Mastering Systemd

Introduction

Agenda

- Managing Systemd - Basics
- Advanced Systemd Service Management
- Using systemd security
- Managing Mounts and Automounts
- Managing Systemd Timers
- Working with Systemd Sockets
- Creating Containers with systemd-nspawn
- Managing Resource Allocation with Cgroups
- Managing Networks with systemd
- Managing User Settings
- Miscellaneous stuff

Lab/Demo Requirements

- Best: run the latest version of Fedora Workstation or Ubuntu
- Second best: run the latest version of CentOS / RHEL 8, or Ubuntu LTS
- Also possible: run any other distribution that is systemd based (features discussed here might not be implemented)

Poll Question 1

Which of the following topics do you consider most interesting?

- Managing Systemd - Basics
- Advanced Systemd Service Management
- Using systemd security
- Managing Mounts and Automounts
- Managing Systemd Timers
- Working with Systemd Sockets
- Creating Containers with systemd-nspawn
- Managing Resource Allocation with Cgroups
- Managing Networks with systemd
- Managing User Settings
- Miscellaneous stuff

Poll Question 2

Rate your own systemd knowledge / experience

- none
- I hate it
- beginner
- comfortable
- advanced
- expert

Poll Question 3

Rate your own Linux knowledge / experience

- none
- I hate it
- beginner
- comfortable
- advanced
- expert

Poll Question 4

Where are you from?

- India
- Asia
- Africa
- Netherlands
- Europe
- North/Central America
- South America
- Australia/Pacific
- Greenland/Antartica

Poll Question 5

Which job title best applies to you?

- sysadmin
- developer
- devops
- DBA
- network engineer
- security specialist
- architect
- other IT support

Poll 6

What is your primary Linux distribution?

- Red Hat / CentOS
- Fedora
- Oracle Linux
- Ubuntu / Debian / Mint
- Kali
- SUSE
- Arch
- Gentoo
- something else

Mastering Systemd

1. Systemd Service Management - basics

Understanding Units

- The Unit is the thing managed by systemd
- Different unit types are available, use **systemctl -t help** for an overview
- Unit files are written in three locations:
  - /etc/systemd/system: administrator environment, highest priority
  - /run/systemd/system: non-persistent
  - /usr/lib/systemd/system: package provided, lowest priority

Finding the Right Man Page

- Many systemd man pages are available
- Start with **man 7 systemd.directives**, which has a complete list indicating which specific mount page to use
- Use **systemd.<unit-type>** for more specific information about a unit type
- Use **systemd.exec** for more information about the systemd runtime environment
- Use **man -k systemd** for a complete overview

Setting the Systemd Editor

- The systemd editor by default is nano
- Set **export SYSTEMD_EDITOR=vim** in /etc/bashrc to permanently change it

Viewing and Customizing Units

- **systemctl cat httpd**
- **systemctl edit httpd**
  - Creates a drop-in in /etc/systemd/system/httpd.d/
- **systemctl show --all httpd**

Understanding Targets

- A target is a group of units that can be managed as one, using **systemctl start**, **systemctl stop**, etc.
- Some targets are isolatable, which means they can be used to define a state in which a system should be started
- Use **systemctl get-default** to see the default target in which your system will boot
- Use **systemctl set-default** to set the default target to something else

Understanding Systemd Journal

- Systemd units generate messages that are captured by the systemd journal
- Control of the journal goes through **journalctl**
- Data logged by systemd-journald by default is not persistent

Mastering Systemd

2. Systemd Service Management - advanced

Understanding Systemd

- Systemd provides 3 main functions
  - A system and service manager
  - A software platform that serves as a basis for developing other software
  - Glue between applications and the kernel, which provides interfaces to kernel functionality
- Systemd offer scripts to start and manage services, mounts, paths, sockets and more (units), but also offers other functionality
  - **systemd-journald** takes care of logging
  - **systemd-udevd** takes care of hardware initialization
  - **systemd-logind** takes care of session management
  - **systemd-networkd** can be used for network configuration
  - **systemd-nspawn** offers container functionality

Understanding Unit Dependencies

- Dependencies can be defined in the unit files
- Use **systemctl list-dependencies** to show current dependencies

Managing Dependencies

- **Requires=** if this unit loads, units listed here will load also. If one of the other units is deactivated, this unit will be deactivated
- **Requisite=** if the units listed here are not already loaded, this unit will fail
- **Wants=** this unit wants to load the units listed here, but won't fail if any of these units fail
- **Before=** will start this unit before the unit mentioned with **Before=**
- **After=** will start this unit after the unit mentioned with **After=**

Writing your Own Unit Files

- Anything can run as a Systemd service
- Use the appropriate options in the **type=** field in the **[Service]** section
  - **Type=simple**: runs the process specified with **ExecStart**
  - **Type=oneshot**: like **simple**, but waits for the process to exit before starting anything else
  - See **man 5 systemd.service** for more details

Using systemd-run

- **systemd-run** allows users to run a command through systemd
- While doing so, the command by default will run in the background through a unit that is automatically generated - watch command output for more details
- Use **journalctl -u run-<some-id>** for the journald entries
- Use **systemctl run -t** to run the command in an interactive terminal, use ^] three times to disconnect

Mastering Systemd

3. Using Systemd Security
   (Most recent versions only)

Understanding Users

- As a default, systemd processes run as a child of the systemd daemon, as root
- Use **User=myuser** in the [Service] section to run as a different user account
- Or, run **systemd-run -p User=myuser** to change the user account
- Alternatively, use **DynamicUser=yes** to assign a unique user ID on the fly
  - This user ID will never make it to /etc/passwd, but is generated and removed on the fly by NSS
  - Dynamic users are great if no connection is required with other services

Userstanding Mount namespaces

- A namespace is a strictly isolated environment created by the Linux kernel
- Using mount namespaces makes it impossible for systemd to access specific directories
  - **ProtectHome=** makes /home read-only or inaccessible, set to **on**, **off** or **read-only**
  - **ProtectSystem=** protects /usr, /boot and /etc, set to **on** or **off** to protect /usr and /boot, and **full** to also make /etc read-only
  - **PrivateTmp=** uses namespaces to give each process a private /tmp directory

Protecting Specific Directories

- **InaccessiblePaths=** makes paths inaccessible
- **ReadOnlyPaths=** makes paths readonly
- **ReadWritePaths=** defines exceptions to ReadOnlyPaths=
- **BindPaths=**
- **ReadOnlyBindPaths=**

Automatic Creation of Directories

- Systemd can automatically create directories for specific services, which will be removed once the service has stopped
  - **ConfigurationDirectory=**
  - **CacheDirectory=**
  - **StateDirectory=**
  - **LogsDirectory=**
  - **RuntimeDirectory=**

More Security Settings

- **ProtectKernelTunables=** disabled modifications to /proc and /sys
- **ProtectKernelModules=** No loading and unloading of kernel modules
- **ProtectControlGroups=** No writes to /sys/fs/cgroup
- **RestrictSUIDSGID=** No SUID and SGID on files
- **MemoryDenyWriteExecute=** no memorymapping that is simultaneously writable and executable
- **RestrictRealTime=** prohibits real-time scheduling
- **RemoveIPC=** removes semaphores, shared memory and message queues
- **SystemCallFilter=** sets a blacklist or whitelist of specific system calls
- **SELinuxContext=** sets SELinux context

Using **systemd-analyze**

- Use **systemd-analyze** to analyze multiple aspects of the working of systemd
- Use **systemd-analyze security my.service** to analyze security settings in a specific unit file
- Unrelated but also cool is **systemd-analyze blame**, which will show how much time was taken by each unit

Mastering Systemd

4. Managing Mounts and Automounts

Demo: (auto)mounting with systemd

- Run an NFS server that exports /var on localhost
- **nfsdata.mount**
- **nfsdata.automount**

Mastering Systemd

5. Managing Systemd Timers

Demo: using timers

- Systemd timers are cron on steroids and add functionality to what cron can do
  - run a command for a specific amount of time
  - run a command after occurance of a specific event
- **systemctl cat fstrim.timer**
- **systemctl cat fstrim.service**
- **man systemd.timer**
- **systemctl status *timer**

NTS

- systemctl start monitor.timer
- journalctl -S today -f -u monitor.service

Mastering Systemd

6. Working with Systemd Sockets

Demo: using sockets

- **yum install tftp-server**
- **systemctl cat tftp.service**
- **systemctl cat tftp.socket**

Mastering Systemd

7. Creating Containers with Systemd Nspawn

Understanding **systemd-nspawn**

- **systemd-nspawn** is a systemd component that allows for running containers
- It basically is chroot on steroids
- Container processes are directly managed by systemd
- There is no multi-node orchestration
- **machinectl** is a related utility to manage containers and virtual machines that are running on top of systemd
- Install the **systemd-container** package if necessary

Demo: running Containers in nspawn

In this demo, a ready-to-use cloud image is used

- **sudo machinectl pull-raw --verify=no https://download.fedoraproject.org/pub/fedora/linux/releases/31/Cloud/x86_64/images/Fedora-Cloud-Base-31-1.9.x86_64.raw.xz Fedora-Cloud-Base-31-1.9.x86-64**
- **sudo machinectl list-images**
- **sudo systemd-nspawn -M Fedora-Cloud-Base-31-1.9.x86-64**

Demo: running containers in nspawn

In this demo, a chroot directory is set up and the container is started from there

- **sudo dnf -y --releasever=31 --installroot=/var/lib/machines/f31 --disablerepo='*' --enablerepo=fedora --enablerepo=updates install systemd passwd dnf fedora-release vim-minimal glibc-minimal-langpack**
- **cd /var/lib/machines/f31; systemd-nspawn -a f31 passwd root**
- **sudo systemd-nspawn -bD /var/lib/machines/f31**

Managing Systemd-nspawn

- Disconnect from a running container using ^] 3 times
- Use **machinectl list** for a list of currently running containers
- **machinectl list-images** shows images
- **machinectl status my-container** shows status information
- **sudo systemctl -M my-container status sshd.service** allows systemd to get service information from the container
- **sudo journalctl -M my-container -u sshd.service** tells journald to get into the container to get information

Mastering Systemd

8. Managing Resource Allocation

Understanding Resource Allocation

- Slice: system level unit for cgroup resource allocation management
- scope: subdivision of a slice to make it easier to manage resources for groups of processes
- service: resource limitations for individual processes

Understanding Cgroups

- RHEL 8 is on Cgroup v1, Fedora 31 and later is on Cgroup v2
- Cgroup v2 in RHEL 8 expected in next major release
- Cgroup v2 integrates much better in systemd
- Relevant parameters in Cgroup v1
  - CPUAccounting, CPUQuota, CPUShares
  - MemoryAccounting, MemoryLimit
  - TaskAccounting, TasksMax
  - BlockIOAccounting, BlockIOWeight, BlockIODeviceWeight
- Relevant parameters in Cgroup v2
  - CPUWeight
  - MemoryMax
  - IO* instead of BlockIO*

Managing Resource Limitations

- **systemctl set-property --runtime httpd CPUShares=2048** (runtime)
- **systemctl set-property httpd CPUShares=2048** (permanent)
- Or put in unit file:

  [Service]

  CPUShare=2048

- New controls in Cgroups v2
  - AllowedCPUs=
  - AllowedMemoryNodes=
- Tip! Consider setting
  - **systemctl set-property system.slice CPUShares=8192**
- **systemd-run -p CPUQuota=15% /usr/bin/stress1**

Monitoring Resource Usage

- **systemd-cgls**
- **systemd-cgtop**

Mastering Systemd

9. Managing Networks with systemd

A primer to systemd-networkd

- All recent distributions can deal with systemd-networkd
- **systemctl disable NetworkManager**
- **systemctl enable systemd-networkd**
- **systemctl enable systemd-resolved**
- **rm /etc/resolv.conf**
- **ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf**
- **mkdir /etc/systemd/network**
- create configuration files in this directory

/etc/systemd/network/20-dhcp.network

```
[Match]
Name=enp3*

[Network]
DHCP=yes
```

Understanding Systemd Users

- User processes are started by systemd as well
- The unit files for user processes are in:
  - ~/.config/systemd/user: user units
  - /usr/lib/systemd/user: maintainer user units
  - /etc/systemd/user: global users applying to all users
- User processes end up in a scope that is reserved for that specific user
- The systemd user environment has its own environment, set in ~/.config/environment.d
  - Use **systemctl --user show-environment** to show it
  - Use **systemctl --user import-environment** to import .bashrc and .bash_profile

/etc/systemd/network/10-static-enp3s0.network

```
[Match]
Name=enp3s0

[Network]
Address=10.0.0.10/24
Gateway=10.0.0.1
DNS=8.8.8.8
```

Mastering Systemd

10. Systemd --user

Running Units as User

- **systemctl --user** gives a generic overview
- **systemctl --user enable --now my.service** runs and enables my.service in the user slice
  - Notice the service is loaded from /usr/lib/systemd/user which is linked to .config/systemd/user, and started in the user slice.
- **journalctl --user-unit=my.service** shows information from the journal

Using **loginctl**

- **loginctl** is used as systemd session manager
- To allow a user container to be started at system start, use **loginctl enable-linger $user**
- **loginctl status $user** will show current settings
- Also cool: **loginctl list-sessions** lists current sessions; **loginctl kill-sessions <id>** terminates a specific session

Demo: Enabling a Container to run as User Systemd Service

- **podman run -d --name mynginx -p8080:80 nginx**
- **sudo loginctl enable-linger $(id -un)**
- **mkdir ~/.config/systemd/user**
- **podman generate systemd --name mynginx --files**
- **systemctl --user daemon-reload**
- **systemctl --user enable container-mynginx.service**

Mastering Systemd

11. Miscellaneous stuff

Aborting "a stop job is running"

- Systemd services have a timeout when they stop
- This timeout allows the service to shut down properly
- Once the job has been scheduled to stop, interrupting the timeout is not possible
- Edit /etc/systemd/system.conf to change the default settings
  - **DefaultTimeoutStartSec=90s**
  - **DefaultTimeoutStopSec=90s**
- Use **TimeoutStopUSec** on specific services to change the timeout
- Notice that after TimeoutStopUSec, the process will get killed which may lead to data loss!

Using **systemd-udevd**

- **systemd-udevd** is responsible for managing hardware
- Use **udevadm monitor** to see hardware events while hardware is connected
- These events relate to files created in the /sys filesystem

Mastering Systemd

Before we end

Poll Question 7

Which of the following topics did you consider most interesting?

- Managing Systemd - Basics
- Advanced Systemd Service Management
- Using systemd security
- Managing Mounts and Automounts
- Managing Systemd Timers
- Working with Systemd Sockets
- Creating Containers with systemd-nspawn
- Managing Resource Allocation with Cgroups
- Managing Networks with systemd
- Managing User Settings
- Miscellaneous stuff