systemd: Converting sysvinit scripts

Welcome back for another installment of the systemd series. Throughout this series, we discuss ways to use systemd to understand and manage your system. This article focuses on how to convert legacy scripts you may have customized on your system.

SysV init scripts

The init system used previously in Linux was called *SysVinit*. Before we start, it's helpful to know that systemd has a lot of compatibility features for SysVinit. In some cases, you may not have to edit files to maintain your custom solutions. But in others, it's helpful to know how to adapt to and use new systemd technology.

A script used to manage a service in SysVinit was known as a *SysV init script*. With some additions, it was known as a LSB, or Linux Standards Base, init script. Fear not, backwards compatibility for legacy scripts remains 99.9% intact with systemd. That means you don't need to rush to convert all your scripts simply to adopt a systemd-based OS like Fedora. However, there are a number of advantages to doing so.

Init scripts are actually *imperative* shell scripts executed by an interpreter. Here, "imperative" means they contain the commands the system runs to manage the service. However, systemd unit files are *declarative* statements of intent. They give hints to systemd to manage a service, but systemd takes care of execution. As a result, unit files are typically much shorter than the SysV init scripts that they replace. That's because they don't have to worry about implementation, just the intent of the service.

The first step in migrating a SysV init script is to examine the script that you wish to convert. Let's use the sshd init script from Fedora 16 as an example. The previous link will show you the script before it was converted to a systemd unit file. Here's a fun fact: everything handled by that 184 line shell script is now handled by 27 lines of systemd configuration, spread across two unit files.

Runlevels vs. targets

Reviewing the init script linked above, let's look more closely at a few important parts. First is this line:

```
# chkconfig: 2345 55 25
```

The first set of numbers, *2345*, indicates in which *runlevels* in the SysV world this script should run. A SysV runlevel is a definition of a system state where certain processes and services should run. There are a specific set of runlevels defined:

- 0: halt
- 1: single-user
- 2: multi-user
- 3: multi-user with networking
- 4: undefined (user defined)
- 5: multi-user with display manager (graphical login)
- 6: reboot

In systemd, there is no concept of runlevels. These are replaced by *targets*. In systemd, there can be an unlimited set of targets, each represented by a unit file with a *.target* suffix. The analogous target to SysV runlevel 3 in systemd is called *multi-user.target*. Similarly, a unit corresponding to runlevel 5 is called *graphical.target*.

Targets can and do depend on each other. So only once the system has reached *multi-user.target* does it start the services specified in *graphical.target*. There are

also implicit dependencies on *basic.target* which, as the name suggests, establishes basic system services and functions. Some of these happen through their own target dependencies, like *network.target*.

Service ordering

The next two numbers in the SysV init script, *55* and *25*, determine the order for starting and stopping the service. Under SysVinit, scripts run strictly in order. The order is determined by naming a set of links in folders. If a new script is placed in the wrong order to start or stop, the service may fail, the system may appear to hang, or other errors may result.

On the other hand, directives in systemd unit files like *Requires=* and *After=* ensure that units run in the correct order. They don't require prior knowledge about other units or services on the system. For instance, if your unit runs a service that requires the network, you can add *Requires=network.target* and *After=network.target* to the unit file. It will only be run once the network is activated.

There are also dependencies included in the old SysV init script. However, they're not enforced if you simply run the script outside the control of the init system. One of the major features of systemd is deterministic results from running an action.

That's a lot of code

Following header information, such as hard and soft dependencies (*Required-Start, Required-Stop, Should-Start*, and *Should-Stop*) is the description of the service. In this case that's *Start up the OpenSSH server daemon*. Following are various functions that implement what the script does. The standards refer to certain *verbs* the script must implement, such as *start*, *stop*, and *restart*.

As mentioned earlier, though, the systemd way of doing things is declarative. Rather than having to define and code a set of functions, you specify the process to execute. There's no need, for example, for a large amount of code for a common task such as retrieving the process ID (PID) the service starts.

Let's look at the start() function of this script to see what it does, and let's examine it line by line. This is a *shell script,* a program of instructions to be run by the shell started by the init system. Don't worry if you don't know the language; the instructions are explained below.

```
start()
{
    [ -x $SSHD ] || exit 5
    [ -f /etc/ssh/sshd_config ] || exit 6
    # Create keys if necessary
    /usr/sbin/sshd-keygen

    echo -n $"Starting $prog: "
    $SSHD $OPTIONS && success || failure
    RETVAL=$?
    [ $RETVAL -eq 0 ] && touch $lockfile
    [ $RETVAL -eq 0 ] && cp -f $XPID_FILE $PID_FILE
    echo
    return $RETVAL
}
```

In order, here's what the init system does:

- Checks if the *sshd* binary exists and is executable, and if not, exits with a status code of 5.
- Checks whether the configuration file */etc/ssh/sshd_config* exists, and if not, exits with status code 6
- Runs the *sshd-keygen* binary to make keys, if they don't already exist
- Sends a message to the screen/log that it's about to start the service
- Runs the service, and outputs a success or failure message for startup of the *sshd* process
- Captures a *return value* for the *sshd* process itself; 0 means *sshd* has successfully become a service daemon
- Based on the return value, marks several marker files to indicate the service is running, in case the system administrator tries to run it again
- Sends out a "next line" message for the screen/log
- Passes the return value out in case it's needed elsewhere

Note this is only *one function* in the SysV sshd init script. Many other functions also need to be coded, including *stop*, *restart,* and others. Those are over a hundred more lines of code! But what does the systemd unit file look like in comparison?

systemd: cuts down on code

In contrast, here's the *sshd.service* unit file from a Fedora 23 installation. Notice how much smaller and easier to read this file is:

```
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
```

```
After=network.target sshd-keygen.service
Wants=sshd-keygen.service


[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s


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

The first few lines again describe the systemd unit, and where you can read more about it. Here are what the remainder of the lines mean, with links to systemd documentation:

- *After* declares ordering, and describes that *network.target* and *sshd-keygen.service* should run first.
- *Wants* describes that *sshd-keygen* should be run in order for this service to start. Note the subtle difference from ordering. systemd considers order and dependencies *orthogonal*, meaning just because *sshd-keygen* comes first doesn't mean it is a dependency of *sshd* — although in this case it is. *Wants* means systemd should run *sshd-keygen.service,* but if that doesn't complete successfully (for instance, if SSH server keys already exist), *sshd* will still run. If *sshd-keygen.service* needed to complete successfully, you'd use *Requires* instead.
- *EnvironmentFile*, similar to SysVinit, is a configuration file with options for *sshd*. It contains a set of *key=value* pairs that will be passed to *sshd*.
- *ExecStart* is the command that runs to start *sshd*. This replaces the entire *start* function in the old initscript. The *$OPTIONS* variable here is what is specified in EnvironmentFile for the variable OPTIONS.
- *ExecReload* is the command that runs if the sysadmin reloads the *sshd* daemon. This replaces the entire *reload* function in the old initscript.
- *KillMode* sets how systemd will stop the service. In this case, systemd will stop the main *sshd* process only. Any additional *sshd* child processes will continue to manage their open SSH sessions. This is not an option used often, but it is important here. For instance, it stops you from killing your own SSH session when you run *systemctl!*
- *Restart* determines how systemd manages the service if it stops unexpectedly. In this case, *on-failure* means virtually any unexpected failure will cause systemd to restart *sshd*. If *sshd* stops normally, though, such as with a *systemctl stop* command, it will not be restarted.
- RestartSec allows 42 seconds of sleep time before *sshd* is restarted due to an

unexpected stop.

- *WantedBy* means when the command *systemctl enable sshd.service* is run, a link is placed in the *multi-user.target.wants* directory by default. This is one of the mechanisms systemd uses to determine units to run for specific targets. So *multi-user.target* and any target that depends on it, will include running *sshd.service*.

There are of course many other directives available, and the systemd docs cover all of them. But if you were to write your own unit file for a simple service, you needn't use even all the options shown here. You might only need a few directives like *After, ExecStart,* and *WantedBy*. If you look in */usr/lib/systemd/system/* on your own Fedora system, you can find many other *.service* unit files that you can follow as a template.

Additional hints

Another notable feature of systemd, though, is that it doesn't abandon compatibility for SysVinit. Some administrators have created their own custom startup scripts such as */etc/rc.d/rc.local.* If you're not prepared to convert it, don't worry — systemd will honor it, using the *rc-local.service* file already included.

Remember, if you are building custom system services for your Fedora, it's best to place them into */etc/systemd/system*. Typically */etc/* is where all system configuration or customization is kept. This allows you to avoid putting customizations in */usr/lib/systemd/system* which is managed by Fedora's package system.

If you'd like to read another example and explanation of converting, check out this blog entry from the "systemd for administrators" series.

Jon Stanley

Jon is a longtime contributor to Fedora, and a former member of FESCo, the Fedora Board, as well as a current member of the Server Working Group. He lives in New York City, USA.

Previous post

Next post

2 Comments

ADD YOURS

Samuel Sieb
November 5, 2015 at 14:16

Under the ExecStart section, there is an incomplete sentence:
“The $OPTIONS variable here is”

jstanley
November 5, 2015 at 21:55

Thanks! I can’t edit it at this point, but will bring it to the attention of folks that can.

Leave a Reply

Your email address will not be published.

Name

Email

Website

Post Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

SUBSCRIBE TO FEDORA MAGAZINE

Search form

Subscribe with RSS

or

Enter your email address below to receive notifications of new posts by email.

Email Address

Email Address

Subscribe