1. INTRODUCTION TO SYSTEMD

1.1. Systemd is a system and service manager for Linux operating systems. It is designed to be backwards compatible with SysV init scripts, and provides a number of features such as parallel startup of system services at boot time, on-demand activation of daemons, support for system state snapshots, or dependency-based service control logic. In Red Hat Enterprise Linux 7, systemd replaces Upstart as the default init system.

1.2. Systemd introduces the concept of systemd units. These units are represented by unit configuration files located in one of the directories listed below, and encapsulate information about system services, listening sockets, saved system state snapshots, and other objects that are relevant to the init system.

1.3. Available systemd Unit Types:

| | | |
|---|---|---|
| 1.3.1. | Service unit | A system service. |
| 1.3.2. | Target unit | A group of systemd units. |
| 1.3.3. | Automount unit | A file system automount point. |
| 1.3.4. | Device unit | A device file recognized by the kernel. |
| 1.3.5. | Mount unit | A file system mount point. |
| 1.3.6. | Path unit | A file or directory in a file system. |
| 1.3.7. | Scope unit | An externally created process. |
| 1.3.8. | Slice unit | A group of units that manage system processes. |
| 1.3.9. | Snapshot unit | A saved state of the systemd manager. |
| 1.3.10. | Socket unit | An inter-process communication socket. |
| 1.3.11. | Swap unit | A swap device or a swap file. |
| 1.3.12. | Timer unit | A systemd timer. |

1.4. Systemd Unit Files Locations:

1.4.1. /usr/lib/systemd/system/

1.4.1.1. Systemd unit files distributed with installed RPM packages.

1.4.2. /run/systemd/system/

1.4.2.1. Systemd unit files created at run time. This directory takes precedence over the directory with installed service unit files.

1.4.3. /etc/systemd/system/

1.4.3.1. Systemd unit files created by systemctl enable as well as unit files added for extending a service. This directory takes precedence over the directory with runtime unit files.

1.5. Main Features:

1.5.1. Socket-based activation — At boot time, systemd creates listening sockets for all system services that support this type of activation, and passes the sockets to these services as soon as they are started. This not only allows systemd to start services in parallel, but also makes it possible to restart a service without losing any message sent to it while it is unavailable: the corresponding socket remains accessible and all messages are queued. Systemd uses socket units for socket-based activation.

1.5.2. Bus-based activation — System services that use D-Bus for inter-process communication can be started on-demand the first time a client application attempts to communicate with them. Systemd uses D-Bus service files for bus-based activation.

1.5.3. Device-based activation — System services that support device-based activation can be started on-demand when a particular type of hardware is plugged in or becomes available. Systemd uses device units for device-based activation.

1.5.4. Path-based activation — System services that support path-based activation can be started on-demand when a particular file or directory changes its state. Systemd uses path units for path-based activation.

1.5.5. System state snapshots — Systemd can temporarily save the current state of all units or restore a previous state of the system from a dynamically created snapshot. To store the current state of the system, systemd uses dynamically created snapshot units.

1.5.6. Mount and automount point management — Systemd monitors and manages mount and automount points. Systemd uses mount units for mount points and automount units for automount points.

1.5.7. Aggressive parallelization — Because of the use of socket-based activation, systemd can start system services in parallel as soon as all listening sockets are in place. In combination with system services that support on-demand activation, parallel activation significantly reduces the time required to boot the system.

1.5.8. Transactional unit activation logic — Before activating or deactivating a unit, systemd calculates its dependencies, creates a temporary transaction, and verifies that this transaction is consistent. If a transaction is inconsistent, systemd automatically attempts to correct it and remove non-essential jobs from it before reporting an error.

1.5.9. Backwards compatibility with SysV init — Systemd supports SysV init scripts as described in the Linux Standard Base Core Specification, which eases the upgrade path to systemd service units.

1.6. Compatibility Changes

1.6.1. Systemd has only limited support for runlevels. It provides a number of target units that can be directly mapped to these runlevels and for compatibility reasons, it is also distributed with the earlier runlevel command. Not all systemd targets can be directly mapped to runlevels, however, and as a consequence, this command might return N to indicate an unknown runlevel. It is recommended that you avoid using the runlevel command if possible.

1.6.2. The systemctl utility does not support custom commands. In addition to standard commands such as start, stop, and status, authors of SysV init scripts could implement support for any number of arbitrary commands in order to provide additional functionality. For example, the init script for iptables in Red Hat Enterprise Linux 6 could be executed with the panic command, which immediately enabled panic mode and reconfigured the system to start dropping all incoming and outgoing packets. This is not supported in systemd and the systemctl only accepts documented commands.

1.6.3. The systemctl utility does not communicate with services that have not been started by systemd. When systemd starts a system service, it stores the ID of its main process in order to keep track of it. The systemctl utility then uses this PID to query and manage the service. Consequently, if a user starts a particular daemon directly on the command line, systemctl is unable to determine its current status or stop it.

1.6.4. Systemd stops only running services. Previously, when the shutdown sequence was initiated, Red Hat Enterprise Linux 6 and earlier releases of the system used symbolic links located in the /etc/rc0.d/ directory to stop all available system services regardless of their status. With systemd, only running services are stopped on shutdown.

1.6.5. System services are unable to read from the standard input stream. When systemd starts a service, it connects its standard input to /dev/null to prevent any interaction with the user.

1.6.6. System services do not inherit any context (such as the HOME and PATH environment variables) from the invoking user and their session. Each service runs in a clean execution context.

1.6.7. When loading a SysV init script, systemd reads dependency information encoded in the Linux Standard Base (LSB) header and interprets it at run time.

1.6.8. All operations on service units are subject to a default timeout of 5 minutes to prevent a malfunctioning service from freezing the system. This value is hardcoded for services that are generated from initscripts and cannot be changed. However, individual configuration files can be used to specify a longer timeout value per service.

2. MANAGING SYSTEM SERVICES

2.1. Previous versions of Red Hat Enterprise Linux, which were distributed with SysV init or Upstart, used init scripts located in the /etc/rc.d/init.d/ directory. These init scripts were typically written in Bash, and allowed the system administrator to control the state of services and daemons in their system. In Red Hat Enterprise Linux 7, these init scripts have been replaced with service units.

2.2. Service units end with the .service file extension and serve a similar purpose as init scripts. To view, start, stop, restart, enable, or disable system services, use the systemctl command. The service and chkconfig commands are still available in the system and work as expected, but are only included for compatibility reasons and should be avoided.

2.3. service name start

2.3.1. systemctl start name.service — Starts a service.

2.4. service name stop

2.4.1. systemctl stop name.service — Stops a service.

2.5. service name restart

2.5.1. systemctl restart name.service — Restarts a service.

2.6. service name condrestart

2.6.1. systemctl try-restart name.service — Restarts a service only if it is running.

2.7. service name reload

2.7.1. systemctl reload name.service — Reloads configuration.

2.8. service name status

2.8.1. systemctl status name.service

2.8.2. systemctl is-enabled name.service — Checks if a service is running.

2.9. service --status-all

2.9.1. systemctl list-units --type service --all — Displays the status of all services.

2.10. chkconfig name on

2.10.1. systemctl enable name.service — Enables a service.

2.11. chkconfig name off

2.11.1. systemctl disable name.service — Disables a service.

2.12. chkconfig --list name

2.12.1. systemctl status name.service

2.12.2. systemctl is-enabled name.service — Checks if a service is enabled.

2.13. chkconfig --list

2.13.1. systemctl list-unit-files --type service — Lists and checks if they are enabled.

2.14. chkconfig --list

2.14.1. systemctl list-dependencies --after — Lists services that are ordered to start before the specified unit.

2.15. chkconfig --list

2.15.1. systemctl list-dependencies --before — Lists services that are ordered to start after the specified unit.

3. Listing Services
    3.1. To list all currently loaded service units, type the following at a shell prompt:
        3.1.1. systemctl list-units --type service
    3.2. For each service unit file, this command displays its full name (UNIT) followed by a note whether the unit file has been loaded (LOAD), its high-level (ACTIVE) and low-level (SUB) unit file activation state, and a short description (DESCRIPTION).
    3.3. By default, the systemctl list-units command displays only active units. If you want to list all loaded units regardless of their state, run this command with the --all or -a command line option:
        3.3.1. systemctl list-units --type service --all
    3.4. You can also list all available service units to see if they are enabled. To do so, type:
        3.4.1. systemctl list-unit-files --type service
    3.5. For each service unit, this command displays its full name (UNIT FILE) followed by information whether the service unit is enabled or not (STATE).

4. Displaying Service Status
    4.1. To display detailed information about a service unit that corresponds to a system service, type the following at a shell prompt:
        4.1.1. systemctl status name.service
    4.2. Available Service Unit Information:
        4.2.1. Loaded
            4.2.1.1. Information whether the service unit has been loaded, the absolute path to the unit file, and a note whether the unit is enabled.
        4.2.2. Active
            4.2.2.1. Information whether the service unit is running followed by a time stamp.
        4.2.3. Main PID
            4.2.3.1. The PID of the corresponding system service followed by its name.
        4.2.4. Status
            4.2.4.1. Additional information about the corresponding system service.
        4.2.5. Process
            4.2.5.1. Additional information about related processes.
        4.2.6. CGroup
            4.2.6.1. Additional information about related Control Groups (cgroups).
    4.3. To only verify that a particular service unit is running, run the following command:
        4.3.1. systemctl is-active name.service
    4.4. Similarly, to determine whether a particular service unit is enabled, type:
        4.4.1. systemctl is-enabled name.service
    4.5. Note that both systemctl is-active and systemctl is-enabled return an exit status of 0 if the specified service unit is running or enabled.
    4.6. To determine what services are ordered to start before the specified service, type the following at a shell prompt:
        4.6.1. systemctl list-dependencies --after name.service
    4.7. To determine what services are ordered to start after the specified service, type the following at a shell prompt:
        4.7.1. systemctl list-dependencies --before name.service

5. Starting and stopping a Service
   5.1. To start a service unit that corresponds to a system service:
      5.1.1. systemctl start name.service
   5.2. To stop a service unit that corresponds to a system service:
      5.2.1. systemctl stop name.service
   5.3. To restart a service unit that corresponds to a system service:
      5.3.1. systemctl restart name.service
   5.4. Replace name with the name of the service unit you want to restart (for example, httpd). This command stops the selected service unit in the current session and immediately starts it again. Importantly, if the selected service unit is not running, this command starts it too. To tell systemd to restart a service unit only if the corresponding service is already running, run the following command as root:
      5.4.1. systemctl try-restart name.service
   5.5. Certain system services also allow you to reload their configuration without interrupting their execution. To do so, type as root:
      5.5.1. systemctl reload name.service
   5.6. Note that system services that do not support this feature ignore this command altogether. For convenience, the systemctl command also supports the reload-or-restart and reload-or-try-restart commands that restart such services instead.
   5.7. To configure a service unit that corresponds to a system service to be automatically started at boot time, type the following at a shell prompt as root:
      5.7.1. systemctl enable name.service
   5.8. Replace name with the name of the service unit you want to enable (for example, httpd). This command reads the [Install] section of the selected service unit and creates appropriate symbolic links to the /usr/lib/systemd/system/name.service file in the /etc/systemd/system/ directory and its subdirectories. This command does not, however, rewrite links that already exist. If you want to ensure that the symbolic links are re-created, use the following command as root:
      5.8.1. systemctl reenable name.service
   5.9. This command disables the selected service unit and immediately enables it again.
   5.10. To prevent a service unit that corresponds to a system service from being automatically started at boot time, type the following at a shell prompt as root:
      5.10.1. systemctl disable name.service
   5.11. Replace name with the name of the service unit you want to disable (for example, bluetooth). This command reads the [Install] section of the selected service unit and removes appropriate symbolic links to the /usr/lib/systemd/system/name.service file from the /etc/systemd/system/ directory and its subdirectories. In addition, you can mask any service unit to prevent it from being started manually or by another service. To do so, run the following command as root:
      5.11.1. systemctl mask name.service
   5.12. This command replaces the /etc/systemd/system/name.service file with a symbolic link to /dev/null, rendering the actual unit file inaccessible to systemd. To revert this action and unmask a service unit, type as root:
      5.12.1. systemctl unmask name.service

6. WORKING WITH SYSTEMD TARGETS

6.1. Previous versions of Red Hat Enterprise Linux, which were distributed with SysV init or Upstart, implemented a predefined set of runlevels that represented specific modes of operation. These runlevels were numbered from 0 to 6 and were defined by a selection of system services to be run when a particular runlevel was enabled by the system administrator. In Red Hat Enterprise Linux 7, the concept of runlevels has been replaced with systemd targets.

6.2. Systemd targets are represented by target units. Target units end with the .target file extension and their only purpose is to group together other systemd units through a chain of dependencies. For example, the graphical.target unit, which is used to start a graphical session, starts system services such as the GNOME Display Manager (gdm.service) or Accounts Service (accounts-daemon.service) and also activates the multi-user.target unit. Similarly, the multi-user.target unit starts other essential system services such as NetworkManager (NetworkManager.service) or D-Bus (dbus.service) and activates another target unit named basic.target.

6.3. Red Hat Enterprise Linux 7 is distributed with a number of predefined targets that are more or less similar to the standard set of runlevels from the previous releases of this system. For compatibility reasons, it also provides aliases for these targets that directly map them to SysV runlevels.

6.4. To determine which target unit is used by default, run the following command:

6.4.1. systemctl get-default

6.5. This command resolves the symbolic link located at /etc/systemd/system/default.target and displays the result.

6.6. To list all currently loaded target units, type the following command at a shell prompt:

6.6.1. systemctl list-units --type target

6.7. For each target unit, this commands displays its full name (UNIT) followed by a note whether the unit has been loaded (LOAD), its high-level (ACTIVE) and low-level (SUB) unit activation state, and a short description (DESCRIPTION).

6.8. By default, the systemctl list-units command displays only active units. If you want to list all loaded units regardless of their state, run this command with the --all or -a command line option:

6.8.1. systemctl list-units --type target --all

6.9. To configure the system to use a different target unit by default, type the following at a shell prompt as root:

6.9.1. systemctl set-default name.target

6.10. To change to a different target unit in the current session, type the following at a shell prompt as root:

6.10.1. systemctl isolate name.target

6.11. Changing to Rescue Mode

6.11.1. Rescue mode provides a convenient single-user environment and allows you to repair your system in situations when it is unable to complete a regular booting process. In rescue mode, the system attempts to mount all local file systems and start some important system services, but it does not activate network interfaces or allow more users to be logged into the system at the same time. In Red Hat Enterprise Linux 7, rescue mode is equivalent to single user mode and requires the root password.

6.11.2. To change the current target and enter rescue mode in the current session, type the following at a shell prompt as root:

6.11.2.1. systemctl rescue

6.11.3. This command is similar to systemctl isolate rescue.target, but it also sends an informative message to all users that are currently logged into the system. To prevent systemd from sending this message, run this command with the --no-wall command line option:

6.11.3.1. systemctl --no-wall rescue

6.12. Changing to Emergency Mode

6.12.1. Emergency mode provides the most minimal environment possible and allows you to repair your system even in situations when the system is unable to enter rescue mode. In emergency mode, the system mounts the root file system only for reading, does not attempt to mount any other local file systems, does not activate network interfaces, and only starts a few essential services. In Red Hat Enterprise Linux 7, emergency mode requires the root password.

6.12.2. To change the current target and enter emergency mode, type the following at a shell prompt as root:

6.12.2.1. systemctl emergency

6.12.3. This command is similar to systemctl isolate emergency.target, but it also sends an informative message to all users that are currently logged into the system. To prevent systemd from sending this message, run this command with the --no-wall command line option:

6.12.3.1. systemctl --no-wall emergency

7. SHUTTING DOWN, SUSPENDING, AND HIBERNATING THE SYSTEM

7.1. In Red Hat Enterprise Linux 7, the systemctl utility replaces a number of power management commands used in previous versions of the Red Hat Enterprise Linux system.

7.1.1. halt

7.1.1.1. systemctl halt — Halts the system.

7.1.2. poweroff

7.1.2.1. systemctl poweroff — Powers off the system.

7.1.3. reboot

7.1.3.1. systemctl reboot — Restarts the system.

7.1.4. pm-suspend

7.1.4.1. systemctl suspend — Suspends the system.

7.1.5. pm-hibernate

7.1.5.1. systemctl hibernate — Hibernates the system.

7.1.6. pm-suspend-hybrid

7.1.6.1. systemctl hybrid-sleep — Hibernates and suspends the system.

7.2. Shutting Down the System

7.2.1. The systemctl utility provides commands for shutting down the system, however the traditional shutdown command is also supported. Although the shutdown command will call the systemctl utility to perform the shutdown, it has an advantage in that it also supports a time argument. This is particularly useful for scheduled maintenance and to allow more time for users to react to the warning that a system shutdown has been scheduled. The option to cancel the shutdown can also be an advantage.

7.2.2. To shut down the system and power off the machine, type the following at a shell prompt as root:

7.2.2.1. systemctl poweroff

7.2.3. To shut down and halt the system without powering off the machine, run the following command as root:

7.2.3.1. systemctl halt

7.2.4. By default, running either of these commands causes systemd to send an informative message to all users that are currently logged into the system. To prevent systemd from sending this message, run the selected command with the --no-wall command line option, for example:

7.2.4.1. systemctl --no-wall poweroff

7.2.5. To shut down the system and power off the machine at a certain time, use a command in the following format as root:

7.2.5.1. shutdown --poweroff hh:mm

7.2.5.2. Where hh:mm is the time in 24 hour clock format. The /run/nologin file is created 5 minutes before system shutdown to prevent new logins. When a time argument is used, an optional message, the wall message, can be appended to the command.

7.2.6. To shut down and halt the system after a delay, without powering off the machine, use a command in the following format as root:

7.2.6.1. shutdown --halt +m

7.2.6.2. Where +m is the delay time in minutes. The now keyword is an alias for +0.

7.2.7. A pending shutdown can be canceled by the root user as follows:

7.2.7.1. shutdown -c

7.3. Restarting the System

7.3.1. To restart the system, run the following command as root:

7.3.1.1. systemctl reboot

7.3.2. By default, this command causes systemd to send an informative message to all users that are currently logged into the system. To prevent systemd from sending this message, run this command with the --no-wall command line option:

7.3.2.1. systemctl --no-wall reboot

7.4. Suspending the System

7.4.1. To suspend the system, type the following at a shell prompt as root:

7.4.1.1. systemctl suspend

7.4.2. This command saves the system state in RAM and with the exception of the RAM module, powers off most of the devices in the machine. When you turn the machine back on, the system then restores its state from RAM without having to boot again. Because the system state is saved in RAM and not on the hard disk, restoring the system from suspend mode is significantly faster than restoring it from hibernation, but as a consequence, a suspended system state is also vulnerable to power outages.

7.5. Hibernating the System

7.5.1. To hibernate the system, type the following at a shell prompt as root:

7.5.1.1. systemctl hibernate

7.5.2. This command saves the system state on the hard disk drive and powers off the machine. When you turn the machine back on, the system then restores its state from the saved data without having to boot again. Because the system state is saved on the hard disk and not in RAM, the machine does not have to maintain electrical power to the RAM module, but as a consequence, restoring the system from hibernation is significantly slower than restoring it from suspend mode.

7.5.3. To hibernate and suspend the system, run the following command as root:

7.5.3.1. systemctl hybrid-sleep

8. CONTROLLING SYSTEMD ON A REMOTE MACHINE

8.1. In addition to controlling the systemd system and service manager locally, the systemctl utility also allows you to interact with systemd running on a remote machine over the SSH protocol. Provided that the sshd service on the remote machine is running, you can connect to this machine by running the systemctl command with the --host or -H command line option:

8.1.1. systemctl --host user_name@host_name command

8.2. Replace user_name with the name of the remote user, host_name with the machine's host name, and command with any of the systemctl commands described above. Note that the remote machine must be configured to allow the selected user remote access over the SSH protocol.

9. CREATING AND MODIFYING SYSTEMD UNIT FILES

9.1. A unit file contains configuration directives that describe the unit and define its behavior. Several systemctl commands work with unit files in the background. To make finer adjustments, system administrator must edit or create unit files manually.

9.2. The /etc/systemd/system/ directory is reserved for unit files created or customized by the system administrator.

9.3. Unit files can be supplemented with a directory for additional configuration files. For example, to add custom configuration options to sshd.service, create the sshd.service.d/custom.conf file and insert additional directives there.

9.4. Also, the sshd.service.wants/ and sshd.service.requires/ directories can be created. These directories contain symbolic links to unit files that are dependencies of the sshd service. The symbolic links are automatically created either during installation according to [Install] unit file options or at runtime based on [Unit] options. It is also possible to create these directories and symbolic links manually.

9.5. Understanding the Unit File Structure

9.5.1. [Unit] — contains generic options that are not dependent on the type of the unit. These options provide unit description, specify the unit's behavior, and set dependencies to other units.

9.5.1.1. Description

9.5.1.1.1. A meaningful description of the unit. This text is displayed for example in the output of the systemctl status command.

9.5.1.2. Documentation

9.5.1.2.1. Provides a list of URIs referencing documentation for the unit.

9.5.1.3. After

9.5.1.3.1. Defines the order in which units are started. The unit starts only after the units specified in After are active. Unlike Requires, After does not explicitly activate the specified units. The Before option has the opposite functionality to After.

9.5.1.4. Requires

9.5.1.4.1. Configures dependencies on other units. The units listed in Requires are activated together with the unit. If any of the required units fail to start, the unit is not activated.

9.5.1.5. Wants

9.5.1.5.1. Configures weaker dependencies than Requires. If any of the listed units does not start successfully, it has no impact on the unit activation. This is the recommended way to establish custom unit dependencies.

9.5.1.6. Conflicts

9.5.1.6.1. Configures negative dependencies, an opposite to Requires.

9.5.2. [unit type] — if a unit has type-specific directives, these are grouped under a section named after the unit type.

9.5.2.1. Type

9.5.2.1.1. Configures the unit process startup type that affects the functionality of ExecStart and related options. One of:

9.5.2.1.1.1. simple – The default value. The process started with ExecStart is the main process of the service.

9.5.2.1.1.2. forking – The process started with ExecStart spawns a child process that becomes the main process of the service. The parent process exits when the startup is complete.

9.5.2.1.1.3. oneshot – This type is similar to simple, but the process exits before starting consequent units.

9.5.2.1.1.4. dbus – This type is similar to simple, but consequent units are started only after the main process gains a D-Bus name.

9.5.2.1.1.5. notify – This type is similar to simple, but consequent units are started only after a notification message is sent via the sd_notify() function.

9.5.2.1.1.6. idle – similar to simple, the actual execution of the service binary is delayed until all jobs are finished, which avoids mixing the status output with shell output of services.

9.5.2.2. ExecStart

9.5.2.2.1. Specifies commands or scripts to be executed when the unit is started. ExecStartPre and ExecStartPost specify custom commands to be executed before and after ExecStart. Type=oneshot enables specifying multiple custom commands that are then executed sequentially.

9.5.2.3. ExecStop

9.5.2.3.1. Specifies commands or scripts to be executed when the unit is stopped.

9.5.2.4. ExecReload

9.5.2.4.1. Specifies commands or scripts to be executed when the unit is reloaded.

9.5.2.5. Restart

9.5.2.5.1. With this option enabled, the service is restarted after its process exits, with the exception of a clean stop by the systemctl command.

9.5.2.6. RemainAfterExit

9.5.2.6.1. If set to True, the service is considered active even when all its processes exited. Default value is False. This option is especially useful if Type=oneshot is configured.

9.5.3. [Install] — contains information about unit installation used by systemctl enable and disable commands.

9.5.3.1. Alias

9.5.3.1.1. Provides a space-separated list of additional names for the unit. Most systemctl commands, excluding systemctl enable, can use aliases instead of the actual unit name.

9.5.3.2. RequiredBy

9.5.3.2.1. A list of units that depend on the unit. When this unit is enabled, the units listed in RequiredBy gain a Require dependency on the unit.

9.5.3.3. WantedBy

9.5.3.3.1. A list of units that weakly depend on the unit. When this unit is enabled, the units listed in WantedBy gain a Want dependency on the unit.

9.5.3.4. Also

9.5.3.4.1. Specifies a list of units to be installed or uninstalled along with the unit.

9.5.3.5. DefaultInstance

9.5.3.5.1. Limited to instantiated units, this option specifies the default instance for which the unit is enabled.

9.5.3.5.2.

9.6. Creating Custom Unit Files

9.6.1. Prepare the executable file with the custom service. This can be a custom-created script, or an executable delivered by a software provider. Make sure the source script is executable (by executing the chmod a+x) and is not interactive.

9.6.2. Create a unit file in the /etc/systemd/system/ directory and make sure it has correct file permissions. Execute as root:

9.6.2.1. touch /etc/systemd/system/name.service

9.6.2.2. chmod 664 /etc/systemd/system/name.service

9.6.2.3. Replace name with a name of the service to be created. Note that file does not need to be executable.

9.6.3. Open the name.service file created in the previous step, and add the service configuration options. The following is an example unit configuration for a network-related service:

9.6.3.1. [Unit]

9.6.3.1.1. Description=service_description

9.6.3.1.2. After=network.target

9.6.3.2. [Service]

9.6.3.2.1. ExecStart=path_to_executable

9.6.3.2.2. Type=forking

9.6.3.2.3. PIDFile=path_to_pidfile

9.6.3.3. [Install]

9.6.3.3.1. WantedBy=default.target

9.6.3.4. Where:

9.6.3.4.1. service_description is an informative description that is displayed in journal log files and in the output of the systemctl status command.

9.6.3.4.2. the After setting ensures that the service is started only after the network is running. Add a space-separated list of other relevant services or targets.

9.6.3.4.3. path_to_executable stands for the path to the actual service executable.

9.6.3.4.4. Type=forking is used for daemons that make the fork system call. The main process of the service is created with the PID specified in path_to_pidfile.

9.6.3.4.5. WantedBy states the target or targets that the service should be started under.

9.6.4. Notify systemd that a new name.service file exists by executing the following command as root:

9.6.4.1. systemctl daemon-reload

9.6.4.2. systemctl start name.service

10. Example. Creating the emacs.service File
    10.1. # touch /etc/systemd/system/emacs.service
    10.2. # chmod 664 /etc/systemd/system/emacs.service
    10.3. Add the following content to the file:
        10.3.1. [Unit]
        10.3.2. Description=Emacs: the extensible, self-documenting text editor
        10.3.3. [Service]
        10.3.4. Type=forking
        10.3.5. ExecStart=/usr/bin/emacs --daemon
        10.3.6. ExecStop=/usr/bin/emacsclient --eval "(kill-emacs)"
        10.3.7. Environment=SSH_AUTH_SOCK=%t/keyring/ssh
        10.3.8. Restart=always
        10.3.9. [Install]
        10.3.10. WantedBy=default.target
    10.4. With the above configuration, the /usr/bin/emacs executable is started in daemon mode on service start. The SSH_AUTH_SOCK environment variable is set using the "%t" unit specifier that stands for the runtime directory. The service also restarts the emacs process if it exits unexpectedly.
    10.5. # systemctl daemon-reload
    10.6. # systemctl start emacs.service
    10.7. As the editor is now registered as a systemd service, you can use all standard systemctl commands. For example, run systemctl status emacs to display the editor's status or systemctl enable emacs to make the editor start automatically on system boot.