# Oracle Linux 8 Managing the System With systemd





Oracle Linux 8 Managing the System With systemd,

G13059-01

Copyright © 2024, Oracle and/or its affiliates.

# Contents

D	r۵	fs	C.	Δ
		1	11.	ᆫ

Documentation License	\
Conventions	V
Documentation Accessibility	V
Access to Oracle Support for Accessibility	V
Diversity and Inclusion	V
About systemd	
systemd Configuration	1-1
systemd Units	1-1
systemd Utilities	
systemctl System State Commands	2-1
Running systemctl on a Remote System	2-1
Configuring System Date and Time Settings	2-2
Configuring System Language (Locale) and Keyboard Settings	2-4
Changing the Language Setting	2-4
Installing Language Locales Individually	2-5
Changing the Keyboard Layout	2-6
Targets	
Displaying Default and Active System-State Targets	3-1
Changing Default and Active System-State Targets	3-3
Service Management	
Starting and Stopping Services	4-1
Enabling and Disabling Services	4-1
Displaying the Status of Services	4-3
Controlling Access to System Resources	4-5
Creating a User-Based systemd Service	4-6



	Changing systemd Service Unit Files	4-7
	About Service Unit Files	4-7
	Configurable Options in Service Unit Files	4-8
5	Working with Timers	
	Using Timer Units to Control Service Unit Runtime	5-2
	Configuring a Realtime Timer Unit	5-3
	Configuring a Monotonic Timer Unit	5-4
	Running a Transient Timer Unit	5-5
6	Core Dumps	
	Enabling Core Dumps	6-1
	Configuring Core Dumps	6-1
	Analyzing Core Dumps	6-2
	Exporting Core Dumps	6-2
7	About Control Groups	
	About Control Groups and systemd	7-2
	Using systemd to Manage cgroups v2	7-4
	About Slices and Resource Allocation in systemd	7-5
	Slices, Services, and Scopes in the cgroup Hierarchy	7-7
	systemd Scopes	7-8
	Setting Resource Controller Options and Creating Custom Slices	7-9
	Using Service Unit Files	7-9
	Using Drop-in Files	7-12
	Using systemctl set-property	7-14



### **Preface**

Oracle Linux 8: Managing the System With systemd describes how to use systemd to manage core system configuration, services, timer units, and resource usage.

### **Documentation License**

The content in this document is licensed under the Creative Commons Attribution—Share Alike 4.0 (CC-BY-SA) license. In accordance with CC-BY-SA, if you distribute this content or an adaptation of it, you must provide attribution to Oracle and retain the original copyright notices.

### Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

### **Documentation Accessibility**

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

# Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

### **Diversity and Inclusion**

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and



the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.



1

# About systemd

systemd is the system initialization and service manager in Oracle Linux. The systemd daemon is the first process that starts after a system boots and is the final process that's running when the system shuts down. systemd controls the final stages of booting and prepares the system for use. It also speeds up booting by loading services concurrently.



#### Tip:

See Use systemd on Oracle Linux for a hands-on tutorial and video demonstrations on working with systemd in Oracle Linux.

For more information about system boot, see Oracle Linux 8: Managing Kernels and System Boot

# systemd Configuration

systemd reads its configuration from files in the /etc/systemd directory. For example, the /etc/systemd/system.conf file controls how systemd handles system initialization.

The systemd daemon starts services during the boot process by reading the symbolic link /etc/systemd/system/default.target. The following example shows the value of /etc/systemd/system/default.target on a system configured to boot to a multiuser mode without a graphical user interface, a target called multi-user.target:

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

/etc/systemd/system/default.target -> /usr/lib/systemd/system/multiuser.target



#### Note:

You can use a kernel boot parameter to override the default system target. See Oracle Linux 8: Managing Kernels and System Boot for information about setting kernel boot parameters.

### systemd Units

systemd organizes the different types of resources it manages into units. Most units are configured in unit configuration files that enable you to configure these units according to system needs. In addition to the files, you can also use systemd runtime commands to configure the units.

To display all the types of units available in systemd, use the following command:

```
sudo systemctl -t help

Available unit types:
service
mount
swap
socket
target
device
automount
timer
path
slice
scope
```

The following list describes some system units that you can manage on an Oracle Linux system by using systemd:

#### Services

Service unit configuration files have the file name format service\_name.service, for example sshd.service, crond.service, and httpd.service.

Service units start and control daemons and the processes of which the daemons consist. The following example shows how you might start the systemd service unit for the Apache HTTP server, httpd.service:

```
sudo systemctl start httpd.service
```

See Service Management for more information.

#### **Targets**

Target unit configuration files have the file name format *target\_name*.target, for example graphical.target.

Targets are similar to runlevels. A system reaches different targets during the boot process as resources get configured. For example, a system reaches network-pre.target before it reaches the target network-online.target.

Many target units have dependencies. For example, the activation of graphical.target (for a graphical session) fails unless multi-user.target (for multiuser system) is also active. See Targets for more information.

### **File System Mount Points**

Mount unit configuration files have the file name format <code>mount\_point\_name.mount</code>. Mount units enable you to mount file systems at boot time. For example, you can run the following command to mount the temporary file system (tmpfs) on /tmp at boot time:

```
sudo systemctl enable tmp.mount
```

### **Devices**

Device unit configuration files have the file name format *device\_unit\_name*.device. Device units are named after the /sys and /dev paths they control. For example, the device /dev/sda5 is exposed in systemd as dev-sda5.device.



Device units enable you to implement device-based activation.

#### **Sockets**

Socket unit configuration files have the file name format socket\_unit\_name.socket.

Each "\*.socket" file needs a corresponding "\*.service" file to configure the service to start on incoming traffic on the socket.

Socket units enable you to implement socket-based activation.

#### **Timers**

Timer unit configuration files have the file name format *timer\_unit\_name*.timer.

Each "\*.timer" file needs a corresponding "\*.service" file to configure the service to start at a configured timer event. A Unit configuration entry can be used to specify a service that's named differently to the timer unit. if required.

Timer units can control when service units are run and can act as an alternative to using the cron daemon. Timer units can be configured for calendar time events, monotonic time events, and can be run asynchronously.

See Working with Timers for more information.

Paths to systemd unit configuration files vary depending on their purpose and whether systemd is running in 'user' or 'system' mode. For example, configuration for units that are installed from packages might be available in /usr/lib/systemd/system or in /usr/local/lib/systemd/system, while a user mode configuration unit is likely to be stored in \$HOME/.config/systemd/user. See the systemd.unit (5) manual page for more information.



# systemd Utilities

systemd provides several command line utilities you can use to view and change the system.

Utility	Purpose	Manual Page
systemctl	Manage units and change the system state.	systemctl(1)
timedatectl	View and change time and date settings on the system.	timedatectl(1)
localectl	View and change language and keyboard settings on the system.	localectl(1)

# systemctl System State Commands

Some systemctl subcommands control the state of the system. Each of these system commands activate a related target.

For more information, see the systemctl (1) manual page.

Command	Description	Target
systemctl halt	Stop all running software, stop the kernel, and leave the hardware powered on.	halt.target
systemctl hibernate	Save the contents of system memory to disk and power off the hardware.	hibernate.target
systemctl hybrid-sleep	Save the contents of system memory to disk and leave the hardware powered on.	hybrid-sleep.target
systemctl poweroff	Halt and power off the system.	poweroff.target
systemctl reboot	Reboot the system.	reboot.target
systemctl suspend	Power off most hardware in the system while preserving power to memory.	suspend.target

# Running systemctl on a Remote System

You can run systemctl commands on a remote system where the sshd service is running. Include the -H option and the hostname with the systemctl command to control the system remotely.

For more information see the systemctl(1) manual page.

The following example shows how to check the status of the crond service on a remote system.

Run the following command: sudo systemctl -H root@10.0.0.2 status crond

The remote system returns results similar to the following:

### **Configuring System Date and Time Settings**

System time is based on the POSIX time standard, where time is measured as the number of seconds that have elapsed from 00:00:00 Coordinated Universal Time (UTC), Thursday, January 1, 1970. A day is defined as 86400 seconds and leap seconds are subtracted automatically.

Date and time representation on a system can be set to match a specific timezone. To list the available timezones, run:

```
timedatectl list-timezones
```

To set the system timezone to match a value returned from the available timezones, you can run:

```
timedatectl set-timezone America/Los Angeles
```

Substitute America/Los\_Angeles with a valid timezone entry.

This command sets a symbolic link from /etc/localtime to point to the appropriate zone information file in /usr/share/zoneinfo/. The setting takes effect immediately. Some long running processes that use /etc/localtime to detect the current system timezone might not detect a change in system timezone until the process is restarted.

Note that timezones are largely used for display purposes or to handle user input. Changing timezone doesn't change the time for the system clock. You can change the presentation for system time in any console by setting the  ${\tt TZ}$  environment variable. For example, to see the current time in Tokyo, you can run:

```
TZ="Asia/Tokyo" date
```



You can check the system's current date and time configuration by running the timedatectl command on its own:

timedatectl

```
Local time: Wed 2021-07-17 00:50:58
EDT

Universal time: Wed 2021-07-17 04:50:58
UTC

RTC time: Wed 2021-07-17
04:50:55

Time zone: America/New_York (EDT, -0400)

System clock synchronized: yes

NTP service: active
```

To set system time manually, use the timedatectl set-time command:

```
timedatectl set-time "2021-07-17 01:59:59"
```

This command sets the current system time based on the time specified assuming the currently set system timezone. The command also updates the system Real Time Clock (RTC).



### Tip:

See Learn How to Localize Your Installation on Oracle Linux for a hands-on tutorial that describes how to use tools to configure system parameters such as date, time, and locale.

Consider configuring the system to use network time synchronization for more accurate time-keeping. Using network time synchronization is important especially when setting up high-availability or when using network-based file systems.

For more information about configuring the network time services that use NTP, see Oracle Linux 8: Setting Up Networking.



### Tip:

See Configure Chrony on Oracle Linux for a hands-on tutorial on setting up and configuring the chronyd service.

If you configure an NTP service, enable NTP by running the following command:

```
timedatectl set-ntp true
```

This command enables and starts the chronyd service, if available.

### Configuring System Language (Locale) and Keyboard Settings

System-wide preferences for language and keyboard are stored in the locale configuration file (/etc/locale.conf). You can query and change these settings as needed using localectl command. Note that the systemd process reads the locale configuration file at boot and applies these settings to every system-wide service, user interface, and user profile, unless they're overridden by other programs or users. For more information about configuring these system-wide settings, see:

- Changing the Language Setting
- Changing the Keyboard Layout



### Note:

System-wide preferences for language and keyboard are also configurable during installation. For details on how to configure these settings at installation, see Oracle Linux 8: Installing Oracle Linux.

### Changing the Language Setting

The system locale language setting defines the language in which text appears in the Linux user interfaces (text-based and graphical).

To query and change the language setting on the system, follow these steps:

1. To check the current language locale set on the system, type:

```
localectl status
```

For example, the following system language locale output indicates: English (en) as the language, US as the country code, and UTF-8 as the codeset.

```
System Locale: LANG=en US.UTF-8
```

2. To list all possible language locales available on the system, type:

```
localectl list-locales
```

To search the output for a specific language locale, use the grep command. For example, to list all possible English locales available for configuration, type:

```
localectl list-locales | grep en
```



3. To set the default language locale on the system, type:

```
sudo localectl set-locale LANG=locale_name
```

#### Where:

 locale\_name is replaced with the name retrieved earlier from the list-locales output.

For example, to set British English as the system language locale, type:

sudo localectl set-locale LANG=en\_GB.utf8

### Note:

Locale options are typically listed in the following format:

LANGUAGE\_COUNTRY.CODESET[@MODIFIERS]. The LANGUAGE is an ISO 639 language code, for example, en for English and COUNTRY is an ISO 3166 country code. The two letter country code in this example is GB for Great Britain and the United Kingdom. The CODESET is the character set or encoding, for example, utf-8.

For more information on how to configure language locale options on the system, see the locale manual page.

### Installing Language Locales Individually

A langpack is a metapackage that consists of dependencies that provide support for a specified language. The dependencies include packages for locales, fonts, and other functionality for using a language on a system.

For a given language, one of the dependencies the language installs is glibc-language. <a href="locale\_code">locale\_code</a>. To reduce storage space required for languages, you can choose to install only the individual glibc locale language packages (glibc-language.

1. To list all language packs already installed on the system and all language packs available on the ol8 appstream repository, type:

```
sudo dnf list langpacks-*
```

For example, the following shows that this system has Spanish, French, Japanese, and Russian language packs installed followed by a truncated list of language packs available on oll appstream.

```
sudo dnf list langpacks-*
Last metadata expiration check: 1:37:24 ago on Thu 18 Apr 2024 08:14:05 AM
EDT.
Installed Packages
                                        1.0-12.el8
langpacks-es.noarch
@ol8 appstream
langpacks-fr.noarch
                                        1.0-12.el8
@ol8 appstream
langpacks-ja.noarch
                                       1.0-12.el8
@ol8 appstream
                                       1.0-12.el8
langpacks-ru.noarch
@ol8 appstream
Available Packages
```



langpacks-af.noarch	1.0-12.el8
ol8_appstream	
langpacks-am.noarch	1.0-12.el8
ol8_appstream	
langpacks-ar.noarch	1.0-12.el8
ol8_appstream	
• • •	

Use dnf to install a language pack. For example, the following installs the Japanese language pack:

```
sudo dnf install langpacks-ja.noarch
```

3. To list all installed and all available glibc Langpack packages, run the following command:

```
sudo dnf list glibc-langpack*
```

4. To install a glibc language pack, run the following command:

```
sudo dnf install glibc-langpack-language code
```

In the previous command, <code>language\_code</code> is the language code you want to install. For example, the following example installs Japanese.

```
sudo dnf install glibc-langpack-ja.x86 64
```

### Changing the Keyboard Layout

The keyboard layout settings enable you to specify a keymap locale for the Linux user interfaces (text-based and graphical).

To query and change the keyboard layout settings on the system, follow these steps:

1. To check the current keyboard layout configuration on the system, type:

```
localectl status
```

For example, the following keyboard layout output indicates a US country code for the virtual console keymap and a US country code for the X11 layout.

2. To list all possible keyboard layout configurations available, type:

```
localectl list-keymaps
```

To search the output for a specific keymap name, use the grep command. For example, to list British compatible keyboard layouts, type:

```
localectl list-keymaps | grep gb
```

3. To set the default keyboard layout on the system, type:

```
sudo localectl set-keymap keymap_name
```

Where:



keymap\_name is replaced with the name of the keymap retrieved earlier from the list-keymaps output.

Note that the keymap name change applies to both the virtual console and the x11 layout settings. If you want the X11 layout to differ from the virtual console keymap, use the --no-convert option, for example:

```
sudo localectl --no-convert set-x11-keymap keymap_name
```

The *no-convert* option retains the previous x11 keyboard layout setting.

For more information on how to use the localectl command line utility to change keyboard system settings, see thelocalectl manual page.



# **Targets**

By using targets, you can control systemd so that it starts only the services that are required for a specific purpose. For example, you set the default target to multi-user.target on a production server so that the graphical user interface isn't used when the system boots. In a case where you need to troubleshoot or perform diagnostics, you might consider setting the target to rescue.target, where only root logs onto the system to run the minimum number of services.

Each run level defines the services that systemd stops or starts. As an example, systemd starts network services for multi-user.target and the X Window System for graphical.target, and stops both services for rescue.target.

Table 3-1 shows the commonly used system-state targets and the equivalent runlevel targets.

Table 3-1 System-State Targets and Equivalent Runlevel Targets

System-State Targets	Equivalent Runlevel Targets	Description
graphical.target	runlevel5.target	Set up a multiuser system with networking and display manager.
multi-user.target	runlevel2.target	Set up a nongraphical
	runlevel3.target	multiuser system with networking.
	runlevel4.target	networking.
poweroff.target	runlevel0.target	Shut down and power off the system.
reboot.target	runlevel6.target	Shut down and reboot the system.
rescue.target	runlevel1.target	Set up a rescue shell.

Note that runlevel\* targets are implemented as symbolic links.

For more information, see the systemd.target(5) manual page.

# Displaying Default and Active System-State Targets

To display the default system-state target, use the systemctl get-default command:

sudo systemctl get-default

graphical.target

To display the active targets on a system, use the systemctl list-units --type target command:

sudo systemctl list-units --type target [--all]

```
UNIT basic.target loaded active active Easic System
cryptsetup.target loaded active active Local Encrypted Volumes
getty.target loaded active active Local Encrypted Volumes
graphical.target loaded active active Local File Systems (Pre)
local-fs-pre.target loaded active active Local File Systems (Pre)
local-fs.target loaded active active Local File Systems
multi-user.target loaded active active Multi-User System
network-online.target loaded active active Network is Online
network-pre.target loaded active active Network (Pre)
network.target loaded active active Network (Pre)
network.target loaded active active NFS client services
nss-user-lookup.target loaded active active User and Group Name Lookups
paths.target loaded active active Paths
remote-fs-pre.target loaded active active Remote File Systems (Pre)
remote-fs.target loaded active active Remote File Systems (Pre)
remote-fs.target loaded active active Remote File Systems
rpc_pipefs.target loaded active active Sices
sockets.target loaded active active Slices
sockets.target loaded active active Sockets
sound.target loaded active active Sound Card
sshd-keygen.target loaded active active Swap
sysinit.target loaded active active System Initialization
timers.target loaded active active System Initialization
timers.target The high-level unit activation state, i.e. generalization of SUB.
```

SUB = The low-level unit activation state, values depend on unit type.

24 loaded units listed. Pass --all to see loaded but inactive units, too.

To show all installed unit files use 'systemctl list-unit-files'.

The output for a system with the graphical target active shows that this target depends on other active targets, including network and sound to support networking and sound.

Use the --all option to include inactive targets in the list.

For more information, see the systemctl(1) and systemd.target(5) manual pages.

# Changing Default and Active System-State Targets

Use the systemctl set-default command to change the default system-state target:

sudo systemctl set-default multi-user.target

Removed /etc/systemd/system/default.target.

Created symlink /etc/systemd/system/default.target → /usr/lib/systemd/system/
multi-user.target

### Note:

This command changes the target to which the default target is linked, but doesn't change the state of the system.

To change the current active system target, use the systematl isolate command, for example:

sudo systemctl isolate multi-user.target

For more information, see the systemctl(1) manual page.



4

# Service Management

Services in an Oracle Linux system are managed by the systemctl subcommand command.

Examples of subcommands are enable, disable, stop, start, restart, reload, and status.

For more information, see the systematl(1) manual page.

# Starting and Stopping Services

To start a service, use the systemctl start command:

sudo systemctl start sshd

To stop a service, use the systematl stop command:

sudo systemctl stop sshd

Changing the state of a service only lasts while the system remains at the same state. If you stop a service and then change the system-state target to one in which the service is configured to run (for example, by rebooting the system), the service restarts. Similarly, starting a service doesn't enable the service to start following a reboot. See Enabling and Disabling Services.

# **Enabling and Disabling Services**

You can use the systemctl command to enable or disable a service from starting when the system boots, for example:

sudo systemctl enable httpd

Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.

The <code>enable</code> command activates a service by creating a symbolic link for the lowest-level system-state target at which the service starts. In the previous example, the command creates the <code>symbolic</code> link <code>httpd.service</code> for the <code>multi-user</code> target.



To start the service at the same time you enable it, include the --now option in the command. For example: sudo systemctl enable --now httpd

### Disabling a service removes the symbolic link:

```
sudo systemctl disable httpd
```

Removed /etc/systemd/system/multi-user.target.wants/httpd.service.

To check whether a service is enabled, use is-enabled subcommand as shown in the following examples:

```
\verb+sudo+ systemctl is-enabled+ \verb+ httpd+
```

disabled

sudo systemctl is-enabled sshd

enabled

After running the systemctl disable command, the service can still be started or stopped by user accounts, scripts, and other processes. However, if you need to ensure that the service might be started inadvertently, for example, by a conflicting service, then use the systemctl mask command as follows:

```
sudo systemctl mask httpd
```

```
Created symlink from '/etc/systemd/system/multi-user.target.wants/httpd.service' to '/dev/null'
```

The mask command sets the service reference to /dev/null. If you try to start a service that has been masked, you will receive an error as shown in the following example:

```
sudo systemctl start httpd
```

Failed to start httpd.service: Unit is masked.

To relink the service reference back to the matching service unit configuration file, use the systemctl unmask command:

```
sudo systemctl unmask httpd
```

For more information, see the systemctl(1) manual page.



# Displaying the Status of Services

To check whether a service is running, use the is-active subcommand. The output would either be *active*) or *inactive*, as shown in the following examples:

```
sudo systemctl is-active httpd
active
systemctl is-active sshd
inactive
```

The status subcommand provides a detailed summary of the status of a service, including a tree that displays the tasks in the control group (CGroup) that the service implements:

```
sudo systemctl status httpd
httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor
preset: disabled)
   Active: active (running) since ...
     Docs: man:httpd.service(8)
 Main PID: 11832 (httpd)
   Status: "Started, listening on: port 80"
    Tasks: 213 (limit: 26213)
   Memory: 32.5M
   CGroup: /system.slice/httpd.service
            -11832 /usr/sbin/httpd -DFOREGROUND
            ├11833 /usr/sbin/httpd -DFOREGROUND

─11834 /usr/sbin/httpd -DFOREGROUND

─11835 /usr/sbin/httpd -DFOREGROUND

           └11836 /usr/sbin/httpd -DFOREGROUND
Jul 17 00:14:32 Unknown systemd[1]: Starting The Apache HTTP Server...
Jul 17 00:14:32 Unknown httpd[11832]: Server configured, listening on: port 80
Jul 17 00:14:32 Unknown systemd[1]: Started The Apache HTTP Server.
```

A cgroup is a collection of processes that are bound together so that you can control their access to system resources. In the example, the cgroup for the httpd service is httpd.service, which is in the system slice.

Slices divide the <code>cgroups</code> on a system into different categories. To display the slice and <code>cgroup</code> hierarchy, use the <code>systemd-cgls</code> command:

```
sudo systemd-cgls
Control group /:
-.slice
-user.slice
  └user-1000.slice
    -user@1000.service
     └init.scope
        -6488 /usr/lib/systemd/systemd --user
        └6492 (sd-pam)
    ∟session-7.scope
      ⊢6484 sshd: root [priv]
      -6498 sshd: root@pts/0
      -6499 -bash
      -6524 sudo systemd-cgls
      -6526 systemd-cqls
      └-6527 less
 -init.scope
 └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 16
 -system.slice
  -rngd.service
   └1266 /sbin/rngd -f --fill-watermark=0
  -irgbalance.service
   └1247 /usr/sbin/irqbalance --foreground
   -libstoragemgmt.service
   └1201 /usr/bin/lsmd -d
   -systemd-udevd.service
   └1060 /usr/lib/systemd/systemd-udevd
   -polkit.service
   L-1241 /usr/lib/polkit-1/polkitd --no-debug
   -chronyd.service
   └1249 /usr/sbin/chronyd
   -auditd.service
    ⊢1152 /sbin/auditd
    └1154 /usr/sbin/sedispatch
   -tuned.service
   └─1382 /usr/libexec/platform-python -Es /usr/sbin/tuned -l -P
   -systemd-journald.service
   └1027 /usr/lib/systemd/systemd-journald
   -atd.service
   └1812 /usr/sbin/atd -f
   -sshd.service
   └1781 /usr/sbin/sshd
```

The system.slice contains services and other system processes. user.slice contains user processes, which run within transient cgroups called *scopes*. In the example, the processes for the user with ID 1000 are running in the scope session-7.scope under the slice / user.slice/user-1000.slice.

You can use the systemctl command to limit the CPU, I/O, memory, and other resources that are available to the processes in service and scope cgroups. See Controlling Access to System Resources.

For more information, see the systemctl(1) and systemd-cgls(1) manual pages.

### Controlling Access to System Resources

Use the systematl command to control a cgroup's access to system resources, for example:

```
sudo systemctl [--runtime] set-property httpd CPUShares=512 MemoryLimit=1G
```

CPUShare controls access to CPU resources. As the default value is 1024, a value of 512 halves the access to CPU time that the processes in the cgroup have. Similarly, MemoryLimit controls the maximum amount of memory that the cgroup can use.



You don't need to specify the .service extension to the name of a service.

If you specify the --runtime option, the setting doesn't persist across system reboots.

Alternatively, you can change the resource settings for a service under the [Service] heading in the service's configuration file in /usr/lib/systemd/system. After editing the file, make systemd reload its configuration files and then restart the service:

```
sudo systemctl daemon-reload
sudo systemctl restart service
```

You can run general commands within scopes and use systemctl to control the access that these transient cgroups have to system resources. To run a command within in a scope, use the systemd-run command:

```
sudo systemd-run --scope --unit=group name [--slice=slice name]
```

If you don't want to create the group under the default system slice, you can specify another slice or the name of a new slice. The following example runs a command named mymonitor in mymon.scope under myslice.slice:

```
sudo systemd-run --scope --unit=mymon --slice=myslice mymonitor
```

Running as unit mymon.scope.



### Note:

If you don't specify the --scope option, the control group is a created as a service rather than as a scope.

You can then use <code>systemctl</code> to control the access that a scope has to system resources in the same way as for a service. However, unlike a service, you must specify the <code>.scope</code> extension, for example:

```
sudo systemctl --runtime set-property mymon.scope CPUShares=256
```

For more information see About Control Groups and the systemctl(1), systemd-cgls(1), and systemd.resource-control(5) manual pages.

### Creating a User-Based systemd Service

In addition to the system-wide systemd files, systemd enables you to create user-based services that you can run from a user level without requiring root access and privileges. These user-based services are under user control and are configurable independent of system services.

The following are some distinguishing features of user-based systemd services:

- User-based systemd services are linked with a specific user account.
- They're created under the associated user's home directory in \$HOME/.config/systemd/user/.
- After these services are enabled, they start when the associated user logs in. This
  behavior differs from that of enabled systemd services which start when the system boots.

This feature is useful when creating podman container services. For more information about podman, see Oracle Linux: Podman User's Guide.

To create a user based service:

1. Create the service's unit file in the ~/.config/systemd/user directory, for example:

```
touch ~/.config/systemd/user/myservice.service
```

2. Open the unit file and specify the values to the options you want to use, such as Description, ExecStart, WantedBy, and so on.

For reference, see Configurable Options in Service Unit Files and the systemd.service (5) and systemd.unit (5) manual pages.

3. Enable the service to start automatically when you log in.

```
sudo systemctl --user enable myservice.service
```



### Note:

When you log out, the service is stopped unless the root user has enabled processes to continue to run for the user.

See Use systemd on Oracle Linux for more information.

4. Start the service.

```
sudo systemctl --user start myservice.service
```

Verify that the service is running.

sudo systemctl --user status myservice.service

### Changing systemd Service Unit Files

To change the configuration of systemd services, copy the files with .service, .target, .mount and .socket extensions from /usr/lib/systemd/system to /etc/systemd/system.

After you have copied the files, you can edit the versions in <code>/etc/systemd/system</code>. The files in <code>/etc/systemd/system</code> take precedence over the versions in <code>/usr/lib/systemd/systemd/system</code>. Files in <code>/etc/systemd/system</code> aren't overwritten when you update a package that touches files in <code>/usr/lib/systemd/system</code>.

To revert to the default systemd configuration for a particular service, you can either rename or delete the copies in /etc/systemd/system.

Another approach for changing the configuration of a service is to create a drop-in file. With this approach, you can preserve the original unit while changing specific parameters of the unit.

Create drop-in files in /etc/systemd/system/unit\_name.d/, where the unit\_name.d directory is an existing unit, then give the drop-in files a .conf file extension. For example:  $/etc/systemd/system/unit_name.d/name_of_drop-in.conf.$  systemd reads the .conf file and applies the settings to the original unit.

The following sections describe the different parts of a service unit file that you can edit and customize for a system.

### **About Service Unit Files**

Services run based on their corresponding service unit files. A service unit file typically contains the following sections, with each section having its respective defined options that determine how a specific service runs:

### [Unit]

Contains information about the service.

### [UnitType]:

Contains options that are specific to the unit type of the file. For example, in a service unit file this section is titled [Service] and contains options that are specific to units of the service type, such as ExecStart or StandardOutput.



Only those unit types that offer options specific to their type have such a section.

#### [Install]

Contains installation information for the specific unit. The information in this section is used by the systemctl enable and systemctl disable commands.

A service unit file might contain the following configurations for a service.

```
[Unit]
Description=A test service used to develop a service unit file template
[Service]
Type=simple
StandardOutput=journal
ExecStart=/usr/lib/systemd/helloworld.sh

[Install]
WantedBy=default.target
```

Configurable Options in Service Unit Files describes some commonly used configured options available under each section. A complete list is also available in the systemd.service(5) and systemd.unit(5) manual pages.

### Configurable Options in Service Unit Files

Each of the following lists deals with a separate section of the service unit file.

### **Description of Options Under [Unit] Section**

The following list provides a general overview of the commonly used configurable options available in the [Unit] section of service unit file:

#### Description

Provides information about the service. The information is displayed when you run the systemctl status command on the unit.

#### Documentation

Contains a space-separated list of URIs referencing documentation for this unit or its configuration.

### After

Configures the unit to only run after the units listed in the option finish starting up. In the following example, if the file *var3*.service has the following entry, then it's only started after units *var1*.service and *var2*.service have started:

```
After=var1.service var2.service
```

### Requires

Configures a unit to have requirement dependencies on other units. If a unit is activated, those listed in its Requires option are also activated.

#### Wants

A less stringent version of the Requires option. For example, a specific unit can be activated even if one of those listed in its Wants option fails to start.



### **Description of Options Under [Service] Section**

This following list gives a general overview of the commonly used configurable options available in the [Service] section of a service unit file.

#### Type

Configures the process start-up type for the service unit.

By default, this parameter's value is simple, which indicates that the service's main process is that which is started by the ExecStart parameter.

Typically, if a service's type is simple, then the definition can be omitted from the file.

### StandardOutput

Configures the how the service's events are logged. For example, consider a service unit file has the following entry:

StandardOutput=journal

In the example, the value <code>journal</code> indicates that the events are recorded in the journal, which can be viewed by using the <code>journalctl</code> command.

#### ExecStart

Specifies the full path and command that starts the service, for example, /usr/bin/npm start.

#### ExecStop

Specifies the commands to run to stop the service started through ExecStart.

#### ExecReload

Specifies the commands to run to trigger a configuration reload in the service.

### Restart

Configures whether the service is to be restarted when the service process exits, is stopped, or when a timeout is reached.



This option doesn't apply when the process is stopped cleanly by a systemd operation, for example a systemctl stop or systemctl restart. In these cases, the service isn't restarted by this configuration option.

#### RemainAfterExit

A Boolean value that configures whether the service is to be considered active even when all of its processes have exited. The default value is no.

### **Description of Options Under [Install] Section**

This following list gives a general overview of the commonly used configurable options available in the [Install] section of service unit file.

#### Alias

A space-separated list of names for a unit.

At installation time, systemctl enable creates symlinks from these names to the unit filename.



Aliases are only effective when the unit is enabled.

#### RequiredBy

Configures the service to be required by other units.

For example, consider a unit file var1.service that has the following configuration added to it:

RequiredBy=var2.service var3.service

When *var1*.service is enabled, both *var2*.service and *var3*.service are granted a Requires dependency upon *var1*.service. This dependency is defined by a symbolic link that's created in the .requires folder of each dependent service (*var2*.service and *var3*.service) that points to the *var1*.service system unit file.

### WantedBy

Specifies a list of units that are to be granted a wants dependency upon the service whose file you're editing.

For example, consider a unit file var1.service that has the following configuration added to it:

WantedBy=var2.service var3.service

When var1. service is enabled, both var2. service and var3. service are granted a Wants dependency upon var1. service. This dependency is defined by a symbolic link that's created in the ".wants" folder of each dependent service (var2. service and var3. service) that points to the system unit file for var1. service.

#### Also

Lists additional units to install or remove when the unit is installed or removed.

### DefaultInstance

The DefaultInstance option applies to template unit files only.

Template unit files enable the creation of multiple units from a single configuration file. The <code>DefaultInstance</code> option specifies the instance for which the unit is enabled if the template is enabled without any explicitly set instance.



5

# **Working with Timers**

Timer unit files are a type of systemd file that the systemctl utility uses to schedule tasks, similar to the cron utility that uses crontab and other cron jobs for the same purpose. Note that the cron daemon runs as a service within systemd, so timer units are preferred because they remove a layer of added processing and offer much more utility and more granular configuration than is available in the cron service.

Typically, packages that use specific services to function in the system include their own systemd timer unit files. Thus, when these packages are installed with Oracle Linux, the timer unit files are automatically included. You can display with the timer files in the system with the following command:

systemctl list-unit-files --type=timer



The list of timer files might differ depending on where Oracle Linux is running, such as in an instance in Oracle Cloud Infrastructure, a physical system, and so on.

Each timer unit file contains parameter settings that manage the schedule of a task. For example, the schedule for running dnf-makecache.service is set in the dnf-makecache.timer file. The file contains the following settings:

```
systemctl cat dnf-makecache.timer
```

```
# /usr/lib/systemd/system/dnf-makecache.timer
[Unit]
Description=dnf makecache --timer
ConditionKernelCommandLine=!rd.live.image
# See comment in dnf-makecache.service
ConditionPathExists=!/run/ostree-booted
Wants=network-online.target
```

[Timer]
OnBootSec=10min
OnUnitInactiveSec=1h
RandomizedDelaySec=60m
Unit=dnf-makecache.service
[Install]
WantedBy=timers.target

The schedule information is specified under the [Timer] section. In the sample configuration, the dnf-makecache.service service is set to automatically run 10 minutes after the system is

booted. The service then goes into idle mode for an hour, as specified by the <code>OnUnitInactiveSec</code> parameter. At the end of the hour, the service runs again. This cycle continues every hour indefinitely.

The RandomizedDelaySec setting provides a value limit for how much a run can be delayed beyond its schedule. In the example, the service is allowed to run one minute later than its schedule at the latest. This parameter is useful for preventing too many jobs that start at the same time on a specified schedule, which would otherwise risk overloading the resources.

OnCalendar is another useful parameter for task scheduling. Suppose that the parameter is set as follows:

OnCalendar=\*:00/10

The  $\star$ : 00 indicates every hour at the top of the hour, while the /10 setting indicates 10 minutes. Therefore, the job is set to run hourly, at ten minutes past the top of the hour.

For a complete list of systemd timer unit file parameters for scheduling a job, see the systemd.timer(5) manual pages.



#### Tip:

For a tutorial on how to use systemd in Oracle Linux, including how to configure systemd timer unit files, see Use systemd on Oracle Linux.

### Using Timer Units to Control Service Unit Runtime

Timer units can be configured to control when service units run. You can use timer units instead of configuring the <code>cron</code> daemon for time-based events. Timer units can be more complicated to configure than creating a crontab entry. However, timer units are more configurable and the services that they control can be configured for better logging and deeper integration with <code>systemd</code> architecture.

Timer units are started, enabled, and stopped similarly to service units. For example, to enable and start a timer unit immediately, type:

```
sudo systemctl enable --now myscript.timer
```

To list all existing timers on the system, to see when they last ran, and when they're next configured to run, type:

```
systemctl list-timers
```

For more information about system timers, see the systemd.timer(5) and systemd.time(7) manual pages.



# Configuring a Realtime Timer Unit

**Realtime timers** activate on a calendar event, similar to events in a crontab. The option OnCalendar specifies when the timer runs a service.

If needed, create a .service file that defines the service to be triggered by the timer unit.
 In the following procedure, the sample service is /etc/systemd/system/
 update.service which is a service unit that runs an update script.

For more information about creating service units, see Creating a User-Based systemd Service.

• Decide the time and frequency for running the service. In this procedure, the timer is configured to run the service every 2 hours from Monday to Friday.

This task shows you how to create a system timer to trigger a service to run based on a calendar event. The definition of the calendar event is similar to entries that you put in a cron job.

Create the /etc/systemd/system/update.timer with the following content:

```
[Unit]
Description="Run the update.service every two hours from Mon to Fri."

[Timer]
OnCalendar=Mon..Fri 00/2
Unit=update.service

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

Defining OnCalendar can vary from a simple wetting such as OnCalendar=weekly definitions that are more detailed. However, the format for defining settings is constant, as follows:

```
DayofWeek Year-Month-Day Hour:Minute:Second
```

The following definition means "the first 4 days of each month at 12:00 o'clock noon, but only if that day is either a Monday or a Tuesday":

```
OnCalendar=Mon, Tue *-*-01..04 12:00:00
```

For other ways to define OnCalendar and for more timer options that you can configure in the system timer file, see the systemd.timer(5) and systemd.time(7) manual pages.

2. Check that all the files related to this timer are configured correctly.

```
systemd-analyze verify /etc/systemd/system/update.*
```

Any detected errors are reported on the screen.

Start the timer.

```
sudo systemctl start update.timer
```



This command starts the timer for the current session only.

4. Ensure that the timer starts when the system is booted.

```
sudo systemctl enable update.timer
```

### Configuring a Monotonic Timer Unit

**Monotonic timers** that activate after a time span relative to a varying starting point, such as a boot event, or when a particular systemd unit becomes active. These timer units stop if the computer is temporarily suspended or shut down. Monotonic timers are configured by using the OnTypeSec option, where Type is the name of the event to which the timer is related. Common monotonic timers include OnBootSec and OnUnitActiveSec.

• If needed, create a .service file that defines the service to be triggered by the timer unit. In the following procedure, the sample service is /etc/systemd/system/update.service which is a service unit that runs an update script.

For more information about creating service units, see Creating a User-Based systemd Service.

Decide the time and frequency for running the service. In this procedure, the timer is configured to run the service 10 minutes after a system boot, and every 2 hours from when the service is last activated.

This task shows you how to create a system timer to trigger a service to run at specific events, which are when the system boots or after 2 hours have lapsed from the timer's activation.

Create the /etc/systemd/system/update.timer with the following content:

```
[Unit]
Description="Run the update.service every two hours from Mon to Fri."

[Timer]
OnBootSec=10min
OnUnitActiveSec=2h
Unit=update.service

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

For more timer options that you can configure in the system timer, see the systemd.timer(5) and systemd.time(7) manual pages.

2. Check that all the files related to this timer are configured correctly.

```
systemd-analyze verify /etc/systemd/system/update.*
```

Any detected errors are reported on the screen.

3. Start the timer.

```
sudo systemctl start update.timer
```

This command starts the timer for the current session only.

**4.** Ensure that the timer starts when the system is booted.

```
sudo systemctl enable update.timer
```

### Running a Transient Timer Unit

Transient timers are temporary timers that are valid only for the current session. These timers can be created to run a program or script directly without requiring service or timer units to be configured within systemd. These units are generated by using the systemd-run command. See the systemd-run (1) manual page for more information.

The parameter options that you would add to the *unit-file*.timer file also serve as arguments when you use systemd-run command to run a transient timer unit.

The following examples show how to use systemd-run to activate transient timers.

• Run update.service after 2 hours have elapsed.

```
sudo systemd-run --on-active="2h" --unit update.service
```

Create ~/tmp/myfile after 1 hour.

```
sudo systemd-run --on-active="1h" /bin/touch ~/tmp/myfile
```

• Run ~/myscripts/update.sh 5 minutes after the service manager is started. Use this syntax to run a service after the service manager has started at user login.

```
sudo systemd-run --on-startup="5m" ~/myscripts/update.sh
```

Run myjob.service 10 minutes after system boot.

```
sudo systemd-run --on-boot="10m" --unit myjob.service
```

Run report.service at the end of the day.

```
sudo systemd-run --on-calendar="17:00:00"
```



6

# **Core Dumps**

Core dumps contain crash information for userspace applications and services running on Oracle Linux. They can be generated on demand by using a debugger, or the systemd-coredump service can be configured to generate them automatically in the event of a process stopping prematurely.

Core dumps contain a log summary of the crash event that typically includes the process ID, owner, termination signal, and a stack trace. For more information, see the systemd-coredump (8) manual pages.

The coredumpctl command can be used to review core dumps that have been written to the system journal or saved as a file. For more information, see the coredumpctl(1) manual pages.

### **Enabling Core Dumps**

Core dumps aren't enabled by default, so you must configure Systemd to generate them.

 Create the /etc/systemd/system.conf.d/10-enable-coredumps.conf configuration file and add the following content:

```
DumpCore=yes
DefaultLimitCORE=infinity
```

2. Restart the systemd daemon to apply the change without restarting Oracle Linux:

```
sudo systemctl daemon-reload
```

# **Configuring Core Dumps**

1. To adjust the scope of the data captured in Systemd core dumps and define where Systemd stores them, change the /etc/systemd/coredump.conf configuration file.

For more information, see the coredump.conf(5) manual pages.

2. Before running the coredumpctl command, remove any core dump size limits that apply to the current shell session:

```
sudo ulimit -c unlimited
```

For more information about the ulimit command, see the ulimit (1) manual pages.

# **Analyzing Core Dumps**

Use the coredumpct1 command to list the core dumps that are available on the system:

```
coredumpctl list
```

 To review more information about the core dumps stored for a particular application, specify the executable as an option:

```
coredumpctl list executable-path
```

 To review all the core dumps that are stored for a failed process on the system, specify the process ID instead:

```
coredumpctl list process-id
```

### **Exporting Core Dumps**

1. To export the core dump for bug reporting purposes, specify the process ID and output file when you run the coredumpctl dump command:

```
coredumpctl dump process-id -o output-file
```

- Optionally, you can export an SOS report with extra information about the system. For more information, see #unique\_41.
- 3. On the same system or a different one, install the gdb package and then step through a core dump with the GNU Debugger by using the coredumpctl debug command:

```
sudo dnf install gdb

coredumpctl debug process-id
```

For more information about the coredumpctl command, see the coredumpctl(1) manual pages.



# **About Control Groups**

Control groups, usually referred to as cgroups, are an Oracle Linux kernel feature that enables processes (PIDs) to be organized into hierarchical groups for the purpose of resource allocation. For example, if you have identified 3 sets of processes that need to be allocated CPU time in a ratio of 150:100:50, you can create 3 cgroups, each with a CPU weight corresponding to one of the 3 values in your ratio, and then assign the appropriate processes to each cgroup.

By default, systemd creates a cgroup for the following:

- Each systemd service set up on the host.
  - For example, a server might have control group <code>NetworkManager.service</code> to group processes owned by the <code>NetworkManager.service</code>, and control group <code>firewalld.service</code> to group processes owned by the <code>firewalld.service</code>, and so on.
- Each user (UID) on the host.

The cgroup functionality is mounted as a virtual file system under /sys/fs/cgroup. Each cgroup has a corresponding folder within /sys/fs/cgroup file system. For example, the cgroups created by systemd for the services it manages can be seen by running the command ls -l /sys/fs/cgroup/system.slice | grep ".service" as shown in the following sample code block:

```
ls -l /sys/fs/cgroup/system.slice | grep ".service"
...root root 0 Mar 22 10:47 atd.service
...root root 0 Mar 22 10:47 auditd.service
...root root 0 Mar 22 10:47 chronyd.service
...root root 0 Mar 22 10:47 crond.service
...root root 0 Mar 22 10:47 dbus-broker.service
...root root 0 Mar 22 10:47 dtprobed.service
...root root 0 Mar 22 10:47 firewalld.service
...root root 0 Mar 22 10:47 httpd.service
...root root 0 Mar 22 10:47 httpd.service
...root root 0 Mar 22 10:47 httpd.service
```

You can also create <code>cgroups</code> of your own by creating your own folders under the <code>/sys/fs/cgroup</code> virtual file system and assigning process IDs (PIDs) to different <code>cgroups</code> according to your system needs. However, the recommended practice is to use <code>systemd</code> to configure <code>cgroups</code> instead of creating the <code>cgroups</code> manually under <code>/sys/fs/cgroup</code>. See Using systemd to Manage <code>cgroups v2</code> for the recommended method of managing <code>cgroups</code> through <code>systemd</code>.

### Note:

### Use systemd to configure cgroups.

Although the recommended method for configuring using systemd to manage cgroups, this topic also covers the manual creation of cgroup folders in the /sys/fs/cgroup file system. However, this coverage is mainly to provide background knowledge of the kernel cgroup feature to which systemd provides access.

Oracle Linux provides two types of control groups:

### Control groups version 1 (cgroups v1)

These groups provide a per-resource controller hierarchy. Each resource, such as CPU, memory, I/O, and so on, has its own control group hierarchy. A disadvantage of this group is the difficulty of establishing proper coordination of resource use among groups that might belong to different process hierarchies.

### Control groups version 2 (cgroups v2)

These groups provide a single control group hierarchy against which all resource controllers are mounted. In this hierarchy, you can obtain better proper coordination of resource uses across different resource controllers. This version is an improvement over cgroups v1 whose over flexibility prevented proper coordination of resource use among the system consumers.

Both versions are present in Oracle Linux. However, by default, the <code>cgroups v1</code> functionality is enabled and mounted on Oracle Linux 8 systems.

For more information about control groups of both versions, see the cgroups (7) and sysfs (5) manual pages.

# About Control Groups and systemd

Control groups can be used by the systemd system and service manager for resource management. Systemd uses these groups to organize units and services that consume resources. For more information about systemd, see About systemd.

Systemd provides different unit types, three of which are for resource control purposes:

- **Service**: A process or a group of processes whose settings are based on a unit configuration file. Services encompass specified processes in a "collection" so that systemd can start or stop the processes as one set. Service names follow the format name.service.
- **Scope**: A group of externally created processes, such as user sessions, containers, virtual machines, and so on. Similar to services, scopes encapsulate these created processes and are started or stopped by the arbitrary processes and then registered by systemd at runtime. Scope names follow the format name.scope.
- Slice: A group of hierarchically organized units in which services and scopes are located. Thus, slices themselves don't contain processes. Rather, the scopes and services in a slice define the processes. Every name of a slice unit corresponds to the path to a location in the hierarchy. Root slices, typically user.slice for all user-based processes and system.slice for system-based processes, are automatically created in the hierarchy. Parent slices exist immediately below the root slice and follow the format parent-name.slice. These root slices can then have subslices on multiple levels.



The service, the scope, and the slice units directly map to objects in the control group hierarchy. When these units are activated, they map directly to control group paths that are built from the unit names. To display the mapping between the <code>systemd</code> resource unit types and control groups, type:

sudo systemd-cgls

```
Working directory /sys/fs/cgroup:
-user.slice (#1243)
 → trusted.invocation id: 50ce3909b2644f919ee420adc39edb4b
  -user-1001.slice (#4167)
   → trusted.invocation id: 02e80a960d4549a7a9c69ce0fb546c26
    -session-2.scope (#4405)
      -2417 sshd: alice [priv]
      -2430 sshd: alice@pts/0
      -2431 -bash
      ├2689 sudo systemd-cgls
      -2691 systemd-cgls
     └-2692 less
    _user@984.service ... (#3827)
     → trusted.delegate: 1
     → trusted.invocation id: 09b47ce9f3124239b75814114353f3f2
      └init.scope (#3861)
        -2058 /usr/lib/systemd/systemd --user
        └2099 (sd-pam)
 -init.scope (#19)
  └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
 -system.slice (#53)
  -chronyd.service (#2467)
   → trusted.invocation id: c0f77aaa9c7844e6bef6a6898ae4dd56
   └1358 /usr/sbin/chronyd -F 2
  -auditd.service (#2331)
   → trusted.invocation id: 756808add6a348609316c9e8c1801846
   └1310 /sbin/auditd
   -tuned.service (#3079)
   → trusted.invocation id: 2c358135fc46464d862b05550338d4f4
   └─1415 /usr/bin/python3 -Es /usr/sbin/tuned -l -P
   -systemd-journald.service (#1651)
   → trusted.invocation id: 7cb7ccb14e044a899aadf47bbb583ada
    └─977 /usr/lib/systemd/systemd-journald
   -atd.service (#3623)
   → trusted.invocation id: 597a7a4e5646468db407801b8562d869
   └1915 /usr/sbin/atd -f
  \vdashsshd.service (#3419)
   → trusted.invocation id: 490504a683fc4311ab0fbeb0864a1a34
   └-1871 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
```

For an example of how to use systemd commands such as systemctl to manage resources, see Controlling Access to System Resources. For further technical details, see the systemctl(1), systemd-cgls(1), and systemd.resource-control(5) manual pages.

# Using systemd to Manage cgroups v2

The preferred method of managing resource allocation with  $\tt cgroups\ v2$  is to use the control group functionality provided by  $\tt systemd$ .



For information on enabling groups v2 functionality on the system, see Oracle Linux 8: Managing Kernels and System Boot

By default, systemd creates a cgroup folder for each systemd service set up on the host. systemd names these folders using the format servicename.service, where servicename is the name of the service associated with the folder.

To see a list of the <code>cgroup</code> folders <code>systemd</code> creates for the services, run the <code>ls</code> command on the <code>system.slice</code> branch of the <code>cgroup</code> file system as shown in the following sample code block:

```
ls /sys/fs/cgroup/system.slice/
```

• • •	• • •	• • •
<pre>app_service1.service</pre>	cgroup.subtree_control	httpd.service
<pre>app_service2.service</pre>	chronyd.service	
	crond.service	
cgroup.controllers	dbus-broker.service	
cgroup.events	dtprobed.service	
cgroup.freeze	firewalld.service	
	gssproxy.service	

In the preceding command block:

• The folders *app\_service1*.service and *app\_service2*.service represent custom application services you might have on your system.

In addition to service control groups, systemd also creates a cgroup folder for each user on the host. To see the cgroups created for each user you can run the 1s command on the user.slice branch of the cgroup file system as shown in the following sample code block:

### ls /sys/fs/cgroup/user.slice/

cgroup.controllers	cgroup.subtree_control	user-1001.slice
cgroup.events	cgroup.threads	user-982.slice
cgroup.freeze	cgroup.type	
• • •	• • •	• • •
• • •	•••	• • •

In the preceding code block:

• Each user cgroup folder is named using the format user-*UID*.slice. So, control group user-1001.slice is for a user whose UID is 1001, for example.

systemd provides high-level access to the cgroups and kernel resource controller features so you do not have to access the file system directly. For example, to set the CPU weight of a

**service called** *app\_service1*.service, **you might choose to run the** systemctl set-property **command as follows**:

sudo systemctl set-property app service1.service CPUWeight=150

Thus, systemd enables you to manage resource distribution at an application level, rather than the process PID level used when configuring cgroups without using systemd functionality.

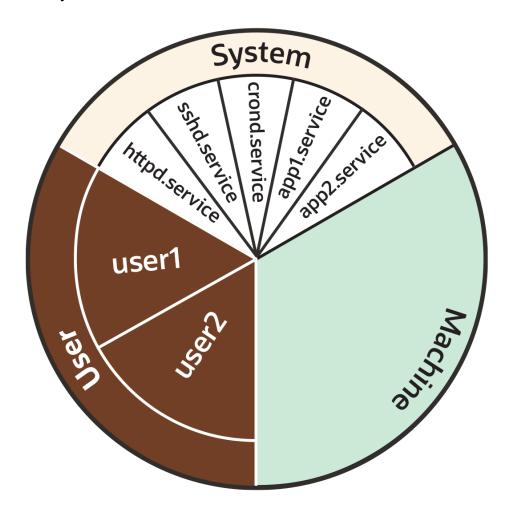
## About Slices and Resource Allocation in systemd

This section looks at the way systemd initially divides each of the default kernel controllers, for example CPU, memory and blkio, into portions called "slices" as illustrated by the following example pie chart:



You can also create your own custom slices for resource distribution, as shown in section Setting Resource Controller Options and Creating Custom Slices.

Figure 7-1 Pie chart illustrating distribution in a resource controller, such as CPU or Memory



As the preceding pie chart shows, by default each resource controller is divided equally between the following 3 slices:

- System (system.slice).
- User (user.slice).
- Machine (machine.slice).

The following list looks at each slice more closely. For the purposes of discussion, the examples in the list focus on the CPU controller.

#### System (system.slice)

This resource slice is used for managing resource allocation amongst daemons and service units.

As shown in the preceding example pie chart, the system slice is divided into further subslices. For example, in the case of CPU resources, we might have sub-slice allocations within the system slice that include the following:

- httpd.service (CPUWeight=100)
- sshd.service (CPUWeight =100)
- crond.service (CPUWeight =100)
- app1.service (CPUWeight =100)
- app2.service (CPUWeight =100)

In the preceding list, *app1*.service and *app2*.service represent custom application services you might have running on your system.

### User (user.slice)

This resource slice is used for managing resource allocation amongst user sessions. A single slice is created for each <code>UID</code> irrespective of how many logins the associated user has active on the server. Continuing with our pie chart example, the sub-slices might be as follows:

- user1 (CPUWeight=100, UID=982)
- *user2* (CPUWeight=100, UID=1001)

### Machine (machine.slice)

This slice of the resource is used for managing resource allocation amongst hosted virtual machines, such as KVM guests, and Linux Containers. The machine slice is only present on a server if the server is hosting virtual machines or Linux Containers.



#### Share allocations do not set a maximum limit for a resource.

For instance, in the preceding examples, the slice user.slice has 2 users: user1 and user2. Each user is allocated an equal share of the CPU resource available to the parent user.slice. However, if the processes associated with user1 are idle, and do not require any CPU resource, then its CPU share is available for allocation to user2 if needed. In such a situation, user2 might even be allocated the entire CPU resource apportioned to the parent user.slice if it is required by other users.

To cap CPU resource, you would need to set the CPUQuota property to the required percentage.

### Slices, Services, and Scopes in the cgroup Hierarchy

The pie chart analogy used in the preceding sections is a helpful way to conceptualize the division of resources into slices. However, in terms of structural organization, the control groups are arranged in a hierarchy. You can view the systemd control group hierarchy on your system by running the systemd-cgls command as follows:



### Tip:

To see the entire <code>cgroup</code> hierarchy, starting from the root slice <code>-.slice</code>, as in the following example, ensure you run <code>systemd-cgls</code> from outside of the control group mount point <code>/sys/fs/cgroup/</code>. Otherwise, If you run the command from within <code>/sys/fs/cgroup/</code>, the output starts from the <code>cgroup</code> location from which the command was run. See <code>systemd-cgls(1)</code> for more information.

```
systemd-cgls
Control group /:
-.slice
 -user.slice (#1429)
 → user.invocation id: 604cf5ef07fa4bb4bb86993bb5ec15e0
  —user-982.slice (#4131)
   → user.invocation id: 9d0d94d7b8a54bcea2498048911136c8
    -session-cl.scope (#4437)
    -2416 /usr/bin/sudo -u ocarun /usr/libexec/oracle-cloud-agent/plugins/
runcommand/runcommand
     -2494 /usr/libexec/oracle-cloud-agent/plugins/runcommand/runcommand
    └user@982.service ... (#4199)
     → user.delegate: 1
      → user.invocation id: 37c7aed7aa6e4874980b79616acf0c82
      └init.scope (#4233)
        -2437 /usr/lib/systemd/systemd --user
        └-2445 (sd-pam)
   -user-1001.slice (#7225)
    → user.invocation id: ce93ad5f5299407e9477964494df63b7
     -session-2.scope (#7463)
       -20304 sshd: oracle [priv]
       -20404 sshd: oracle@pts/0
      -20405 -bash
      -20441 systemd-cgls
      └20442 less
     -user@1001.service ... (#7293)
     → user.delegate: 1
      → user.invocation id: 70284db060c1476db5f3633e5fda7fba
      └init.scope (#7327)
        -20395 /usr/lib/systemd/systemd --user
        └20397 (sd-pam)
 -init.scope (#19)
  └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 28
```

```
└system.slice (#53)
  -dbus-broker.service (#2737)
   → user.invocation id: 2bbe054a2c4d49809b16cb9c6552d5a6
    ├-1450 /usr/bin/dbus-broker-launch --scope system --audit
    L1457 dbus-broker --log 4 --controller 9 --machine-id
852951209c274cfea35a953ad2964622 --max-bytes 536870912 --max-fds 4096 --max-
matches 131072 -- audit
  -chronyd.service (#2805)
   → user.invocation id: e264f67ad6114ad5afbe7929142faa4b
   └1482 /usr/sbin/chronyd -F 2
   -auditd.service (#2601)
   → user.invocation id: f7a8286921734949b73849b4642e3277
    ├1421 /sbin/auditd
    └1423 /usr/sbin/sedispatch
   -tuned.service (#3349)
   → user.invocation id: fec7f73678754ed687e3910017886c5e
   └─1564 /usr/bin/python3 -Es /usr/sbin/tuned -l -P
   -systemd-journald.service (#1837)
   → user.invocation id: bf7fb22ba12f44afab3054aab661aedb
    └1068 /usr/lib/systemd/systemd-journald
   -atd.service (#3961)
   → user.invocation id: 1c59679265ab492482bfdc9c02f5eec5
    └2146 /usr/sbin/atd -f
   -sshd.service (#3757)
   → user.invocation id: 57e195491341431298db233e998fb180
   └2097 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
   -crond.service (#3995)
   → user.invocation id: 4f5b380a53db4de5adcf23f35d638ff5
   └2150 /usr/sbin/crond -n
```

The preceding sample output shows how all "\*.slice" control groups reside under the root slice -.slice. Beneath the root slice you can see the user.slice and system.slice control groups, each with their own child cgroup sub-slices.

Examining the systemd-cgls command output you can see how, with the exception of root -.slice, all processes are on leaf nodes. This arrangement is enforced by cgroups v2, in a rule called the "no internal processes" rule. See cgroups (7) for more information about the "no internal processes" rule.

The output in the preceding systemd-cgls command example also shows how slices can have descendent child control groups that are systemd scopes. systemd scopes are reviewed in the following section.

### systemd Scopes

systemd scope is a systemd unit type that groups together system service worker processes that have been launched independently of systemd. The scope units are transient cgroups created programmatically using the bus interfaces of systemd.

For example, in the following sample code, the user with UID 1001 has run the systemd-cgls command, and the output shows session-2.scope has been created for processes the user

has spawned independently of systemd (including the process for the command itself , 21380 sudo systemd-cgls):



In the following example, the command has been run from within the control group mount point /sys/fs/cgroup/. Hence, instead of the root slice, the output starts from the cgroup location from which the command was run.

```
sudo systemd-cgls
Working directory /sys/fs/cgroup:
-user.slice (#1429)
 → user.invocation id: 604cf5ef07fa4bb4bb86993bb5ec15e0
 → trusted.invocation id: 604cf5ef07fa4bb4bb86993bb5ec15e0
  _user-1001.slice (#7225)
   → user.invocation id: ce93ad5f5299407e9477964494df63b7
   → trusted.invocation id: ce93ad5f5299407e9477964494df63b7
     -session-2.scope (#7463)
      —20304 sshd: oracle [priv]
      -20404 sshd: oracle@pts/0
      -20405 -bash
      -21380 sudo systemd-cgls
      -21382 systemd-cgls
      └21383 less
     -user@1001.service ... (#7293)
     → user.delegate: 1
     → trusted.delegate: 1
      → user.invocation id: 70284db060c1476db5f3633e5fda7fba
      → trusted.invocation id: 70284db060c1476db5f3633e5fda7fba
      └init.scope (#7327)
         -20395 /usr/lib/systemd/systemd --user
        └20397 (sd-pam)
```

### Setting Resource Controller Options and Creating Custom Slices

systemd provides the following methods for setting resource controller options, such as CPUWeight, CPUQuota, and so on, to customize resource allocation on your system:

- Using service unit files.
- Using drop-in files.
- Using the systematl set-property command.

The following sections provide example procedures for using each of these methods to configure resources and slices in your system.

# **Using Service Unit Files**

To set options in a service unit file, perform the following steps:

1. Create file /etc/systemd/system/myservice1.service with the following content:

```
[Service]
Type=oneshot
ExecStart=/usr/lib/systemd/generate_load.sh
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes
[Install]
WantedBy=multi-user.target
```

2. The service created in the preceding step requires a bash script /usr/lib/systemd/generate load.sh. Create the file with the following content:

```
#!/bin/bash
for i in {1..4};do while : ; do : ; done & done
```

3. Make the script runnable:

```
sudo chmod +x /usr/lib/systemd/generate load.sh
```

4. Enable and start the service:

```
sudo systemctl enable myservice1 --now
```

**5.** Run the systemd-cgls command and confirm the service myservice1 is running under system.slice:

```
Control group /:
-.slice
...
|-user.slice (#1429)
...
-system.slice (#53)
...
|-myservice1.service (#7939)
| - user.invocation_id: e227f8f288444fed92a976d391e6a897
| -22325 /bin/bash /usr/lib/systemd/generate_load.sh
| -22326 /bin/bash /usr/lib/systemd/generate_load.sh
| -22327 /bin/bash /usr/lib/systemd/generate_load.sh
| -22328 /bin/bash /usr/lib/systemd/generate_load.sh
| -pmie.service (#4369)
| - user.invocation_id: 68fcd40071594481936edf0f1d7a8e12
```

6. Create a custom slice for the service.

Add the line Slice=my\_custom\_slice.slice to the [Service] section in the myservice1.service file, created in a previous step, as shown in the following code block:

```
[Service]
Slice=my_custom_slice.slice
Type=oneshot
ExecStart=/usr/lib/systemd/generate_load.sh
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes
[Install]
WantedBy=multi-user.target
```

### **NOT\_SUPPORTED:**

### Use underscores instead of dashes to separate terms in slice names.

In systemd, a dash in a slice name is a special character: in systemd, dashes in slice names are used to describe the full cgroup path to the slice (starting from the root slice). For example, if you specify a slice name as "my-custom-slice.slice", instead of creating a slice of that name, systemd creates the following cgroups path underneath the root slice: my.slice/my-custom.slice/my-custom-slice.slice.

7. After editing the file, ensure systemd reloads its configuration files and then restart the service:

```
sudo systemctl daemon-reload
sudo systemctl restart myservice1
```

8. Run the systemd-cgls command and confirm the service myservice1 is now running under custom slice my custom slice:

```
Lirqbalance.service (#2907)
| → user.invocation_id: 00d64c9b9d224f179496a83536dd60bb
| L1464 /usr/sbin/irqbalance --foreground
```

## Using Drop-in Files

To use a drop-in file to configure resources, perform the following steps:

1. Create the directory for your service drop-in file.



### Tip:

The "drop-in" directory for drop-in files for a service is located at /etc/systemd/system/service\_name.service.d where service\_name is the name of the service.

Continuing with our example with service myservice1, we would run the following command:

```
sudo mkdir -p /etc/systemd/system/myservice1.service.d/
```

2. Create 2 drop-in files called 00-slice.conf and 10-CPUSettings.conf in the myservice1.service.d directory created in the preceding step.



- Multiple drop-in files with different names are applied in lexicographic order.
- These drop-in files take precedence over the service unit file.
- 3. Add the following contents to 00-slice.conf

```
[Service]
Slice=my_custom_slice2.slice
MemoryAccounting=yes
CPUAccounting=yes
```

4. And add the following contents to 10-CPUSettings.conf

```
[Service]
CPUWeight=200
```

- 5. Create a second service (myservice2) and assign it a different CPUWeight to that assigned to myservice1:
  - a. Create file /etc/systemd/system/myservice2.service with the following contents:

```
[Service]
Slice=my_custom_slice2.slice
Type=oneshot
```



```
ExecStart=/usr/lib/systemd/generate_load2.sh
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes

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

b. The service created in the preceding step requires a bash script /usr/lib/systemd/generate load2.sh. Create the file with the following content:

```
#!/bin/bash
for i in {1..4};do while : ; do : ; done & done
```

c. Make the script runnable:

```
sudo chmod +x /usr/lib/systemd/generate load2.sh
```

d. Create a drop in file /etc/systemd/system/myservice2.service.d/10-CPUSettings.conf for myservice2 with the following contents:

```
[Service]
CPUWeight=400
```

**6.** Ensure systemd reloads its configuration files, and restart myservice1, and also enable and start myservices2:

```
sudo systemctl daemon-reload
sudo systemctl restart myservice1
sudo systemctl enable myservice2 --now
```

7. Run the systemd-cgtop command to display control groups ordered by their resource usage. You can see from the following sample output how, in addition to the resource usage of each slice, the systemd-cgtop command displays resource usage breakdown within each slice, so you can use it to confirm your CPU weight has been divided as expected.

systemd-cgtop

```
Control Group
                                                  Tasks %CPU
                                                                 Memory
Input/s Output/s
                                                    228 198.8
712.5M
                                                      8 198.5
my custom slice2.slice
1.8M
my custom slice2.slice/myservice2.service
                                                      4 132.8
944.0K
my custom slice2.slice/myservice1.service
                                                          65.6
976.0K
user.slice
                                                     18
                                                           0.9
43.9M
user.slice/user-1001.slice
                                                           0.9
user.slice/user-1001.slice/session-2.scope
                                                           0.9
```

```
9.4M - - - system.slice 60 0.0
```

## Using systemctl set-property

The systemctl set-property command places the configuration files under the following location:

/etc/systemd/system.control



### **Caution:**

You must not manually edit the files systematl set-property command creates.



The systemctl set-property command does not recognize every resource-control property used in the system-unit and drop-in files covered earlier in this topic.

The following procedure demonstrates how you can use the systemctl set-property command to configure resource allocation:

 Continuing with our example, create another service file at location /etc/systemd/ system/myservice3.service with the following content:

```
[Service]
Type=oneshot
ExecStart=/usr/lib/systemd/generate_load3.sh
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes
[Install]
WantedBy=multi-user.target
```

2. Set the slice for the service to be my\_custom\_slice2 (the same slice used by the services created in from earlier steps) by adding the following line to the [Service] section in the myservice3.service file:

Slice=my\_custom\_slice2.slice



The slice must be set in the service-unit file because the systemctl set-property command does not recognize the Slice property.

3. The service created in the preceding step requires a bash script /usr/lib/systemd/generate load3.sh. Create the file with the following content:

```
#!/bin/bash
for i in {1..4};do while : ; do : ; done & done
```

4. Make the script runnable:

```
sudo chmod +x /usr/lib/systemd/generate load3.sh
```

5. Ensure systemd reloads its configuration files, and then enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable myservice3 --now
```

- **6.** Run systemd-cgtop to confirm all 3 services, myservice1, myservice2, and myservice3, are all running in the same slice.
- 7. Use systemctl set-property command to set the CPUWeight for myservice3 to 800:

```
sudo systemctl set-property myservice3.service CPUWeight=800
```

8. Confirm that a drop-in file has been created for you under /etc/systemd/ system.control/myservice3.service.d. However, you must not edit the file:

```
cat /etc/systemd/system.control/myservice3.service.d/50-CPUWeight.conf
```

```
# This is a drop-in unit file extension, created via "systemctl set-
property"
# or an equivalent operation. Do not edit.
[Service]
CPUWeight=800
```

9. Ensure systemd reloads its configuration files, and restart all the services:

```
sudo systemctl daemon-reload
sudo systemctl restart myservice1
sudo systemctl restart myservice2
sudo systemctl restart myservice3
```

**10.** Run the systemd-cgtop command to confirm your CPU weight has been divided as expected:

```
systemd-cgtop
```

Control Group	Tasks	%CPU
Memory Input/s Output/s		
	235	200.0
706.1M		
my_custom_slice2.slice	12	198.4
2.9M		
<pre>my_custom_slice2.slice/myservice3.service</pre>	4	112.7
976.0K		

<pre>my_custom_slice2.slice/myservice2.service</pre>	4	56.9
996.0K		
<pre>my_custom_slice2.slice/myservice1.service</pre>		28.8
988.0K		
user.slice	18	0.9
44.1M		
user.slice/user-1001.slice	6	0.9
13.9M		
user.slice/user-1001.slice/session-2.scope	4	0.9
9 5M		

