Pid Eins

レナート
لينارت

POSTED ON FR 01 OKTOBER 2010

systemd for Administrators, Part III

Here's the third installment of my ongoing series about systemd for administrators.

How Do I Convert A SysV Init Script Into A systemd Service File?

Traditionally, Unix and Linux services (*daemons*) are started via SysV init scripts. These are Bourne Shell scripts, usually residing in a directory such as `/etc/rc.d/init.d/` which when called with one of a few standardized arguments (verbs) such as `start`, `stop` or `restart` controls, i.e. starts, stops or restarts the service in question. For starts this usually involves invoking the daemon binary, which then forks a background process (more precisely *daemonizes*). Shell scripts tend to be slow, needlessly hard to read, very verbose and fragile. Although they are immensly flexible (after all, they are just code) some things are very hard to do properly with shell scripts, such as ordering parallized execution, correctly supervising processes or just configuring execution contexts in all detail. systemd provides compatibility with these shell scripts, but due to the shortcomings pointed out it is recommended to install native systemd service files for all daemons installed. Also, in contrast to SysV init scripts which have to be adjusted to the distribution systemd service files are compatible with any kind of distribution running systemd (which become more and more these days...). What follows is a terse guide how to take a SysV init script and translate it into a native systemd service file. Ideally, upstream projects should ship and install systemd service files in their tarballs. If you have successfully converted a SysV script according to the guidelines it might hence be a good idea to submit the file as patch to upstream. How to prepare a patch like that will be discussed in a later installment, suffice to say at this point that the daemon(7) manual page shipping with systemd contains a lot of useful information regarding this.

So, let's jump right in. As an example we'll convert the init script of the ABRT daemon into a systemd service file. ABRT is a standard component of every Fedora install, and is an acronym for Automatic Bug Reporting Tool, which pretty much

describes what it does, i.e. it is a service for collecting crash dumps. Its SysV script I have uploaded here.

The first step when converting such a script is to read it (surprise surprise!) and distill the useful information from the usually pretty long script. In almost all cases the script consists of mostly boilerplate code that is identical or at least very similar in all init scripts, and usually copied and pasted from one to the other. So, let's extract the interesting information from the script linked above:

- A description string for the service is "*Daemon to detect crashing apps*". As it turns out, the header comments include a redundant number of description strings, some of them describing less the actual service but the init script to start it. systemd services include a description too, and it should describe the service and not the service file.
- The LSB header[1] contains dependency information. systemd due to its design around socket-based activation usually needs no (or very little) manually configured dependencies. (For details regarding socket activation see the original announcement blog post.) In this case the dependency on `$syslog` (which encodes that abrtd requires a syslog daemon), is the only valuable information. While the header lists another dependency (`$local_fs`) this one is redundant with systemd as normal system services are always started with all local file systems available.
- The LSB header suggests that this service should be started in runlevels 3 (multi-user) and 5 (graphical).
- The daemon binary is `/usr/sbin/abrtd`

And that's already it. The entire remaining content of this 115-line shell script is simply boilerplate or otherwise redundant code: code that deals with

synchronizing and serializing startup (i.e. the code regarding lock files) or that outputs status messages (i.e. the code calling echo), or simply parsing of the verbs (i.e. the big case block).

From the information extracted above we can now write our systemd service file:

```
[Unit]
Description=Daemon to detect crashing apps
After=syslog.target

[Service]
ExecStart=/usr/sbin/abrtd
Type=forking

[Install]
WantedBy=multi-user.target
```

A little explanation of the contents of this file: The `[Unit]` section contains generic information about the service. systemd not only manages system services, but also devices, mount points, timer, and other components of the system. The generic term for all these objects in systemd is a *unit*, and the `[Unit]` section encodes information about it that might be applicable not only to services but also in to the other unit types systemd maintains. In this case we set the following unit settings: we set the description string and configure that the daemon shall be started after Syslog[2], similar to what is encoded in the LSB header of the original init script. For this Syslog dependency we create a dependency of type `After=` on a systemd unit `syslog.target`. The latter is a special target unit in systemd and is the standardized name to pull in a syslog implementation. For more information about these standardized names see the systemd.special(7). Note that a dependency of type `After=` only encodes the suggested ordering, but does not actually cause syslog to be started when abrtd is -- and this is exactly what we

want, since abrtd actually works fine even without syslog being around. However, if both are started (and usually they are) then the order in which they are is controlled with this dependency.

The next section is `[Service]` which encodes information about the service itself. It contains all those settings that apply only to services, and not the other kinds of units systemd maintains (mount points, devices, timers, ...). Two settings are used here: `ExecStart=` takes the path to the binary to execute when the service shall be started up. And with `Type=` we configure how the service notifies the init system that it finished starting up. Since traditional Unix daemons do this by returning to the parent process after having forked off and initialized the background daemon we set the type to `forking` here. That tells systemd to wait until the start-up binary returns and then consider the processes still running afterwards the daemon processes.

The final section is `[Install]`. It encodes information about how the suggested installation should look like, i.e. under which circumstances and by which triggers the service shall be started. In this case we simply say that this service shall be started when the `multi-user.target` unit is activated. This is a special unit (see above) that basically takes the role of the classic SysV Runlevel 3[3]. The setting `WantedBy=` has little effect on the daemon during runtime. It is only read by the `systemctl enable` command, which is the recommended way to enable a service in systemd. This command will simply ensure that our little service gets automatically activated as soon as `multi-user.target` is requested, which it is on all normal boots[4].

And that's it. Now we already have a minimal working systemd service file. To test it we copy it to `/etc/systemd/system/abrtd.service` and invoke `systemctl`

`daemon-reload`. This will make systemd take notice of it, and now we can start the service with it: `systemctl start abrtd.service`. We can verify the status via `systemctl status abrtd.service`. And we can stop it again via `systemctl stop abrtd.service`. Finally, we can enable it, so that it is activated by default on future boots with `systemctl enable abrtd.service`.

The service file above, while sufficient and basically a 1:1 translation (feature- and otherwise) of the SysV init script still has room for improvement. Here it is a little bit updated:

```
[Unit]
Description=ABRT Automated Bug Reporting Tool
After=syslog.target

[Service]
Type=dbus
BusName=com.redhat.abrt
ExecStart=/usr/sbin/abrtd -d -s

[Install]
WantedBy=multi-user.target
```

So, what did we change? Two things: we improved the description string a bit. More importantly however, we changed the type of the service to `dbus` and configured the D-Bus bus name of the service. Why did we do this? As mentioned classic SysV services *daemonize* after startup, which usually involves double forking and detaching from any terminal. While this is useful and necessary when daemons are invoked via a script, this is unnecessary (and slow) as well as counterproductive when a proper process babysitter such as systemd is used. The reason for that is that the forked off daemon process usually has little relation to the original process started by systemd (after all the daemonizing scheme's whole idea is to remove this relation), and hence it is difficult for systemd to figure out

after the fork is finished which process belonging to the service is actually the main process and which processes might just be auxiliary. But that information is crucial to implement advanced babysitting, i.e. supervising the process, automatic respawning on abnormal termination, collectig crash and exit code information and suchlike. In order to make it easier for systemd to figure out the main process of the daemon we changed the service type to `dbus`. The semantics of this service type are appropriate for all services that take a name on the D-Bus system bus as last step of their initialization[5]. ABRT is one of those. With this setting systemd will spawn the ABRT process, which will no longer fork (this is configured via the `-d -s` switches to the daemon), and systemd will consider the service fully started up as soon as `com.redhat.abrt` appears on the bus. This way the process spawned by systemd is the main process of the daemon, systemd has a reliable way to figure out when the daemon is fully started up and systemd can easily supervise it.

And that's all there is to it. We have a simple systemd service file now that encodes in 10 lines more information than the original SysV init script encoded in 115. And even now there's a lot of room left for further improvement utilizing more features systemd offers. For example, we could set `Restart=restart-always` to tell systemd to automatically restart this service when it dies. Or, we could use `OOMScoreAdjust=-500` to ask the kernel to please leave this process around when the OOM killer wreaks havoc. Or, we could use `CPUSchedulingPolicy=idle` to ensure that abrtd processes crash dumps in background only, always allowing the kernel to give preference to whatever else might be running and needing CPU time.

For more information about the configuration options mentioned here, see the respective man pages systemd.unit(5), systemd.service(5), systemd.exec(5). Or,

browse all of systemd's man pages.

Of course, not all SysV scripts are as easy to convert as this one. But gladly, as it turns out the vast majority actually are.

That's it for today, come back soon for the next installment in our series.

Footnotes

[1] The LSB header of init scripts is a convention of including meta data about the service in comment blocks at the top of SysV init scripts and is defined by the Linux Standard Base. This was intended to standardize init scripts between distributions. While most distributions have adopted this scheme, the handling of the headers varies greatly between the distributions, and in fact still makes it necessary to adjust init scripts for every distribution. As such the LSB spec never kept the promise it made.

[2] Strictly speaking, this dependency does not even have to be encoded here, as it is redundant in a system where the Syslog daemon is socket activatable. Modern syslog systems (for example rsyslog v5) have been patched upstream to be socket-activatable. If such a init system is used configuration of the `After=syslog.target` dependency is redundant and implicit. However, to maintain compatibility with syslog services that have not been updated we include this dependency here.

[3] At least how it used to be defined on Fedora.

[4] Note that in systemd the graphical bootup (`graphical.target`, taking the role of SysV runlevel 5) is an implicit superset of the console-only bootup (`multi-user.target`, i.e. like runlevel 3). That means hooking a service into the latter will also hook it into the former.

[5] Actually the majority of services of the default Fedora install now take a name on the bus after startup.

Category: projects

← BACK TO INDEX