An Introduction to systemd

...

Erik Johnson

What is systemd?

- Replacement for sysvinit
- Manages your services/daemons
- Integrated logging (journal)
- Easy-to-write service files (units)
- Aims to standardize management of several system management tasks, including (but not limited to) the following:
  - Network configuration
    - Static/DHCP IP configuration, bridging, DNS configuration, etc.
  - System Time/Timezone
  - Power management (ACPI)
  - Scheduled tasks
- A lot more

What is systemd?

Don’t you mean “Systemd” or “SystemD”

- No, it’s ***systemd***, uncapitalized
- The project is actually quite particular about the spelling
- There is an entire paragraph about the reason for the spelling on the project’s homepage: https://www.freedesktop.org/wiki/Software/systemd/
- Spell it “systemd” or suffer the merciless wrath of pedants on the internet

How systemd Differs from Traditional Init Systems

- Linux-only
  - Relies upon cgroups to track daemons and the processes they spawn, rather than manually keeping track of PIDs
  - cgroups are a built-in feature of the Linux kernel which tracks processes when they fork/exec other processes, allowing for service-level resource tracking (CPU, memory, etc.) and limits
  - cgroups can also be used in Linux to organize ps output to show process hierarchy: ps auxf
- Socket-activated services
  - systemd listens for activity on a network socket, FIFO, etc. and spawns an instance of a service when activity is detected
- Intelligent service startup
  - Services which need to talk to network interfaces will wait for the network stack to be initialized before starting
  - No more creative ordering of service startup to achieve this

How systemd Differs from Traditional Init Systems

- Unit files (instead of init scripts)
  - Does not spawn shells to start/stop services
  - Leads to quicker system startup/shutdown (though performance gain may be less noticeable on newer hardware)
- Binary logging (a.k.a. “the journal”)
  - Each log entry is associated with its unit file, allowing for easy filtering of log messages
  - Can replace syslog, but also supports passing through log messages to a syslog daemon so you get both kinds of logging
    - Many distros set this up for you out-of-the-box for convenience, so you may still see the log files you expect to see in **/var/log**

How systemd Differs from Traditional Init Systems

- Targets instead of runlevels
  - Allows for more logical organization of services
  - **multi-user.target** is equivalent to SysV runlevel 3
  - **graphical.target** is equivalent to SysV runlevel 5
  - **reboot.target** is equivalent to SysV runlevel 6
  - **network.target** is reached when the network management stack is reached
  - There are a lot more, to see all active targets run: **`systemctl list-units --type=target`**
  - Add a unit to a target by adding a **`WantedBy`** in the unit file's **`[Install]`** section

Unit Files vs. Init Scripts

- Init scripts are shell scripts
  - With no standard way of initializing daemons, there are almost as many ways of managing init scripts as there are Linux distributions
  - An init script written for SuSE Linux will need to be rewritten/tweaked to work in RHEL, Ubuntu, etc.
- Since there are no competing implementations in systemd, unit files have a standard syntax, making them more portable from one distribution to another
- Most distros have a library of additional functions to implement common tasks (finding pid of daemon, killing all PIDs belonging to a daemon, getting status of daemon), due to these features not being built into init
  - For example, RHEL <= 6 puts these in **/etc/rc.d/init.d/functions**
- These tasks are handled by systemd and do not require these helper functions

Unit Files vs. Init Scripts

- Unit files are easier to read/write than init scripts
- An init script would not fit on this slide without making the text so small that a magnifying glass would be required
- By contrast, a unit file is clear and concise, using the well-known “ini-file” format with bracketed sections and key/value pairs:

```
[Unit]
Description=OpenSSH Daemon
Wants=sshdgenkeys.service
After=sshdgenkeys.service
After=network.target

[Service]
ExecStart=/usr/bin/sshd -D
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=always

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

Unit Files

- Unit file location: **/usr/lib/systemd**
  - Do not edit these files, as they will be owned by individual software packages and will be overwritten when these packages are upgraded
  - If you need to make changes to a unit file, copy it to the same path (relative to **/usr/lib/systemd**) within **/etc/systemd**
    - Example: copy **/usr/lib/systemd/system/sshd.service** to **/etc/systemd/system/sshd.service** and make your changes there
- Any modifications to unit files require that you restart systemd
  - `systemctl daemon-reload`

Working With Units

- **systemctl** (not to be confused with **sysctl**) is used to manage units
  - Starting a unit
    - `systemctl start sshd.service`
  - Stopping a unit
    - `systemctl stop sshd.service`
  - Restarting a unit
    - `systemctl restart sshd.service`
  - Enable a unit to start at boot
    - `systemctl enable sshd.service`
  - Disabling service so it does not run at boot
    - `systemctl disable sshd.service`
  - Displaying the contents of a unit file
    - `systemctl cat sshd.service`

Working With Units

- **systemctl status** is used to get information about a unit

```
% systemctl status sshd.service
● sshd.service - OpenSSH Daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; disabled; vendor
preset: disabled)
   Active: active (running) since Wed 2017-04-19 22:09:50 CDT; 8s ago
 Main PID: 833 (sshd)
    Tasks: 1 (limit: 4915)
   Memory: 752.0K
      CPU: 8ms
   CGroup: /system.slice/sshd.service
           └─833 /usr/bin/sshd -D

Apr 19 22:09:50 tardis systemd[1]: Started OpenSSH Daemon.
Apr 19 22:09:50 tardis sshd[833]: Server listening on 0.0.0.0 port 22.
Apr 19 22:09:50 tardis sshd[833]: Server listening on :: port 22.
```

ACPI Support

- **systemd-logind** can replace **acpid** for window managers that use it to handle power-related ACPI events
- Edit **/etc/systemd/logind.conf** (or **systemd-logind.conf**, depending on the distro) and set the following parameters:
  - **`HandlePowerKey`** – Power off system when power button is pressed
  - **`HandleSleepKey`** – Suspend system when sleep key is pressed
  - **`HandleLidSwitch`** – Suspend system when laptop lid is closed
- Run **man logind.conf** for more information on valid values for the above parameters
- You'll need to restart **systemd-logind.service** for changes to this config file to take effect

ACPI Support

- Full-fledged desktop environments such as GNOME, KDE, XFCE, etc. (which have their own ACPI handlers) will not require this file to be configured, and will likely have a GUI to configure ACPI event-handling
- Configuring **systemd-logind** is more helpful for users of tiling window managers with no desktop environment

Sleep, Hibernate, Shutdown, etc.

- Sleep (Suspend to RAM)
  - `systemctl suspend`
- Hibernate (Suspend to Disk)
  - `systemctl hibernate`
- **/sbin/shutdown** tasks
  - Reboot
    - `systemctl reboot`
  - Halt System (without powering off)
    - `systemctl halt`
  - Power Off System
    - `systemctl poweroff`

The Journal

- All services managed by systemd send log entries to the journal
  - This takes the place of traditional syslog
- systemd can be configured to send log entries to a socket, to which traditional syslog daemons such as **syslog-ng** or **rsyslog** can listen
  - Most distros will set this up for you, but in distros like Arch this must be configured manually
- Journal entries are lost on reboot unless the directory **/var/log/journal** exists

The Journal

- **journalctl** is used to interact with the journal
  - Show all messages by a specific executable
    - `journalctl /full/path/to/executable`
  - Show all messages by a specific PID (ex. 456)
    - `journalctl _PID=456`
  - Show all messages by a specific unit
    - `journalctl _SYSTEMD_UNIT=sshd.service`
  - Show all messages in journal
    - `journalctl`
- Similar to the tail command, the **-f** flag can be used to follow the journal, while the **-n** flag can be used to limit results to a number of most recent messages
- Run **man journalctl** for the full list of options

Timers

- Timer units (ending in **.timer**) activate a service unit of the same name
  - e.g. **foo.timer** activates **foo.service**
- 2 types
  - **Monotonic:** activates at a fixed time/interval starting when the system is booted
    - Defined by setting one or more of **OnActiveSec**, **OnBootSec**, **OnStartupSec**, **OnUnitActiveSec**, or **OnUnitInactiveSec** in the timer unit
  - **Realtime:** activates at a specific calendar event (like a cron job)
    - Defined by setting **OnCalendar** in the timer unit
- The **systemd.timer** and **systemd.time** manpages contain more documentation

Timer Example (foo.timer)

- Monotonic

```
[Unit]
Description=Run foo hourly and on boot

[Timer]
OnBootSec=15min
OnUnitActiveSec=1h

[Install]
WantedBy=timers.target
```

- Starts **foo.service** 15 minutes after boot and hourly thereafter

- Realtime

```
[Unit]
Description=Run foo weekly

[Timer]
OnCalendar=weekly
Persistent=true

[Install]
WantedBy=timers.target
```

- Starts **foo.service** at midnight every Monday morning

Timer Example (service unit)

- Here's an example of the corresponding service unit (**foo.service**)
  - Notice there is no **`[Install]`** section
  - This is because it is the *timer* that is enabled/started using **systemctl**

```
[Unit]
Description=Update foo

[Service]
Type=simple
ExecStart=/usr/bin/update-foo
```

Timer Accuracy

- Timers do not trigger at the precise time specified for the timer
- A timer setting called **`AccuracySec`** (default: 1min) helps establish a time range in which the timer will trigger
  - A randomized value between the time the timer expires and the time period specified by **`AccuracySec`** will be chosen
  - For timers which execute on a repeating schedule, this value will remain stable (i.e. it will not be random for every repetition of the timer)
- This allows for a natural spreading of jobs executed by a number of hosts, to prevent all of them running the same job and potentially overloading a database or other shared resource
- For timers which must execute as close as possible to the specified time, set **`AccuracySec=1us`** (1 microsecond) in the timer unit

Timers as a Cron Replacement

- Pros
  - Easy to start a job independently of the timer (service unit can be run with **systemctl start**)
  - Very granular control over the environment used by the command being executed by the service unit (see **systemd.exec** manpage)
  - Job runs and their output are logged to the journal for easy access/troubleshooting
- Cons
  - Not as simple as configuring a cron job; two unit files need to be created instead of adding a single line to the crontab
  - No built-in emailing of output from jobs
    - This can be achieved by creating a service unit that calls a script to send the message, and then triggering it by adding an **OnFailure** to the service unit
    - Example: https://wiki.archlinux.org/index.php/Systemd/Timers#MAILTO
    - **OnFailure** is not limited to services activated by timers, it can be used on *any* service unit

Transient Timers

- Using **systemd-run**, a transient timer and service can be created to run a single command
  - e.g. `systemd-run --on-active=1m touch /tmp/foo`
  - **--on-active=**, **--on-boot=**, **--on-startup=**, **--on-unit-active=**, and **--on-unit-inactive=** can be used to make the timer monotonic, while **--on-calendar=** can be used to make the timer realtime
- The same accuracy mechanic that applies to regular timers also applies to transient timers
  - By default, the timer will execute a random amount of time between when the specified time is reached, and one minute after
  - To modify the accuracy, use `--timer-property=AccuracySec=`
    - e.g. `--timer-property=AccuracySec=100ms`

Instance Names

- Some unit files naturally lend themselves to multiple instances (e.g. **openvpn**)
- Unit files which support multiple instances contain an @ sign before the suffix
  - e.g. **openvpn-client@.service**
- When this sort of unit file is used, the instance name goes after the @ sign
  - e.g. **openvpn-client@vpn_name.service**
- In the unit file, the instance name is represented by the **%i** placeholder
  - There are a number of other placeholders that can be used in unit files, the systemd.unit manpage contains a section called **SPECIFIERS**

Unit File Example with Instance Name

```
[Unit]
Description=OpenVPN tunnel for %I
After=syslog.target network-online.target
Wants=network-online.target
Documentation=man:openvpn(8)
Documentation=https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
Documentation=https://community.openvpn.net/openvpn/wiki/HOWTO

[Service]
Type=notify
PrivateTmp=true
WorkingDirectory=/etc/openvpn/client
ExecStart=/usr/bin/openvpn --suppress-timestamps --nobind --config %i.conf
CapabilityBoundingSet=CAP_IPC_LOCK CAP_NET_ADMIN CAP_NET_RAW CAP_SETGID CAP_SETUID CAP_SYS_CHROOT
CAP_DAC_OVERRIDE
LimitNPROC=10
DeviceAllow=/dev/null rw
DeviceAllow=/dev/net/tun rw
ProtectSystem=true
ProtectHome=true

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

Per-user systemd Instances

- systemd provides a PAM session module (enabled by default on virtually all distros which use systemd) which will launch a per-user instance of systemd

```
% ps aux | grep 'systemd --user' | grep -v grep
erik      7839  0.0  0.0  55812  7196 ?        Ss   Mar24   0:09 /usr/lib/systemd/systemd --user
```

- Per-user unit files are placed in **`~/.config/systemd/user/`**
- **systemctl**, **journalctl**, **systemd-run**, etc. all support a **`--user`** flag which tells those commands to connect to the per-user systemd instance
- Users can run their own services, timers, etc. without privileged access
  - All processes spawned by a per-user systemd instance will be run as the user of course, and not the root user

Network Management

- systemd provides a component called **systemd-networkd** which, when enabled (systemd-networkd.service) will allow network interfaces to automatically be configured as they are detected
- This is not enabled by default, and in fact RHEL/CentOS by default uses their own service unit to manage network interfaces (keeping their old configuration method from prior RHEL/CentOS release cycles)
- Network configuration files provided by system packages are found in **/lib/systemd/network**, while new ones should be placed in **/etc/systemd/network** to avoid conflicts
- Documentation for these configuration files can be found in the **systemd-networkd** manpage, which lists a couple other manpages to read

Configuring Network Interfaces

- Interface configuration files must end in **.network**
- DHCP Example

```
[Match]
Name=enp1s0

[Network]
DHCP=ipv4
```

- Static IP example

```
[Match]
Name=enp1s0

[Network]
Address=10.1.10.9/24
Gateway=10.1.10.1
```

NOTE: globbing is supported in the **`Name`** match. This allows for USB network interfaces (which may be named differently depending on the port they are plugged into) to be matched

Configuring Virtual Interfaces

- Interface configuration files must end in **.netdev**
- Bridge example

```
[NetDev]
Name=br0
Kind=bridge
```

- Unlike **.network** files, globbing is not supported
  - We're creating a specific interface, so we need a unique name
- Documentation can be found in the **systemd.netdev** manpage
- A **.network** file would still be necessary to assign an IP address to the bridge

Binding an Interface to a Bridge

- Instead of configuring DHCP or a static IP address, the **`Bridge`** option is used to bind the interface to the bridge

```
[Match]
Name=enp1s0

[Network]
Bridge=br0
```

- Remember, the bridge interface is the one with the IP address assigned to it

More on Network Management

- Any changes to configuration files requires a restart of **systemd-networkd.service**
- For DNS servers assigned via DNS, you will also need to enable and start **systemd-resolved.service** and then symbolically link **/etc/resolv.conf** to **/run/systemd/resolve/resolv.conf**
  - `ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf`
  - It may be a good idea to back up the old **/etc/resolv.conf** first
- The current status of the network interfaces can be viewed by running `networkctl`

Helpful Links

- systemd mainpage: https://www.freedesktop.org/wiki/Software/systemd/
- Arch Wiki links:
  - https://wiki.archlinux.org/index.php/Systemd
  - https://wiki.archlinux.org/index.php/Systemd-networkd
  - https://wiki.archlinux.org/index.php/Systemd/User
  - https://wiki.archlinux.org/index.php/Init/Rosetta