~ systemd.timer, an alternative to cron

Posted on Thu 8 2022 to Programming

There will come a point in time during your time administering a Linux server where you will want to perform a job on a schedule. Perhaps you want to rotate some TLS certificates before they expire, or delete old files that are no longer needed. Typically for this, you would use cron, perhaps the most widely used job scheduler for UNIX like systems. You would fire up the crontab for the user, punch in the schedule followed by the command, then write and quit. If you wanted to monitor the job, then you would add `MAILTO` at the top to receive the cron logs should the job fail.

systemd offers an alternative to cron via systemd.timer, one that I prefer over cron for reasons I will get into later. With systemd.timer, you specify a `*.timer` file and a corresponding `*.service`, what with the latter being the job you want to perform. For example, using the example of certificate rotation, we might have a `certrotate.service` file that looks like this,

```
[Unit]
Description=Rotate TLS certificates

[Service]
Type=oneshot
ExecStart=/usr/local/bin/uacme -d /etc/uacme.d -h /usr/local/
```

and the corresponding `certrotate.timer` file,

```
[Unit]
Description=Weekly rotation of TLS certificates

[Timer]
OnCalendar=weekly
# Set to true so we can store when the timer last triggered
# on disk.
Persistent=true
```

with both of these in place we can then start the timer.

```
$ sudo systemctl start certrotate.timer
```

The above is certainly more verbose than cron, what with the main difference being that two files are required, one for the job itself, and another for the timer. The implementation of timers in systemd offers up the following benefits over that of cron:

- It allows for independent job execution via `systemctl start`
- Jobs can be configured to have dependencies
- Job output will automatically be written to systemd-journald
- Templated unit files

On the first point, you can make the argument that this is possible via cron. Since cron does just execute the arbitrary commands, so you could run the same commands in the crontab via the terminal. This is true, however systemd offers up some convenience in this regard. With the above example for certificate rotation I can fire of the job like so,

```
$ sudo systemctl start certrotate
```

without having to memorise the full sequence of flags and arguments to pass. This makes debugging jobs easier, and should one fail you can check the status via `systemctl status`, or the entire log with `journalctl -u`.

With the second point, since jobs are just regular `*.service` files, they can be configured to have dependencies via `Wants` and `Requires`, more details can be found under systemd.unit. This can allow for more sophisticated orchestration between jobs on a system, should any of them depend on another job being completed first.

Regarding the final point, systemd offers the ability to have templated unit files via specifiers. This reduces the overhead of managing jobs that have repetitive logic. With the above example we currently only rotate the certificate for the domain `example.com`, what if we also want to rotate the certificates for any subdomains too? First we would rename both the `*.timer` and `*.service` files so a `@` precedes the suffix, so `certrotate@.timer`, and `certrotate@.server`. Then, in `certrotate@.service` we would modify `ExecStart` to use `%i` for the instance name of the service,

```
ExecStart=/usr/local/bin/uacme -d /etc/uacme.d -h /usr/local/
```

then we can perform the following to have the changes applied,

Feeds

Atom

RSS

Links

About

GitHub

Categories

Programming

Ramblings

```
$ sudo systemctl daemon-reload
$ sudo systemctl start certrotate@example.com.timer
```

and this would be the same for each subsequent domain too.

```
$ sudo systemctl start certrotate@api.example.com.timer
$ sudo systemctl start certrotate@about.example.com.timer
```

And my favourite thing about systemd.timer is how it provides and easy overview of the jobs running on your server, and when they will next execute,

```
$ systemctl list-timers
NEXT                         LEFT          LAST
Fri 2022-12-09 00:00:00 UTC  4h 57min left Thu 2022-12-08 00
Fri 2022-12-09 00:00:00 UTC  4h 57min left Thu 2022-12-08 00
Fri 2022-12-09 06:06:34 UTC  11h left      Thu 2022-12-08 06
Fri 2022-12-09 08:19:01 UTC  13h left      Thu 2022-12-08 18
Fri 2022-12-09 09:55:30 UTC  14h left      Thu 2022-12-08 09
```

Now, one thing that systemd.timer lacks is the `MAILTO` functionality offered by cron. But this can be easily reimplemented by implementing another oneshot service that is invoked via `OnFailure`. For further information on this, see the MAILTO section from the Arch wiki on how to achieve this.

If you are someone who administers a Linux server, one that uses systemd specifically, I encourage you to invest some time in using systemd timers. Sure, they're more verbose than cron, but I've found the tradeoff in that regard to be one worth making. They're easy to manage, since they are just standalone services at the end of the day, make deduplication of jobs a cinch with unit specifiers, and offer easy monitoring via `systemctl list-timers` as shown above.