Pid Eins

レナート
لينارت

POSTED ON MO 18 JULI 2011

systemd for Administrators, Part IX

Here's the ninth installment of my ongoing series on systemd for Administrators:

On /etc/sysconfig and /etc/default

So, here's a bit of an opinion piece on the `/etc/sysconfig/` and `/etc/default` directories that exist on the various distributions in one form or another, and why I believe their use should be faded out. Like everything I say on this blog what follows is just my personal opinion, and not the gospel and has nothing to do with the position of the Fedora project or my employer. The topic of `/etc/sysconfig` has been coming up in discussions over and over again. I hope with this blog story I can explain a bit what we as systemd upstream think about these files.

A few lines about the historical context: I wasn't around when /etc/sysconfig was introduced -- suffice to say it has been around on Red Hat and SUSE distributions since a long long time. Eventually /etc/default was introduced on Debian with very similar semantics. Many other distributions know a directory with similar semantics too, most of them call it either one or the other way. In fact, even other Unix-OSes sported a directory like this. (Such as SCO. If you are interested in the details, I am sure a Unix greybeard of your trust can fill in what I am leaving vague here.) So, even though a directory like this has been known widely on Linuxes and Unixes, it never has been standardized, neither in POSIX nor in LSB/FHS. These directories very much are something where distributions distuingish themselves from each other.

The semantics of `/etc/default` and `/etc/sysconfig` are very losely defined only. What almost all files stored in these directories have in common though is that they are sourcable shell scripts which primarily consist of environment variable assignments. Most of the files in these directories are sourced by the SysV init scripts of the same name. The Debian Policy Manual (9.3.2) and the Fedora Packaging Guidelines suggest this use of the directories, however both

distributions also have files in them that do not follow this scheme, i.e. that do not have a matching SysV init script -- or not even are shell scripts at all.

Why have these files been introduced? On SysV systems services are started via init scripts in `/etc/rc.d/init.d` (or a similar directory). `/etc/` is (these days) considered the place where system configuration is stored. Originally these init scripts were subject to customization by the administrator. But as they grew and become complex most distributions no longer considered them true configuration files, but more just a special kind of programs. To make customization easy and guarantee a safe upgrade path the customizable bits hence have been moved to separate configuration files, which the init scripts then source.

Let's have a quick look what kind of configuration you can do with these files. Here's a short incomprehensive list of various things that can be configured via environment settings in these source files I found browsing through the directories on a Fedora and a Debian machine:

- Additional command line parameters for the daemon binaries
- Locale settings for a daemon
- Shutdown time-out for a daemon
- Shutdown mode for a daemon
- System configuration like system locale, time zone information, console keyboard
- Redundant system configuration, like whether the RTC is in local timezone
- Firewall configuration data, not in shell format (!)
- CPU affinity for a daemon
- Settings unrelated to boot, for example including information how to install a new kernel package, how to configure nspluginwrap or whether to do library

prelinking
- Whether a specific service should be started or not
- Networking configuration
- Which kernel modules to statically load
- Whether to halt or power-off on shutdown
- Access modes for device nodes (!)
- A description string for the SysV service (!)
- The user/group ID, umask to run specific daemons as
- Resource limits to set for a specific daemon
- OOM adjustment to set for a specific daemon

Now, let's go where the beef is: what's wrong with `/etc/sysconfig` (resp. `/etc/default`)? Why might it make sense to fade out use of these files in a systemd world?

- For the majority of these files the reason for having them simply does not exist anymore: systemd unit files are not programs like SysV init scripts were. Unit files are simple, declarative descriptions, that usually do not consist of more than 6 lines or so. They can easily be generated, parsed without a Bourne interpreter and understood by the reader. Also, they are very easy to modify: just copy them from `/lib/systemd/system` to `/etc/systemd/system` and edit them there, where they will not be modified by the package manager. The need to separate code and configuration that was the original reason to introduce these files does not exist anymore, as systemd unit files do not include code. These files hence now are a solution looking for a problem that no longer exists.
- They are inherently distribution-specific. With systemd we hope to encourage standardization between distributions. Part of this is that we want that unit

files are supplied with upstream, and not just added by the packager -- how it has usually been done in the SysV world. Since the location of the directory and the available variables in the files is very different on each distribution, supporting `/etc/sysconfig` files in upstream unit files is not feasible. Configuration stored in these files works against de-balkanization of the Linux platform.

- Many settings are fully redundant in a systemd world. For example, various services support configuration of the process credentials like the user/group ID, resource limits, CPU affinity or the OOM adjustment settings. However, these settings are supported only by some SysV init scripts, and often have different names if supported in multiple of them. OTOH in systemd, all these settings are available equally and uniformly for all services, with the same configuration option in unit files.
- Unit files know a large number of easy-to-use process context settings, that are more comprehensive than what most `/etc/sysconfig` files offer.
- A number of these settings are entirely questionnabe. For example, the aforementiond configuration option for the user/group ID a service runs as is primarily something the distributor has to take care of. There is little to win for administrators to change these settings, and only the distributor has the broad overview to make sure that UID/GID and name collisions do not happen.
- The file format is not ideal. Since the files are usually sourced as shell scripts, parse errors are very hard to decypher and are not logged along the other configuration problems of the services. Generally, unknown variable assignments simply have no effect but this is not warned about. This makes these files harder to debug than necessary.
- Configuration files sources from shell scripts are subject to the execution parameters of the interpreter, and it has many: settings like IFS or LANG tend

to modify drastically how shell scripts are parsed and understood. This makes them fragile.

- Interpretation of these files is slow, since it requires spawning of a shell, which adds at least one process for each service to be spawned at boot.
- Often, files in `/etc/sysconfig` are used to "fake" configuration files for daemons which do not support configuration files natively. This is done by glueing together command line arguments from these variable assignments that are then passed to the daemon. In general proper, native configuration files in these daemons are the much prettier solution however. Command line options like "-k", "-a" or "-f" are not self-explanatory and have a very cryptic syntax. Moreover the same switches in many daemons have (due to the limited vocabulary) often very much contradicting effects. (On one daemon `-f` might cause the daemon to daemonize, while on another one this option turns exactly this behaviour off.) Command lines generally cannot include sensible comments which most configuration files however can.
- A number of configuration settings in `/etc/sysconfig` are entirely redundant: for example, on many distributions it can be controlled via `/etc/sysconfig` files whether the RTC is in UTC or local time. Such an option already exists however in the 3rd line of the `/etc/adjtime` (which is known on all distributions). Adding a second, redundant, distribution-specific option overriding this is hence needless and complicates things for no benefit.
- Many of the configuration settings in `/etc/sysconfig` allow disabling services. By this they basically become a second level of enabling/disabling over what the init system already offers: when a service is enabled with `systemctl enable` or `chkconfig` on these settings override this, and turn the daemon of even though the init system was configured to start it. This of course is very confusing to the user/administrator, and brings virtually no

benefit.

- For options like the configuration of static kernel modules to load: there are nowadays usually much better ways to load kernel modules at boot. For example, most modules may now be autoloaded by udev when the right hardware is found. This goes very far, and even includes ACPI and other high-level technologies. One of the very few exceptions where we currently do not do kernel module autoloading is CPU feature and model based autoloading which however will be supported soon too. And even if your specific module cannot be auto-loaded there's usually a better way to statically load it, for example by sticking it in `/etc/load-modules.d` so that the administrator can check a standardized place for all statically loaded modules.
- Last but not least, /etc already is intended to be the place for system configuration ("Host-specific system configuration" according to FHS). A subdirectory beneath it called `sysconfig` to place system configuration in is hence entirely redundant, already on the language level.

What to use instead? Here are a few recommendations of what to do with these files in the long run in a systemd world:

- Just drop them without replacement. If they are fully redundant (like the local/UTC RTC setting) this is should be a relatively easy way out (well, ignoring the need for compatibility). If systemd natively supports an equivalent option in the unit files there is no need to duplicate these settings in `sysconfig` files. For a list of execution options you may set for a service check out the respective man pages: systemd.exec(5) and systemd.service(5). If your setting simply adds another layer where a service can be disabled, remove it to keep things simple. There's no need to have multiple ways to disable a service.

- Find a better place for them. For configuration of the system locale or system timezone we hope to gently push distributions into the right direction, for more details see previous episode of this series.
- Turn these settings into native settings of the daemon. If necessary add support for reading native configuration files to the daemon. Thankfully, most of the stuff we run on Linux is Free Software, so this can relatively easily be done.

Of course, there's one very good reason for supporting these files for a bit longer: compatibility for upgrades. But that's is really the only one I could come up with. It's reason enough to keep compatibility for a while, but I think it is a good idea to phase out usage of these files at least in new packages.

If compatibility is important, then systemd will still allow you to read these configuration files even if you otherwise use native systemd unit files. If your `sysconfig` file only knows simple options `EnvironmentFile=-/etc/sysconfig/foobar` (See systemd.exec(5) for more information about this option.) may be used to import the settings into the environment and use them to put together command lines. If you need a programming language to make sense of these settings, then use a programming language like shell. For example, place an short shell script in `/usr/lib/<your package>/` which reads these files for compatibility, and then `exec`'s the actual daemon binary. Then spawn this script instead of the actual daemon binary with `ExecStart=` in the unit file.

And this is all for now. Thank you very much for your interest.

Category: projects

← BACK TO INDEX