Use systemd on Oracle Linux

docs.oracle.com/en/learn/use_systemd/

Note:

- This tutorial is available in an Oracle-provided free lab environment.
- It uses example values for Oracle Cloud Infrastructure credentials, tenancy, and compartments. When completing your lab, substitute these values with ones specific to your cloud environment.

Introduction

In this tutorial, you learn how to use the **systemctl** command line utility to manage and view systemd units that are controlled by systemd. This tutorial is targeted at users of Oracle Linux 8 or later.

systemd is the first process that starts at boot and is the final process to terminate at system shutdown. systemd is primarily used to manage system services or processes and system initialization at boot. However, systemd is also capable of handling many other tasks and functions as well including event logging, device management, user login, task scheduling, time synchronization and system boot. Many features in systemd are not fully utilized as users may be more comfortable with alternate software for these purposes or different Linux distributions may have preferred approaches to system configuration.

Different types of behaviour or functions within systemd are handled in systemd units. For example, daemon processes or system services are run as service units, while system states are usually defined as target units. Timer units can be defined to schedule tasks similarly to how you might use the system cron service and a mount unit can be used to configure a mount point similarly to how you might configure a mount point in the system fstab.

systemd is used to manage system level processes and functions, but it is also capable of managing processes running in user space. Users on a system can configure and manage their own services and systemd can even be configured to allow these services to continue running after the user has terminated their session.

Objectives

- Discover different systemd unit types
- Use systemd target units
- Learn common **systemctl** command syntax
- Create your own systemd timer unit in user space
- Configure systemd to allow user space processes to run after logout

What Do You Need?

A system with Oracle Linux installed.

Explore systemd Unit Files

> **Note:** When using the free lab environment, see Oracle Linux Lab Basics for connection and other usage instructions.

1. Open a terminal and connect via ssh to the *ol-server* instance if not already connected.

```
ssh oracle@<hostname or ip address>
```

2. Run the **systemctl** command to list all systemd units that are currently loaded by systemd:

   ```
   systemctl
   ```

   Use the Space or PgDn keys on your keyboard to page through the output.

   This command is equivalent to running:

   ```
   systemctl list-units
   ```

   The output shows all currently active configuration units that systemd is managing. In the output you should notice that there are units named with different suffixes, including units named with the '.device', '.mount','.service', '.target' and '.timer' suffixes.

   Units are active in the sense that they are started, running, mounted or plugged in, depending on their purpose. Units can be inactive in the sense that they are stopped, unmounted or disconnected. If you want to see all units regardless of whether they are active or not, you can run:

   ```
   systemctl list-units --all
   ```

   Output from these commands shows a selection of the different types of systemd units:

   - `automount`: Provides automount capabilities for on-demand mounting of file systems and parallelized boot-up.
   - `mount`: Controls mount points in the file system current date and time displays.
   - `path`: Can activate services when file system path information changes.
   - `scope`: Similar to service units but manages foreign processes instead of starting them.
   - `service`: Starts and controls daemons and the processes they consist of.
   - `slice`: Used to group units that manage system processes, such as service units and scope units, in the hierarchical cgroup tree for resource management purposes.
   - `socket`: Encapsulates local interprocess communication (IPC) or network sockets in the system, which are useful for socket-based activation.
   - `target`: Used to group units or to provide well-known synchronization points during boot-up.
   - `timer`: Used to trigger activation of other units using timers. These provide an alternative to tasks that may have been previously managed using the cron service.
   - `device`: Exposes kernel devices in systemd and can also be used to implement device-based activation.
   - `swap`: Encapsulates memory swap partitions or swap files.

3. Restrict the unit listing to a particular unit type by using the `--type` option.

    - Use the `systemctl list-units --type services` command to list the currently active service units on your system.

      ```
      systemctl list-units --type service
      ```

    - Run the same command again, but include the `--all` option to see all loaded units, including those that are inactive, if any.

      ```
      systemctl list-units --type service --all
      ```

You can repeat these commands for each of the service types that are available, so that you restrict information only to the type that you are interested in working with at any time.

Work with systemd Target Units

Target units are used to group different units together to bring the system to a particular state so that it is ready to function for a particular purpose.

1. List the available `target` units.

   ```
   systemctl list-units --type target
   ```

   Units can require other units to load or can be configured to conflict with particular units. For example, the `multi-user.target` requires the `basic.target` to function and it also conflicts with the `rescue.service` and the `rescue.target` units. Units also specify other units that they want to load to be able to function.

   In this way, targets can be chained together to set up a particular state, but are also modular enough to be reused to trigger an alternate state.

2. View the default target unit.

   The default target unit defines the default system state after boot.

    - Use the `systemctl get-default` command to view which target unit is used by default. The default target unit is represented by the `/etc/systemd/system/default.target` file.

      ```
      systemctl get-default
      ```

    - Use the `ls -l` command to list information about the `/etc/systemd/system/default.target` file.

      ```
      ls -l /etc/systemd/system/default.target
      ```

      > **Note:** The `default.target` file is a symbolic link to the current default target unit file.

3. Change the default target unit.

    - Use the `systemctl set-default` command to change the default target unit to the `graphical.target` unit.

      ```
      systemctl set-default graphical.target
      ```

    - Use the `ls -l` command to confirm that the `default.target` file is now a symbolic link to the `graphical.target` file.

      ```
      ls -l /etc/systemd/system/default.target
      ```

      > **Note:** Changing the default target unit removes the existing `default.target` symbolic link and re-creates the symbolic link, which points to the new default target unit.

4. Explore a target for more information.

    - Use the `systemctl show` command to get more information about any specific target.

      ```
      systemctl show multi-user.target
      ```

      The output shows all of the parameters for the target that you specify. Note that you can identify the units that the target requires, wants and conflicts with and that you can also see which units this target must run before and after.

    - Use the `systemctl list-dependencies` command to show the tree of dependencies that are required or wanted for a particular target to reach its state:

      ```
      systemctl list-dependencies default.target
      ```

      This command shows you all of the units that are started when you start the default target. The chain of units is presented recursively in a tree that makes it possible to fully assess what the target achieves when it starts. If you have set the `graphical.target` as the default target, you can see that the system wants to run the `display-manager.service` to load the graphical display but it also runs the `multi-user.target` to do everything required prior to running the graphical display.

Using systemctl to Enable, Disable and Mask Units

Units can be disabled or enabled and can also be masked so that they never run under any circumstance. Some units are static in that they are always available, usually because they are dependencies for other units to work. The `systemctl list-units` command can only be used to show units that are active or inactive on the system.

1. List all of the units that are available on the system, along with their state:

   ```
   systemctl list-unit-files
   ```

   Many of the units that are available are static. Enabled units start at boot time. Disabled units are units that are available on the system but which are not configured to start at boot. Masked units are available on the system but have been actively set to a state in which they cannot be started at all.

2. Use the `systemctl status` command to view detailed information about the `nfs-server.service` unit.

   ```
   systemctl status nfs-server.service
   ```

   The `systemctl` command allows you to drop the `.service` extension when referring to service units.

   The status command indicates whether a unit is enabled, active, inactive, disabled or masked.

   For scripted solutions, `systemctl` provides short commands to output status in a single line:

   - Use the `systemctl is-active` command to check if the `nfs-server` service is running (active) or not running (inactive).

     ```
     systemctl is-active nfs-server
     ```

   - Use the `systemctl is-enabled` command to check if the `nfs-server` service is enabled or disabled. With the service enabled, the service starts on a system reboot.

     ```
     systemctl is-enabled nfs-server
     ```

3. Enable a service to start at boot.

   Use the `systemctl enable` command to enable the `nfs-server` service.

   ```
   sudo systemctl enable --now nfs-server
   ```

   You must run the `systemctl` command with administrator privileges if the command changes system state or configuration. You can use the `--now` option to additionally start the service at the same time that you enable it.

   > **Note:** The command enables the service by creating a symbolic link for the lowest-level system-state target at which the service starts. In the output, the command created the symbolic link `nfs-server.service` for the `multi-user` target.

   Use the `systemctl status command` to confirm the `nfs-server` service is now enabled and running.

   ```
   systemctl status nfs-server
   ```

4. Disable and stop a service.

   Use the `systemctl disable` command to disable the `nfs-server` service. Also note that the `systemctl disable` command deletes the systemctl link for the service.

   ```
   sudo systemctl disable nfs-server
   ```

   Use the `systemctl stop` command to stop the `nfs-server` service.

   ```
   sudo systemctl stop nfs-server
   ```

   You are able to combine these steps by using the `--now` option when you disable the service.

5. Mask and unmask a unit.

   In some cases, you may wish to disable units from starting at all. Typically, you may do this if a particular unit conflicts with some other functionality on the system, or for a policy reason.

   Use the `systemctl mask` command to mask the `nfs-server` service:

   ```
   sudo systemctl mask nfs-server
   ```

   A symbolic link is created to ensure that the systemd unit configuration points to /dev/null. This prevents the service from being enabled or started.

   Confirm that you are unable to start the nfs-server unit while it is masked:

   ```
   sudo systemctl start nfs-server
   ```

   The service is unable to start and an error is returned to indicate that the service is masked.

   Unmask the unit to return it to its original state and to allow users to start or enable the service.

   ```
   sudo systemctl unmask nfs-server
   ```

Set up systemd for User Space Units

In general, systemd is used to manage units at a system level. Users require administrator level access to the system to manage systemd units that are configured in this way. In some environments and for some unit types, users may wish to use systemd's ability to run units within user space. For instance, users may wish to schedule tasks using systemd's timer unit capabilities; or users may want to run specific applications or services as service units that should not require root level permission to run.

systemd starts a systemd-user process for a user at login. Units located in the following directories are processed in the following order for the user:

- `/usr/lib/systemd/user/`: user space units provided by installed packages
- `$HOME/.local/share/systemd/user/`: user space units from packages that are installed in the home directory
- `/etc/systemd/user/`: global system-wide user units that should run in user space for all users
- `$HOME/.config/systemd/user/`: user created units

You can indicate to systemd that you are working in user space by using the `--user` option for any systemd command.

1. List currently available unit files for your user.

   ```
   systemctl --user list-unit-files
   ```

   Notice that the list of available units is significantly shorter than when you issued the same command without the `--user` option.

   The majority of these unit files, on a new system are located in `/usr/lib/systemd/user`. List the files in this directory to view the units located here:

   ```
   ls -la /usr/lib/systemd/user/
   ```

2. Create a directory to host your own systemd unit files.

   ```
   mkdir -p $HOME/.config/systemd/user
   ```

3. Create your own systemd service unit.

   ```
   cat << EOF > $HOME/.config/systemd/user/uptime.service
   [Unit]
   Description="Logs system uptime and load average"
   Wants=uptime.timer

   [Service]
   ExecStart=/usr/bin/uptime

   [Install]
   WantedBy=default.target

   EOF
   ```

   This service unit provides three configuration sections.

   The `Unit` section provides a description for the unit and any requirements. In this case, a `Wants` entry defines a weak requirement for a timer unit that does not exist yet. Units that are listed as `Wants` entries are run if they are available but do not prevent the parent unit from running if they are not found or fail to run.

   The `Service` section defines the behavior of this specific service unit when it is run. We rely on many default values for the available options here and only specify the `ExecStart`line, which specifies the command that is run when the service is started. In this case, the `uptime` command is run to log the system uptime and load values.

   The `Install` section defines how the service should be installed onto the system when it is enabled. Notably, the service is added as a service that is `WantedBy` the ‘default.target’. This would mean that the service is enabled as part of the default target for this user.

4. Run the systemd unit and check its output.

   Since you have added a new unit, it is usually a good idea to reload the systemd configuration before attempting to run the service:

   ```
   systemctl --user daemon-reload
   ```

   Now start the new unit.

   ```
   systemctl --user start uptime
   ```

   Check that the command has run as expected. You can check that the service has run by checking its status:

   ```
   systemctl --user status uptime
   ```

   > **Note:** These commands use the `--user` option to run within user space.

   To check the output from the `uptime` command that was run, use the `journalctl` command to view the log and specify the tag option to view logs specific to the command:

   ```
   journalctl -t uptime
   ```

   You can enable this service so that it starts when your user first logs into the system.

   ```
   systemctl --user enable uptime
   ```

   Note that the service runs when the user first logs into the system. It does not start automatically at system boot. Usually, services that run in user space terminate after the user logs out or all user sessions are terminated. Enabling persistence for user services is discussed later in this tutorial.

Work with systemd Timer Units

In this exercise, you build on the previous exercise to create and enable a timer unit to regularly run another systemd unit at a particular time or interval. Timer units can be defined at both the system level and the user level and can be used to define when systemd should run another unit. Timer units provide granular control over scheduled events and can act as an alternative to using the cron daemon to handle more subtle configurations.

Many system services include timer units to control when they run. A great example of a timer unit is included in the `dnf-automatic` package that can be used to keep your system up to date when it performs regular dnf updates automatically. To see this in action at a system level, install the package and enable the timer unit:

```
sudo dnf install -y dnf-automatic
sudo systemctl enable dnf-automatic.timer
```

You can view the unit file to see how this unit is configured:

```
cat /usr/lib/systemd/system/dnf-automatic.timer
```

Notable content in this unit, include a `Wants` line that expects the `network-online.target` to be met. The `OnCalendar` entry in the `Timer` section of the configuration suggests that this action runs daily at 06h00. Also of interest is the `RandomizedDelaySec` entry, which can help prevent timer units from all firing at exactly the same time and pushing up system load suddenly.

The example provided here is part of a much more complex set of units. To better understand how timer units work, add a timer unit in user space to schedule the `uptime.service` that you created in the previous exercise so that it runs at a regular interval.

1. Create a timer unit file.

   ```
   cat <<EOF > $HOME/.config/systemd/user/uptime.timer
   [Unit]
   Description=Timer for the uptime service that logs uptime
   Requires=uptime.service

   [Timer]
   Unit=uptime.service
   OnCalendar=*-*-* *:*:00

   [Install]
   WantedBy=timers.target

   EOF
   ```

   This file specifies that the `uptime.service` is required for this timer unit to run. This is a much stronger requirement than anything defined in a `Wants` definition and the unit does not run if the requirement is not satisfied.

   The `Timer` section defines that it loads the `uptime.service` unit using an `OnCalendar` entry. The `OnCalendar` entry functions similarly to the options in a crontab definition but provides more granularity. In this case, the unit is defined to run every minute at 00 seconds.

2. Since you have modified the systemd configuration, reload systemd daemons and restart the uptime service so that it can pick up the new timer unit:

   ```
   systemctl --user daemon-reload
   systemctl --user restart uptime
   ```

3. List the units to check that the `uptime.service` and `uptime.timer` units are running.

   ```
   systemctl --user list-units
   ```

4. Monitor the log output in the journal to see the uptime output triggered by the uptime service running every minute.

   ```
   journalctl -f -t uptime
   ```

   After a couple of minutes, several lines of output should have displayed. If you were paying close attention, you may notice that the uptime command does not always trigger exactly on the minute. This is an intentional feature within systemd timer functionality. Timer jobs are triggered with a randomizer that can allow a task to trigger with up to a minute in delay. This helps to prevent timers from all triggering at exactly the same time. You can force a timer to be incredibly accurate by setting the accuracy to within a nanosecond of the scheduled event, by adding the following configuration entry to the timer unit `Timer` section:

   ```
   AccuracySec=1us
   ```

   However, for most tasks, allowing for a degree of inaccuracy is sensible to prevent tasks from running too synchronously.

   You can use the Ctrl-C key combination to exit the journal when you have finished monitoring.

By default, services and processes that are started and owned by a user are terminated when the user logs out or when all sessions for the user have terminated. There are several methods that you can use to change this default behavior within systemd. Two options are explored here.

Use the loginctl command to enable systemd linger users

The **loginctl** command can be used to change the default behavior for a specific user and to enable processes for that user to 'linger' after the user's session is terminated.

1. Use the **loginctl** utility to enable linger for a specific user. In this instance, enable the systemd linger behavior for the 'oracle' user:

   ```
   sudo loginctl enable-linger oracle
   ```

2. To verify that the setting is applied, check for a file with the same name as the user in the `/var/lib/systemd/linger` directory.

   ```
   ls /var/lib/systemd/linger/oracle
   ```

   The command should verify that the file exists.

Edit the systemd logind.conf file

Systemd manages user login events and provides a configuration file that can be edited to set default behavior for different events related to the user's session. This configuration file is located at `/etc/systemd/logind.conf`.

1. Dump the contents of the existing configuration at `/etc/systemd/logind.conf` to the screen to review:

   ```
   cat /etc/systemd/logind.conf
   ```

   The majority of options are commented out but display the compile time default values. There are three options in this file that can control how systemd handles processes running in user space when the user's session terminates.

   - `KillUserProcesses`: this option can control whether or not user processes are terminated by default when the session ends. Setting this option to 'no' allows systemd to run user processes after any user logs out of the system.
   - `KillExcludeUsers`: If the `KillUserProcesses` option is enabled, this option allows you to specify a space separated list of users for which systemd allows processes to continue to run after the session terminates. Adding a username to this list behaves similarly to adding a user to the systemd linger group using the **loginctl** command.
   - `KillOnlyUsers`: If the `KillUserProcesses` option is disabled, this parameter can be used to specify a space separated list of users for which processes should be terminated after logout.

Video Demonstration

Video demonstrations on systemd are provided at https://www.youtube.com/watch?v=9uDvnZKhU8A and https://www.youtube.com/watch?v=Tkxs-wfZrnw if you need more information on working with the systemd on Oracle Linux 8.

Watch Video At: https://youtu.be/9uDvnZKhU8A

systemd System and Service Manager on Oracle Linux 8

Watch Video At: https://youtu.be/Tkxs-wfZrnw

More Information

- Systemd documentation: https://systemd.io/
- `systemd(1)` manual page
- `systemctl(1)` manual page
- `journalctl(1)` manual page
- `systemd.unit(5)` manual page
- `logind.conf(5)` manual page
- Oracle Linux 8: Managing Core System Configuration
- Oracle Linux Documentation

More Learning Resources

Explore other labs on docs.oracle.com/learn or access more free learning content on the Oracle Learning YouTube channel. Additionally, visit education.oracle.com/learning-explorer to become an Oracle Learning Explorer.

For product documentation, visit Oracle Help Center.
