Pid Eins

レナート
لينارت

POSTED ON FR 19 NOVEMBER 2010

systemd for Administrators, Part IV

Here's the fourth installment of my ongoing series about systemd for

administrators.

Killing Services

Killing a system daemon is easy, right? Or is it?

Sure, as long as your daemon persists only of a single process this might actually be somewhat true. You type `killall rsyslogd` and the syslog daemon is gone. However it is a bit dirty to do it like that given that this will kill all processes which happen to be called like this, including those an unlucky user might have named that way by accident. A slightly more correct version would be to read the `.pid` file, i.e. `kill `cat /var/run/syslogd.pid``. That already gets us much further, but still, is this really what we want?

More often than not it actually isn't. Consider a service like Apache, or crond, or atd, which as part of their usual operation spawn child processes. Arbitrary, user configurable child processes, such as cron or at jobs, or CGI scripts, even full application servers. If you kill the main apache/crond/atd process this might or might not pull down the child processes too, and it's up to those processes whether they want to stay around or go down as well. Basically that means that terminating Apache might very well cause its CGI scripts to stay around, reassigned to be children of init, and difficult to track down.

systemd to the rescue: With `systemctl kill` you can easily send a signal to all processes of a service. Example:

```
# systemctl kill crond.service
```

This will ensure that SIGTERM is delivered to all processes of the crond service,

not just the main process. Of course, you can also send a different signal if you wish. For example, if you are bad-ass you might want to go for SIGKILL right-away:

```
# systemctl kill -s SIGKILL crond.service
```

And there you go, the service will be brutally slaughtered in its entirety, regardless how many times it forked, whether it tried to escape supervision by double forking or fork bombing.

Sometimes all you need is to send a specific signal to the main process of a service, maybe because you want to trigger a reload via SIGHUP. Instead of going via the PID file, here's an easier way to do this:

```
# systemctl kill -s HUP --kill-who=main crond.service
```

So again, what is so new and fancy about killing services in systemd? Well, for the first time on Linux we can actually properly do that. Previous solutions were always depending on the daemons to actually cooperate to bring down everything they spawned if they themselves terminate. However, usually if you want to use SIGTERM or SIGKILL you are doing that because they actually do not cooperate properly with you.

How does this relate to `systemctl stop`? `kill` goes directly and sends a signal to every process in the group, however `stop` goes through the official configured way to shut down a service, i.e. invokes the stop command configured with `ExecStop=` in the service file. Usually `stop` should be sufficient. `kill` is the tougher version, for cases where you either don't want the official shutdown command of a service to run, or when the service is hosed and hung in other ways.

(It's up to you BTW to specify signal names with or without the SIG prefix on the -s switch. Both works.)

It's a bit surprising that we have come so far on Linux without even being able to properly kill services. systemd for the first time enables you to do this properly.

Category: projects

← BACK TO INDEX