Linux

Systemd

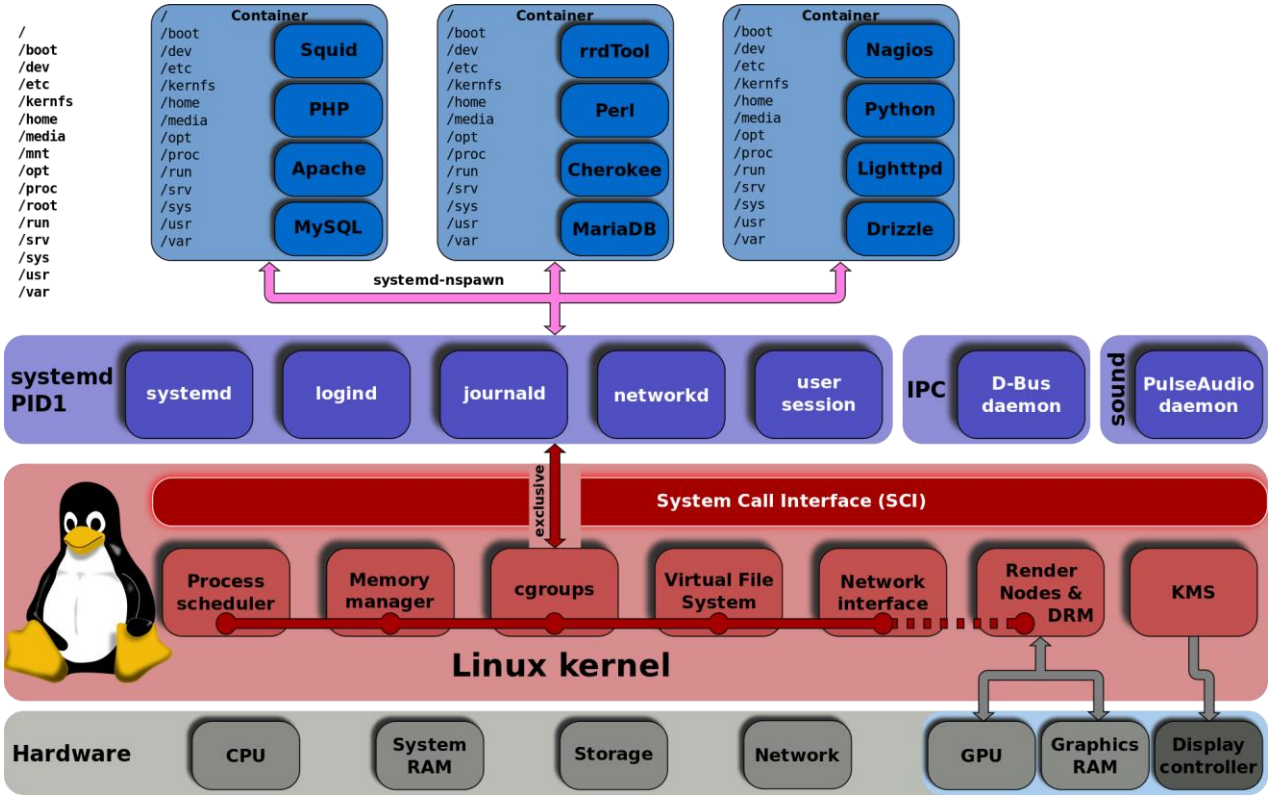
Introduction

- systemd is an array of system components for Linux-based operating systems
- systemd's primary component is a "system and service manager"—an init system used to bootstrap user space and manage user processes.
- It also provides replacements for various daemons and utilities, including device management, login management, network connection management, event logging, container management and a bootloader.

systemd's history

- Initially developed in 2010 by two RedHat engineers: Lennart Poettering and Kay Sievers
- Originally inspired by MacOS's launchd
- The idea was to create an integrated set of crucial system components, which could:
    - Dynamically react to changes in both hardware and software – event-driven approach, e.g. starting bluetoothd when Bluetooth dongle is plugged in
    - Keep track of services and the processes started by them – no more rogue processes not killed by service stop xxx
    - Parallelize everything – system and service startup, etc
    - Get rid of using shell in boot-up process – shell is quite slow
    - Have declarative syntax instead of turing-full shell scripts
- All of that was not possible with the existing solutions (by that time most popular init system was sysVinit, followed by Upstart)
- By 2021, systemd is the default init system for most Linux distributions

systemd’s architecture (with satellite services)



Init system component

- Is the first process executed by the kernel during the booting of a system (pid 1).
- Responsible for bringing up rest of userspace, e.g.
  - Mounting “real’ root filesystem and all other filesystems
  - Starting and managing services
  - Handling timers, sockets, other events – hardware cold/hotplug, path change, d-bus activations.
  - …
- Reparents orphan processes
- Relies on cgroups for service supervision and control of services execution environment

cgroups trivia

- Abbreviated from control groups
- Linux kernel feature, which is:
    - A way to hierarchally group and label processes
    - A way to then apply resource limits to these groups
- Also used by docker, LXC, other containerization software.

cgroups and systemd

- Each service runs in its own cgroup
- That allows for:
  - Tracking all processes created by a service – including spawned ones and double-forks
  - Per-service resource monitoring and limitation
- Also isolates user sessions – each session gets its own cgroup
- Using cgroups removes the need for hacky solutions with race conditions (PID files)

- Further reading: Control Groups vs. Control Groups from Lennart's blog

systemd-cgls: cgroup hierarchy visualized

```
Unit system.slice (/system.slice):
├─NetworkManager-dispatcher.service
│ └─717525 /usr/lib/nm-dispatcher
├─systemd-udevd.service
│ ├─   405 /usr/lib/systemd/systemd-udevd
│ ├─717520 /usr/lib/systemd/systemd-udevd
│ └─717521 /usr/lib/systemd/systemd-udevd
├─polkit.service
│ └─634 /usr/lib/polkit-1/polkitd --no-debug
├─rtkit-daemon.service
│ └─960 /usr/lib/rtkit-daemon
├─iwd.service
│ └─507 /usr/lib/iwd/iwd
├─systemd-journald.service
│ └─387 /usr/lib/systemd/systemd-journald
├─NetworkManager.service
│ ├─610 /usr/bin/NetworkManager --no-daemon
│ └─744 /usr/bin/dnsmasq --conf-file=/dev/null --no-hosts --keep-in-foreground --bind-interfaces --except-interface=
├─greetd.service
│ └─617 greetd
├─systemd-resolved.service
│ └─471 /usr/lib/systemd/systemd-resolved
├─dbus.service
│ └─483 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
├─systemd-timesyncd.service
│ └─472 /usr/lib/systemd/systemd-timesyncd
└─systemd-logind.service
  └─488 /usr/lib/systemd/systemd-logind
```

systemd-cgtop: per-group resource usage

Control Group	Tasks	%CPU	Memory	Input/s	Output/s
user.slice	681	24.1	6.4G	-	-
user.slice/user-1000.slice	681	24.1	6.3G	-	-
user.slice/user-1000.slice/session-3.scope	647	24.1	6.1G	-	-
/	893	23.9	7.5G	-	-
system.slice	28	0.4	220.4M	-	-
system.slice/systemd-resolved.service	1	0.4	6.9M	-	-
user.slice/user-1000.slice/user@1000.service	34	0.0	191.5M	-	-
dev-hugepages.mount	-	-	4.0K	-	-
dev-mqueue.mount	-	-	56.0K	-	-
init.scope	1	-	22.2M	-	-
proc-sys-fs-binfmt_misc.mount	-	-	4.0K	-	-
sys-fs-fuse-connections.mount	-	-	4.0K	-	-
sys-kernel-config.mount	-	-	4.0K	-	-
sys-kernel-debug.mount	-	-	4.0K	-	-
system.slice/NetworkManager.service	4	-	19.3M	-	-
system.slice/boot.mount	-	-	20.0K	-	-
system.slice/dbus.service	1	-	3.6M	-	-
system.slice/greetd.service	1	-	5.8M	-	-
system.slice/iwd.service	1	-	2.6M	-	-
system.slice/polkit.service	12	-	17.3M	-	-
system.slice/rtkit-daemon.service	3	-	716.0K	-	-
system.slice/sysroot.mount	-	-	4.0K	-	-
system.slice/system-modprobe.slice	-	-	420.0K	-	-
system.slice/system-openvpn\x2dclient.slice	-	-	3.6M	-	-
system.slice/system-systemd\x2dbacklight.slice	-	-	92.0K	-	-
system.slice/system-systemd\x2dcoredump.slice	-	-	624.0K	-	-
system.slice/system-systemd\x2dcryptsetup.slice	-	-	8.0K	-	-
system.slice/system-systemd\x2dfsck.slice	-	-	240.0K	-	-
system.slice/systemd-journald.service	1	-	77.2M	-	-

Essential services: systemd-logind

- Manages user logins. Is responsible for:
    - Keeping track of user processes using cgroups – that also allows to limit the resources available to them
    - Device access management (e.g. true multiseat)
    - Handling power/sleep hardware keys and idling (e.g. running a lock screen after some minutes)
    - Providing applications with an ability to inhibit sleep/shutdown (e.g. video player)
    - Providing polkit-based access for operations such as sleep or shutdown
- `loginctl` command is used for controlling it – e.g. `loginctl list-sessions` to show active login sessions.

Satellite services

- Non-essential services include:
  - systemd-networkd, which manages network-related configuration.
  - systemd-timesyncd: simple SNTP client, used for syncing time from the network.
  - systemd-resolved: recursive DNS server, used to provide caching and per-interface DNS resolution
  - systemd-homed: manager for portable home directories
  - systemd-oomd: userspace OOM killer

systemctl: managing units

- Unit is something that systemd can control: services (.service), sockets (.socket), mountpoints (.mount and .automount), targets (.target), etc
- By default, systemctl assumes that you are trying to control a service (`systemctl start foo` is the same as `systemctl start foo.service`)
- Several units can be specified in one command (e.g. `systemctl start i2pd nginx`)
- Basic commands:

Command	Effect
`systemctl start foo`	Tries to start foo.service
`systemctl stop foo`	Tries to stop foo.service
`systemctl restart foo`	Tries to restart foo.service (equivalent to stop followed by a start)
`systemctl reload foo`	Asks units to reload their configuration (Works only if .service file defines a command for it)
`systemctl enable foo`	Enables the unit to be auto-started (for example, on boot)
`systemctl disable foo`	Disables auto-start of a unit

systemctl: checking system state

- List units with systemctl list-units (or just systemctl)
  - List only failed units: systemctl –failed
  - List only services: systemctl --type=service
- System status overview: systemctl status
  - Can also be used on any unit: systemctl status foo.service:

```
● NetworkManager.service - Network Manager
     Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled; vendor preset: disabled)
    Drop-In: /usr/lib/systemd/system/NetworkManager.service.d
             └─NetworkManager-ovs.conf
     Active: active (running) since Wed 2021-09-08 14:46:55 MSK; 4 days ago
       Docs: man:NetworkManager(8)
   Main PID: 610 (NetworkManager)
      Tasks: 4 (limit: 18395)
     Memory: 19.3M
        CPU: 34.551s
     CGroup: /system.slice/NetworkManager.service
             ├─610 /usr/bin/NetworkManager --no-daemon
             └─744 /usr/bin/dnsmasq --conf-file=/dev/null --no-hosts --keep-in-foreground --bind-interfaces --except-interface=lo --clear-on-reload --strict-order

сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3091] device (tun_vps1): state change: unmanaged -> unavailable (reason 'connection-assumed',
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3121] device (tun_vps1): state change: unavailable -> disconnected (reason 'connection-assume
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3132] device (tun_vps1): Activation: starting connection 'tun_vps1' (de7565be-7b10-47ca-a829-
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3154] device (tun_vps1): state change: disconnected -> prepare (reason 'none', sys-iface-stat
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3159] device (tun_vps1): state change: prepare -> config (reason 'none', sys-iface-state: 'ex
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3162] device (tun_vps1): state change: config -> ip-config (reason 'none', sys-iface-state: '
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3165] device (tun_vps1): state change: ip-config -> ip-check (reason 'none', sys-iface-state:
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3182] device (tun_vps1): state change: ip-check -> secondaries (reason 'none', sys-iface-stat
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3183] device (tun_vps1): state change: secondaries -> activated (reason 'none', sys-iface-sta
сен 12 20:27:01 hostname.com NetworkManager[610]: <info>  [1631467621.3188] device (tun_vps1): Activation: successful, device activated.
```

Unit files: basics

- Declarative syntax - .ini file format with few additions
- Parts of a unit file (or even entire unit) can be overridden (check `systemd-delta` utility)
- `systemd-cat` can be used to get unit's config file
- `systemctl edit` can be used to edit these
- Alternatively, look in:
    - /lib/systemd/system/ - for distribution-provided services
    - /etc/systemd/system/ - for local overrides and custom units

Example: named

```
# /usr/lib/systemd/system/named.service
[Unit]
Description=Internet domain name server
After=network.target

[Service]
ExecStart=/usr/bin/named -f -u named
ExecReload=/usr/bin/kill -HUP $MAINPID

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

Unit files: options

- Can be found in man pages for systemd.unit ([Unit] section), systemd.service ([Service] section) and systemd.exec (execution-related documentation)
- `systemctl show`: allows you to show all options that are applied to a unit
- Some commonly used options:
- [Unit] section:
    - Description: human readable description
    - Docs: link to a documetation
    - Dependencies (before/after, etc): used to configure ordering dependencies. For example, named will only be started after network has been initialized
- [Service] section:
    - ExecStart: what to execute on service start. Could be any executable.
    - ExecReload: which command to use when asking service to reload its configuration
- [Install] section:
    - WantedBy: which target to attach this unit to when it’s being enabled.

Unit file: options (2)

- Some very useful options:
  - Restarting a crashed service:
    - `Restart=on-failure`
  - Sourcing environmental variables from a file:
    - `EnvironmentFile=/etc/sysconfig/sshd`
  - Run unit as a non-root user/group:
    - `User=nobody and Group=nobody`
  - Setting working directory for a service:
    - `WorkingDirectory=/etc/openvpn/client`
  - Limit the amount of file descriptors (files, sockets, etc) service can open simultaneously:
    - `LimitNOFILE=128`
  - Disallow service that needs to be run as root to access parts of a system it doesn't need:
    - `ProtectHome=yes`
    - `ProtectSystem=strict`
    - … and many others

Timer units: basics

- Cron, which has all the power of systemd – dependencies, execution environment configuration, etc
- Realtime timers: calendar events, similar to cron
  - They use format described in systemd.time(7)
  - Example: every minute `(*-*-* *:*:00)`, every hour `(*-*-* *:00:00)` every day `(*-*-* 00:00:00)`
  - If a timer's activation was missed (for example, system was shut down) and option `Persistent=true` was used, the timer will be started as soon as the system is booted up.
- Monotonic timers, relative to a point in time
  - 5 minutes after system startup: `OnBootSec=5m`
  - 1 hour after the last service activation: `OnUnitActiveSec=1h` (should be combined with `OnBootSec` to ensure that it runs after a reboot)

Timer units: example (1)

```
# /usr/lib/systemd/system/shadow.timer
[Unit]
Description=Daily verification of password and group files

[Timer]
OnCalendar=daily
AccuracySec=12h
Persistent=true
```

Timer units: example (2)

```
# /usr/lib/systemd/system/shadow.service
[Unit]
Description=Verify integrity of password and group files
After=systemd-sysusers.service

[Service]
Type=simple
# Always run both checks, but fail the service if either fails
ExecStart=/bin/sh -c '/usr/bin/pwck -r || r=1; /usr/bin/grpck -r && exit $r'
Nice=19
IOSchedulingClass=best-effort
IOSchedulingPriority=7
```

Timer management

- List all timers:
  - `systemctl list-timers`
- Start timer at boot:
  - `systemctl enable myscript.timer`
- Start timer right now:
  - `systemctl start myscript.timer`

Instantiated services

- Allows user to create multiple instances of one service with different parameters
    - For example: multiple instances of an openvpn process, each with different configuration file used
- Unit files of instantiated services must end with `@.service` (For example – `openvpn-client@.service`)
- Each instance is a unit, which means:
    - Each instance can be managed individually (e.g. `systemctl start/stop/restart openvpn-client@vps1.service`)
    - Logging is separate
    - Any instance can be overridden - creating `/etc/systemd/system/openvpn-client@vps4.service` will affect only `vps4` instance
- Using instantiated services requires user to set some variables:
    - `%i` – escaped instance name: systemd doesn't allow special character in unit names
    - `%I` – unescaped service name: for documentation and human-readable descriptions
- To get the full list of variables, check man for `systemd.unit(5)`

Instantiated services: example

```
# /usr/lib/systemd/system/openvpn-client@.service
[Unit]
Description=OpenVPN tunnel for %I
After=syslog.target network-online.target
Wants=network-online.target

[Service]
WorkingDirectory=/etc/openvpn/client
ExecStart=/usr/bin/openvpn --suppress-timestamps --nobind --config %i.conf
User=openvpn

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

Socket-activated services

- systemd creates a socket for the application, and passes pending connections once the service is ready
- Benefits:
    - Other services don’t need to wait for socket-activated service to get ready
        - They can instantly try to access it; the attempt will block until the service is ready to accept connections
    - Services can be started on-demand, they don’t need to be constantly running
- That synergizes well with instanced services – for example, a user can spawn a script per connection
- Socket service consists of two parts: .socket and .service (or @.service, if you want to create an instance per connection)

Socket-activated services: example .socket

```
# sshd.socket
[Unit]
Description=SSH Socket for Per-Connection Servers

[Socket]
ListenStream=22
Accept=yes

[Install]
WantedBy=sockets.target
```

Socket-activated services: example @.service

```
# sshd@.service
[Unit]
Description=SSH Per-Connection Server

[Service]
ExecStart=-/usr/sbin/sshd -i
StandardInput=socket
```

Socket-activated services: checking state

- There's one instance of `sshd@.service` per connection:

```
$ systemctl --full | grep ssh
sshd@172.31.0.52:22-172.31.0.4:47779.service  loaded active running       SSH Per-Connection Server
sshd@172.31.0.52:22-172.31.0.54:52985.service loaded active running       SSH Per-Connection Server
sshd.socket                                   loaded active listening     SSH Socket for Per-Connection…
```

systemd-journald

- Collects log messages from a variety of sources:
    - Kernel log messages (via kmsg)
    - Syslog messages sent via syslog function (e.g. via logger command)
    - Standard output and error streams of service units
    - Native Journal API
    - Audit records from kernel audit subsystem
- journald also forwards everything collected to system's syslog server – for further processing and remote collection
- Stores them in an indexed binary format under /var/log/journal (by default)
- Rotation is handled automatically and trasparently based on configured free space thresholds
- Also stores metadata with the logs – for example PID, timestamp and systemd unit from which the log message was sent
- `journalctl is used to control the daemon`

Systemd-journald: example message in JSON


```
{
"__CURSOR" :
"s=c4ee459c883148549d114c566bc0b979;i=12b782;b=6c92864cbcc64a5fabebe04147953894;m=42d22604a2;t=58bc87981a1f5;x
=8daf632187
"__REALTIME_TIMESTAMP" : "1561068031812085",
"__MONOTONIC_TIMESTAMP" : "286993548450",
"_BOOT_ID" : "6c92864cbcc64a5fabebe04147953894",
"SYSLOG_FACILITY" : "3",
"_UID" : "0",
"_GID" : "0",
...
"UNIT" : "apache2.service",
"CODE_LINE" : "2039",
"CODE_FUNCTION" : "unit_notify",
"MESSAGE" : "apache2.service: Unit entered failed state.",
"_SOURCE_REALTIME_TIMESTAMP" : "1561068031809136"
}
```

Journalctl basics

- View the full log: `journalctl`
- Since last boot: `journalctl --boot`
- For a time interval: `journalctl --since=today` or `journalctl --until=yesterday`
- Filter by unit name: `journalctl --unit=sshd`
- Continuously tail the output: `journalctl -f`
- Show 100 last entries: `journalctl -n 100`
- Further reading: Lennart's blog

THANK YOU