily started independently of their timers. This simplifies debugging.
- Each job can be configured to run in a specific environment (see systemd.exec(5)).
- Jobs can be attached to cgroups.
- Jobs can be set up to depend on other *systemd* units.
- Jobs are logged in the *systemd* journal for easy debugging.

Caveats

Some things that are easy to do with cron are difficult to do with timer units alone:

- Creation: to set up a timed job with *systemd* you need to create two files and run `systemctl` commands, compared to adding a single line to a crontab.
- Emails: there is no built-in equivalent to cron's `MAILTO` for sending emails on job failure. See systemd#Notifying with e-mail for an example of setting up a similar functionality using `OnFailure=`.

Also note that user timer units will only run during an active user login session by default. However, lingering can enable services to run at boot even when the user has no active login session.

Using a crontab

Several of the caveats can be worked around by installing a package that parses a traditional crontab to configure the timers. systemd-cron-next-git[AUR] and systemd-cron[AUR] are two such packages. These can provide the missing `MAILTO` feature.

Also, like with crontabs, a unified view of all scheduled jobs can be obtained with `systemctl`. See #Management.

Manually

Outside of migrating from an existing crontab, using the same periodicity as cron can be desired. To avoid the tedious task of creating a timer for each service to start periodically, use a template unit, for example:

```
/etc/systemd/system/monthly@.timer
```

```
[Unit]
Description=Monthly Timer for %i service

[Timer]
OnCalendar=*-*-1 02:00:00
AccuracySec=6h
RandomizedDelaySec=1h
Persistent=true
Unit=%i.service

[Install]
WantedBy=default.target
```

Note See systemd.timer(5) § OPTIONS for the importance of using `RandomizedDelaySec` and not only `AccuracySec` to avoid all units started by the timer firing at once.

Then one only needs to enable/start `monthly@unit_name.timer`.

Tip The template units can be nested, e.g. one could enable/start `monthly@btrfs-scrub@mnt-$(systemd-escape bbb76c63-e4ac-4e39-8897-a120c5d30686).timer`.

Tips and tricks

Handling "time to live"

Some software will track the time elapsed since they last ran, for example blocking the update of a database if the last download ended less than 24 hours ago.

By default, timers do not track when the task they launched has ended. To work around this, we can use `OnUnitInactiveSeconds`:

```
/etc/systemd/system/daily-inactive@.timer
```

```
[Unit]
Description=Launch %i service 24hours after it deactivated

[Timer]
OnUnitInactiveSec=1day1sec
Unit=%i.service
Persistent=true

[Install]
WantedBy=default.target
```

Tip With `Restart=on-failure` along with `RestartSec`, it is possible to have a unit rerun after failure and success according to different schedules, see systemd.service(5) § OPTIONS.

Desktop notifications

The systemd-timer-notify[AUR] provides an automatic desktop notification that helps you notice when a systemd service is triggered by a timer and is running. The notification will automatically close when the service finishes.

This can be helpful for understanding why CPU usage is high or for preventing a shutdown when a backup service hasn't finished.

For more details and configuration options, visit https://gitlab.com/Zesko/systemd-timer-notify

See also

- systemd.timer(5)
- Fedora:Features/SystemdCalendarTimers
- Gentoo:Systemd#Timer services

- **systemd-cron** — provides systemd units to run cron scripts; using *systemd-crontab-generator* to convert crontabs

  https://github.com/systemd-cron/systemd-cron || systemd-cron[AUR]

- **systemd-cron-next** — tool to generate timers/services from crontab and anacrontab files

  https://github.com/systemd-cron/systemd-cron-next || systemd-cron-next-git[AUR]

- Systemd Timers for Scheduling Tasks
